**What is RBAC**

In kubernetes there are several authorization mechanism like RBAC, ABAC.

With RBAC we can add constraints to access kubernetes resources. For example we can give permission to listing pods for specific namespace to a ServiceAccount and prevent it to delete any resource. We can do it for a specific namespace or cluster wide as well.

**Key points of RBAC**

There are 3 important concept in RBAC.

* **Subject :**Users, groups or service accounts.
* **Resources :**Kubernetes API objects which we will operate on.
* **Verbs :** The operations which we want to do with our resources.

**Role & ClusterRole**

Role and ClusterRole contains set of rules to access & modify kubernetes resources. Difference between them is Role works in particular namespace while ClusterRole is cluster wide which is obvious from it’s name.

If you need define permissions inside a namespace use Role, if you need that cluster wide use ClusterRole.

Creating Role via kubectl is simple like I showed you below:

kubectl create role my-custom-role --verb=list --resource=pods --namespace k8boss

We can also specify multiple verbs and resources as well:

kubectl create role my-custom-role --verb=list --verb=get --resource=pods --resource=services --namespace k8boss

If we want to make it via yaml:

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

name: my-custom-role

namespace: k8boss

rules:

- apiGroups:

- "\*"

resources:

- services

- pods

verbs:

- get

- list

kubectl apply -f role.yaml

If we did will not specify namespace in our role.yaml it will considered as default namespace.

ClusterRole is not much differs:

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: my-custom-cluster-role

rules:

- apiGroups:

- "\*"

resources:

- services

- pods

verbs:

- get

- list

It is almost same with Role definition only without a namespace.

**ServiceAccount**

Service accountsare used by processes inside pods to interact with the Kubernetes API. If you write custom application to work with kubernetes you will probably need to create a custom ServiceAccount with some specialized roles assigned to it. For example when you deploy kubernetes dashboard you can give additional roles to kubernetes-dashboard service account or create a custom ServiceAccount with a cluster-admin role bindings.

Creating a ServiceAccount is so simple with kubectl:

kubectl create serviceaccount custom-sa -n k8boss

Or yaml:

apiVersion: v1

kind: ServiceAccount

metadata:

name: custom-sa

namespace: k8boss

After creating ServiceAccount lets show it’s yaml content:

kubectl get sa custom-sa -n k8boss -o yaml :

apiVersion: v1

kind: ServiceAccount

metadata:

creationTimestamp: "2020-04-13T22:11:49Z"

name: custom-sa

namespace: k8boss

resourceVersion: "1653"

selfLink: /api/v1/namespaces/test/serviceaccounts/test-sa

uid: 8fb1df02-6bab-43f1-afe8-3ede5f4a1710

secrets:

- name: custom-sa-token-jcq7h

As you can see there is an additional field “secrets” which we did not specify while creating ServiceAccount. A secret has been created which related to our ServiceAccount by kubernetes.

Lets take a look our secret:

kubectl get secret custom-sa-token-jcq7h -n k8boss -o yaml :

apiVersion: v1

data:

ca.crt: 

namespace: dGVzdA==

token: 

kind: Secret

metadata:

annotations:

kubernetes.io/service-account.name: custom-sa

kubernetes.io/service-account.uid: 8fb1df02-6bab-43f1-afe8-3ede5f4a1710

creationTimestamp: "2020-04-13T22:11:49Z"

name: custom-sa-token-jcq7h

namespace: k8boss

resourceVersion: "1652"

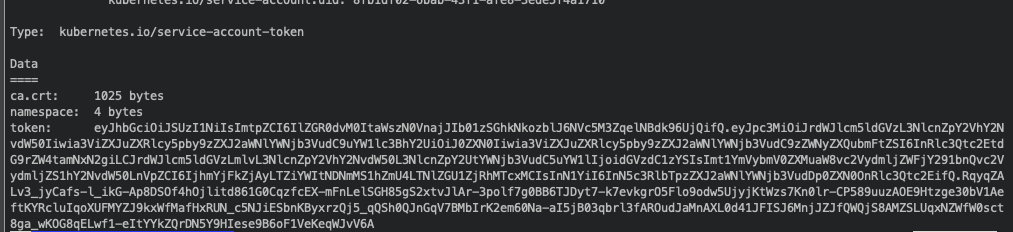
selfLink: /api/v1/namespaces/k8boss/secrets/custom-sa-token-jcq7h

uid: 45956fb0-2449-48c8-a0d3-d1a088a1f4ea

type: kubernetes.io/service-account-token

We can see a token key value here. We can use these secret to access kubernetes-dashboard. But we need to decode it because kubernetes stores secret values in base64 encoded format. Simply we can describe secret and take the token value:

kubectl describe secret custom-sa -n k8boss :



**RoleBinding & ClusterRoleBinding**

Role bindings as indicates from names makes us bind roles to subjects (user or set of users or service accounts). Again the only difference is in usage, we use RoleBinding to bind Roles while ClusterRoleBinding to bind ClusterRoles. But as an addition we can use RoleBinding to bind a ClusterRole to a Role within a namespace.

Before creating role binding we can ask our service account’s capabilities with:

kubectl auth can-i list pods --as=system:serviceaccount:k8boss:custom-sa -n k8boss

You will see “no” as output of that command.

Lets create a role binding with kubectl:

kubectl create rolebinding my-custom-role-binding --role=my-custom-role --serviceaccount=k8boss:custom-sa --namespace k8boss

Equivalent yaml file:

apiVersion: v1

items:

- apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: my-custom-role-binding

namespace: k8boss

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: my-custom-role

subjects:

- kind: ServiceAccount

name: custom-sa

namespace: k8boss

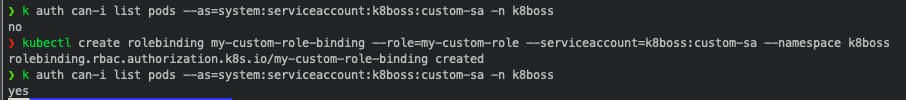
ClusterRoleBinding is not much differs:

kubectl create clusterrolebinding my-custom-clusterrole-binding --clusterrole=my-custom-cluster-role --serviceaccount=k8boss:custom-sa

Now again lets ask our capabilities:

kubectl auth can-i list pods --as=system:serviceaccount:k8boss:custom-sa -n k8boss

Now you can see “yes” as an answer.



**Aggregated ClusterRoles**

Last thing we will mention is Aggregated ClusterRoles. Sometimes we might want to combine several cluster role into one.

To do this we can use aggregationRule field of ClusterRole. We can select roles using label selectors. For example lets assume we have two ClusterRole one for listing pods one for listing services. So we need another ClusterRole to list pods and services. We can now use aggregationRule to combine our two ClusterRoles.

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: my-custom-cluster-role

labels:

rbac.example.com/aggregate-to-pod-list: "true"

rules:

- apiGroups:

- "\*"

resources:

- pods

verbs:

- list

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: my-custom-cluster-role-2

labels:

rbac.example.com/aggregate-to-service-list: "true"

rules:

- apiGroups:

- "\*"

resources:

- service

verbs:

- list

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: custom-aggregated-role

aggregationRule:

clusterRoleSelectors:

- matchLabels:

rbac.example.com/aggregate-to-pod-list: "true"

rbac.example.com/aggregate-to-service-list: "true"

rules: [] # The control plane automatically fills in the rules