RBAC Role Based Access Control

What is RBAC?

Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within your organization.

RBAC authorization uses the rbac.authorization.k8s.io API to drive authorization decisions, allowing you to dynamically configure policies through the Kubernetes API.

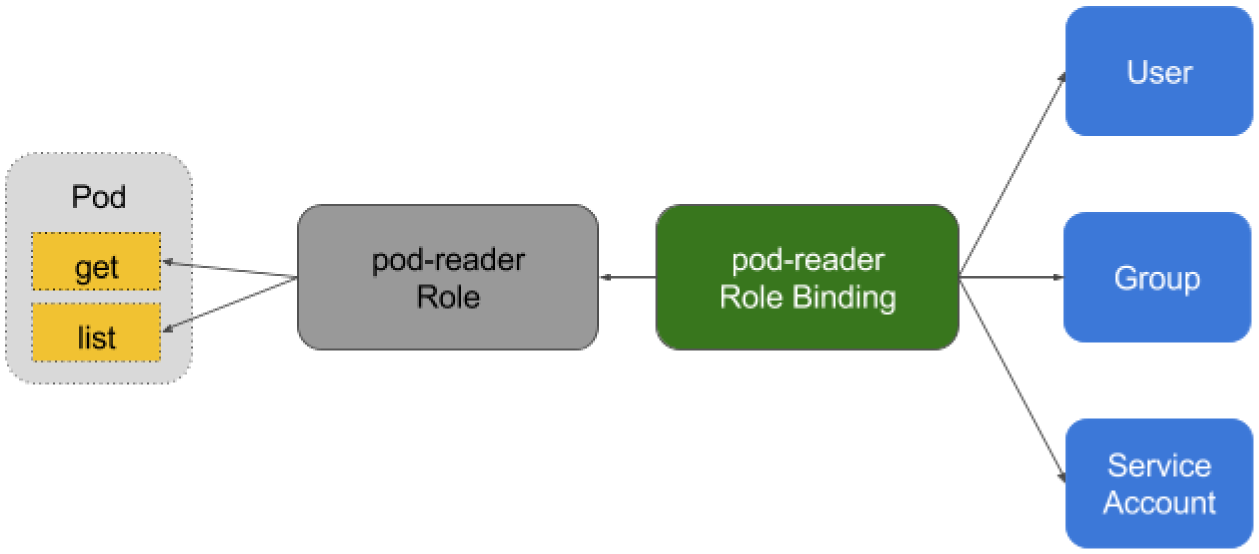
The RBAC API declares four kinds of Kubernetes object:*Role*, *ClusterRole*, *RoleBinding* and *ClusterRoleBinding*.

In order to fully grasp the idea of RBAC, we must understand that three elements are involved:

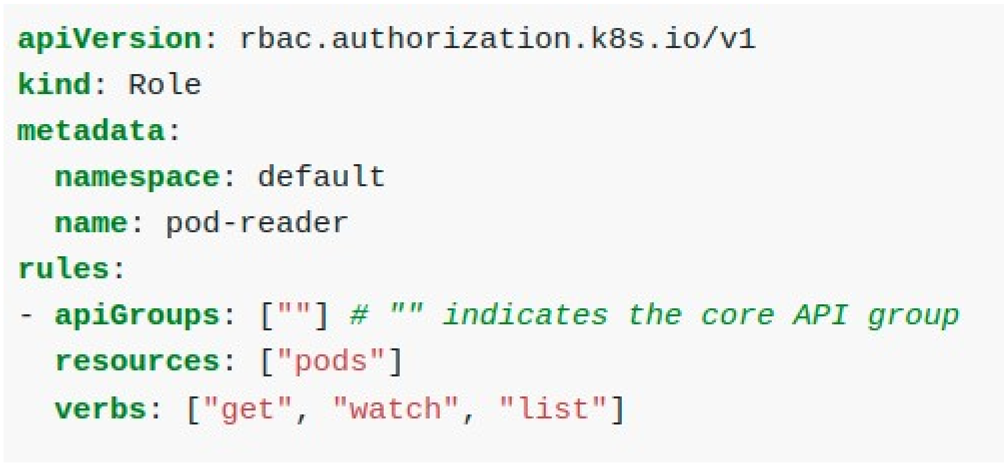
-Subjects: The set of users and processes that want to access the Kubernetes API.

-Resources: The set of Kubernetes API Objects available in the cluster. Examples include Pods, Deployments, Services, Nodes, and PersistentVolumes, among others.

-Verbs: The set of operations that can be executed to the resources above. Different verbs are available (examples: get, watch, create, delete, etc.), but ultimately all of them are Create, Read, Update or Delete (CRUD) operations.



An RBAC Role or ClusterRole contains rules that represent a set of permissions. Permissions are purely additive (there are no "deny" rules).

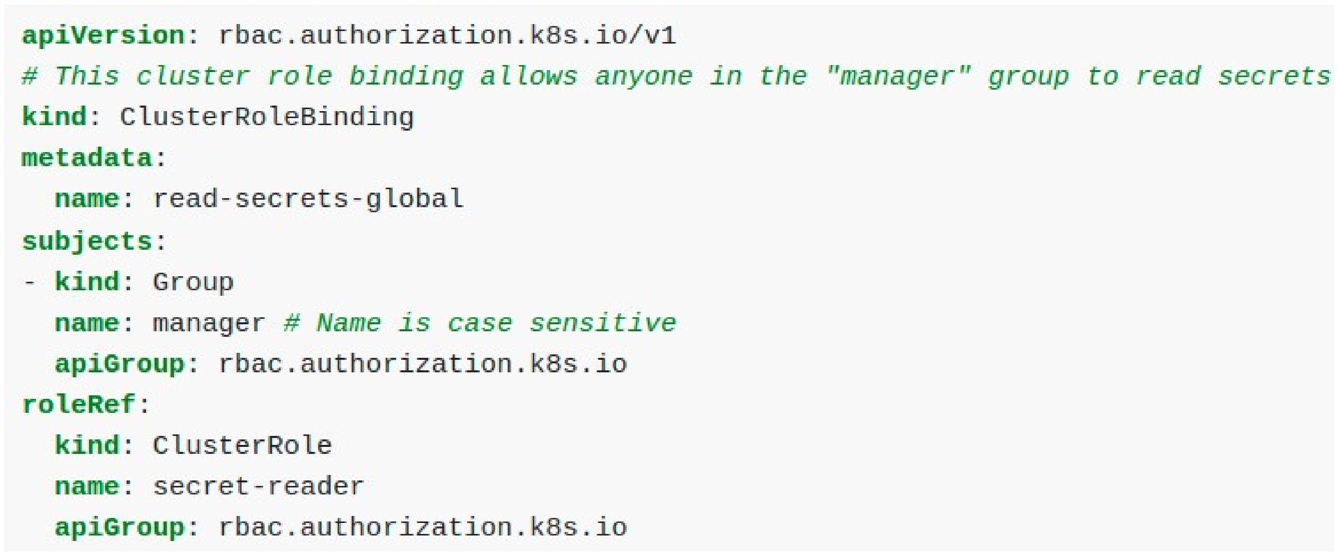


```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

A Role is a collection of permissions. For example, a role could be defined to include read permission on pods and list permission for pods. A ClusterRole is just like a Role, but can be used anywhere in the cluster.

A role binding grants the permissions defined in a role to a user or set of users. It holds a list of *subjects* (users, groups, or service accounts), and a reference to the role being granted. A RoleBinding grants permissions within a specific namespace whereas a ClusterRoleBinding grants that access cluster-wide.

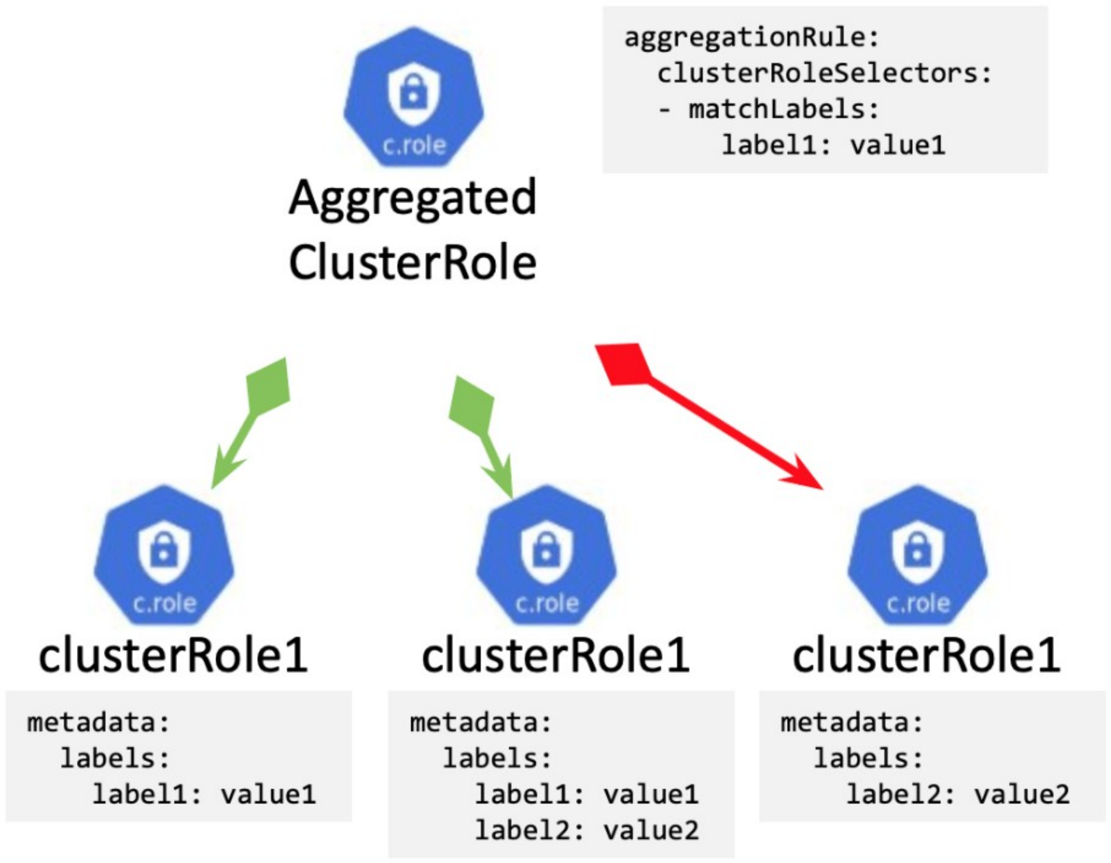
To grant permissions across a whole cluster, you can use a ClusterRoleBinding. The following ClusterRoleBinding allows any user in the group "manager" to read secrets in any namespace.



```yaml
apiVersion: rbac.authorization.k8s.io/v1
# This cluster role binding allows anyone in the "manager" group to read secrets
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

Cluster roles, unlike regular Roles, can aggregate other Cluster Roles.

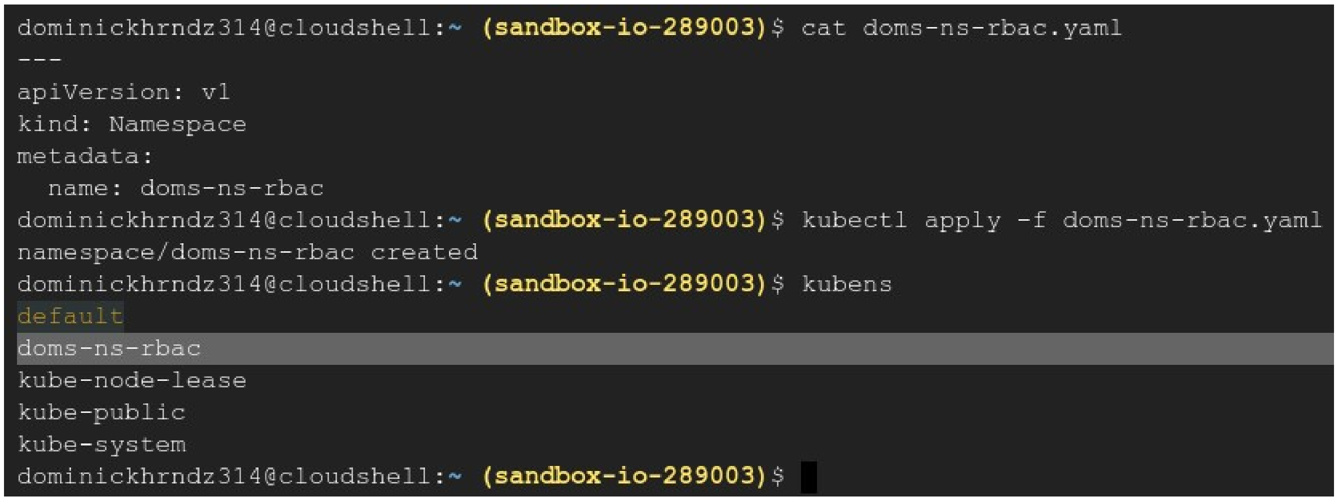
You must specify an aggregation rule for your cluster role which is essentially a set of label matching rules. Kubernetes will then find all cluster roles that match those label matching rules and aggregate them into this Cluster Role dynamically. You can add and remove matching cluster roles and the aggregated cluster role will change the set of permissions accordingly — very useful for predefined cluster roles.



Hands On – Test RBAC by creating a Service Account

I. Create a namespace called yourname-ns-rbac:

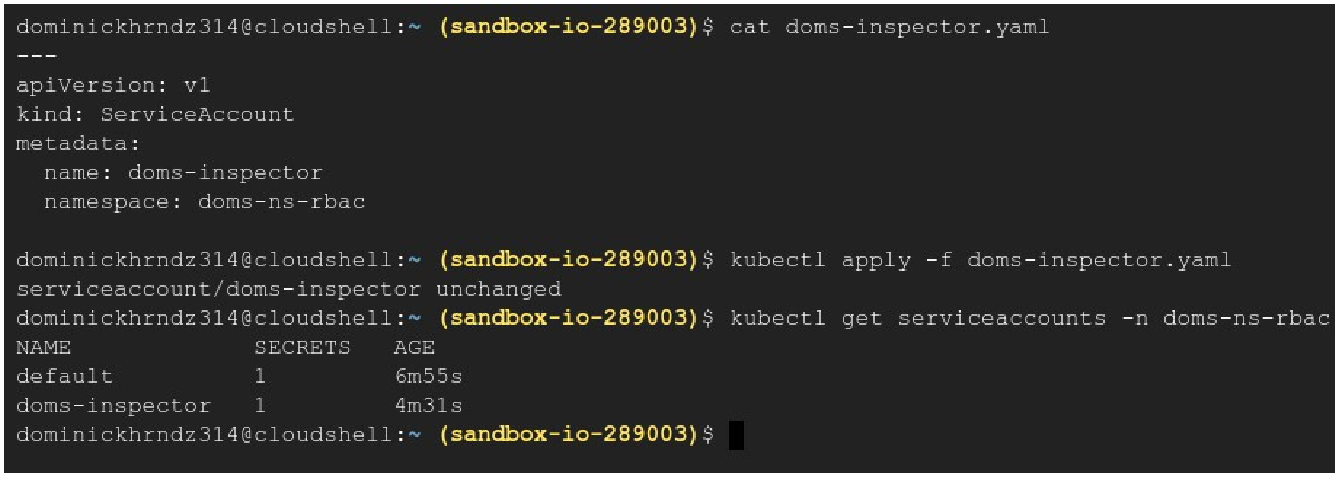
Start by creating a new namespace specifically for RBAC. Apply the the config and then run the kubens commands to ensure your new namespace has been created.



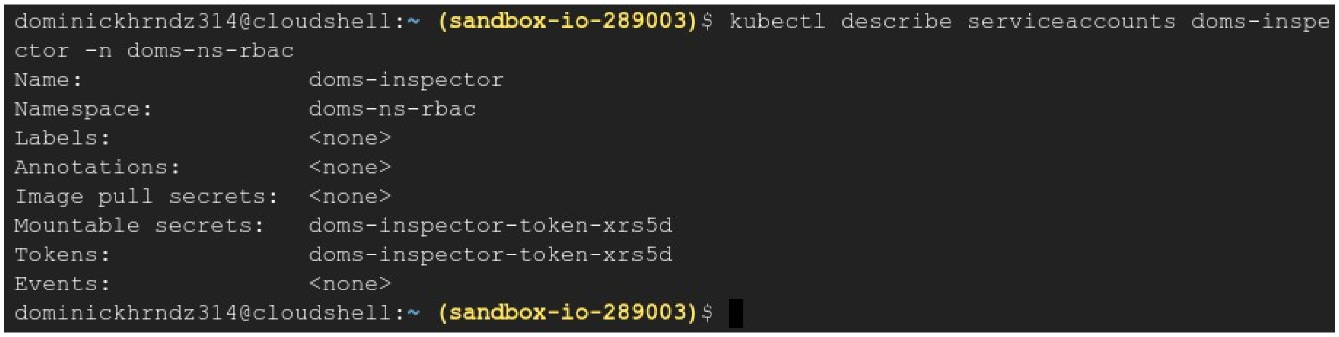
```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-ns-rbac.yaml
---
apiVersion: v1
kind: Namespace
metadata:
  name: doms-ns-rbac
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-ns-rbac.yaml
namespace/doms-ns-rbac created
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubens
default
doms-ns-rbac
kube-node-lease
kube-public
kube-system
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

II. Create a service account called yourname-inspector for the rbac namespace:

Now, let’s create our service account for your new namespace. Apply the config then verify by running the kubectl get serviceaccounts n yourname-ns-rbac command. Describe the service for more details.



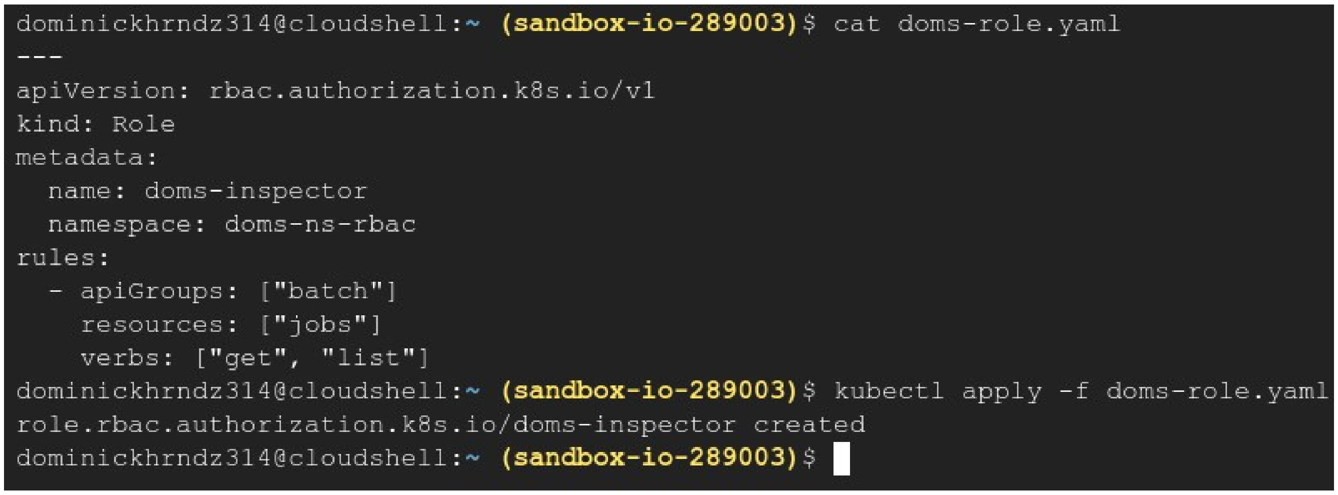
```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-inspector.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: doms-inspector
  namespace: doms-ns-rbac

dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-inspector.yaml
serviceaccount/doms-inspector unchanged
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get serviceaccounts -n doms-ns-rbac
NAME             SECRETS   AGE
default          1         6m55s
doms-inspector   1         4m31s
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```



```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl describe serviceaccounts doms-inspe
ctor -n doms-ns-rbac
Name:                doms-inspector
Namespace:           doms-ns-rbac
Labels:              <none>
Annotations:         <none>
Image pull secrets:  <none>
Mountable secrets:   doms-inspector-token-xrs5d
Tokens:              doms-inspector-token-xrs5d
Events:              <none>
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

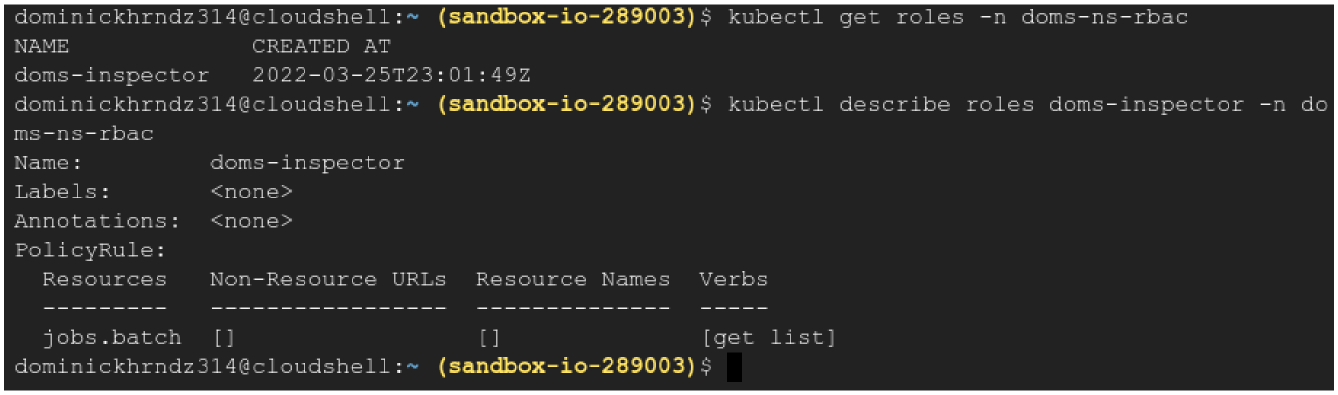
III. Create a role that has rules to 'get' and 'list' job objects:

Next, create the role which will permit you ‘get’ and ‘list’ ‘jobs’ in the new namespace. Notice we **DO NOT** allow this for deployments in our config. Apply and verify.



```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-role.yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: doms-inspector
  namespace: doms-ns-rbac
rules:
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["get", "list"]
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-role.yaml
role.rbac.authorization.k8s.io/doms-inspector created
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

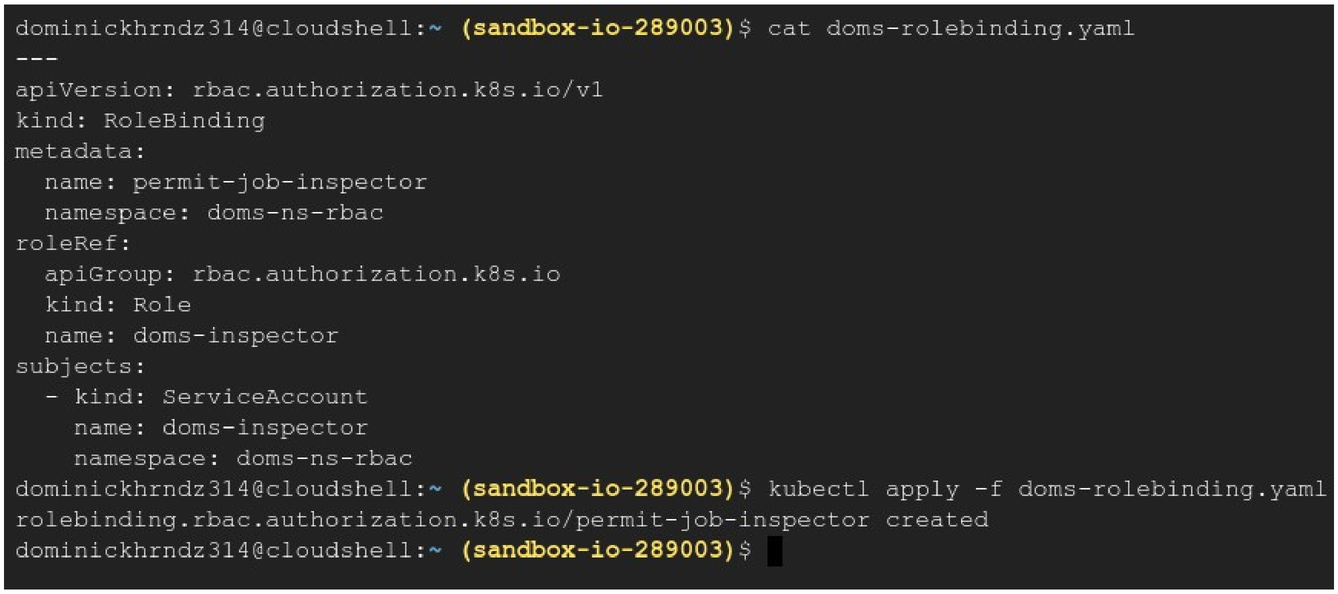
```
kubectl describe roles yourname-inspector -n yourname-ns-rbac
```



```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get roles -n doms-ns-rbac
NAME             CREATED AT
doms-inspector   2022-03-25T23:01:49Z
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl describe roles doms-inspector -n do
ms-ns-rbac
Name:         doms-inspector
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources   Non-Resource URLs  Resource Names  Verbs
  ---------   -----------------  --------------  -----
  jobs.batch  []                 []              [get list]
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

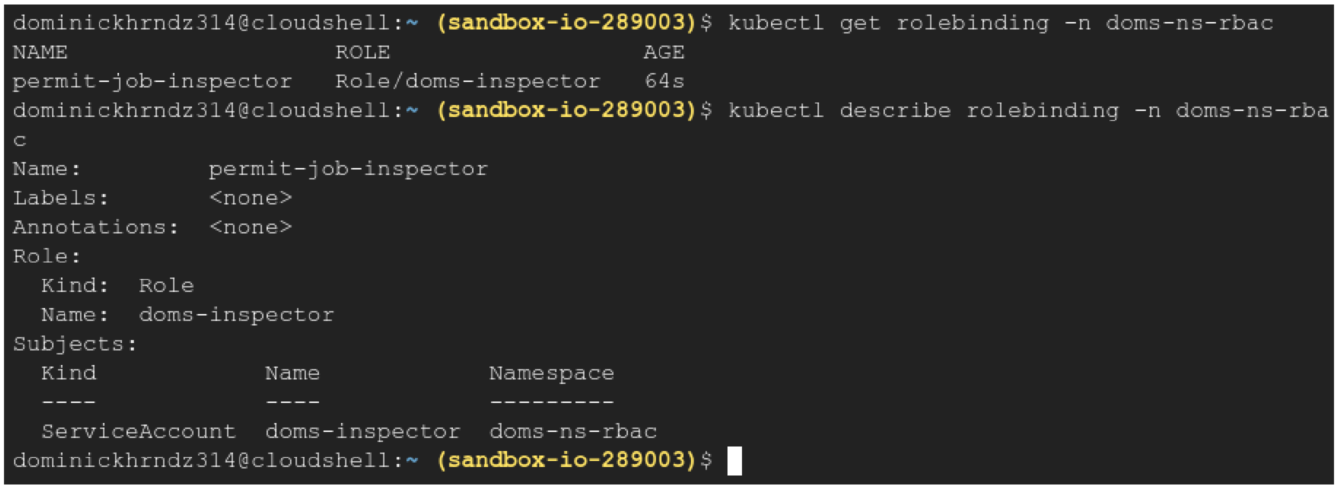
IV. Create a RoleBinding that binds the service account 'yourname-inspector' to the role created in step 3:

Now, let’s bind the role to the service account we just created. In order to do this we need to create a configuration that will include the service account name that we created along with the new namespace. Be sure that your config includes these two fields or the test will fail.



```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-rolebinding.yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: permit-job-inspector
  namespace: doms-ns-rbac
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: doms-inspector
subjects:
  - kind: ServiceAccount
    name: doms-inspector
    namespace: doms-ns-rbac
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-rolebinding.yaml
rolebinding.rbac.authorization.k8s.io/permit-job-inspector created
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

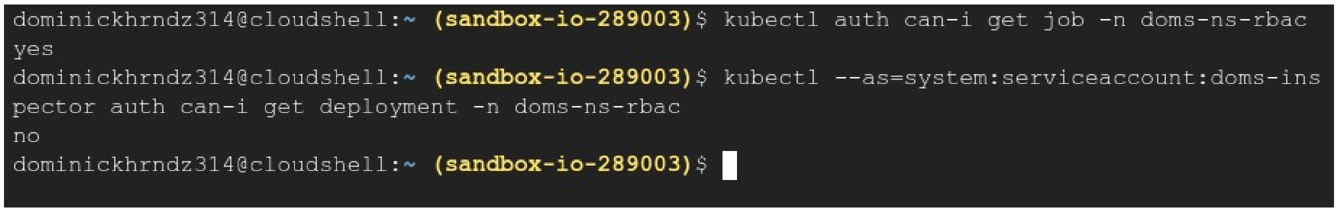
`kubectl describe rolebinding -n yourname-ns-rbac`



```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get rolebinding -n doms-ns-rbac
NAME                   ROLE                  AGE
permit-job-inspector   Role/doms-inspector   64s
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl describe rolebinding -n doms-ns-rba
c
Name:         permit-job-inspector
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  doms-inspector
Subjects:
  Kind            Name            Namespace
  ----            ----            ---------
  ServiceAccount  doms-inspector  doms-ns-rbac
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

V. Prove the job-inspector service account can "get" job objects but not deployment objects:

Finally, we will test that we can only 'get' 'jobs' within our new namespace, but not 'deployments' as specified in our roles.yaml file.



```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl auth can-i get job -n doms-ns-rbac
yes
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl --as=system:serviceaccount:doms-ins
pector auth can-i get deployment -n doms-ns-rbac
no
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

If you received the 'yes' for 'jobs' and 'no' for 'deployments' you have finished this training.