**Security Context Constraints.**   
  
**What is an SCC?**   
An SCC (Security Context Constraint) is a security policy object in OpenShift that defines what a pod can and cannot do, particularly around:

* Linux capabilities
* Running as privileged or not
* User/group IDs
* Volume types allowed
* Host network/port access

It acts as a security gate applied at the pod level, ensuring workloads run within allowed constraints.   
  
**Why is SCC important?**   
Because OpenShift is built with strong security defaults, SCCs help:

* Prevent escalation of privileges
* Limit access to host resources
* Enforce multi-tenancy isolation
* Control workload security boundaries

**Example: Default SCCs**   
OpenShift ships with predefined SCCs, such as:

SCC Name / Description

* **restricted** - The most secure SCC, used by default. Doesn’t allow privileged containers.
* **anyuid** - Allows pods to run as any UID (e.g., required by some legacy apps).
* **privileged** - Allows privileged containers. Use with caution.
* **hostaccess** - Allows access to host IPC, PID, and network.
* **hostnetwork** - Allows use of the host’s network namespace.

**Real-World Example**   
Suppose you're deploying an app that requires root access. By default, it will fail under the restricted SCC.   
  
**You'd then:**   
oc adm policy add-scc-to-user anyuid -z default -n myproject   
So that the pod in myproject can run as any UID.   
  
SCCs (Security Context Constraints) in OpenShift involves controlling what security rules are applied to pods, and granting specific users or service accounts access to those SCCs.   
  
SCCs in OpenShift are enforced based on the service account a pod uses — not directly by namespace.   
  
because all pods run as a service account, even if you don’t explicitly define one.   
By default, pods run as the default service account in their namespace.   
So:

* You don’t need to explicitly create a custom service account if you're fine using default.
* But you can’t apply an SCC directly to a namespace and expect all pods in that namespace to inherit it — you must assign the SCC to a service account.

**Practical Flow**

**You create a namespace:**   
oc new-project myapp

You deploy a pod using the default service account   
By default, that pod will be constrained by the restricted SCC   
if your pod needs more permissions (e.g., to run as root), you must assign a different SCC (e.g., anyuid) to that service account:

oc adm policy add-scc-to-user anyuid -z default -n myapp   
  
**Question Answer**

* Can I bind an SCC to a namespace directly?

No, SCCs are bound to service accounts.

* Do I have to define a service account?

  Not necessarily — you can use default.

* Is SCC applied at the namespace level?

Indirectly, only via the service account(s) used in that namespace.   
  
  
**SCCs in OpenShift can be bound not only to service accounts but also to users and groups.**   
  
**how you bind an SCC to a user, step-by-step.**

**1. Bind SCC to a User**   
oc adm policy add-scc-to-user <scc-name> <username>   
**Example:**   
oc adm policy add-scc-to-user anyuid haseeb   
This grants the anyuid SCC to the user named haseeb.   
  
**2. Check the Binding**   
**To verify which SCCs a user can access:**   
oc describe clusterrolebinding | grep -A 5 haseeb

**or check in the SCC object directly:**   
oc get scc anyuid -o yaml

**Look for users: section like this:**   
users:   
- system:serviceaccount:myproject:default   
- haseeb   
  
  
**3. Revoke SCC from a User**   
oc adm policy remove-scc-from-user anyuid haseeb   
  
**Important Notes:**   
**The username must match the OpenShift identity. You can get logged-in users with:**   
oc whoami   
  
**For OAuth or LDAP integrated clusters, usernames may include domains, like:**   
johndoe   
or   
johndoe@domain.com   
  
You typically assign SCCs to service accounts for workloads, and users when the user is creating/running pods directly (e.g., via CLI).   
  
  
**What does it mean to bind an SCC to a user?**   
In OpenShift, when a user (e.g. you) creates a pod or deployment, the platform checks which Security Context Constraints (SCCs) that user is allowed to use.

Think of SCCs like security rules for pods — and OpenShift needs to know whether you, the user, are trusted enough to use a certain SCC (like privileged or anyuid)   
  
**How SCC Binding Works**

When a pod is created, OpenShift checks:

* Which SCCs are available to the user (or the service account used).
* Whether the pod’s spec matches the permissions in any of those SCCs.
* If no match → pod creation fails with an error like:
* unable to validate against any security context constraint.

**Binding SCC to a User**

**Let’s say you (user haseeb) are trying to run a pod from the CLI like:**oc run nginx --image=nginx --command -- bash

But this image tries to run as UID 0 (root), which the default restricted SCC doesn’t allow.

So your pod will fail unless you're allowed to use a less restrictive SCC — like anyuid.

**To allow this, you run:**   
oc adm policy add-scc-to-user anyuid haseeb

This gives you, the user, permission to use the anyuid SCC, so when you create a pod, OpenShift says:

“Yes, this user is allowed to use anyuid, and this pod matches its rules → Allow.”

**Result**   
**Now, when you run:**   
oc run nginx --image=nginx

The pod works — because your user has access to an SCC (anyuid) that allows running as any UID.

**Removing SCC Access:**

**To take away access from the user:**   
oc adm policy remove-scc-from-user anyuid Haseeb

Now your pod would again be denied if it needs permissions outside restricted.

**Where is this recorded?**   
You can see which users are bound to an SCC by checking:   
oc get scc anyuid -o yaml   
  
**You'll see:**   
users:   
- haseeb   
  
  
**Example of a complete YAML that demonstrates:**

A Pod trying to run as UID 0 (root)   
It will only succeed if the user or service account has access to an SCC like anyuid   
  
**Scenario: Pod that runs as root**   
Pod YAML (nginx-pod.yaml)   
  
apiVersion: v1   
kind: Pod   
metadata:   
  name: nginx-root   
  namespace: myproject   
spec:   
  containers:   
  - name: nginx   
    image: nginx   
    command: ["/bin/bash", "-c", "sleep 3600"]   
    securityContext:   
      runAsUser: 0 # trying to run as root   
  restartPolicy: Never  
  
 **This pod will fail by default in OpenShift because:**   
The restricted SCC (used by default) does not allow runAsUser: 0   
  
  
**Fix: Bind the SCC to Your User**   
From CLI, give your user (haseeb) permission to use anyuid:   
oc adm policy add-scc-to-user anyuid haseeb   
  
**Or if using a service account:**   
oc create serviceaccount mysa -n myproject   
oc adm policy add-scc-to-user anyuid -z mysa -n myproject   
  
**Then update the pod YAML to use the service account:**spec:   
serviceAccountName: mysa   
containers:   
- name: nginx   
image: nginx   
...   
  
**End Result**:   
Pod now runs as root (UID 0)   
Only works if your user or SA has the right SCC (anyuid)   
  
 **Let's compare the three most common SCCs:**   
  
 **1. restricted SCC (Default & Most Secure)**   
 Most pods run with this by default. It's designed for maximum safety in multi-tenant environments.

 **What It Does NOT Allow:**

* Running as root (runAsUser: 0)
* Accessing host network, ports, or PID/IPC
* Mounting hostPath volumes
* Privileged containers

**What It Allows:**

* Must run as non-root UID
* Only certain volume types (e.g., ConfigMap, Secret, PVC)
* SELinux set to MustRunAs
* Safe container usage

 **2. anyuid SCC (Run as Any User, incl. root)**   
 Used for apps that need to run as root inside the container, but don’t need host access.

**What It Allows:**

* Run as UID 0 (root) or any user
* Still blocks host networking, host volumes, privileged mode
* Same safe volume types as restricted

 **What It Does NOT Allow:**

* Host network, PID, IPC
* HostPath volumes
* Privileged containers

 Use Case: NGINX, Apache, or legacy apps that require root inside the container — but not on the host.   
  
**3. privileged SCC (Full Host Access)**   
 Use with extreme caution — gives pod nearly full control over the node.

**What It Allows:**

* Privileged containers
* Access to host network, IPC, PID
* Mount any volume type (including hostPath)
* Run as any UID
* Add Linux capabilities (CAP\_SYS\_ADMIN, etc.)

**What It Does NOT Enforce:**

* Basically removes most isolation
* Not safe for multi-tenant clusters unless you fully trust the workload

**Use Case: Low-level tools like:**   
Storage/volume plugins   
Monitoring agents (e.g., node-exporter)   
Debug containers   
SELinux tuning tools