Introduction to Red Hat OpenShift security context constraints

Red Hat OpenShift employs several practices to secure a cluster's environment, one of which is to prevent a container process from running as the root user. Containers are still able to run in this environment because most applications don't need root or privileged access, but this limit is a significant problem for an application that needs access to Linux's protected resources. An OpenShift cluster administrator can configure security context constraints (SCC) to enable access to specific protected resources. With an SCC, a deployer can enable an application to have access to protected resources.

This article is part 1 of a two-part series on security context constraints (SCC). This one introduces SCCs and gives an overview of how SCCs solve the access problem. Part 2, "Allow pods to access protected resources in Red Hat OpenShift using security context constraints," digs into the details of how SCCs are implemented and administered. It assumes you have a general understanding of how to deploy an application to an OpenShift cluster and how the cluster manages a workload.

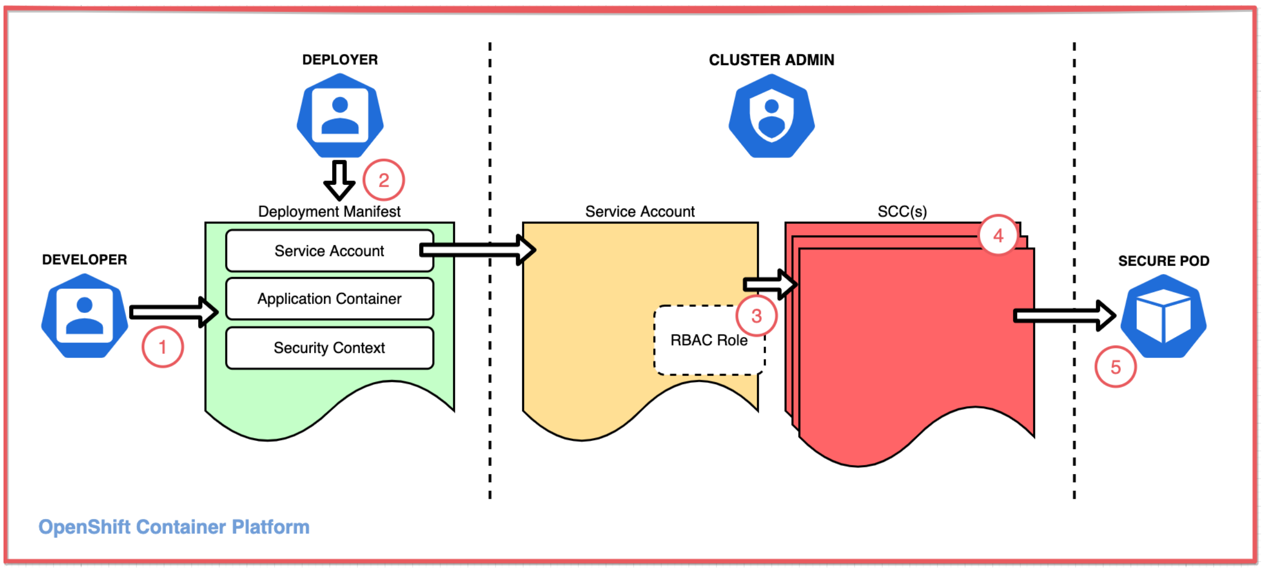
Deploying a secure pod

An OpenShift cluster provides several features for securing the cluster and the applications running in the cluster, including container security, certificate management, vulnerability scanning, and authentication and authorization. Security context constraints address one aspect of security, namely enabling an application the ability to access protected resources such as shared file systems and privileged ip ports. To prevent even a rogue or hacked application from gaining access to resources it shouldn't be able to access, the access is configured by the pod running the application container and is enforced by the cluster. The application can only access the resources that the pod requests and that the cluster approves.

An application's access to protected resources is an agreement between three personas:

- A **developer** who writes an application that accesses protected resources
- A **deployer** who writes the deployment manifest that must request the access the application requires
- An **administrator** who decides whether to grant the deployment the access it requests

This diagram illustrates the components and process that allow an application to access resources:



1. A developer writes an application
2. A deployer creates a **deployment manifest** to deploy the application with a pod spec that configures:
    - A **security context** (for the pod and/or for each container) that requests the access needed by the application
    - A **service account** to grant the requested access
3. An administrator assigns a **security context constraint** to the service account that grants the requested access

    - The SCC can be assigned directly to the service account or indirectly via an RBAC role or group
4. The SCC may be one of OpenShift's predefined SCCs or may be a custom SCC
5. If the SCC grants the access, the admission process allows the pod to deploy

> **NOTE**: An OpenShift service account is a special type of user account that can be used programmatically without using a regular user's credentials.

Now that we know the personas involved and the general process that they follow, let's explain the components they use in more detail.

What is an SCC?

Security context constraints (SCCs) control the access that pods have to underlying resources, similarly to how role-based access control (RBAC) controls access for users. SCCs enable an administrator to control the resources a pod is allowed to access. By default, a pod is assigned an SCC called `restricted` that blocks access to protected resources. For an application to access protected resources, the pod must have access to SCCs that allow it.

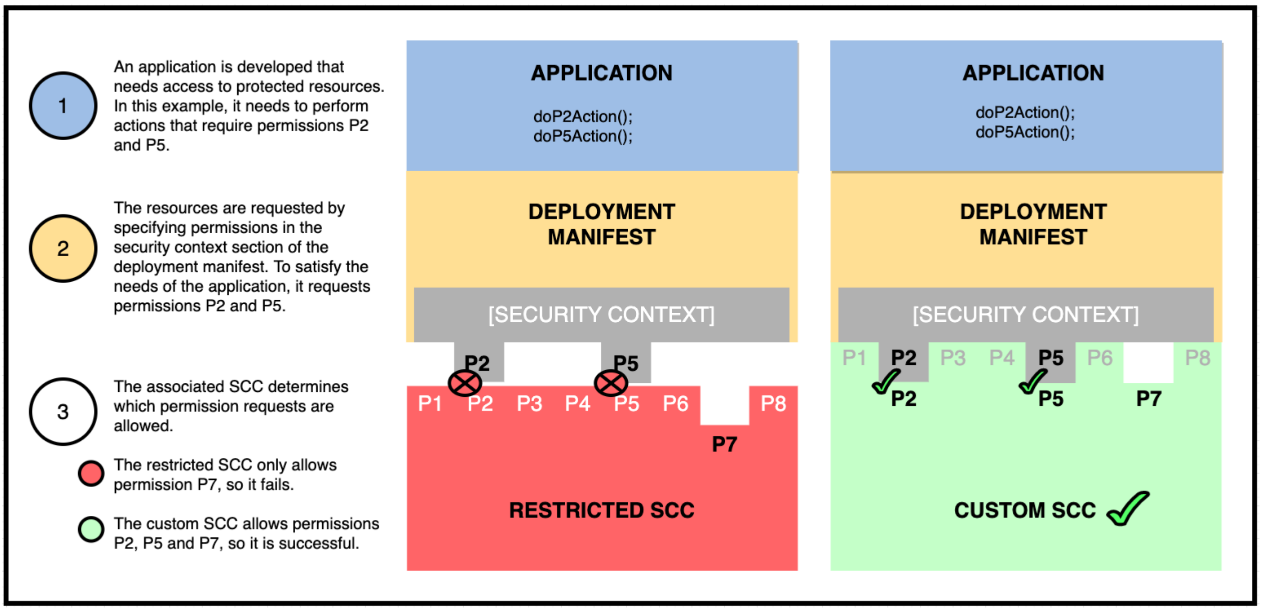
> **NOTE**: SCC enforcement is implemented using SELinux and AppArmor, security modules included in the kernel of all Red Hat Linux distributions. The nodes in an OpenShift v4 cluster can only run on a Red Hat Linux distribution, guaranteeing that the kernel will include those modules.

How a pod requests additional access

While an SCC grants access to protected resources, each pod that wants to use that access must request it. To request access to the resources its application needs, a pod specifies those permissions in the **security context** field of the pod manifest. The manifest also specifies the service account that the pod expects will be able to grant this access. When the manifest is deployed, the cluster associates the pod with the service account, which is associated with the SCC. For the cluster to deploy the pod, the SCC must grant the permissions that the pod requests.

One way to envision this relationship is to think of the SCC as a lock protecting system resources and the manifest being the key. The pod is allowed to deploy only if the key fits.

This diagram illustrates that relationship:


1
An application is developed that needs access to protected resources. In this example, it needs to perform actions that require permissions P2 and P5.
2
The resources are requested by specifying permissions in the security context section of the deployment manifest. To satisfy the needs of the application, it requests permissions P2 and P5.
3
The associated SCC determines which permission requests are allowed.
The restricted SCC only allows permission P7, so it fails.
The custom SCC allows permissions P2, P5 and P7, so it is successful.
APPLICATION
doP2Action();
doP5Action();
DEPLOYMENT MANIFEST
[SECURITY CONTEXT]
P2
P5
P1 P2 P3 P4 P5 P6 P8
P7
RESTRICTED SCC
APPLICATION
doP2Action();
doP5Action();
DEPLOYMENT MANIFEST
[SECURITY CONTEXT]
P1 P2 P3 P4 P5 P6 P8
P2
P5
P7
CUSTOM SCC

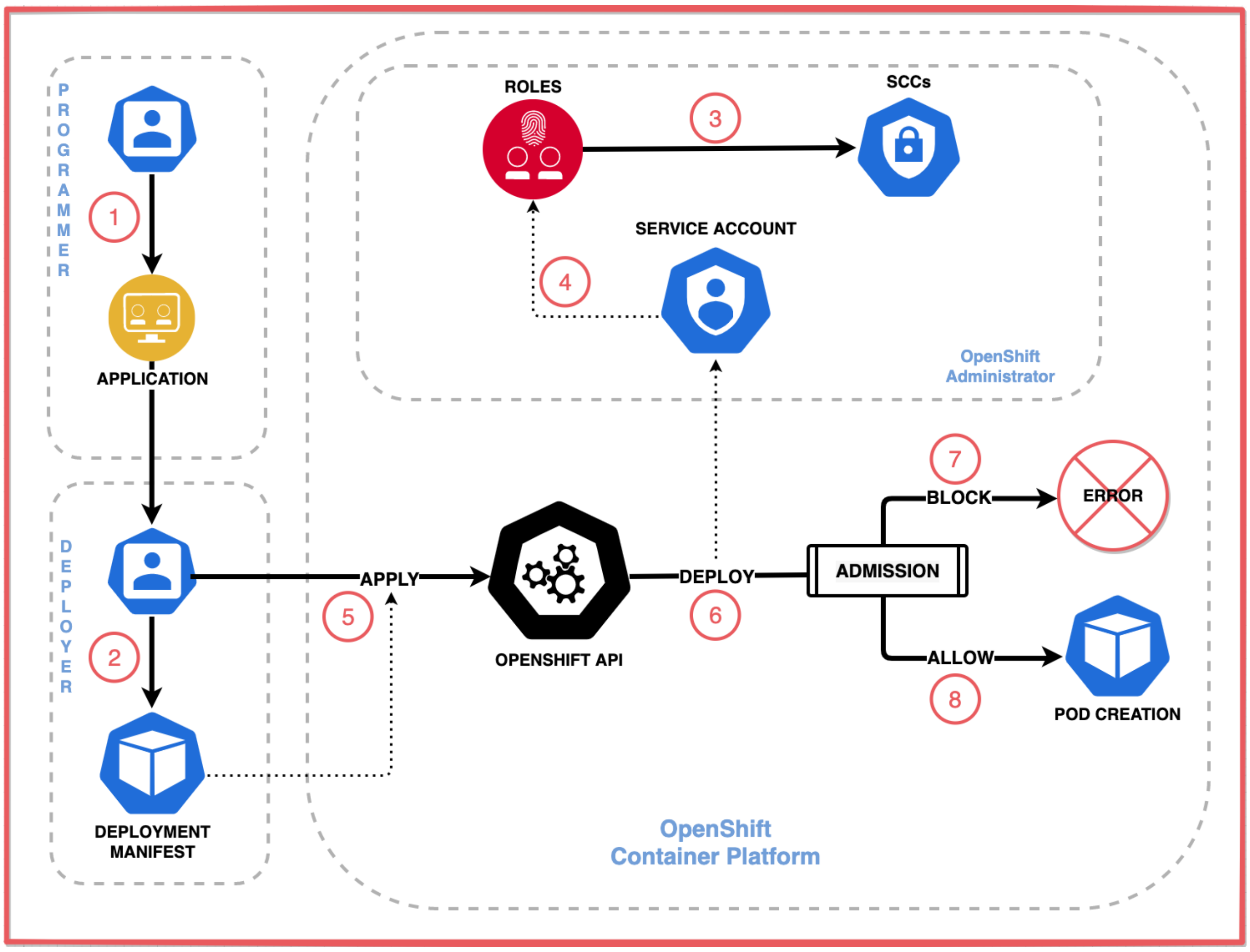
The diagram shows a deployment that will be blocked by the SCC on the left and will be allowed by the SCC on the right. In both examples, the pod specifies in its security context that it needs access to two permissions arbitrarily labeled P2 and P5. In the first example, a very restrictive SCC does not grant the access the manifest requests in its security context, so the cluster will refuse to deploy the pod. In the second example, a custom SCC does grant the access the manifest requests in its security context, so the cluster will proceed with deploying the pod.

> **IMPORTANT**: As a best-practice, each custom SCC should grant as little access as possible. If protected resources are like rooms in a building, only allow the users access to the rooms they need. While an SCC can be used to allow a pod to run as a privileged user or even the root user, this should be granted to pods very sparingly.

The big picture

To better understand how SCCs control access on a Red Hat OpenShift cluster, lets walk through a deployment scenario to show how the cluster's admission process uses the deployment manifest, service account, and SCC together to determine whether to deploy a pod.

This diagram illustrates the deployment process:



1. The **developer** implements an application or service that requires access to protected resources, and delivers the application to the deployer.
2. The **deployer** creates a deployment manifest for the application. The manifest specifies a security context and a service account.
3. The **administrator** creates a role and assigns it an SCC.
4. The **administrator** creates a service account and binds it to the role.
5. The **deployer** applies the deployment manifest, thereby deploying the application.
6. OpenShift processes the deployment manifest and attempts to deploy the pod. The deployment process will determine which SCC to use based on the sevice account specified in the manifest. The **admission process** compares the security context of the manifest against the SCC and decides whether to block or allow the pod to deploy.

7. **BLOCK**: Some requested permissions are not granted, so the deployment fails.
8. **ALLOW**: All requested permissions are granted, so the deployment creates the pod and runs the application.

If the pod is denied the requested permissions, the administrator will need to:

- Determine if the additional requests made in the manifest are in fact needed.
- If so, assign the requested permissions to the SCC or select an SCC that already has the requested permissions.

Continue learning

To dig deeper into the details of how SCCs work, check out part 2, "Allow pods to access protected resources in Red Hat OpenShift using security context constraints" of this article series.

To get hands-on experience using SCCs, check out the accompanying tutorial, "Use security context constraints to restrict and empower your OpenShift workloads."