neueda

futureproof your workforce

SECURITY AND AUTHORIZATION OVERVIEW

NEUEDA.COM

Objectives

- Authorization through RBAC
- Service Accounts
- Security Context Constraints

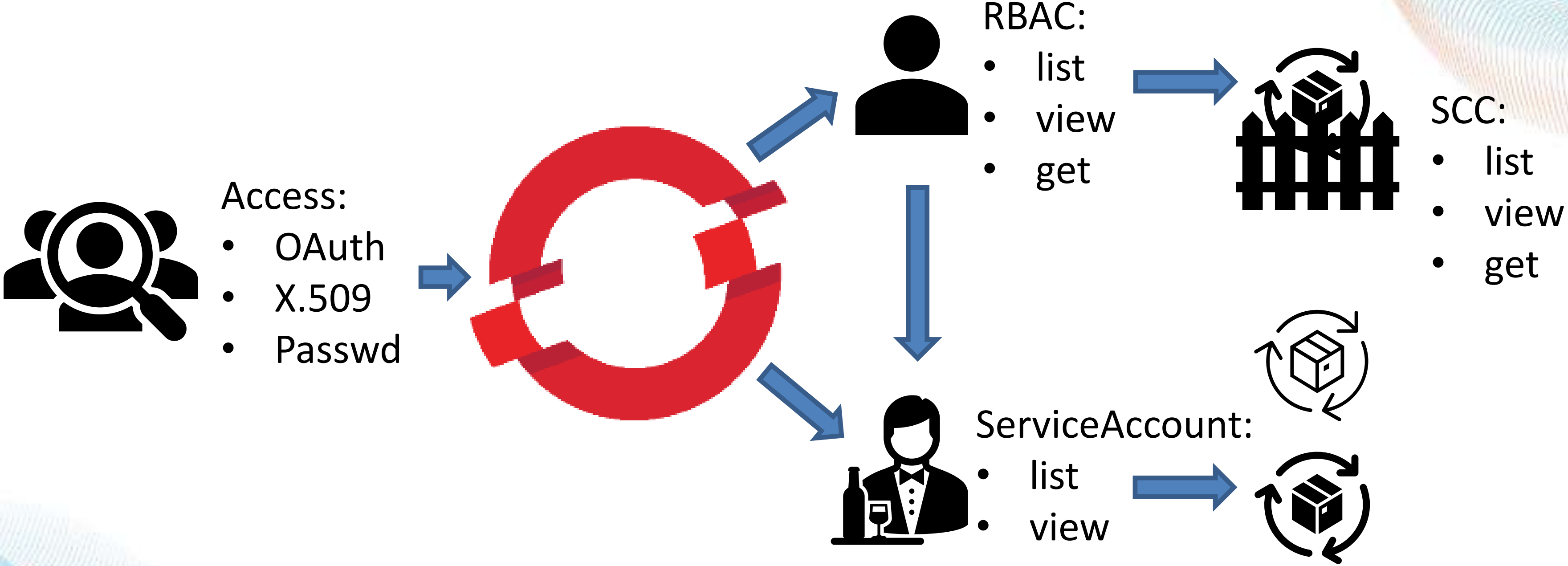
Authorization and Security

- Security of and OpenShift cluster and its workloads is critical
- To even interact with a cluster, users must first authenticate
    - This will depend on how your cluster is set up
        - OAuth access tokens
        - X.509 client certificates
        - (less secure) passwords

Authorization and Security

- A user has certain permissions assigned using **RBAC objects**
  - Rules
  - Roles
  - Bindings
- **Service accounts** can also be used to control API access without sharing a regular users’ credentials
- **Security Context Constraints** can be used to control the actions a pod can perform and the resources it can access

Overview



Access:

- OAuth
- X.509
- Passwd

RBAC:

- list
- view
- get

SCC:

- list
- view
- get

ServiceAccount:

- list
- view

RBAC

Role-based Access Control (RBAC)

- RBAC objects determine whether a user is allowed to perform a given action within a project
- Admins (and devs, locally) determine these permissions using

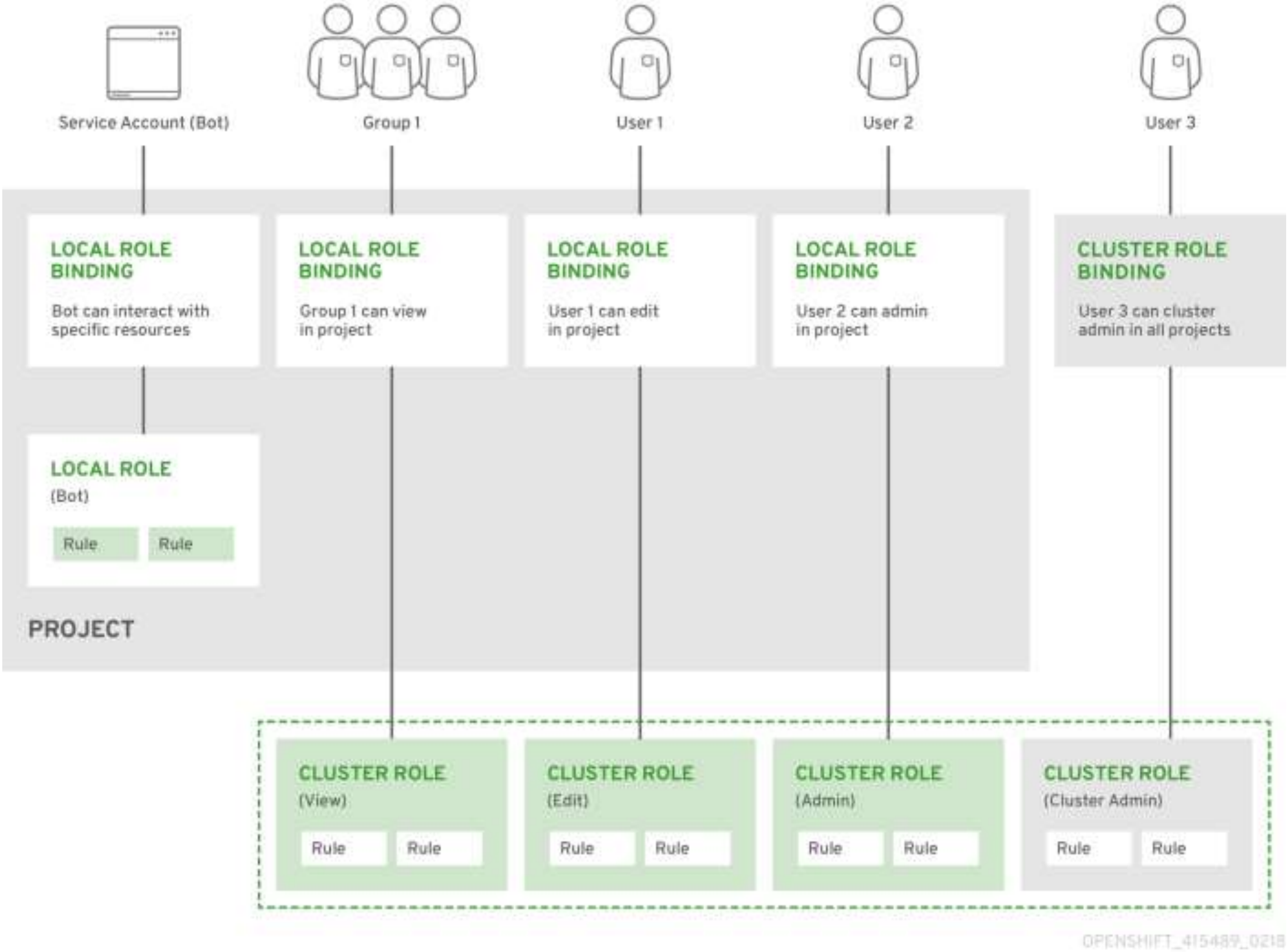
Rules	Sets of permitted verbs on a set of objects (get, list, create, update, delete)
Roles	Collections of rules
Bindings	Associations between users/groups with a role

Two Levels of RBAC Roles and Bindings

- Cluster RBAC
  - Roles and bindings across all projects
  - Cluster Roles exist cluster-wide
  - Cluster Role Bindings can reference only Cluster Roles
- Local RBAC
  - Roles and bindings scoped to a given project
  - Local Roles exist only in a single project, but …
  - Local Role Bindings can reference both cluster and local roles
    - E.g. the cluster role *view* must be bound to a user using a (local) role binding for that user to view that project.

Default Cluster Roles

Cluster Role	Description
admin	A project manager. If used in a local binding, an admin has rights to view any resource in the project and modify any resource in the project except for quota.
basic-user	A user that can get basic information about projects and users.
cluster-admin	A super-user that can perform any action in any project. When bound to a user with a local binding, they have full control over quota and every action on every resource in the project.
cluster-status	A user that can get basic cluster status information.
cluster-reader	A user that can get or view most of the objects but cannot modify them.
edit	A user that can modify most objects in a project but does not have the power to view or modify roles or bindings.
self-provisioner	A user that can create their own projects.
view	A user who cannot make any modifications, but can see most objects in a project. They cannot view or modify roles or bindings.


Service Account (Bot)
Group 1
User 1
User 2
User 3
LOCAL ROLE BINDING
Bot can interact with specific resources
LOCAL ROLE BINDING
Group 1 can view in project
LOCAL ROLE BINDING
User 1 can edit in project
LOCAL ROLE BINDING
User 2 can admin in project
CLUSTER ROLE BINDING
User 3 can cluster admin in all projects
LOCAL ROLE
(Bot)
Rule
Rule
PROJECT
CLUSTER ROLE
(View)
Rule
Rule
CLUSTER ROLE
(Edit)
Rule
Rule
CLUSTER ROLE
(Admin)
Rule
Rule
CLUSTER ROLE
(Cluster Admin)
Rule
Rule
OPENSHIFT_415489_0218

Evaluating Authorization

- OpenShift evaluates authorization using
  - Identity: user name and groups the user belongs to
  - Action: this usually consists of
    - The project in which the action is to take place
    - The action itself (verb)
    - The resource name, which is the API endpoint you access

Authorization Steps

- Use identity + project-scoped action to find all bindings applying to the user or their groups
- Bindings then locate Roles
- Roles then locate Rules
- The Action is checked against the Rules
- If no match is found, deny by default

SERVICE ACCOUNTS

Service Accounts

- Exist within each project
- Allow a component to directly access the API
- Provide a flexible way to control API access without sharing a regular user’s credentials
- E.g. service accounts can allow API calls so that
    - Replication controllers can create or delete pods
    - Applications inside pods can access service discovery
    - External applications can perform monitoring or integration tasks

Service accounts names, groups and secrets

- Every service account has a local name, and a full name scoped as follows: **system:serviceaccount:<project>:<name>**
- Every service account is also a member of two groups:

  system:serviceaccounts: all service accounts in the system

  system:serviceaccounts:<project>: all service accounts in the specified project
- Every service account automatically has two secrets, an API token, and credentials for the OpenShift Container Registery

Creating Service Accounts

- View all service accounts in the current project:

```
$oc get sa
```

- Create a new service account in the current project:

```
$oc create sa <service_account_name>
```

- View the secrets for the service account:

```
$oc describe sa <service_account_name>
```

- Grant a role to service account:

```
$oc policy add-role-to-user view -z <service_account_name>
```

Default cluster service accounts

- There are three cluster-wide service accounts for infrastructure-controllers:

Service Account	Description
replication-controller	Assigned the system:replication-controller role
deployment-controller	Assigned the system:deployment-controller role
build-controller	Assigned the system:build-controller role. Additionally, the build-controller service account is included in the privileged security context constraint to create privileged build pods.

Default project service accounts

Service Account	Usage
builder	Used by build pods. It is given the system:image-builder role, which allows pushing images to any imagestream in the project using the internal Docker registry.
deployer	Used by deployment pods and given the system:deployer role, which allows viewing and modifying replication controllers and pods in the project.
default	Used to run all other pods unless they specify a different service account.

- All project service accounts are given the `system:image-puller` role which allows pulling image from any project imagestream.

SECURITY CONTEXT CONSTRAINTS

Security Context Constraints

- Security Context Constraints (SCC) control permissions for pods
    - Similar to the way RBAC resources control user access
- Includes:
    - Actions a pod can perform
    - Resources it can access
- SCCs can define a set of conditions necessary for a pod to be accepted on the system.

Linux Security (1)

- Linux has privileged (ID 0) and unprivileged processes (ID ≠ 0)
- Privileged processes have unfettered access to all OS objects, and their actions are not verified by the kernel.
  - User, group etc permissions are not checked before access to an object is granted to a privileged process
- Unprivileged processes are subject to full permission checking based on process credentials (user ID, group ID, etc).
  - Kernel makes an iterative check trying to match user’s credentials to target object’s permissions to grant/deny access.

Linux Security (2)

- Linux also has a concept of **capabilities**
  - Superuser or root features that can be enabled or disabled in a very granular way.
  - E.g. suppose a process requires the ability to bind a socket to an Internet domain privileged port (port number < 1024)
  - Then that process could be granted the `CAP_NET_BIND_SERVICE` capability to achieve this.

Linux Security (3)

- There are also Linux kernel security modules
  - SELinux, AppArmor
  - Add on top of capabilities
  - Give even more fine grained security rules by using access security policies or program profiles

Container Security

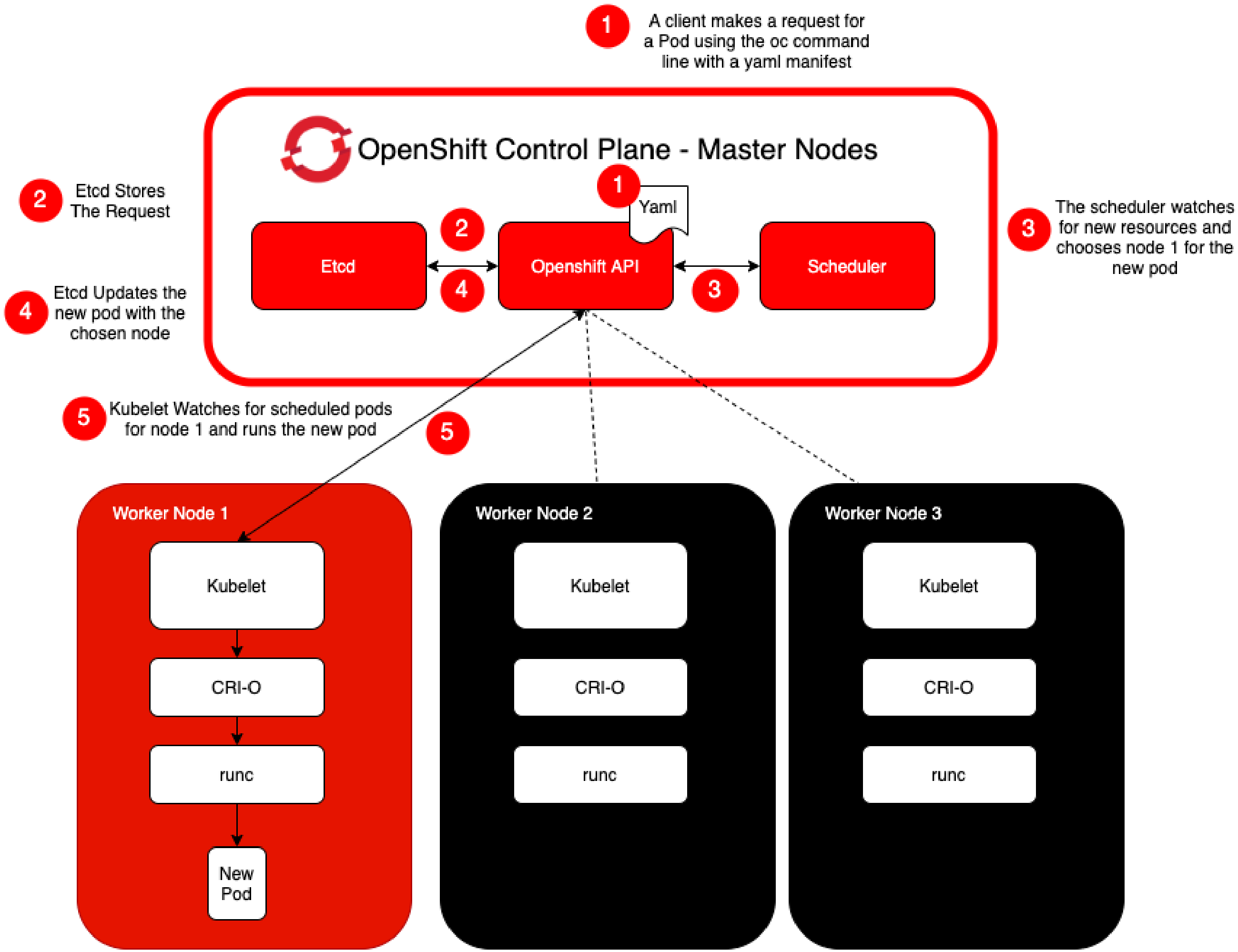
- What are containers?
  - *Processes* segregated by namespaces and cgroups.
- Containers have all the security features Linux has
  - Privileged/unprivileged processes
  - Capabilities
  - SELinux and AppArmor

Container Security (2)

- runc is the software that creates the container process
  - Needs a file system image and a bundle with the process configuration
  - The process configuration for OCI runtimes is detailed here
  - This has fields like:
    - apparmorProfile
    - selinuxLable
    - Capabilities
    - etc

Container Security on OpenShift

- OpenShift uses the CRI-O container engine to create and manage containers (it runs `runc`)
  - The process configuration is packaged for `runc` by CRI-O
  - The resource requesting the container (Pod) will also request the desired privileges



1 A client makes a request for a Pod using the oc command line with a yaml manifest
OpenShift Control Plane - Master Nodes
2 Etcd Stores The Request
4 Etcd Updates the new pod with the chosen node
3 The scheduler watches for new resources and chooses node 1 for the new pod
5 Kubelet Watches for scheduled pods for node 1 and runs the new pod
1
Yaml
Etcd
2
4
Openshift API
3
Scheduler
5
Worker Node 1
Kubelet
CRI-O
runc
New Pod
Worker Node 2
Kubelet
CRI-O
runc
Worker Node 3
Kubelet
CRI-O
runc

https://cloud.redhat.com/blog/introduction-to-security-contexts-and-sccs

Pod YAML Manifest

- This file contains the request for privileges through two objects
  - PodSecurityContext (relates to the Pod level privileges)
  - SecurityContext (which relates specifically to containers)
- This means you can apply a security context to the whole pod, or to specific containers in the pod.
- The SecurityContext takes precedence over the PodSecurityContext
- Using these objects the calling process can request any level of security desired
  - No matter what the RBAC of the calling user is!

neueda

futureproof your workforce

Pod YAML Manifest (2)

- Example manifest with capabilities on securityContext field:

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo-4
spec:
  containers:
  - name: sec-ctx-4
    image: gcr.io/google-samples/node-hello:1.0
    securityContext:
      capabilities:
        add: ["NET_ADMIN", "SYS_TIME"]
```

Security Context Constraints

- How to ensure a specific Pod or Container doesn’t request more privilege than it should?
- **Security Context Constraints!**
  - Check beforehand if the system can permit a pod or container configuration request.

Security Context Constraints (2)

```
$ oc get scc restricted -o yaml
```

- “`restricted`” is the default SCC
  - pretty basic permissions
  - will accept Pod configurations that don't request special security contexts

Security Context Constraints (3)

- SCC resources allow an OpenShift admin to decide whether
  - an entire pod can run in privileged mode
  - access directories and volumes on the host namespace
  - use special SELinux contexts
  - what ID the container process can use etc …
- All this <u>before</u> the Pod gets requested to the API and passed to the container runtime process.

Predefined SCCs

- OpenShift comes preinstalled with 8 SCCs

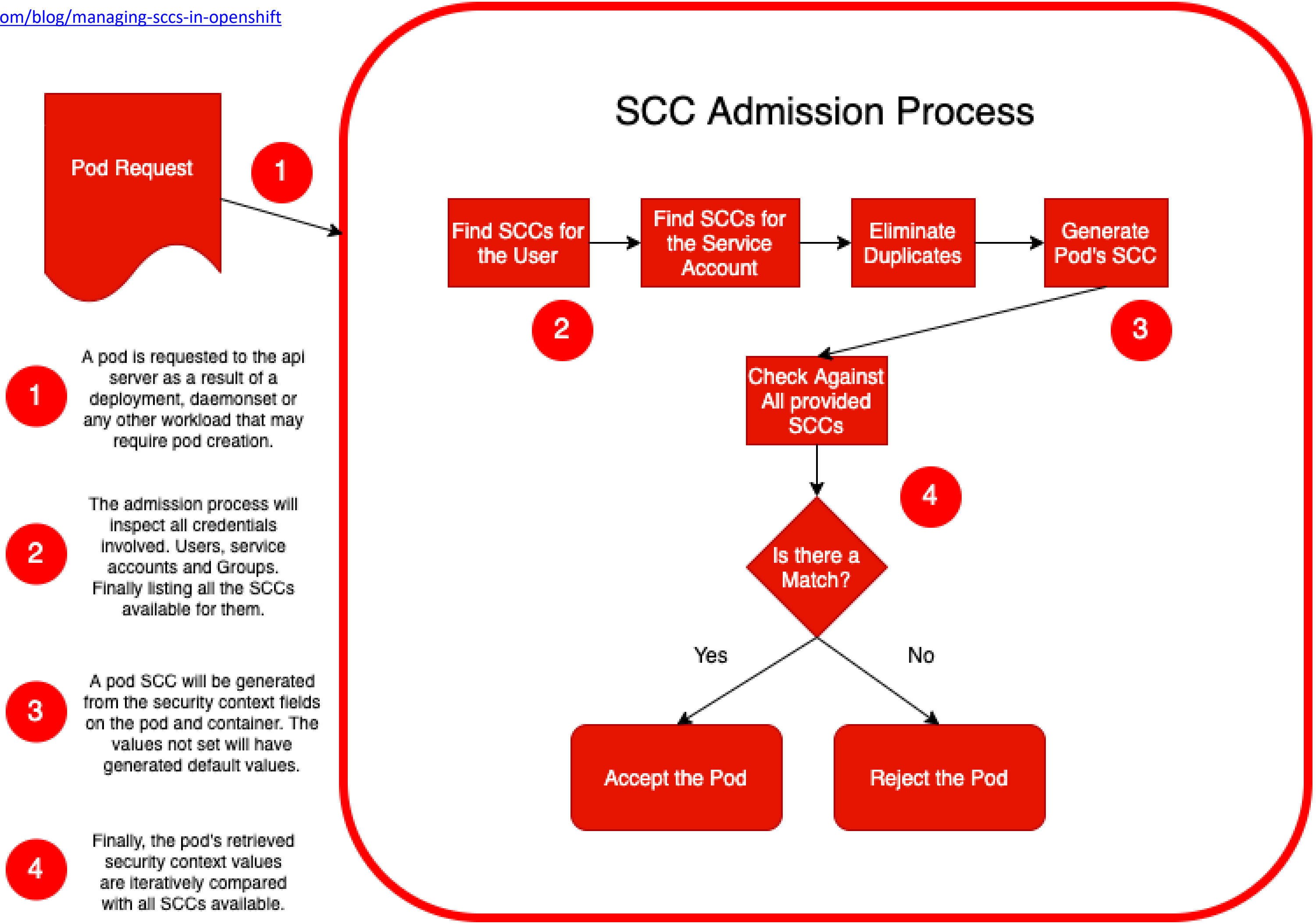
restricted	The most secure. Runs with allocated UID and SELinux context
nonroot	Like restricted, but can have any nonroot UID
anyuid	Equivalent to using UID 0 inside and outside container.
hostmount-anyuid	Like anyuid, but allows mounting host volumes as well.
hostnetwork	Pod can "see and use" the host network stack directly.
node-explorer	Designed for Prometheus to retrieve metrics from the cluster.
hostaccess	Allows access to all host namespaces, file systems and PIDS.
privileged	Total control of Pod and host.

Caution!!!

Granting Additional Permissions

- The restricted SCC is applied by default
- If additional permissions are required, we need to use a different predefined or custom SCC
- When a user requests a Pod, the user’s credentials are used to authorize.
- Then a service account is allocated to the pod (usually default)
- Based on the user & service account the SCC admission process checks the set of available SCCs for a match between the requested SC and the constraints.
- No match, no Pod.

https://cloud.redhat.com/blog/managing-sccs-in-openshift



1 A pod is requested to the api server as a result of a deployment, daemonset or any other workload that may require pod creation.

2 The admission process will inspect all credentials involved. Users, service accounts and Groups. Finally listing all the SCCs available for them.

3 A pod SCC will be generated from the security context fields on the pod and container. The values not set will have generated default values.

4 Finally, the pod's retrieved security context values are iteratively compared with all SCCs available.

Deploying using oc

- The oc command line is often used by cluster admins
- But if a cluster admin deploys pods using oc, that pod can have elevated permissions!
    - Not secure
    - Confusing when unprivileged users attempt same operations.
- Best to test a deployment with lower privileges:

```
$ oc apply -f my-deployment.yaml --as=my-unprivileged-user
```

Binding and removing SCCs

- This can be done in the command line
  - For users

```
oc adm policy add-scc-to-user <scc-name-here> <user-name>
oc adm policy remove-scc-from-user <scc-name-here> <user-name>
```

  - For Service Accounts

```
oc adm policy add-scc-to-user <scc-name-here> -z <service-account-name>
oc adm policy remove-scc-from-user <scc-name-here> -z <service-account-name>
```

  - For User Groups

```
oc adm policy add-scc-to-group <scc-name-here> <group-name>
oc adm policy remove-scc-from-group <scc-name-here> <group-name>
```

Further SCC Topics

- SCCs can be used with RBACs
- SCCs can be managed with the Operator Lifecycle Manager
  - You can use and SCC with your operator to deploy your application.
- More info:
  - Managing security context constraints | Authentication and authorization | OpenShift Container Platform 4.11
  - Managing SCCs in OpenShift (redhat.com)

Summary

- Authorization through RBAC
- Service Accounts
- Security Context Constraints

Questions and Comments?

neueda
futureproof your workforce