# Red Hat OpenShift Service on AWS 4

## Authentication and authorization

Securing Red Hat OpenShift Service on AWS clusters.

Last Updated: 2023-05-25

# Red Hat OpenShift Service on AWS 4 Authentication and authorization

Securing Red Hat OpenShift Service on AWS clusters.

## Legal Notice



## Abstract

This document provides information about securing Red Hat OpenShift Service on AWS (ROSA) clusters.

# Table of Contents

# CHAPTER 1. ASSUMING AN AWS IAM ROLE FOR A SERVICE ACCOUNT

Red Hat OpenShift Service on AWS clusters that use the AWS Security Token Service (STS) include a pod identity webhook for use with pods that run in user-defined projects.

You can use the pod identity webhook to enable a service account to automatically assume an AWS Identity and Access Management (IAM) role in your own pods. If the assumed IAM role has the required AWS permissions, the pods can run AWS SDK operations by using temporary STS credentials.

## 1.1. UNDERSTANDING THE POD IDENTITY WEBHOOK WORKFLOW IN USER-DEFINED PROJECTS

When you install a Red Hat OpenShift Service on AWS cluster that uses the AWS Security Token Service (STS), pod identity webhook resources are included by default.

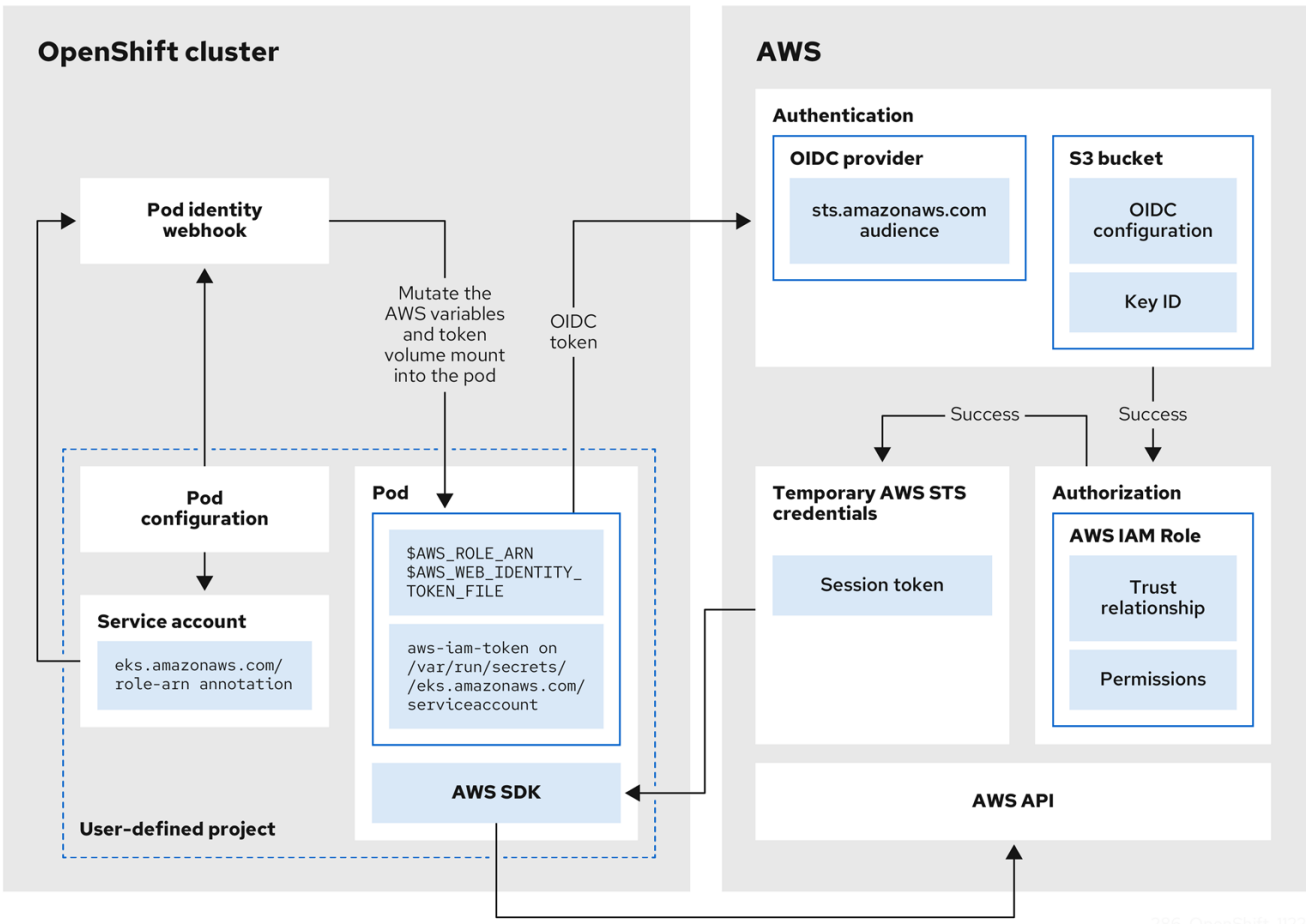
You can use the pod identity webhook to enable a service account in a user-defined project to assume an AWS Identity and Access Management (IAM) role in a pod in the same project. When the IAM role is assumed, temporary STS credentials are provided for use by the service account in the pod. If the assumed role has the necessary AWS privileges, the service account can run AWS SDK operations in the pod.

To enable the pod identity webhook for a pod, you must create a service account with an **eks.amazonaws.com/role-arn** annotation in your project. The annotation must reference the Amazon Resource Name (ARN) of the AWS IAM role that you want the service account to assume. You must also reference the service account in your **Pod** specification and deploy the pod in the same project as the service account.

### Pod identity webhook workflow in user-defined projects

The following diagram illustrates the pod identity webhook workflow in user-defined projects:

Figure 1.1. Pod identity webhook workflow in user-defined projects




The workflow has the following stages:

1. Within a user-defined project, a user creates a service account that includes an **eks.amazonaws.com/role-arn** annotation. The annotation points to the ARN of the AWS IAM role that you want your service account to assume.

2. When a pod is deployed in the same project using a configuration that references the annotated service account, the pod identity webhook mutates the pod. The mutation injects the following components into the pod without the need to specify them in your **Pod** or **Deployment** resource configurations:

   - An **$AWS_ARN_ROLE** environment variable that contains the ARN for the IAM role that has the permissions required to run AWS SDK operations.

   - An **$AWS_WEB_IDENTITY_TOKEN_FILE** environment variable that contains the full path in the pod to the OpenID Connect (OIDC) token for the service account. The full path is **/var/run/secrets/eks.amazonaws.com/serviceaccount/token**.

   - An **aws-iam-token** volume mounted on the mount point **/var/run/secrets/eks.amazonaws.com/serviceaccount**. An OIDC token file named **token** is contained in the volume.

3. The OIDC token is passed from the pod to the OIDC provider. The provider authenticates the service account identity if the following requirements are met:

   - The identity signature is valid and signed by the private key.

- The **sts.amazonaws.com** audience is listed in the OIDC token and matches the audience configured in the OIDC provider.

**NOTE**

The pod identity webhook applies the **sts.amazonaws.com** audience to the OIDC token by default.

In Red Hat OpenShift Service on AWS with STS clusters, the OIDC provider is created during install and set as the service account issuer by default. The **sts.amazonaws.com** audience is set by default in the OIDC provider.

- The OIDC token has not expired.
- The issuer value in the token contains the URL for the OIDC provider.

4. If the project and service account are in the scope of the trust policy for the IAM role that is being assumed, then authorization succeeds.
5. After successful authentication and authorization, temporary AWS STS credentials in the form of a session token are passed to the pod for use by the service account. By using the credentials, the service account is temporarily granted the AWS permissions enabled in the IAM role.
6. When you run AWS SDK operations in the pod, the service account provides the temporary STS credentials to the AWS API to verify its identity.

## 1.2. ASSUMING AN AWS IAM ROLE IN YOUR OWN PODS

Follow the procedures in this section to enable a service account to assume an AWS Identity and Access Management (IAM) role in a pod deployed in a user-defined project.

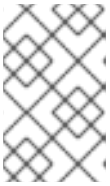
You can create the required resources, including an AWS IAM role, a service account, a container image that includes an AWS SDK, and a pod deployed by using the image. In the example, the AWS Boto3 SDK for Python is used. You can also verify that the pod identity webhook mutates the AWS environment variables, the volume mount, and the token volume into your pod. Additionally, you can check that the service account assumes the AWS IAM role in your pod and can successfully run AWS SDK operations.

### 1.2.1. Setting up an AWS IAM role for a service account

Create an AWS Identity and Access Management (IAM) role to be assumed by a service account in your Red Hat OpenShift Service on AWS cluster. Attach the permissions that are required by your service account to run AWS SDK operations in a pod.

**Prerequisites**

- You have the permissions required to install and configure IAM roles in your AWS account.
- You have access to a Red Hat OpenShift Service on AWS cluster that uses the AWS Security Token Service (STS). Admin-level user privileges are not required.
- You have the Amazon Resource Name (ARN) for the OpenID Connect (OIDC) provider that is configured as the service account issuer in your Red Hat OpenShift Service on AWS with STS cluster.



NOTE

In Red Hat OpenShift Service on AWS with STS clusters, the OIDC provider is created during install and set as the service account issuer by default. If you do not know the OIDC provider ARN, contact your cluster administrator.

- You have installed the AWS CLI (**aws**).

**Procedure**

1. Create a file named **trust-policy.json** with the following JSON configuration:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Federated": "<oidc_provider_arn>" 1
         },
         "Action": "sts:AssumeRoleWithWebIdentity",
         "Condition": {
            "StringEquals": {
               "<oidc_provider_name>:sub": "system:serviceaccount:<project_name>:
<service_account_name>" 2
            }
         }
      }
   ]
}
```

1. Replace **<oidc_provider_arn>** with the ARN of your OIDC provider, for example **arn:aws:iam::<aws_account_id>:oidc-provider/rh-oidc.s3.us-east-1.amazonaws.com/1v3r0n44npxu4g58so46aeohduomfres**.

2. Limits the role to the specified project and service account. Replace **<oidc_provider_name>** with the name of your OIDC provider, for example **rh-oidc.s3.us-east-1.amazonaws.com/1v3r0n44npxu4g58so46aeohduomfres**. Replace **<project_name>:<service_account_name>** with your project name and service account name, for example **my-project:test-service-account**.

NOTE

Alternatively, you can limit the role to any service account within the specified project by using **"<oidc_provider_name>:sub": "system:serviceaccount:<project_name>:*"**. If you supply the * wildcard, you must replace **StringEquals** with **StringLike** in the preceding line.

2. Create an AWS IAM role that uses the trust policy that is defined in the **trust-policy.json** file:

```
$ aws iam create-role \
   --role-name <aws_iam_role_name> \ 1
   --assume-role-policy-document file://trust-policy.json 2
```

1. Replace **<aws_iam_role_name>** with the name of your IAM role, for example **pod-identity-test-role**.
2. References the **trust-policy.json** file that you created in the preceding step.

**Example output:**

```
ROLE    arn:aws:iam::<aws_account_id>:role/<aws_iam_role_name>        2022-09-
28T12:03:17+00:00       /       AQWMS3TB4Z2N3SH7675JK   <aws_iam_role_name>
ASSUMEROLEPOLICYDOCUMENT        2012-10-17
STATEMENT       sts:AssumeRoleWithWebIdentity   Allow
STRINGEQUALS    system:serviceaccount:<project_name>:<service_account_name>
PRINCIPAL       <oidc_provider_arn>
```

Retain the ARN for the role in the output. The format of the role ARN is **arn:aws:iam::<aws_account_id>:role/<aws_iam_role_name>**.

3. Attach any managed AWS permissions that are required when the service account runs AWS SDK operations in your pod:

```
$ aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/ReadOnlyAccess \ 1
    --role-name <aws_iam_role_name> 2
```

1. The policy in this example adds read-only access permissions to the IAM role.
2. Replace **<aws_iam_role_name>** with the name of the IAM role that you created in the preceding step.

4. Optional: Add custom attributes or a permissions boundary to the role. For more information, see Creating a role to delegate permissions to an AWS service in the AWS documentation.

### 1.2.2. Creating a service account in your project

Add a service account in your user-defined project. Include an **eks.amazonaws.com/role-arn** annotation in the service account configuration that references the Amazon Resource Name (ARN) for the AWS Identity and Access Management (IAM) role that you want the service account to assume.

**Prerequisites**

- You have created an AWS IAM role for your service account. For more information, see *Setting up an AWS IAM role for a service account*.
- You have access to a Red Hat OpenShift Service on AWS with AWS Security Token Service (STS) cluster. Admin-level user privileges are not required.
- You have installed the OpenShift CLI (**oc**).

**Procedure**

1. In your Red Hat OpenShift Service on AWS cluster, create a project:

```
$ oc new-project <project_name> 1
```

1. Replace **<project_name>** with the name of your project. The name must match the project name that you specified in your AWS IAM role configuration.

NOTE

You are automatically switched to the project when it is created.

2. Create a file named **test-service-account.yaml** with the following service account configuration:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: <service_account_name> 1
  namespace: <project_name> 2
  annotations:
    eks.amazonaws.com/role-arn: "<aws_iam_role_arn>" 3
```

1. Replace **<service_account_name>** with the name of your service account. The name must match the service account name that you specified in your AWS IAM role configuration.
2. Replace **<project_name>** with the name of your project. The name must match the project name that you specified in your AWS IAM role configuration.
3. Specifies the ARN of the AWS IAM role that the service account assumes for use within your pod. Replace **<aws_iam_role_arn>** with the ARN for the AWS IAM role that you created for your service account. The format of the role ARN is **arn:aws:iam::<aws_account_id>:role/<aws_iam_role_name>**.

3. Create the service account in your project:

```
$ oc create -f test-service-account.yaml
```

**Example output:**

```
serviceaccount/<service_account_name> created
```

4. Review the details of the service account:

```
$ oc describe serviceaccount <service_account_name> 1
```

1. Replace **<service_account_name>** with the name of your service account.

**Example output:**

```
Name:                <service_account_name> 1
Namespace:           <project_name> 2
Labels:              <none>
```

```
Annotations:        eks.amazonaws.com/role-arn: <aws_iam_role_arn> 3
Image pull secrets:  <service_account_name>-dockercfg-rnjkq
Mountable secrets:   <service_account_name>-dockercfg-rnjkq
Tokens:              <service_account_name>-token-4gbjp
Events:              <none>
```

1. Specifies the name of the service account.
2. Specifies the project that contains the service account.
3. Lists the annotation for the ARN of the AWS IAM role that the service account assumes.

### 1.2.3. Creating an example AWS SDK container image

The steps in this procedure provide an example method to create a container image that includes an AWS SDK.

The example steps use Podman to create the container image and Quay.io to host the image. For more information about Quay.io, see Getting Started with Quay.io. The container image can be used to deploy pods that can run AWS SDK operations.

**NOTE**

In this example procedure, the AWS Boto3 SDK for Python is installed into a container image. For more information about installing and using the AWS Boto3 SDK, see the AWS Boto3 documentation. For details about other AWS SDKs, see AWS SDKs and Tools Reference Guide in the AWS documentation.

**Prerequisites**

- You have installed Podman on your installation host.
- You have a Quay.io user account.

**Procedure**

1. Add the following configuration to a file named **Containerfile**:

```
FROM ubi9/ubi 1
RUN dnf makecache && dnf install -y python3-pip && dnf clean all && pip3 install
boto3>=1.15.0 2
```

1. Specifies the Red Hat Universal Base Image version 9.
2. Installs the AWS Boto3 SDK by using the **pip** package management system. In this example, AWS Boto3 SDK version 1.15.0 or later is installed.

2. From the directory that contains the file, build a container image named **awsboto3sdk**:

```
$ podman build -t awsboto3sdk .
```

3. Log in to Quay.io:

```
$ podman login quay.io
```

4. Tag the image in preparation for the upload to Quay.io:

```
$ podman tag localhost/awsboto3sdk quay.io/<quay_username>/awsboto3sdk:latest 1
```

1 Replace **<quay_username>** with your Quay.io username.

5. Push the tagged container image to Quay.io:

```
$ podman push quay.io/<quay_username>/awsboto3sdk:latest 1
```

1 Replace **<quay_username>** with your Quay.io username.

6. Make the Quay.io repository that contains the image public. This publishes the image so that it can be used to deploy a pod in your Red Hat OpenShift Service on AWS cluster:

   a. On https://quay.io/, navigate to the **Repository Settings** page for repository that contains the image.

   b. Click **Make Public** to make the repository publicly available.

### 1.2.4. Deploying a pod that includes an AWS SDK

Deploy a pod in a user-defined project from a container image that includes an AWS SDK. In your pod configuration, specify the service account that includes the **eks.amazonaws.com/role-arn** annotation.

With the service account reference in place for your pod, the pod identity webhook injects the AWS environment variables, the volume mount, and the token volume into your pod. The pod mutation enables the service account to automatically assume the AWS IAM role in the pod.

**Prerequisites**

- You have created an AWS Identity and Access Management (IAM) role for your service account. For more information, see *Setting up an AWS IAM role for a service account* .
- You have access to a Red Hat OpenShift Service on AWS cluster that uses the AWS Security Token Service (STS). Admin-level user privileges are not required.
- You have installed the OpenShift CLI (**oc**).
- You have created a service account in your project that includes an **eks.amazonaws.com/role-arn** annotation that references the Amazon Resource Name (ARN) for the IAM role that you want the service account to assume.
- You have a container image that includes an AWS SDK and the image is available to your cluster. For detailed steps, see *Creating an example AWS SDK container image* .

**NOTE**

In this example procedure, the AWS Boto3 SDK for Python is used. For more information about installing and using the AWS Boto3 SDK, see the AWS Boto3 documentation. For details about other AWS SDKs, see AWS SDKs and Tools Reference Guide in the AWS documentation.

**Procedure**

1. Create a file named **awsboto3sdk-pod.yaml** with the following pod configuration:

```
apiVersion: v1
kind: Pod
metadata:
  namespace: <project_name> 1
  name: awsboto3sdk 2
spec:
  serviceAccountName: <service_account_name> 3
  containers:
  - name: awsboto3sdk
    image: quay.io/<quay_username>/awsboto3sdk:latest 4
    command:
    - /bin/bash
    - "-c"
    - "sleep 100000" 5
  terminationGracePeriodSeconds: 0
  restartPolicy: Never
```

1. Replace **<project_name>** with the name of your project. The name must match the project name that you specified in your AWS IAM role configuration.
2. Specifies the name of the pod.
3. Replace **<service_account_name>** with the name of the service account that is configured to assume the AWS IAM role. The name must match the service account name that you specified in your AWS IAM role configuration.
4. Specifies the location of your **awsboto3sdk** container image. Replace **<quay_username>** with your Quay.io username.
5. In this example pod configuration, this line keeps the pod running for 100000 seconds to enable verification testing in the pod directly. For detailed verification steps, see *Verifying the assumed IAM role in your pod*.

2. Deploy an **awsboto3sdk** pod:

```
$ oc create -f awsboto3sdk-pod.yaml
```

**Example output:**

```
pod/awsboto3sdk created
```

### 1.2.5. Verifying the assumed IAM role in your pod

After deploying an **awsboto3sdk** pod in your project, verify that the pod identity webhook has mutated the pod. Check that the required AWS environment variables, volume mount, and OIDC token volume are present within the pod.

You can also verify that the service account assumes the AWS Identity and Access Management (IAM) role for your AWS account when you run AWS SDK operations in the pod.

### Prerequisites

- You have created an AWS IAM role for your service account. For more information, see *Setting up an AWS IAM role for a service account*.
- You have access to a Red Hat OpenShift Service on AWS cluster that uses the AWS Security Token Service (STS). Admin-level user privileges are not required.
- You have installed the OpenShift CLI (**oc**).
- You have created a service account in your project that includes an **eks.amazonaws.com/role-arn** annotation that references the Amazon Resource Name (ARN) for the IAM role that you want the service account to assume.
- You have deployed a pod in your user-defined project that includes an AWS SDK. The pod references the service account that uses the pod identity webhook to assume the AWS IAM role required to run the AWS SDK operations. For detailed steps, see *Deploying a pod that includes an AWS SDK*.

**NOTE**

In this example procedure, a pod that includes the AWS Boto3 SDK for Python is used. For more information about installing and using the AWS Boto3 SDK, see the AWS Boto3 documentation. For details about other AWS SDKs, see AWS SDKs and Tools Reference Guide in the AWS documentation.

### Procedure

1. Verify that the AWS environment variables, the volume mount, and the OIDC token volume are listed in the description of the deployed **awsboto3sdk** pod:

```
$ oc describe pod awsboto3sdk
```

**Example output:**

```
Name:         awsboto3sdk
Namespace:    <project_name>
...
Containers:
  awsboto3sdk:
    ...
    Environment:
      AWS_ROLE_ARN:                 <aws_iam_role_arn> 1
      AWS_WEB_IDENTITY_TOKEN_FILE:
/var/run/secrets/eks.amazonaws.com/serviceaccount/token 2
    Mounts:
      /var/run/secrets/eks.amazonaws.com/serviceaccount from aws-iam-token (ro) 3
```

```
...
Volumes:
  aws-iam-token: 4
    Type:                 Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  86400
...
```

1. Lists the **AWS_ROLE_ARN** environment variable that was injected into the pod by the pod identity webhook. The variable contains the ARN of the AWS IAM role to be assumed by the service account.
2. Lists the **AWS_WEB_IDENTITY_TOKEN_FILE** environment variable that was injected into the pod by the pod identity webhook. The variable contains the full path of the OIDC token that is used to verify the service account identity.
3. Lists the volume mount that was injected into the pod by the pod identity webhook.
4. Lists the **aws-iam-token** volume that is mounted onto the **/var/run/secrets/eks.amazonaws.com/serviceaccount** mount point. The volume contains the OIDC token that is used to authenticate the service account to assume the AWS IAM role.

2. Start an interactive terminal in the **awsboto3sdk** pod:

```
$ oc exec -ti awsboto3sdk -- /bin/sh
```

3. In the interactive terminal for the pod, verify that the **$AWS_ROLE_ARN** environment variable was mutated into the pod by the pod identity webhook:

```
$ echo $AWS_ROLE_ARN
```

**Example output:**

```
arn:aws:iam::<aws_account_id>:role/<aws_iam_role_name> 1
```

1. The output must specify the ARN for the AWS IAM role that has the permissions required to run AWS SDK operations.

4. In the interactive terminal for the pod, verify that the **$AWS_WEB_IDENTITY_TOKEN_FILE** environment variable was mutated into the pod by the pod identity webhook:

```
$ echo $AWS_WEB_IDENTITY_TOKEN_FILE
```

**Example output:**

```
/var/run/secrets/eks.amazonaws.com/serviceaccount/token 1
```

1. The output must specify the full path in the pod to the OIDC token for the service account.

5. In the interactive terminal for the pod, verify that the **aws-iam-token** volume mount containing the OIDC token file was mounted by the pod identity webhook:

```
$ mount | grep -is 'eks.amazonaws.com'
```

**Example output:**

```
tmpfs on /run/secrets/eks.amazonaws.com/serviceaccount type tmpfs
(ro,relatime,seclabel,size=13376888k)
```

6. In the interactive terminal for the pod, verify that an OIDC token file named **token** is present on the **/var/run/secrets/eks.amazonaws.com/serviceaccount/** mount point:

```
$ ls /var/run/secrets/eks.amazonaws.com/serviceaccount/token
```

**Example output:**

```
/var/run/secrets/eks.amazonaws.com/serviceaccount/token 1
```

1 The OIDC token file in the **aws-iam-token** volume that was mounted in the pod by the pod identity webhook. The token is used to authenticate the identity of the service account in AWS.

7. In the pod, verify that AWS Boto3 SDK operations run successfully:

   a. In the interactive terminal for the pod, start a Python 3 shell:

```
$ python3
```

   b. In the Python 3 shell, import the **boto3** module:

```
>>> import boto3
```

   c. Create a variable that includes the Boto3 **s3** service resource:

```
>>> s3 = boto3.resource('s3')
```

   d. Print the names of all of the S3 buckets in your AWS account:

```
>>> for bucket in s3.buckets.all():
...     print(bucket.name)
...
```

**Example output:**

```
<bucket_name>
<bucket_name>
<bucket_name>
...
```

If the service account successfully assumed the AWS IAM role, the output lists all of the S3 buckets that are available in your AWS account.

## 1.3. ADDITIONAL RESOURCES

- For more information about using AWS IAM roles with service accounts, see IAM roles for service accounts in the AWS documentation.
- For information about AWS IAM role delegation, see Creating a role to delegate permissions to an AWS service in the AWS documentation.
- For details about AWS SDKs, see AWS SDKs and Tools Reference Guide in the AWS documentation.
- For more information about installing and using the AWS Boto3 SDK for Python, see the AWS Boto3 documentation.
- For general information about webhook admission plugins for OpenShift, see Webhook admission plugins in the OpenShift Container Platform documentation.

# CHAPTER 2. MANAGING SECURITY CONTEXT CONSTRAINTS

In Red Hat OpenShift Service on AWS, you can use security context constraints (SCCs) to control permissions for the pods in your cluster.

Default SCCs are created during installation and when you install some Operators or other components. As a cluster administrator, you can also create your own SCCs by using the OpenShift CLI (**oc**).

IMPORTANT

Do not modify the default SCCs. Customizing the default SCCs can lead to issues when some of the platform pods deploy or ROSA is upgraded. Additionally, the default SCC values are reset to the defaults during some cluster upgrades, which discards all customizations to those SCCs.

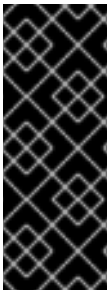
Instead of modifying the default SCCs, create and modify your own SCCs as needed. For detailed steps, see Creating security context constraints.

## 2.1. ABOUT SECURITY CONTEXT CONSTRAINTS

Similar to the way that RBAC resources control user access, administrators can use security context constraints (SCCs) to control permissions for pods. These permissions determine the actions that a pod can perform and what resources it can access. You can use SCCs to define a set of conditions that a pod must run with to be accepted into the system.

Security context constraints allow an administrator to control:

- Whether a pod can run privileged containers with the **allowPrivilegedContainer** flag
- Whether a pod is constrained with the **allowPrivilegeEscalation** flag
- The capabilities that a container can request
- The use of host directories as volumes
- The SELinux context of the container
- The container user ID
- The use of host namespaces and networking
- The allocation of an **FSGroup** that owns the pod volumes
- The configuration of allowable supplemental groups
- Whether a container requires write access to its root file system
- The usage of volume types
- The configuration of allowable **seccomp** profiles



IMPORTANT

Do not set the **openshift.io/run-level** label on any namespaces in Red Hat OpenShift Service on AWS. This label is for use by internal Red Hat OpenShift Service on AWS components to manage the startup of major API groups, such as the Kubernetes API server and OpenShift API server. If the **openshift.io/run-level** label is set, no SCCs are applied to pods in that namespace, causing any workloads running in that namespace to be highly privileged.

### 2.1.1. Default security context constraints

The cluster contains several default security context constraints (SCCs) as described in the table below. Additional SCCs might be installed when you install Operators or other components to Red Hat OpenShift Service on AWS.

IMPORTANT

Do not modify the default SCCs. Customizing the default SCCs can lead to issues when some of the platform pods deploy or ROSA is upgraded. Additionally, the default SCC values are reset to the defaults during some cluster upgrades, which discards all customizations to those SCCs.

Instead of modifying the default SCCs, create and modify your own SCCs as needed. For detailed steps, see *Creating security context constraints*.

**Table 2.1. Default security context constraints**

| Security context constraint | Description |
|---|---|
| **anyuid** | Provides all features of the **restricted** SCC, but allows users to run with any UID and any GID. |
| **hostaccess** | Allows access to all host namespaces but still requires pods to be run with a UID and SELinux context that are allocated to the namespace.<br><br>**WARNING**<br>This SCC allows host access to namespaces, file systems, and PIDs. It should only be used by trusted pods. Grant with caution. |

| Security context constraint | Description |
| --- | --- |
| **hostmount-anyuid** | Provides all the features of the **restricted** SCC, but allows host mounts and running as any UID and any GID on the system.<br><br>**WARNING**<br><br>This SCC allows host file system access as any UID, including UID 0. Grant with caution. |
| **hostnetwork** | Allows using host networking and host ports but still requires pods to be run with a UID and SELinux context that are allocated to the namespace.<br><br>**WARNING**<br><br>If additional workloads are run on control plane hosts, use caution when providing access to **hostnetwork**. A workload that runs **hostnetwork** on a control plane host is effectively root on the cluster and must be trusted accordingly. |
| **hostnetwork-v2** | Like the **hostnetwork** SCC, but with the following differences:<br><br>• **ALL** capabilities are dropped from containers.<br>• The **NET_BIND_SERVICE** capability can be added explicitly.<br>• **seccompProfile** is set to **runtime/default** by default.<br>• **allowPrivilegeEscalation** must be unset or set to**false** in security contexts. |

| Security context constraint | Description |
| --- | --- |
| **node-exporter** | Used for the Prometheus node exporter.<br><br>**WARNING**<br><br>This SCC allows host file system access as any UID, including UID 0. Grant with caution. |
| **nonroot** | Provides all features of the **restricted** SCC, but allows users to run with any non-root UID. The user must specify the UID or it must be specified in the manifest of the container runtime. |
| **nonroot-v2** | Like the **nonroot** SCC, but with the following differences:<br>• **ALL** capabilities are dropped from containers.<br>• The **NET_BIND_SERVICE** capability can be added explicitly.<br>• **seccompProfile** is set to **runtime/default** by default.<br>• **allowPrivilegeEscalation** must be unset or set to**false** in security contexts. |

| Security context constraint | Description |
| --- | --- |
| **privileged** | Allows access to all privileged and host features and the ability to run as any user, any group, any FSGroup, and with any SELinux context.<br><br>**WARNING**<br>This is the most relaxed SCC and should be used only for cluster administration. Grant with caution.<br><br>The **privileged** SCC allows:<br>• Users to run privileged pods<br>• Pods to mount host directories as volumes<br>• Pods to run as any user<br>• Pods to run with any MCS label<br>• Pods to use the host's IPC namespace<br>• Pods to use the host's PID namespace<br>• Pods to use any FSGroup<br>• Pods to use any supplemental group<br>• Pods to use any seccomp profiles<br>• Pods to request any capabilities<br><br>**NOTE**<br>Setting **privileged: true** in the pod specification does not necessarily select the **privileged** SCC. The SCC that has **allowPrivilegedContainer: true** and has the highest prioritization will be chosen if the user has the permissions to use it. |

| Security context constraint | Description |
|---|---|
| **restricted** | Denies access to all host features and requires pods to be run with a UID, and SELinux context that are allocated to the namespace.<br><br>The **restricted** SCC:<br><br>• Ensures that pods cannot run as privileged<br>• Ensures that pods cannot mount host directory volumes<br>• Requires that a pod is run as a user in a pre-allocated range of UIDs<br>• Requires that a pod is run with a pre-allocated MCS label<br>• Allows pods to use any FSGroup<br>• Allows pods to use any supplemental group<br><br>In clusters that were upgraded from Red Hat OpenShift Service on AWS 4.10 or earlier, this SCC is available for use by any authenticated user. The **restricted** SCC is no longer available to users of new Red Hat OpenShift Service on AWS 4.11 or later installations, unless the access is explicitly granted. |
| **restricted-v2** | Like the **restricted** SCC, but with the following differences:<br><br>• **ALL** capabilities are dropped from containers.<br>• The **NET_BIND_SERVICE** capability can be added explicitly.<br>• **seccompProfile** is set to **runtime/default** by default.<br>• **allowPrivilegeEscalation** must be unset or set to**false** in security contexts.<br><br>This is the most restrictive SCC provided by a new installation and will be used by default for authenticated users.<br><br>  <br>NOTE<br><br>The **restricted-v2** SCC is the most restrictive of the SCCs that is included by default with the system. However, you can create a custom SCC that is even more restrictive. For example, you can create an SCC that restricts **readOnlyRootFilesystem** to **true**. |

### 2.1.2. Security context constraints settings

Security context constraints (SCCs) are composed of settings and strategies that control the security features a pod has access to. These settings fall into three categories:

| Category | Description |
|---|---|
| Controlled by a boolean | Fields of this type default to the most restrictive value. For example, **AllowPrivilegedContainer** is always set to **false** if unspecified. |

| Category | Description |
| --- | --- |
| Controlled by an allowable set | Fields of this type are checked against the set to ensure their value is allowed. |
| Controlled by a strategy | Items that have a strategy to generate a value provide:<br>• A mechanism to generate the value, and<br>• A mechanism to ensure that a specified value falls into the set of allowable values. |

CRI-O has the following default list of capabilities that are allowed for each container of a pod:

- **CHOWN**
- **DAC_OVERRIDE**
- **FSETID**
- **FOWNER**
- **SETGID**
- **SETUID**
- **SETPCAP**
- **NET_BIND_SERVICE**
- **KILL**

The containers use the capabilities from this default list, but pod manifest authors can alter the list by requesting additional capabilities or removing some of the default behaviors. Use the **allowedCapabilities**, **defaultAddCapabilities**, and **requiredDropCapabilities** parameters to control such requests from the pods. With these parameters you can specify which capabilities can be requested, which ones must be added to each container, and which ones must be forbidden, or dropped, from each container.

NOTE

You can drop all capabilites from containers by setting the **requiredDropCapabilities** parameter to **ALL**. This is what the **restricted-v2** SCC does.

### 2.1.3. Security context constraints strategies

**RunAsUser**

- **MustRunAs** - Requires a **runAsUser** to be configured. Uses the configured **runAsUser** as the default. Validates against the configured **runAsUser**.

  **Example MustRunAs snippet**

```
...
runAsUser:
  type: MustRunAs
  uid: <id>
...
```

- **MustRunAsRange** - Requires minimum and maximum values to be defined if not using pre-allocated values. Uses the minimum as the default. Validates against the entire allowable range.

  Example **MustRunAsRange** snippet

```
...
runAsUser:
  type: MustRunAsRange
  uidRangeMax: <maxvalue>
  uidRangeMin: <minvalue>
...
```

- **MustRunAsNonRoot** - Requires that the pod be submitted with a non-zero **runAsUser** or have the **USER** directive defined in the image. No default provided.

  Example **MustRunAsNonRoot** snippet

```
...
runAsUser:
  type: MustRunAsNonRoot
...
```

- **RunAsAny** - No default provided. Allows any **runAsUser** to be specified.

  Example **RunAsAny** snippet

```
...
runAsUser:
  type: RunAsAny
...
```

SELinuxContext

- **MustRunAs** - Requires **seLinuxOptions** to be configured if not using pre-allocated values. Uses **seLinuxOptions** as the default. Validates against **seLinuxOptions**.

- **RunAsAny** - No default provided. Allows any **seLinuxOptions** to be specified.

SupplementalGroups

- **MustRunAs** - Requires at least one range to be specified if not using pre-allocated values. Uses the minimum value of the first range as the default. Validates against all ranges.

- **RunAsAny** - No default provided. Allows any **supplementalGroups** to be specified.

FSGroup

- **MustRunAs** - Requires at least one range to be specified if not using pre-allocated values. Uses the minimum value of the first range as the default. Validates against the first ID in the first range.
- **RunAsAny** - No default provided. Allows any **fsGroup** ID to be specified.

### 2.1.4. Controlling volumes

The usage of specific volume types can be controlled by setting the **volumes** field of the SCC.

The allowable values of this field correspond to the volume sources that are defined when creating a volume:

- **awsElasticBlockStore**
- **azureDisk**
- **azureFile**
- **cephFS**
- **cinder**
- **configMap**
- **csi**
- **downwardAPI**
- **emptyDir**
- **fc**
- **flexVolume**
- **flocker**
- **gcePersistentDisk**
- **gitRepo**
- **glusterfs**
- **hostPath**
- **iscsi**
- **nfs**
- **persistentVolumeClaim**
- **photonPersistentDisk**
- **portworxVolume**
- **projected**
- **quobyte**

- **rbd**
- **scaleIO**
- **secret**
- **storageos**
- **vsphereVolume**
- ***** (A special value to allow the use of all volume types.)
- **none** (A special value to disallow the use of all volumes types. Exists only for backwards compatibility.)

The recommended minimum set of allowed volumes for new SCCs are **configMap**, **downwardAPI**, **emptyDir**, **persistentVolumeClaim**, **secret**, and **projected**.

NOTE

This list of allowable volume types is not exhaustive because new types are added with each release of Red Hat OpenShift Service on AWS.

NOTE

For backwards compatibility, the usage of **allowHostDirVolumePlugin** overrides settings in the **volumes** field. For example, if **allowHostDirVolumePlugin** is set to false but allowed in the **volumes** field, then the **hostPath** value will be removed from **volumes**.

### 2.1.5. Admission control

*Admission control* with SCCs allows for control over the creation of resources based on the capabilities granted to a user.

In terms of the SCCs, this means that an admission controller can inspect the user information made available in the context to retrieve an appropriate set of SCCs. Doing so ensures the pod is authorized to make requests about its operating environment or to generate a set of constraints to apply to the pod.

The set of SCCs that admission uses to authorize a pod are determined by the user identity and groups that the user belongs to. Additionally, if the pod specifies a service account, the set of allowable SCCs includes any constraints accessible to the service account.

Admission uses the following approach to create the final security context for the pod:

1. Retrieve all SCCs available for use.
2. Generate field values for security context settings that were not specified on the request.
3. Validate the final settings against the available constraints.

If a matching set of constraints is found, then the pod is accepted. If the request cannot be matched to an SCC, the pod is rejected.

A pod must validate every field against the SCC. The following are examples for just two of the fields that must be validated:

NOTE

These examples are in the context of a strategy using the pre-allocated values.

**An FSGroup SCC strategy of MustRunAs**

If the pod defines a **fsGroup** ID, then that ID must equal the default **fsGroup** ID. Otherwise, the pod is not validated by that SCC and the next SCC is evaluated.

If the **SecurityContextConstraints.fsGroup** field has value **RunAsAny** and the pod specification omits the **Pod.spec.securityContext.fsGroup**, then this field is considered valid. Note that it is possible that during validation, other SCC settings will reject other pod fields and thus cause the pod to fail.

**A SupplementalGroups SCC strategy of MustRunAs**

If the pod specification defines one or more **supplementalGroups** IDs, then the pod's IDs must equal one of the IDs in the namespace's **openshift.io/sa.scc.supplemental-groups** annotation. Otherwise, the pod is not validated by that SCC and the next SCC is evaluated.

If the **SecurityContextConstraints.supplementalGroups** field has value **RunAsAny** and the pod specification omits the **Pod.spec.securityContext.supplementalGroups**, then this field is considered valid. Note that it is possible that during validation, other SCC settings will reject other pod fields and thus cause the pod to fail.

### 2.1.6. Security context constraints prioritization

Security context constraints (SCCs) have a priority field that affects the ordering when attempting to validate a request by the admission controller.

A priority value of **0** is the lowest possible priority. A nil priority is considered a **0**, or lowest, priority. Higher priority SCCs are moved to the front of the set when sorting.

When the complete set of available SCCs is determined, the SCCs are ordered in the following manner:

1. The highest priority SCCs are ordered first.
2. If the priorities are equal, the SCCs are sorted from most restrictive to least restrictive.
3. If both the priorities and restrictions are equal, the SCCs are sorted by name.

By default, the **anyuid** SCC granted to cluster administrators is given priority in their SCC set. This allows cluster administrators to run pods as any user by specifying **RunAsUser** in the pod's **SecurityContext**.

## 2.2. ABOUT PRE-ALLOCATED SECURITY CONTEXT CONSTRAINTS VALUES

The admission controller is aware of certain conditions in the security context constraints (SCCs) that trigger it to look up pre-allocated values from a namespace and populate the SCC before processing the pod. Each SCC strategy is evaluated independently of other strategies, with the pre-allocated values, where allowed, for each policy aggregated with pod specification values to make the final values for the various IDs defined in the running pod.

The following SCCs cause the admission controller to look for pre-allocated values when no ranges are defined in the pod specification:

1. A **RunAsUser** strategy of **MustRunAsRange** with no minimum or maximum set. Admission looks for the **openshift.io/sa.scc.uid-range** annotation to populate range fields.

2. An **SELinuxContext** strategy of **MustRunAs** with no level set. Admission looks for the **openshift.io/sa.scc.mcs** annotation to populate the level.

3. A **FSGroup** strategy of **MustRunAs**. Admission looks for the **openshift.io/sa.scc.supplemental-groups** annotation.

4. A **SupplementalGroups** strategy of **MustRunAs**. Admission looks for the **openshift.io/sa.scc.supplemental-groups** annotation.

During the generation phase, the security context provider uses default values for any parameter values that are not specifically set in the pod. Default values are based on the selected strategy:

1. **RunAsAny** and **MustRunAsNonRoot** strategies do not provide default values. If the pod needs a parameter value, such as a group ID, you must define the value in the pod specification.

2. **MustRunAs** (single value) strategies provide a default value that is always used. For example, for group IDs, even if the pod specification defines its own ID value, the namespace's default parameter value also appears in the pod's groups.

3. **MustRunAsRange** and **MustRunAs** (range-based) strategies provide the minimum value of the range. As with a single value **MustRunAs** strategy, the namespace's default parameter value appears in the running pod. If a range-based strategy is configurable with multiple ranges, it provides the minimum value of the first configured range.

NOTE

**FSGroup** and **SupplementalGroups** strategies fall back to the **openshift.io/sa.scc.uid-range** annotation if the **openshift.io/sa.scc.supplemental-groups** annotation does not exist on the namespace. If neither exists, the SCC is not created.

NOTE

By default, the annotation-based **FSGroup** strategy configures itself with a single range based on the minimum value for the annotation. For example, if your annotation reads **1/3**, the **FSGroup** strategy configures itself with a minimum and maximum value of **1**. If you want to allow more groups to be accepted for the **FSGroup** field, you can configure a custom SCC that does not use the annotation.

NOTE

The **openshift.io/sa.scc.supplemental-groups** annotation accepts a comma-delimited list of blocks in the format of **<start>/<length** or **<start>-<end>**. The **openshift.io/sa.scc.uid-range** annotation accepts only a single block.

## 2.3. EXAMPLE SECURITY CONTEXT CONSTRAINTS

The following examples show the security context constraints (SCC) format and annotations:

**Annotated privileged SCC**

```
allowHostDirVolumePlugin: true
allowHostIPC: true
```

```
allowHostNetwork: true
allowHostPID: true
allowHostPorts: true
allowPrivilegedContainer: true
allowedCapabilities: (1)
- '*'
apiVersion: security.openshift.io/v1
defaultAddCapabilities: [] (2)
fsGroup: (3)
  type: RunAsAny
groups: (4)
- system:cluster-admins
- system:nodes
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: 'privileged allows access to all privileged and host
      features and the ability to run as any user, any group, any fsGroup, and with
      any SELinux context.  WARNING: this is the most relaxed SCC and should be used
      only for cluster administration. Grant with caution.'
  creationTimestamp: null
  name: privileged
priority: null
readOnlyRootFilesystem: false
requiredDropCapabilities: (5)
- KILL
- MKNOD
- SETUID
- SETGID
runAsUser: (6)
  type: RunAsAny
seLinuxContext: (7)
  type: RunAsAny
seccompProfiles:
- '*'
supplementalGroups: (8)
  type: RunAsAny
users: (9)
- system:serviceaccount:default:registry
- system:serviceaccount:default:router
- system:serviceaccount:openshift-infra:build-controller
volumes: (10)
- '*'
```

(1) A list of capabilities that a pod can request. An empty list means that none of capabilities can be requested while the special symbol * allows any capabilities.

(2) A list of additional capabilities that are added to any pod.

(3) The **FSGroup** strategy, which dictates the allowable values for the security context.

(4) The groups that can access this SCC.

(5) A list of capabilities to drop from a pod. Or, specify **ALL** to drop all capabilities.

6 The **runAsUser** strategy type, which dictates the allowable values for the security context.

7 The **seLinuxContext** strategy type, which dictates the allowable values for the security context.

8 The **supplementalGroups** strategy, which dictates the allowable supplemental groups for the security context.

9 The users who can access this SCC.

10 The allowable volume types for the security context. In the example, * allows the use of all volume types.

The **users** and **groups** fields on the SCC control which users can access the SCC. By default, cluster administrators, nodes, and the build controller are granted access to the privileged SCC. All authenticated users are granted access to the **restricted-v2** SCC.

### Without explicit runAsUser setting

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext: 1
  containers:
  - name: sec-ctx-demo
    image: gcr.io/google-samples/node-hello:1.0
```

1 When a container or pod does not request a user ID under which it should be run, the effective UID depends on the SCC that emits this pod. Because the **restricted-v2** SCC is granted to all authenticated users by default, it will be available to all users and service accounts and used in most cases. The **restricted-v2** SCC uses **MustRunAsRange** strategy for constraining and defaulting the possible values of the **securityContext.runAsUser** field. The admission plugin will look for the **openshift.io/sa.scc.uid-range** annotation on the current project to populate range fields, as it does not provide this range. In the end, a container will have **runAsUser** equal to the first value of the range that is hard to predict because every project has different ranges.

### With explicit runAsUser setting

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000 1
  containers:
    - name: sec-ctx-demo
      image: gcr.io/google-samples/node-hello:1.0
```

1 A container or pod that requests a specific user ID will be accepted by Red Hat OpenShift Service on AWS only when a service account or a user is granted access to a SCC that allows such a user ID. The SCC can allow arbitrary IDs, an ID that falls into a range, or the exact user ID specific to the

request.

This configuration is valid for SELinux, fsGroup, and Supplemental Groups.

## 2.4. CREATING SECURITY CONTEXT CONSTRAINTS

You can create security context constraints (SCCs) by using the OpenShift CLI (**oc**).

**IMPORTANT**

Creating and modifying your own SCCs are advanced operations that might cause instability to your cluster. If you have questions about using your own SCCs, contact Red Hat Support. For information about contacting Red Hat support, see *Getting support*.

**Prerequisites**

- Install the OpenShift CLI (**oc**).
- Log in to the cluster as a user with the **cluster-admin** role.

**Procedure**

1. Define the SCC in a YAML file named **scc-admin.yaml**:

```
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: scc-admin
allowPrivilegedContainer: true
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: RunAsAny
fsGroup:
  type: RunAsAny
supplementalGroups:
  type: RunAsAny
users:
- my-admin-user
groups:
- my-admin-group
```

Optionally, you can drop specific capabilities for an SCC by setting the **requiredDropCapabilities** field with the desired values. Any specified capabilities are dropped from the container. To drop all capabilities, specify **ALL**. For example, to create an SCC that drops the **KILL**, **MKNOD**, and **SYS_CHROOT** capabilities, add the following to the SCC object:

```
requiredDropCapabilities:
- KILL
- MKNOD
- SYS_CHROOT
```

NOTE

You cannot list a capability in both **allowedCapabilities** and **requiredDropCapabilities**.

CRI-O supports the same list of capability values that are found in the Docker documentation.

2. Create the SCC by passing in the file:

```
$ oc create -f scc-admin.yaml
```

**Example output**

```
securitycontextconstraints "scc-admin" created
```

**Verification**

- Verify that the SCC was created:

```
$ oc get scc scc-admin
```

**Example output**

```
NAME      PRIV    CAPS    SELINUX    RUNASUSER    FSGROUP    SUPGROUP
PRIORITY    READONLYROOTFS    VOLUMES
scc-admin    true    []    RunAsAny    RunAsAny    RunAsAny    RunAsAny    <none>    false
[awsElasticBlockStore azureDisk azureFile cephFS cinder configMap downwardAPI
emptyDir fc flexVolume flocker gcePersistentDisk gitRepo glusterfs iscsi nfs
persistentVolumeClaim photonPersistentDisk quobyte rbd secret vsphere]
```

## 2.5. ROLE-BASED ACCESS TO SECURITY CONTEXT CONSTRAINTS

You can specify SCCs as resources that are handled by RBAC. This allows you to scope access to your SCCs to a certain project or to the entire cluster. Assigning users, groups, or service accounts directly to an SCC retains cluster-wide scope.

NOTE

You cannot assign a SCC to pods created in one of the default namespaces: **default**, **kube-system**, **kube-public**, **openshift-node**, **openshift-infra**, **openshift**. These namespaces should not be used for running pods or services.

To include access to SCCs for your role, specify the **scc** resource when creating a role.

```
$ oc create role <role-name> --verb=use --resource=scc --resource-name=<scc-name> -n
<namespace>
```

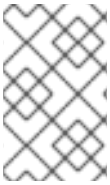
This results in the following role definition:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
```

```
metadata:
...
  name: role-name 1
  namespace: namespace 2
...
rules:
- apiGroups:
  - security.openshift.io 3
  resourceNames:
  - scc-name 4
  resources:
  - securitycontextconstraints 5
  verbs: 6
  - use
```

1. The role's name.
2. Namespace of the defined role. Defaults to **default** if not specified.
3. The API group that includes the **SecurityContextConstraints** resource. Automatically defined when **scc** is specified as a resource.
4. An example name for an SCC you want to have access.
5. Name of the resource group that allows users to specify SCC names in the **resourceNames** field.
6. A list of verbs to apply to the role.

A local or cluster role with such a rule allows the subjects that are bound to it with a role binding or a cluster role binding to use the user-defined SCC called **scc-name**.



NOTE

Because RBAC is designed to prevent escalation, even project administrators are unable to grant access to an SCC. By default, they are not allowed to use the verb **use** on SCC resources, including the **restricted-v2** SCC.

## 2.6. REFERENCE OF SECURITY CONTEXT CONSTRAINTS COMMANDS

You can manage security context constraints (SCCs) in your instance as normal API objects using the OpenShift CLI (**oc**).

### 2.6.1. Listing security context constraints

To get a current list of SCCs:

```
$ oc get scc
```

**Example output**

```
NAME                              PRIV    CAPS                    SELINUX     RUNASUSER          FSGROUP
SUPGROUP    PRIORITY     READONLYROOTFS   VOLUMES
anyuid                            false   <no value>              MustRunAs   RunAsAny           RunAsAny
```

```
RunAsAny    10          false
["configMap","downwardAPI","emptyDir","persistentVolumeClaim","projected","secret"]
hostaccess                  false   <no value>          MustRunAs   MustRunAsRange     MustRunAs
RunAsAny    <no value>   false
["configMap","downwardAPI","emptyDir","hostPath","persistentVolumeClaim","projected","secret"]
hostmount-anyuid            false   <no value>          MustRunAs   RunAsAny           RunAsAny
RunAsAny    <no value>   false
["configMap","downwardAPI","emptyDir","hostPath","nfs","persistentVolumeClaim","projected","secret"]

hostnetwork                 false   <no value>          MustRunAs   MustRunAsRange     MustRunAs
MustRunAs   <no value>   false
["configMap","downwardAPI","emptyDir","persistentVolumeClaim","projected","secret"]
hostnetwork-v2              false   ["NET_BIND_SERVICE"]   MustRunAs   MustRunAsRange
MustRunAs   MustRunAs   <no value>   false
["configMap","downwardAPI","emptyDir","persistentVolumeClaim","projected","secret"]
node-exporter               true    <no value>          RunAsAny    RunAsAny           RunAsAny
RunAsAny    <no value>   false          ["*"]
nonroot                     false   <no value>          MustRunAs   MustRunAsNonRoot   RunAsAny
RunAsAny    <no value>   false
["configMap","downwardAPI","emptyDir","persistentVolumeClaim","projected","secret"]
nonroot-v2                  false   ["NET_BIND_SERVICE"]   MustRunAs   MustRunAsNonRoot
RunAsAny    RunAsAny    <no value>   false
["configMap","downwardAPI","emptyDir","persistentVolumeClaim","projected","secret"]
privileged                  true    ["*"]               RunAsAny    RunAsAny           RunAsAny    RunAsAny
<no value>   false          ["*"]
restricted                  false   <no value>          MustRunAs   MustRunAsRange     MustRunAs
RunAsAny    <no value>   false
["configMap","downwardAPI","emptyDir","persistentVolumeClaim","projected","secret"]
restricted-v2               false   ["NET_BIND_SERVICE"]   MustRunAs   MustRunAsRange
MustRunAs   RunAsAny    <no value>   false
["configMap","downwardAPI","emptyDir","persistentVolumeClaim","projected","secret"]
```

### 2.6.2. Examining security context constraints

You can view information about a particular SCC, including which users, service accounts, and groups the SCC is applied to.

For example, to examine the **restricted** SCC:

```
$ oc describe scc restricted
```

#### Example output

```
Name:                           restricted
Priority:                       <none>
Access:
  Users:                        <none> (1)
  Groups:                       <none> (2)
Settings:
  Allow Privileged:             false
  Allow Privilege Escalation:   true
  Default Add Capabilities:     <none>
  Required Drop Capabilities:   KILL,MKNOD,SETUID,SETGID
  Allowed Capabilities:         <none>
```

```
  Allowed Seccomp Profiles:           <none>
  Allowed Volume Types:
configMap,downwardAPI,emptyDir,persistentVolumeClaim,projected,secret
  Allowed Flexvolumes:                <all>
  Allowed Unsafe Sysctls:             <none>
  Forbidden Sysctls:                  <none>
  Allow Host Network:                 false
  Allow Host Ports:                   false
  Allow Host PID:                     false
  Allow Host IPC:                     false
  Read Only Root Filesystem:          false
  Run As User Strategy: MustRunAsRange
    UID:                              <none>
    UID Range Min:                    <none>
    UID Range Max:                    <none>
  SELinux Context Strategy: MustRunAs
    User:                             <none>
    Role:                             <none>
    Type:                             <none>
    Level:                            <none>
  FSGroup Strategy: MustRunAs
    Ranges:                           <none>
  Supplemental Groups Strategy: RunAsAny
    Ranges:                           <none>
```

1. Lists which users and service accounts the SCC is applied to.
2. Lists which groups the SCC is applied to.

## 2.7. ADDITIONAL RESOURCES

- Getting support