Semperti

Openshift

Sprint 4 - Managing Kubernetes Security

Agenda

- Authenticating to the API Server
- Authentication Plugins
- Users
- ServiceAccount
- Authorization

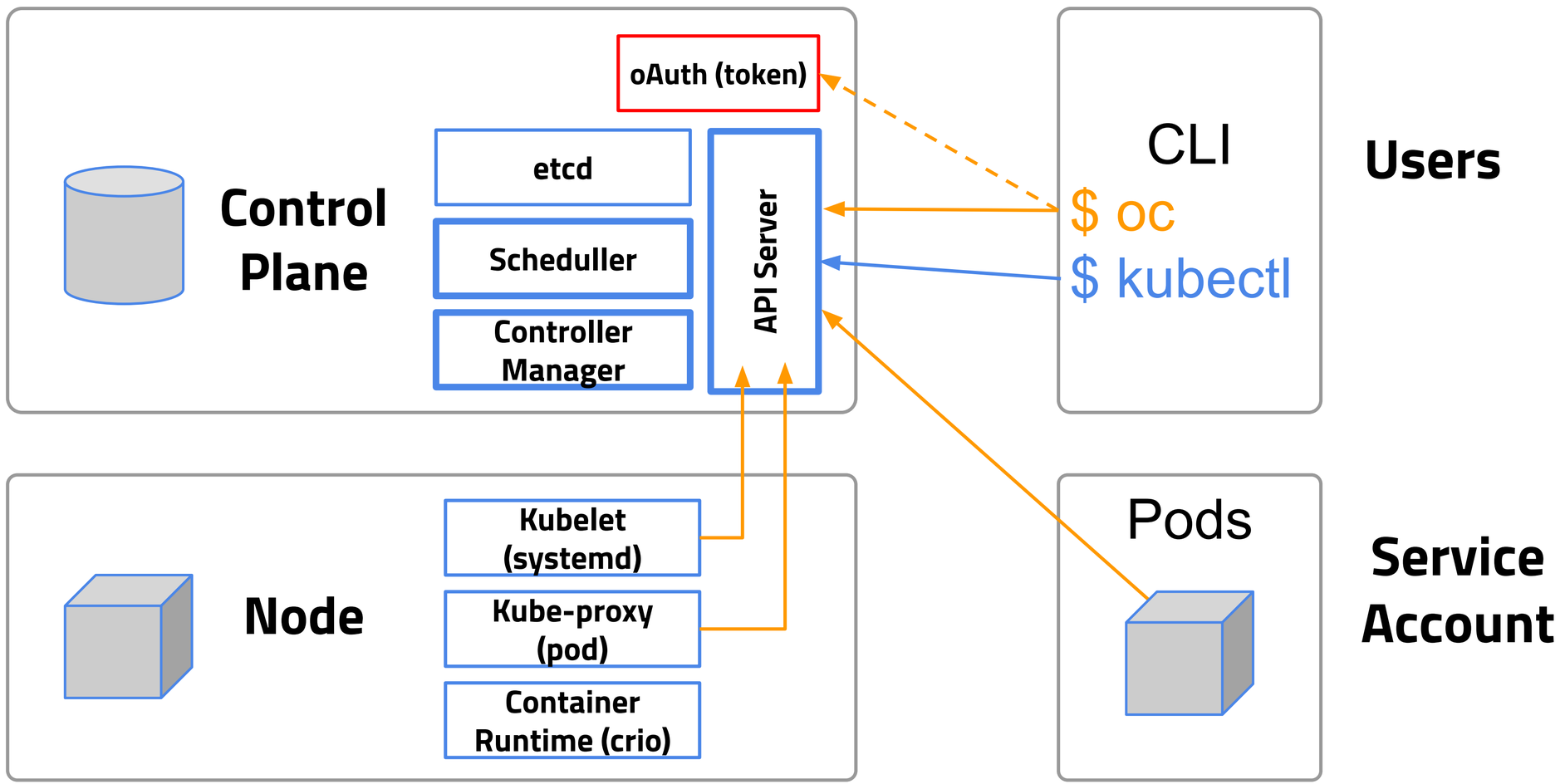
Securing the API Server

Authentication

Authorization

Admission Control

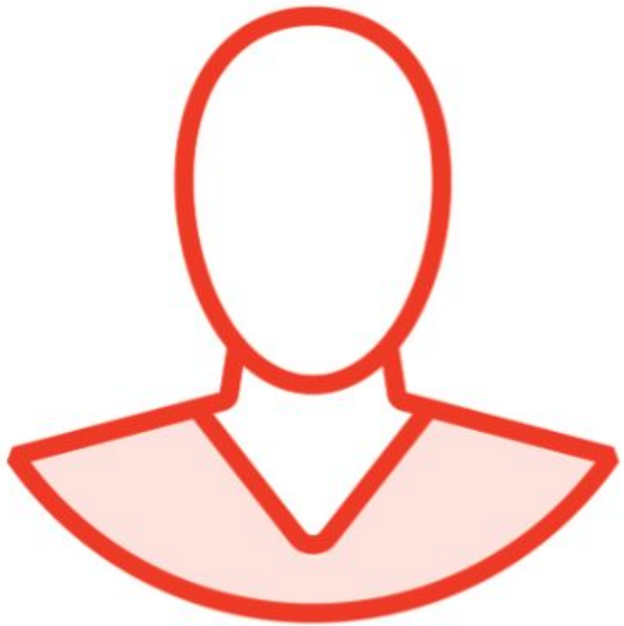
API Server - Authentication



Authentication Plugins

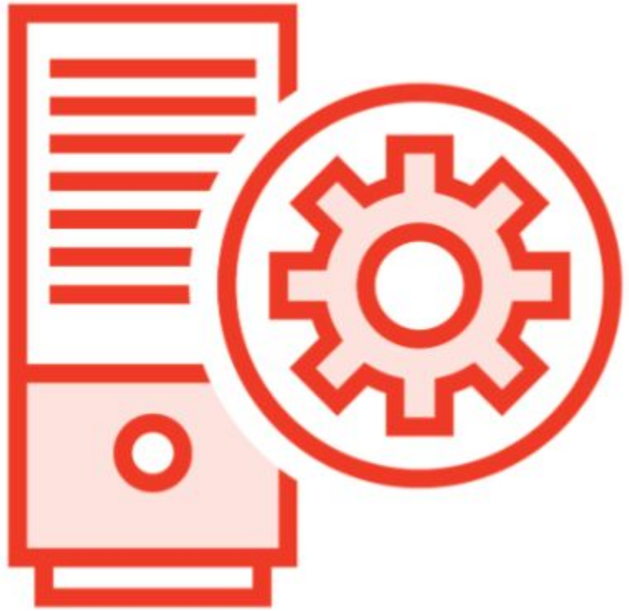
Client Certificates	Authentication Tokens	Basic HTTP	Openshift Authentication
Most commonly used	HTTP Authorization Header in the client request	Static password file	Several Identity Provider
Default when using kubeadm	Service Accounts	Only read during API Server startup	Users populated from Identity Provider like LDAL or AD
Common Name (CN) is the username	Bootstrap Tokens and Static File	Simple to set up and use (Dev)	Physical User Devs and Admin

Users in Kubernetes



- Users are managed by external systems
- Users API Object in Openshift
  - *No Users API Object in Kubernetes*
- Authentication plugin implements authentication
- Authentication is pluggable
- Usernames used for access control and logging
- Users can be aggregated into groups

Service Accounts



- Authenticate Pods to the API Server
- Apply permissions for authorization
- Namespaced API Object
- Default *ServiceAccount* per Namespace
- All Pods must have a *ServiceAccount* defined
- Create *ServiceAccounts* Object

Service Accounts Credentials

- Credential stored as a Secret
  - CA Certificate
  - Token
  - Namespace
- Interact with the API server
- Image pull secret
- Mounted inside a Pod as files using a Volume
- /var/run/secrets/kubernetes.io/serviceaccount

Authentication Process Token

Process authentication in Openshift have the follow steps:

Users	OAuth+Identity Provider	Users receive Token	Reutilization the Token
Foward to OAuth Service	Negotiation together Identity Provider	User receive the token from OAuth Server	The token included in all request to the API Server

*The end of life of the OAuth Token is 86000 seconds (**24hs**) after this the user must be do re login*

https://docs.openshift.com/container-platform/4.5/security/audit-log-view.html

Authentication

Command line

- Login

$ oc login -u USER -p PASSWORD API_URL

- OAuth Token

$ oc get oauthaccesstokens

- Usuario Conectado

$ oc whoami shows current user

- Service Account Token

$ oc sa get-token SERVICE_ACCOUNT

- Web UI

$ oc whoami --show-console

- Server API

$ oc whoami --show-server

- Token

$ oc whoami --show-token

- Oauth Introspection

oc login --loglevel=9 -u USER -p PASSWORD API_URL

ManagingRole Based Access Controls

Semperti

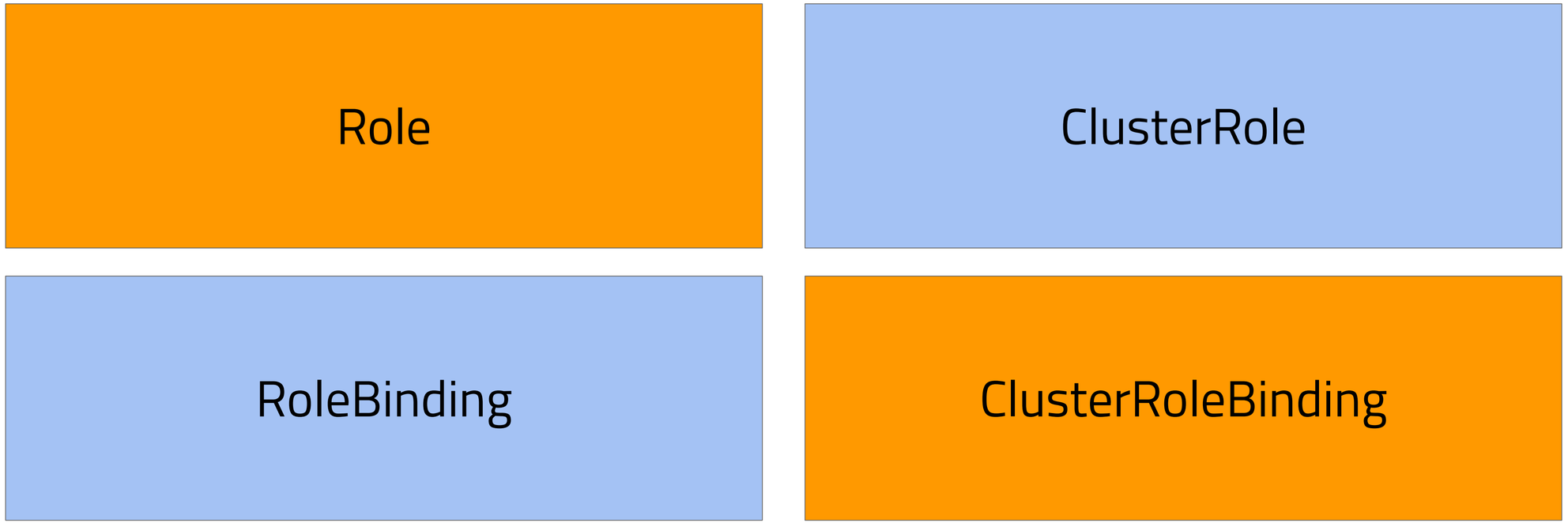
Resumen

- What is Role Based Access Control (RBAC)
- API Objects for configuring RBAC
  - Role and ClusterRole
  - RoleBinding and ClusterRoleBinding

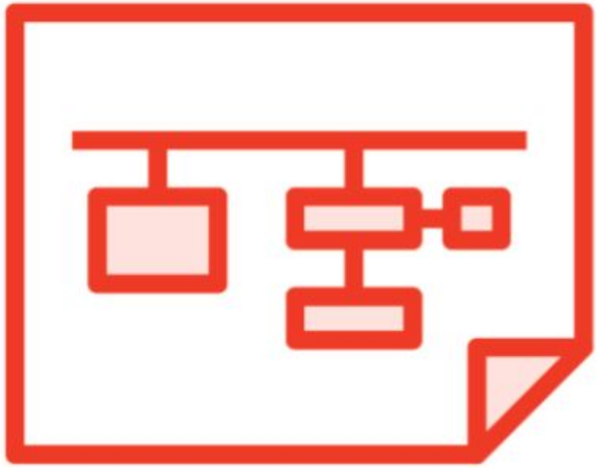
Role Based Access Control (RBAC)

- Authorization plugin enabled on the API Server
- Allowing a requestor to perform actions on resources
- RESTful API semantics
  - Verb on Noun
- Default deny, rules are written to permit actions on the resource
- Subjects - users, groups or ServiceAccounts

API Objects for Implementing RBAC Rules



Roles

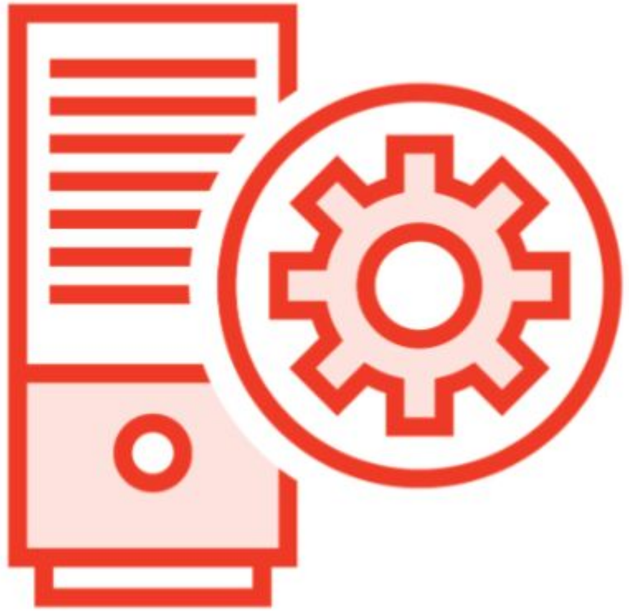


- `Roles` are what can be done to Resources
- `Roles` are made up of one or many `Rules`
- Verbs on resources
  - Get Pods, Create Deployment
- Default deny, add permissions to Resources
- There is no deny permission
- Roles are namespaced

CluterRoles

- Similar to a `Role`, enables access to Resources
- Cluster scoped resources
  - `Nodes`, `PersistentVolumes`
- Give access across more than one namespace
- or all namespaces
- Defining `Roles` in each namespace can
- increase administrative overhead and can be
- error prone

RoleBinding



- Role/ClusterRole only say what can be do
- Defines the Subjects and refers to a Role/ClusterRole
- Who can do what defined in a Role/ClusterRoleRole and *RoleBinding are used in namespaced scoped security*
- ClusterRole and RoleBinding *are used provide access to more than one namespace or the whole cluster*

ClusterRoleBinding

- ClusterRoleBinding grants access cluster-wide
- Combing a ClusterRole with a ClusterRoleBinding
- Will scope security independent of namespace
  - Non-namespaced
  - Cluster-scoped resources

What to use when?

- Use `Role` and a `RoleBinding` to scope security to a single namespace
- Use `ClusterRole` and `RoleBinding` to scope security to several or all namespaces
- Use `ClusterRole` and `ClusterRoleBinding` to scope security to all namespaces OR cluster-scoped resources

Default ClusterRoles

cluster-admin	admin	edit	view
Cluster-wide Super User	Full access within a Namespace	Read/write within a Namespace	Read-Only within a Namespace
RoleBinding - full admin within a Namespace	RoleBinding - full admin within a Namespace	NOT view/edit RolesRoleBindingsResource Quotas	NOT view/edit RolesRoleBindingsResource Quotas
RoleBinding - full admin within a Namespace	Edit RolesRoleBindings	Access to Secrets	No Access to Secrets

Defining Roles and ClusterRoles

Rules

- apiGroups
  - An empty string designates the Core API group
- Resources
  - Pods, Services, Deployments, Nodes and more
- Verbs
  - get, list, create, update, patch, watch, delete, deletecollection

Roles/ClusterRoles can have several Rules defined

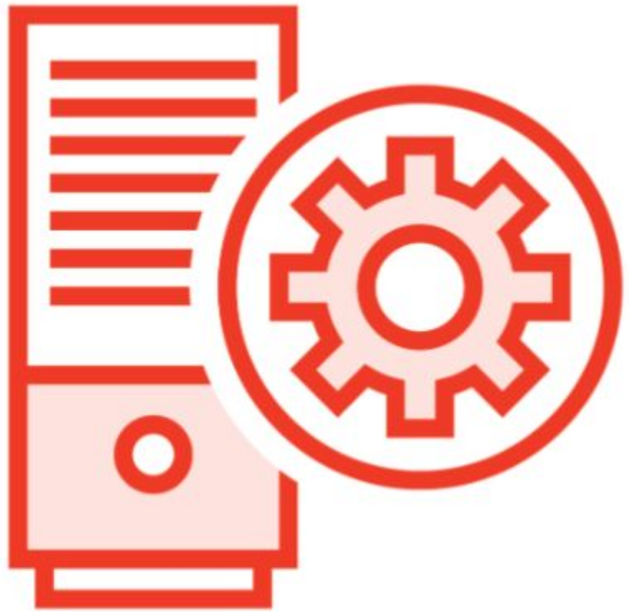
Defining RoleBindings and ClusterRoleBindings

roleRef

- RoleBinding -> Role/ClusterRole
- ClusterRoleBinding -> ClusterRole

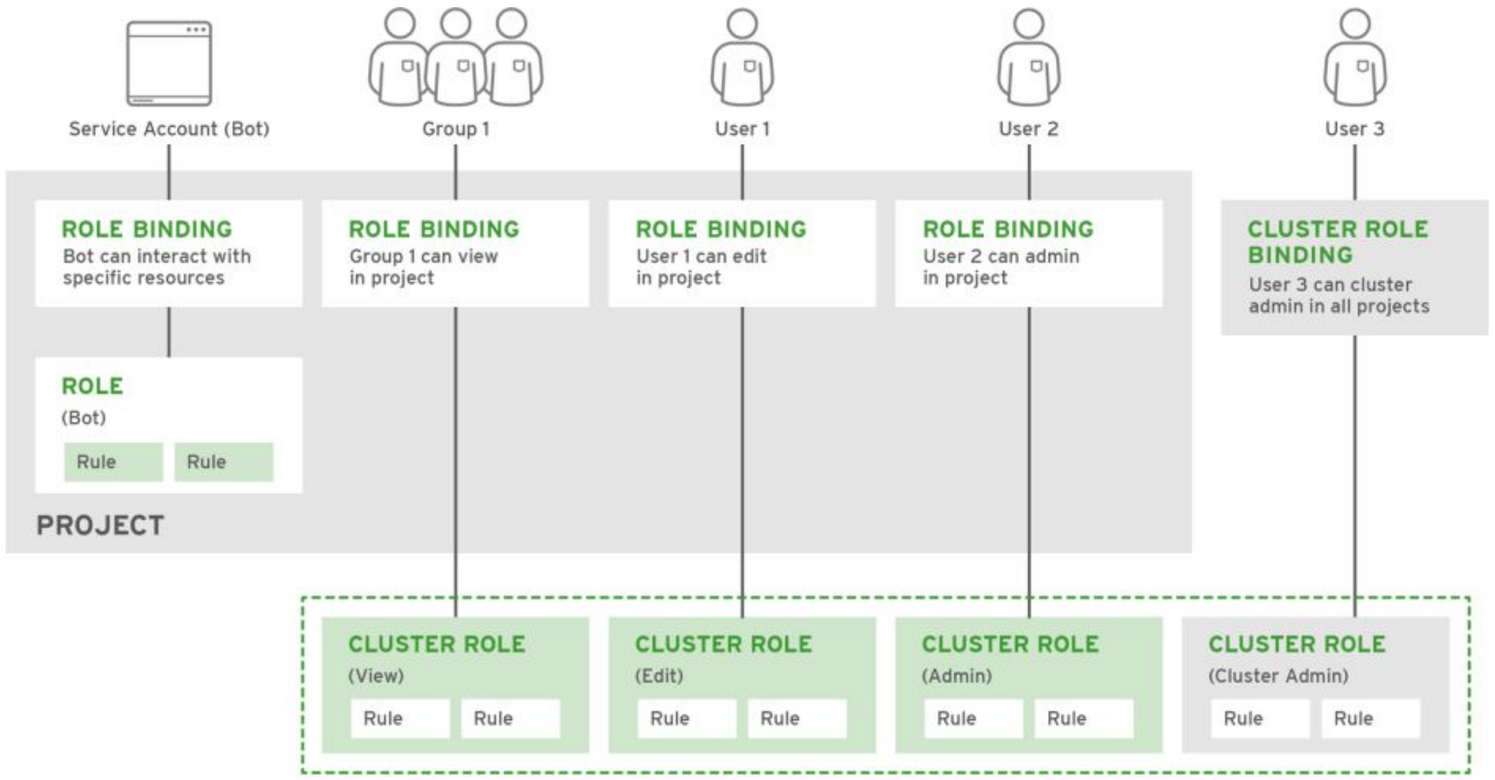
Subjects

- kind (User/Group/ServiceAccount)
- Name
- Namespace



Autorización

Vision General



TIP: El usuario con acceso para crear RoleBindings o ClusterRoleBindings puede otorgar acceso, un usuario no puede otorgar acceso que no tiene

Role and RoleBinding

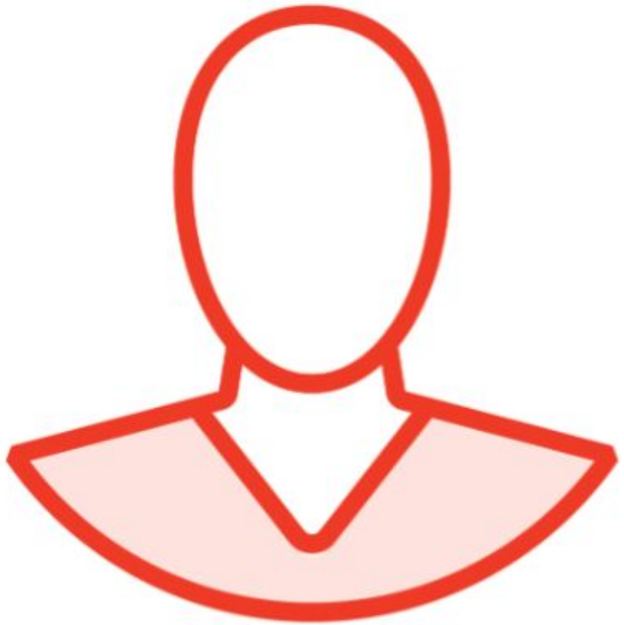
```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
Metadata:
  name: demorole
  namespace: ns1
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
```

```
apiVersion: rbac.authorization.k8s.io/v1
 kind: RoleBinding
 Metadata:
   name: demorolebinding
   namespace: ns1
roleRef:
   apiGroup: rbac.authorization.k8s.io
   kind: Role
   name: demorole
Subjects:
 - apiGroup: rbac.authorization.k8s.io
   kind: User
   name: demouser
```

Role and RoleBinding

```
oc create role demorole \
--verb=get,list \
--resource=pods \
--namespace ns1

oc create rolebinding demorolebinding \
 --role=demorole \
 --user=demouser \
 --namespace ns1
```



Managing Cluster Roles and Roles

Role-Based Access Control

Agregar Role Bindings en un namespace

Add cluster role to user to manage resources in namespace:	`oc policy add-role-to-user CLUSTER_ROLE USER -n NAMESPACE`
Add namespace role to user to manage resources in namespace:	`oc policy add-role-to-user ROLE USER -n NAMESPACE --role-namespace=NAMESPACE`
Add cluster role to group to manage resources in namespace:	`oc policy add-role-to-group CLUSTER_ROLE GROUP -n NAMESPACE`
Add namespace role to group to manage resources in namespace:	`oc policy add-role-to-group ROLE GROUP -n NAMESPACE --role-namespace=NAMESPACE`

Remover Role Bindings en un namespace

Remove cluster role from group in namespace	`oc policy remove-role-from-group CLUSTER_ROLE GROUP -n NAMESPACE`
Remove namespace role from group in namespace	`oc policy remove-role-from-group ROLE GROUP -n NAMESPACE --role-namespace=NAMESPACE`
Remove all role bindings for group in namespace	`oc policy remove-user GROUP -n NAMESPACE`

Managing Cluster Roles and Roles

Role-Based Access Control

Remover Role Bindings en un namespace

Remove cluster role from user in namespace	`oc policy remove-role-from-user CLUSTER_ROLE USER -n NAMESPACE`
Remove namespace role from user in namespace	`oc policy remove-role-from-user ROLE USER -n NAMESPACE --role-namespace=NAMESPACE`
Remove all role bindings for user in namespace	`oc policy remove-user USER -n NAMESPACE`

Cluster Role Binding Management

Add cluster role to user	`oc adm policy add-cluster-role-to-user CLUSTER_ROLE USER`
Add cluster role to group	`oc adm policy add-cluster-role-to-group CLUSTER_ROLE GROUP`
Remove cluster role from user	`oc adm policy remove-cluster-role-from-user CLUSTER_ROLE USER`
Remove cluster role from group	`oc adm policy remove-cluster-role-from-group CLUSTER_ROLE GROUP`

Service Account (SA)
Security Context Contraints (SCC)

Semperti

Service Account

Overview

Service accounts are non-person identities for application integrations:

- Each pod runs as service account
- Used by external agents to access cluster API
- Access managed with role bindings and cluster role bindings

Name	Description
`builder`	Service account usado para build pods, push images
`deployer`	Service account usado para desplegar pods, implementar DeploymentConfig rollout, rollback.
`default`	Service account usado para los pods.

Examples:

- DeploymentConfigs use deployer pod that runs with deployer service account
- Applications in pod containers make API calls for discovery
- Jenkins servers use API to create agents running in containers
- Operators use cluster API to watch custom resources and API to manage ConfigMaps, deployments, etc.

Command Line

```
oc create serviceaccount NAME -n NAMESPACE

oc get serviceaccount NAME -n NAMESPACE

oc describe serviceaccount NAME -n NAMESPACE

oc delete serviceaccount NAME -n NAMESPACE
```

Service Account

Service Account Token

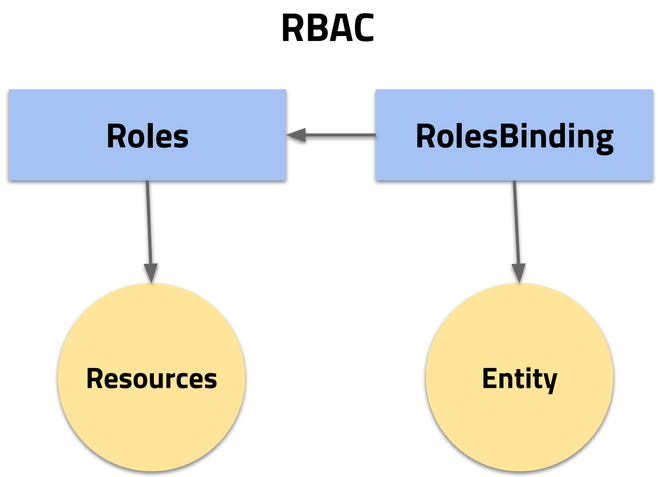
- Service accounts use tokens to authenticate to cluster API
- Within running container, find service account token in /run/secrets/kubernetes.io/serviceaccount/token file
- Tokens used by external agents to act as service account with OpenShift® API
  - Allow external agent to access integrated container image registry
  - Allow existing Jenkins server to run agents as pods within cluster

- List service account tokens:

  oc get secret --field-selector=type=kubernetes.io/service-account-token -n NAMESPACE
- Get active token for service account:

  oc serviceaccount get-token SERVICE_ACCOUNT -n NAMESPACE
- Get specific token from secret:

  oc describe secret SECRET -n NAMESPACE



Security Context Constraints

SCCs

- Role-based access control controls what users can do
- Security context constraints (SCCs), in contrast, control:
  - Actions pod can perform
  - What pod can access
- SCCs define conditions pod must run with to be accepted into system

SCCs let administrator control:

- Capabilities container can request to be added
- Use of host directories as volumes
- SELinux context of container
- User ID
- Use of host namespaces and networking
- Allocation of FSGroup that owns pod's volumes
- Configuration of allowable supplemental groups
- Requirement for use of read-only root file system
- Usage of volume types
- Configuration of allowable secure computing mode (seccomp) profiles

- Add SSC User

```
oc adm policy add-scc-to-user SCC_NAME  USER_NAME
```

- Add SSC Group

```
oc adm policy add-scc-to-group SCC_NAME GROUP_NAME
```

- Remove SSC User

```
oc adm policy remove-scc-from-user SCC_NAME USER_NAME
```

- Remove SSC Group

```
oc adm policy remove-scc-from-group SCC_NAME USER_NAME
```

Security Context Constraints

SCC CLI

SCC	Description
anyuid	Allow containers to run as any user ID, *including* root user (uid=0)
hostaccess	Allow containers to access host file systems, network, and process table with restricted user ID
hostmount-anyuid	Allow containers to access host file system using host mounts, run as any user ID
hostnetwork	Allow containers access to host networking, host ports
node-exporter	Reserved for use by Prometheus node exporter
nonroot	Allow containers to run as any user ID *except* root user (uid=0)
privileged	Allow access to all privileged and host features—most relaxed access, use only for cluster administration
restricted	Deny access to all host features, require pod containers to run with restricted UID (default SCC)

Muchas gracias!

Gonzalo Acosta <gonzalo.acosta@semperti.com>
Emilio Buchailliot <emilio.buchaillot@semperti.com >

Semperti