Authorization

» RBAC

Exploring Kubernetes Roles

A role contains rules that represent a set of permissions.

For role definitions limited to a specific namespace, we use the *role* resource.

List any roles in the default namespace

```
kubectl get roles
```

List any roles in the `kube-system` namespace

```
kubectl get roles -n kube-system
```

The prefix `system:` is reserved for Kubernetes system use.

Take a closer look at the details of the `system:controller:bootstrap-signer` role within the `kube-system` namespace.

This role permits get,list,and watch access to Kubernetes secrets.

```
kubectl get role system:controller:bootstrap-signer -o yaml -n kube-system
```

List any rolebindings in the `kube-system` namespace

```
kubectl get rolebinding -n kube-system
```

View the details of the `system:controller:bootstrap-signer` rolebinding.

This rolebinding associates the `system:controller"bootstrap-signer` role with the `bootstrap-signer` service account.

```
kubectl get role system:controller:bootstrap-signer -o yaml -n kube-system
```

Does the bootstrap-signer service account exist? It should return nothing.

```
kubectl get sa -n kube-system
```

A *ClusterRole* can be used to grant the same permissions as a *Role*, but can also grant access to non-namespaced resources (nodes, namespaces, componentStatus).

List any ClusterRoles in the cluster

```
kubectl get clusterrole
```

Take a closer look at the details of the `admin` and `cluster-admin` roles. Both contain the same privileges.

```
kubectl get clusterrole admin cluster-admin -o yaml
```

List any clusterrolebindings in the cluster:

```
kubectl get clusterrolebinding
```

View the details of the `admin-user` and `cluster-admin` ClusterRoles.

```
kubectl get clusterrolebinding admin-user cluster-admin -o yaml
```

Take note of the subjects list. Both of these bindings are are applying the `cluster-admin` ClusterRole to specific subjects (user, group, or serviceaccount).

It’s important to note that there is no concept of user or group resources in Kubernetes. These must managed by a Kubernetes Administrator using any of the various supported authenticator modules compatible with K8s (x509 certs, bearer token, static password/token file, OIDC, etc.). Multiple authenticators can be specified in which case each one can be tried in sequence.

Let’s take a look at the the signed client certificate we are currently using with `kubectl`

```
grep 'client-certificate-data' generated/auth/kubeconfig | cut -c 30- | base64 -D | openssl x509 -text
```

Observe the `O=system:masters` for the organization subject. This automatically authenticates kubelet as a member of the system:masters group as defined in the `cluster-admin` RoleBinding.

Rather than generate new x509 certificates signed by the cluster CA, let’s create a new service account in the default namespace

```
kubectl create serviceaccount testuser
```

If we wanted this user only to be able to view resources in the default namespace, we can create a new role.

```
cat > intern-role.yaml<<EOF
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: default
  name: intern
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - endpoints
  - persistentvolumeclaims
  - pods
  - replicationcontrollers
  - replicationcontrollers/scale
  - serviceaccounts
  - services
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - bindings
  - events
  - limitranges
  - namespaces/status
  - pods/log
  - pods/status
  - replicationcontrollers/status
  - resourcequotas
  - resourcequotas/status
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - get
```

```
    - get
    - list
    - watch
  - apiGroups:
    - apps
    resources:
    - deployments
    - deployments/scale
    - statefulsets
    verbs:
    - get
    - list
    - watch
  - apiGroups:
    - autoscaling
    resources:
    - horizontalpodautoscalers
    verbs:
    - get
    - list
    - watch
  - apiGroups:
    - batch
    resources:
    - cronjobs
    - jobs
    - scheduledjobs
    verbs:
    - get
    - list
    - watch
  - apiGroups:
    - extensions
    resources:
    - daemonsets
    - deployments
    - deployments/scale
    - ingresses
    - replicasets
    - replicasets/scale
    - replicationcontrollers/scale
    verbs:
    - get
    - list
    - watch
```

```
EOF
```

Create the `intern` role within the default namespace

```
kubectl create -f intern-role.yaml
```

Now we can bind the `intern` role to `testuser` service account

```
kubectl create rolebinding intern-testuser --role=intern --serviceaccount=default:testuser
```

Let’s set a variable for our `testuser` secret name

```
export TEST_USER_SECRET=`kubectl get sa testuser --output=jsonpath={.secrets[*].name}`
```

Using the secret name, let’s extract the JSON Web Token (bearer token)

```
export TEST_USER_TOKEN=`kubectl get secret $TEST_USER_SECRET -o jsonpath={.data.token} | base64 -D`
```

We can now set our kubeconfig file with new credentials

```
kubectl config set-credentials --token=$TEST_USER_TOKEN testuser
```

Set a new context for our new credentials. We be using the same cluster.

```
kubectl config set-context --cluster=local --user=testuser testuser
```

Enable the context

```
kubectl config use-context "testuser"
```

Verify the modification we just made

```
kubectl config view
```

List pods in the default namespace

```
kubectl get pods
```

Attempt list pods in an alternate namespace. We should be denied.

```
kubectl get pods -n kube-system
```

Switch back to our previous context.

```
kubectl config use-context <default-context-name>
```

We will explore additional ways to manage users and groups in the Dex lab.