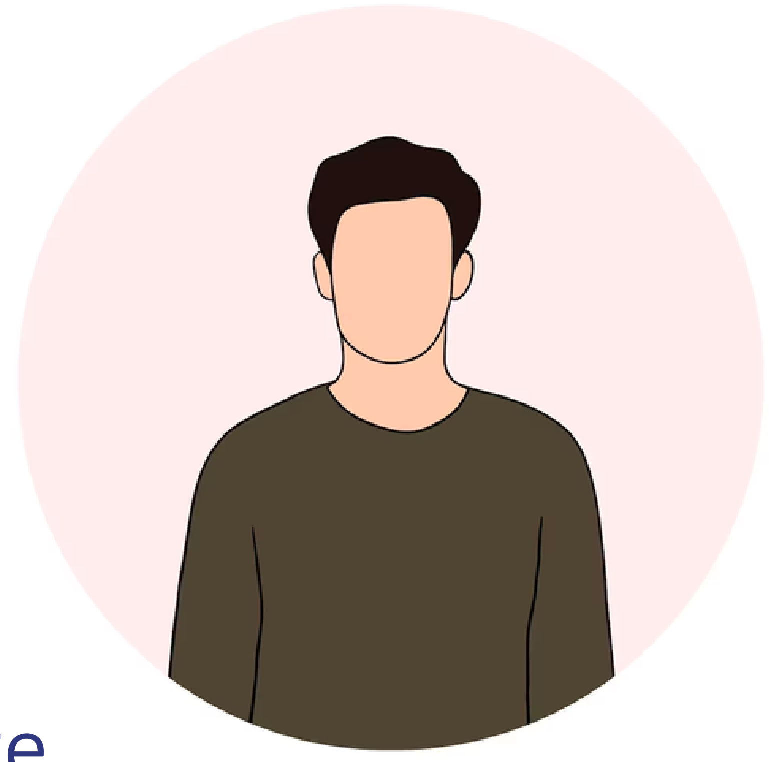
Lets do some hands on:

Lets create a user, But Kubernetes API will not allow us to create user directly. so there are multiple ways to create user

- client certificates
- bearer tokens
- authenticating proxy
- HTTP basic auth.



I will choose client certificate to create a user which is very easy to create.
This certificates are used to create users. When a user perform any command like **kubectl get po** then K8's API will authenticate and authorize the request.

authentication: permission to login
authorization: permission to work on resources

Steps to create certificate.

- lets create a folder: **mkdir mustafa-user && cd mustafa-user**
- Generate a key using openssl : **openssl genrsa -out mustafa.key 2048**
- Generate a Client Sign Request (CSR) : **openssl req -new -key mustafa.key -out mustafa.csr -subj "/CN=mustafa/O=group1"**
- Generate the certificate (CRT): **openssl x509 -req -in mustafa.csr -CA ~/.minikube/ca.crt -CAkey ~/.minikube/ca.key -CAcreateserial -out mustafa.crt -days 500**

Steps to create user.

- lets create a user: **kubectl config set-credentials mustafa --client-certificate=mustafa.crt --client-key=mustafa.key**

Till now we created user and certificates, but we didn’t add that user to our cluster. Lets do it!

- Set a context entry in kubeconfig: **kubectl config set-context my-context --cluster=minikube --user=mustafa**

Context is used to switch the users in K8s cluster. Because in K8s we cant switch using users, we will add users to context and we will use context to switch b/w users.

- To see the user: **kubectl config view**
- Switch to devops user: **kubectl config use-context my-context**
- Now lets check to create some resources
  - kubectl get po
  - kubectl run pod-1 --image=nginx
  - kubectl create ns dev
- ok! thats cool, thats won’t work, just come back to minikube config and perform same commands, that will work. (**kubectl config use-context minikube**)
- because we created user and attached that user to cluster. But we didn’t mention permissions for that user. minikube context is a admin user which will have all permissions by default
- so lets create a role and attach that role to user.

There are four components to RBAC in Kubernetes:

1. roles
2. Cluster roles
3. role bindings
4. ClusterRoles

1. Roles are the basic building blocks of RBAC. A role defines a set of permissions for a user or group. For example, you might create a role called “admin” that gives the user or group full access to all resources in the cluster.

(or)

Role is a set of permissions that define what actions (verbs) are allowed on specific resources (API groups and resources) within a particular Namespace.
Roles are Namespace-scoped, meaning they apply only to resources within that Namespace.

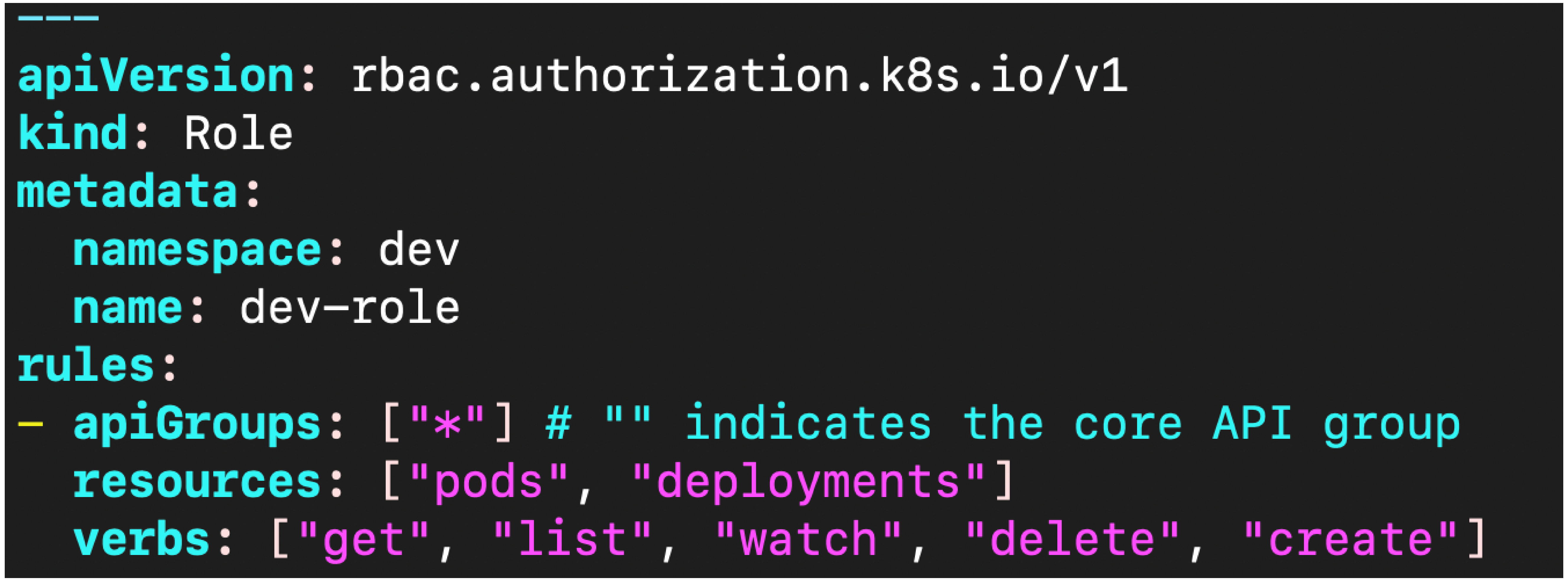
2. ClusterRoles are special roles that apply to all users in the cluster. For example, you might create a ClusterRole called "cluster-admin" that gives the user or group full access to all resources in the cluster.
These are not Namespace-specific. They define permissions for cluster-wide resources.

3. Role bindings connect users/groups to roles. For example, you might bind the "admin" role to the "admin" user. This would give the "admin" user all the permissions defined in the "admin" role.
It grants the permissions to particular namespaces only.

4. ClusterRoleBindings similar to RoleBindings, ClusterRoleBindings associate ClusterRoles with users/groups across the entire cluster.
These are not Namespace-specific. They grants the permissions for cluster-wide resources.

Lets create a namespace called dev : **kubectl create ns dev**

Now lets create a role that gives permissions to K8s resources



```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: dev
  name: dev-role
rules:
- apiGroups: ["*"] # "" indicates the core API group
  resources: ["pods", "deployments"]
  verbs: ["get", "list", "watch", "delete", "create"]
```

Imperative way : **kubectl create role admin --verb=get,list,watch,create,update,delete --resource=pods,deployments -n dev**

Check it : **kubectl get role -n dev**

Now attach this role to user (Role binding):

```yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-rolebinding
  namespace: dev
subjects:
- kind: User
  name: mustafa # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role #this must be Role or ClusterRole
  name: dev-role # must match the name of the Role
  apiGroup: rbac.authorization.k8s.io
```

Imperative way : **kubectl create rolebinding role-binding-name --role=role-name --user=user-name**

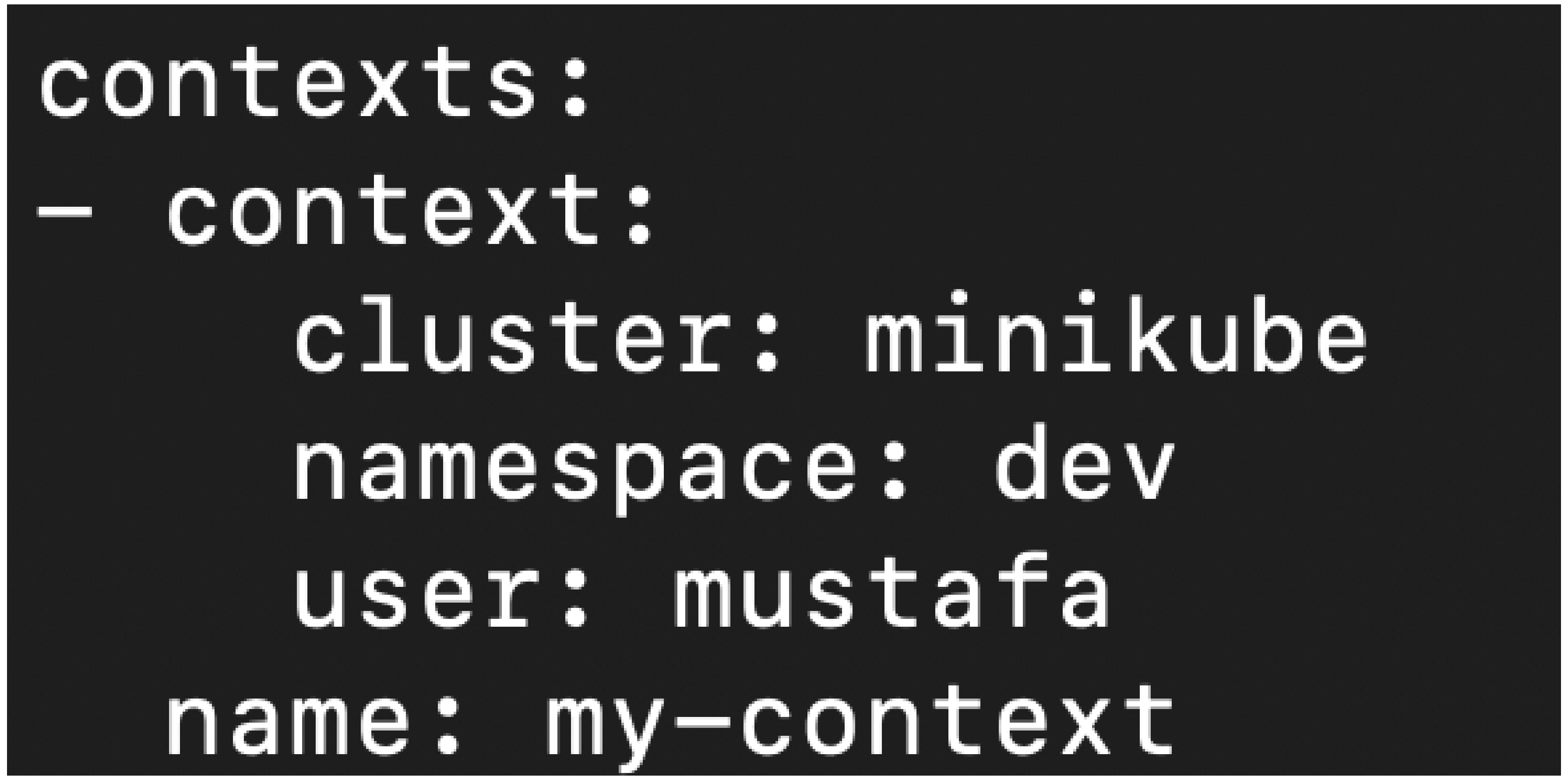
Check it : **kubectl get rolebinding -n dev**

we can also check with the permissions : kubectl can-i delete pod --as mustafa

So far we added role and attached that role to user (role binding), Now when we go to dev namespace and login as devops user, we will able to work with pods and deployments only.

Switch to dev ns: **kubectl config set-context --current --namespace=dev**
Login as devops user using context : **kubectl config use-context my-context**

make sure to check check the configs before going to test the role (**kubectl config view --minify)**



```
contexts:
- context:
    cluster: minikube
    namespace: dev
    user: mustafa
  name: my-context
```

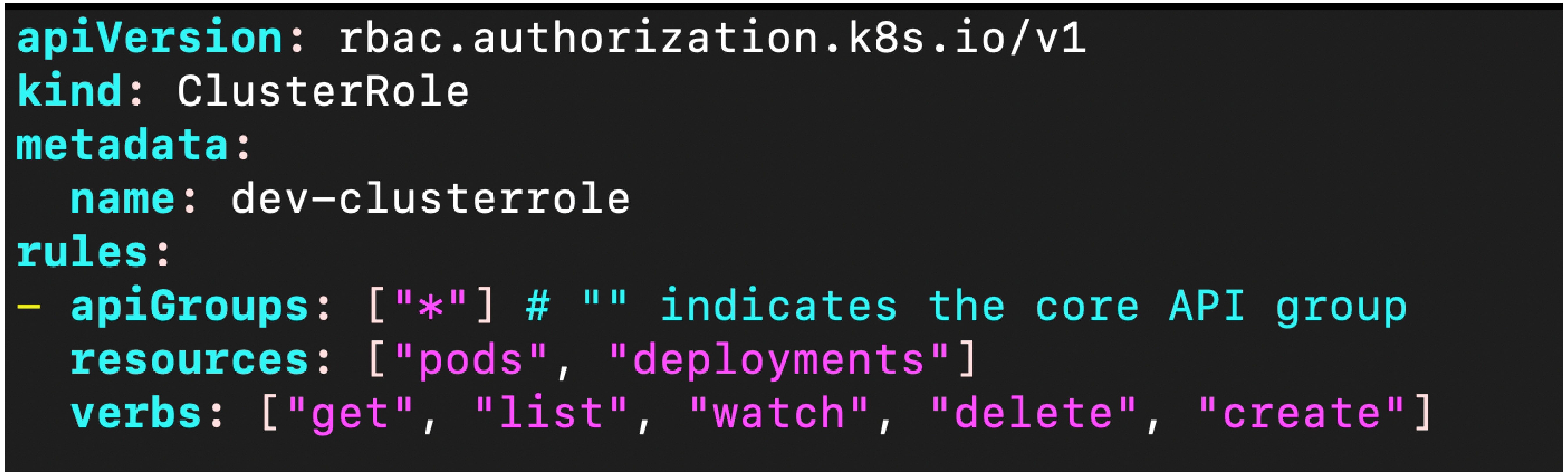
Now we can able to work with pods and deployments.

Note: for only one namespace (dev).

Mustafa user cant get any resources from any other namespaces because we used rolebinding. If mustafa want to access the resources from all namespaces we can use clusterrolebinding

First lets change the context to minikube and go to default namespace

Lets create cluster role:



```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: dev-clusterrole
rules:
- apiGroups: ["*"] # "" indicates the core API group
  resources: ["pods", "deployments"]
  verbs: ["get", "list", "watch", "delete", "create"]
```

Imperative way : **kubectl create clusterrole my-cluster-role --verb=get,list,create,update,delete --resource=pods,services**

Check it : **kubectl get clusterrole**

Lets create cluster role binding:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dev-clusterrolebinding
subjects:
- kind: User
  name: mustafa # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole #this must be Role or ClusterRole
  name: dev-clusterrole # must match the name of the Role
  apiGroup: rbac.authorization.k8s.io
```

Imperative way : **kubectl create clusterrolebinding cluster-binding-name --clusterrole=role-name --user=user-name**
Check it : **kubectl get clusterrolebinding**

After creating the clusterrole and attached that role to cluster role binding, its time to check weather mustafa user can have access to work with resources in all namespaces or not.