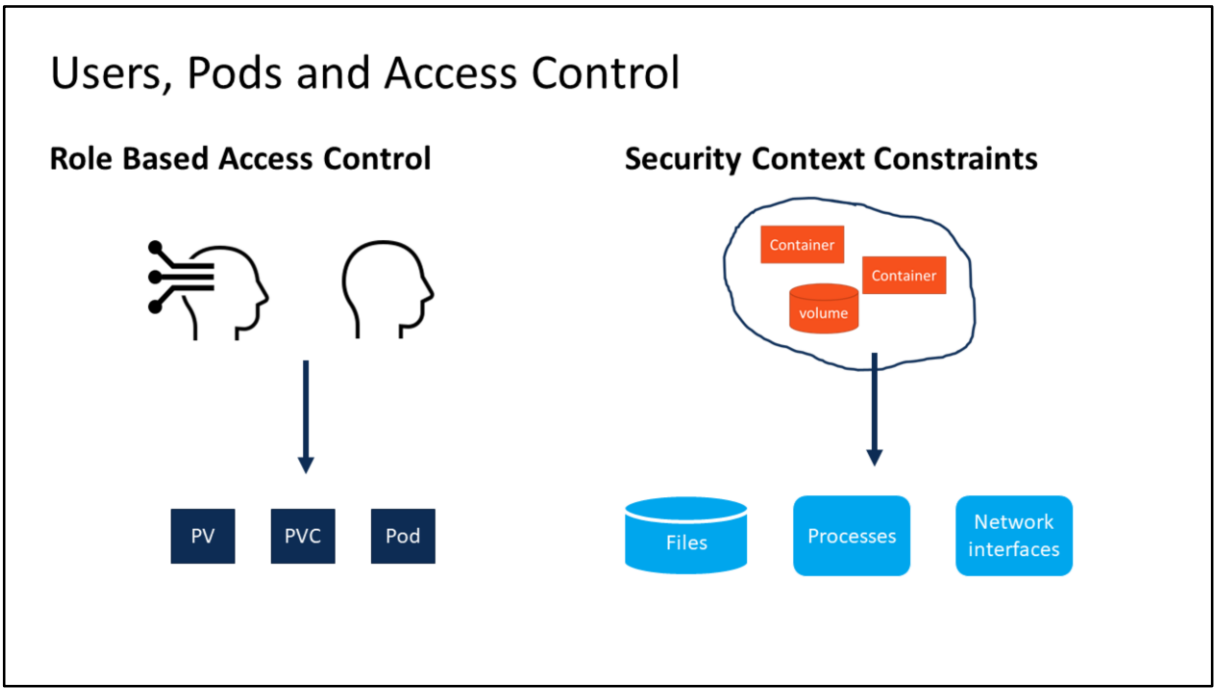
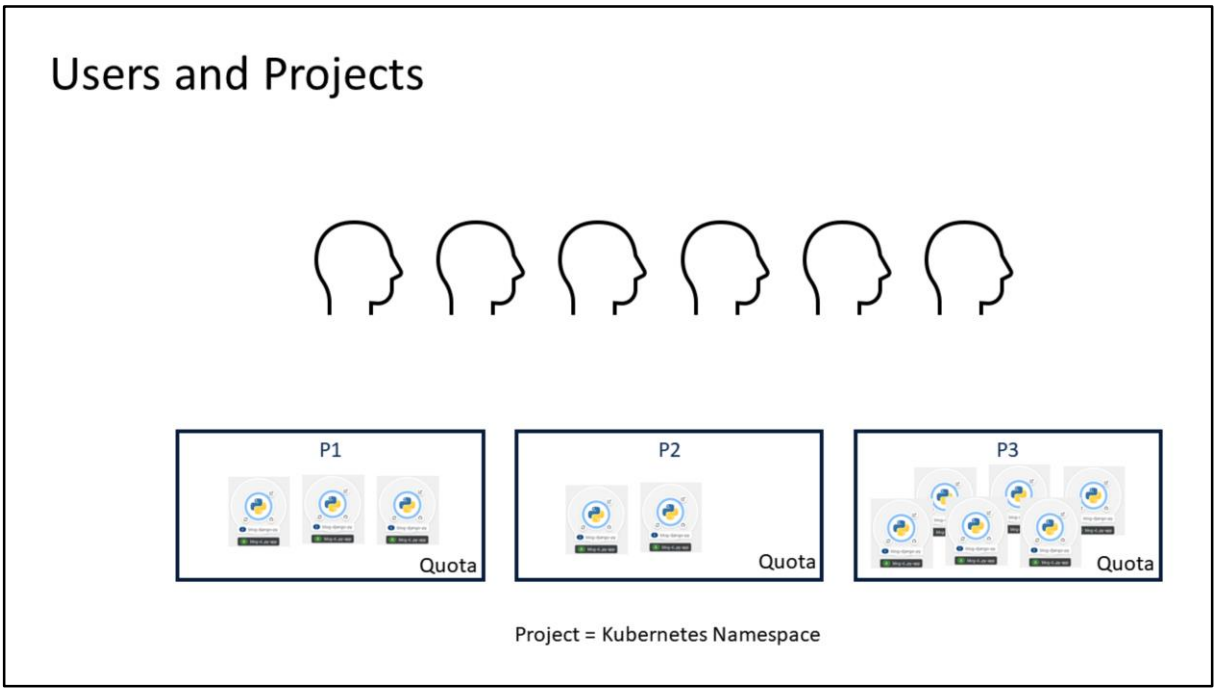
Module 5 OpenShift Security

RBAC resources and OAuth

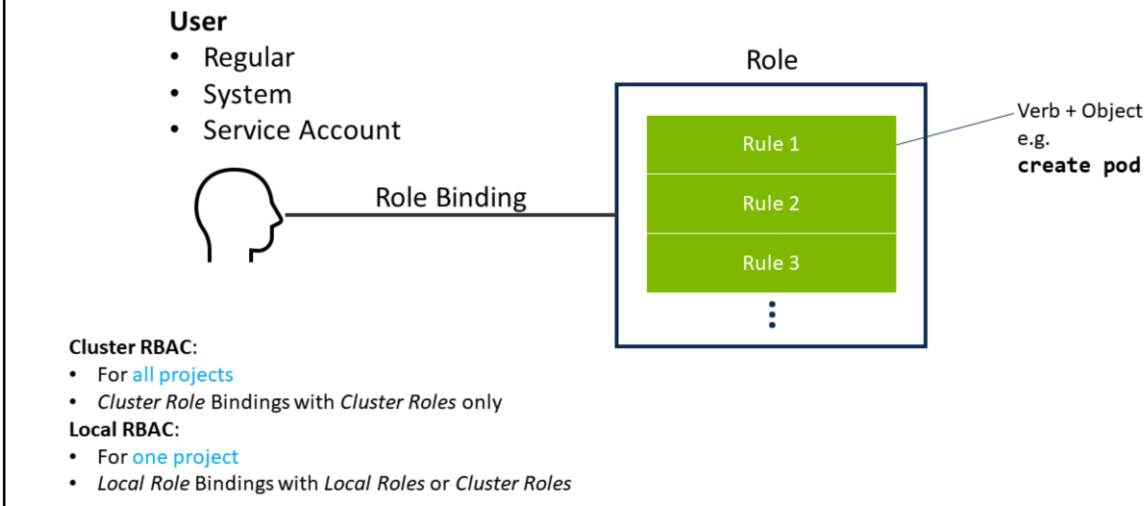


Role Based Access Control, RBAC, configures what users are allowed to do with which OpenShift resources.
OpenShift has three categories of users: Regular users, System users, and Service Accounts. Regular and system users represent human users of an OpenShift cluster. Service accounts are given to pods. Security Contexts are attached to pods, not human users. They describe what a pod can do with operating system resources like files and processes. A pod can request security contexts in order to perform tasks at the operating system level. Most such pods carry out administrative functions in an OpenShift cluster. Security Context Constraints, SCCs, are lists of rules that limit which security context items a pod can request.



Projects are collections of OpenShift resources. They are **namespaces**, i.e. inside a project, a resource can be named without worrying if the same name exists in the cluster already. They are also **access units** - administrators grant users access to one or more projects. Project **quotas** limit the amount of resources in a project.
What a user can do in a project is determined by its role binding (see next page). Users can have privileges to just view project resources, or change them, or they can even have an administrative role in a project. Users may also have permission to create their own projects.
The resources in a project are **objects** such as Kubernetes workloads, **policies** that contain project access rules, **constraints** (quotas), and **service accounts** for bots that automatically perform actions.

RBAC Principles - Users and their Roles



Cluster RBAC:
- For all projects
- *Cluster Role* Bindings with *Cluster Roles* only

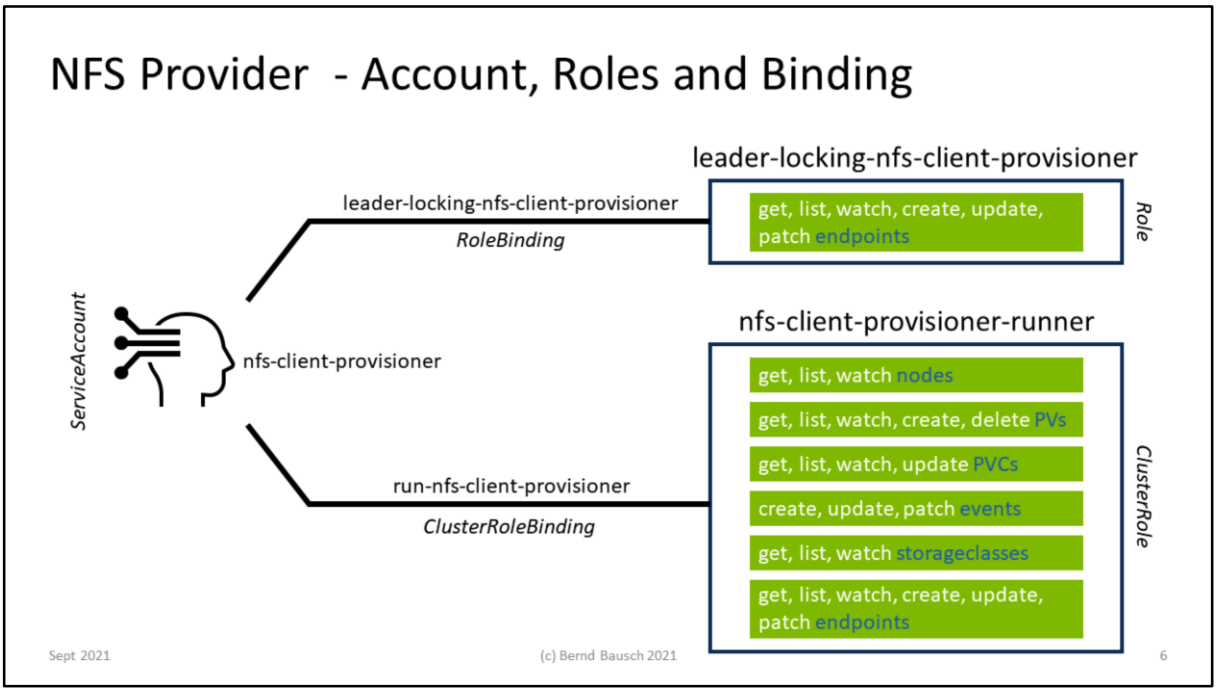
Local RBAC:
- For one project
- *Local Role* Bindings with *Local Roles* or *Cluster Roles*

RBAC - Cluster Roles

Cluster Role	Rights
cluster-admin	Cluster binding: Perform any action in any project Local binding: Quota control and any action on any resource in the project
admin	Local binding: View and modify any resource in the project (limited by quota)
self-provisioner	Create projects
edit	Local binding: Modify project's objects except roles or bindings
view	Local binding: View project's objects except roles or bindings
basic-user	Get basic info about projects and users
cluster-status	Get basic cluster status info

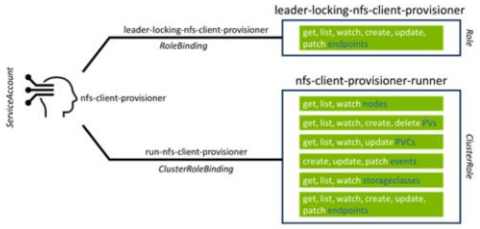
Sept 2021 (c) Bernd Bausch 2021 5

While it is possible to modify the global cluster roles, it is not recommended to do so.
These roles can be used in the global cluster context (global binding) and a project context (local binding).
A user that has cluster binding to the cluster-admin role is the cluster's superuser. A user can also have the cluster-admin role bound locally. In this case, it has full powers in a project.
The admin role makes sense with local binding only and can be seen as a project superuser that can't change the quota.
The self-provisioner role makes sense with cluster binding only. A user with this role can create projects.
The edit and view roles can be described as regular users in a project that can change or just view project resources. They can't access roles or bindings.
The two last roles make sense with cluster binding. They allow a user to view projects and other cluster information.


NFS Provider - Account, Roles and Binding
leader-locking-nfs-client-provisioner
leader-locking-nfs-client-provisioner
RoleBinding
get, list, watch, create, update, patch endpoints
Role
ServiceAccount
nfs-client-provisioner
nfs-client-provisioner-runner
get, list, watch nodes
get, list, watch, create, delete PVs
get, list, watch, update PVCs
create, update, patch events
get, list, watch storageclasses
get, list, watch, create, update, patch endpoints
ClusterRole
run-nfs-client-provisioner
ClusterRoleBinding
Sept 2021
(c) Bernd Bausch 2021
6

NFS Provider - Account, Roles and Binding

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nfs-client-provisioner
  namespace: default
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: nfs-client-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
    ...
```



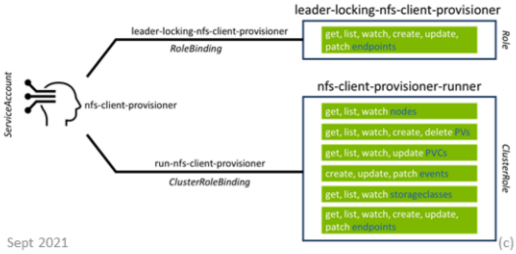

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: run-nfs-client-provisioner
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    namespace: default
roleRef:
  kind: ClusterRole
  name: nfs-client-provisioner-runner
  apiGroup: rbac.authorization.k8s.io
```

Sept 2021 (c) Bernd Bausch 2021 7

NFS Provider - Account, Roles and Binding

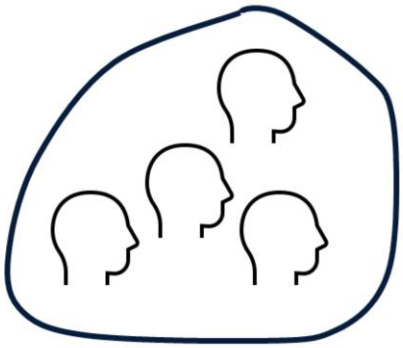
```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  namespace: default
rules:
  - apiGroups: [""]
    resources: ["endpoints"]
    verbs: ["get", "list", "watch", "create",
            "update", "patch"]
```

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  namespace: default
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    namespace: default
roleRef:
  kind: Role
  name: leader-locking-nfs-client-provisioner
  apiGroup: rbac.authorization.k8s.io
```



Sept 2021 (c) Bernd Bausch 2021 8

RBAC Principles - Groups

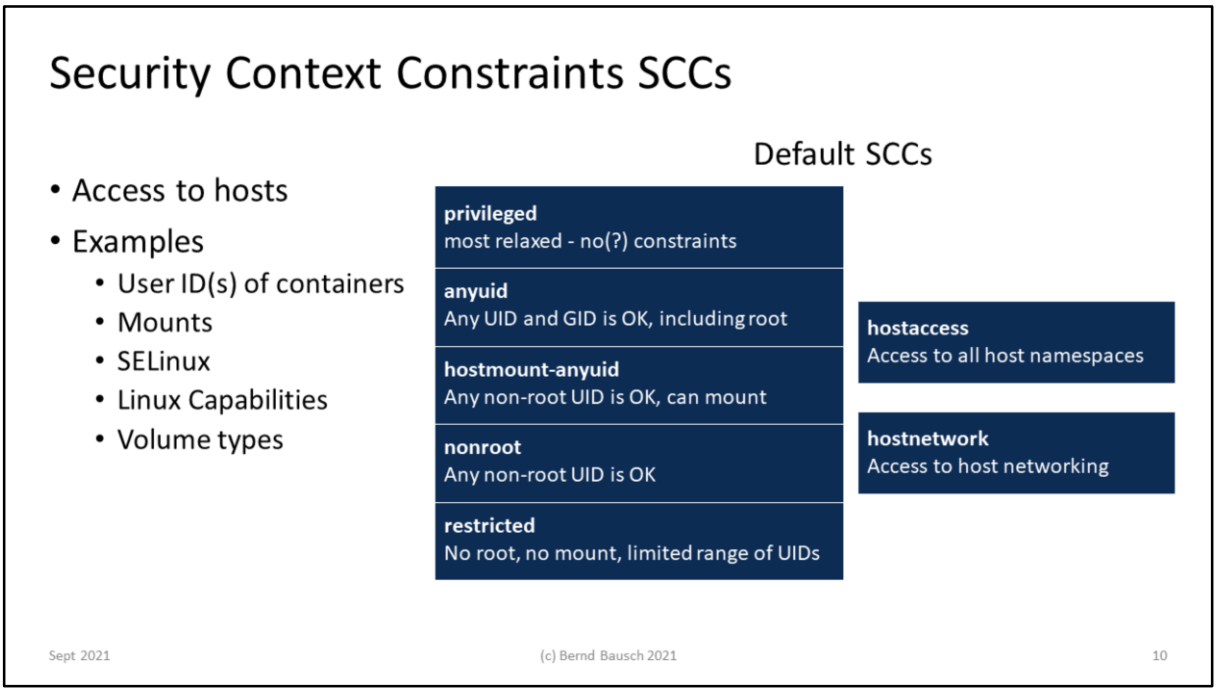


Regular group

- Set up manually
- Convenient for granting permissions

System Group a.k.a. Virtual Group

- Set up automatically
- `system:authenticated`
  group of authenticated users
- `system:authenticated:oauth`
  group of users with OAuth token
- `system:unauthenticated`
  group of unauthenticated users

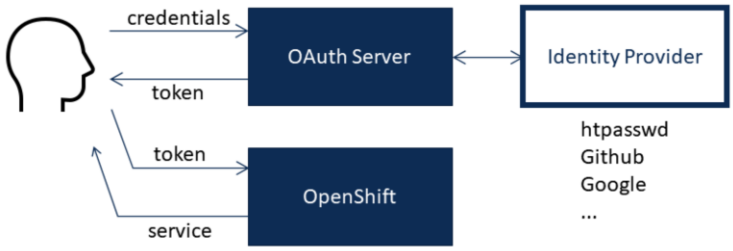


RBAC controls users' access to cluster resources. The actions that pods can perform and the resources they can access are controlled by a different concept, **Security Context Constraints** or **SCCs**.

Authentication

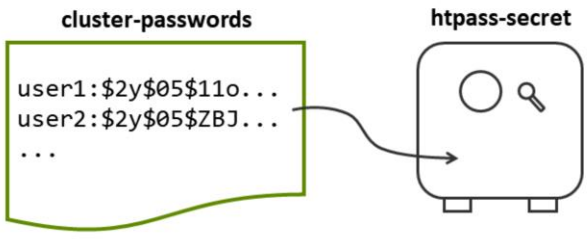
OAuth
Obtain access token from cluster
Use token for authentication

X.509 Certificates
TLS
Created by API server
Used by controllers



Sept 2021 (c) Bernd Bausch 2021 11

Create an HTPASSWD identity provider




```
$ oc create secret generic htpass-secret
    --from-file=htpasswd=cluster-passwords
    -n openshift-config

$ oc apply -f identity-provider.yaml
```

identity-provider.yaml

```
apiVersion:
config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: local_users
    mappingMethod: claim
    type: HTPasswd
    htpasswd:
      fileData:
        name: htpass-secret
```

Sept 2021 (c) Bernd Bausch 2021 12

OpenShift supports several identity providers such as LDAP directories, Keystone (OpenStack identity service), Google etc. A very simple identity provider is htpasswd, which is based on a file residing on a controller. It's a plain text file that contains the names and encrypted passwords of user accounts. On the right, a manifest for a list of identity providers. It's a custom resource of kind OAuth. The list displayed here features a single provider named local_users. It refers to a secret (named htpass-secret in this example) that contains the users and their passwords in htpasswd format. htpasswd is a fairly standard utility for creating simple web-site accounts. The oc create secret command shows how this secret can be created from the htpasswd file. To add the identity provider to the cluster, the manifest must be applied.