KubeCon | CloudNativeCon

Europe 2019

Smarter Kubernetes Access Control: **A Simpler Approach to Auth**

Rob Scott | ReactiveOps | @robertjscott

Outline

- Challenges of authorization
- Quick recap of RBAC basics
- Understanding who has access to what in your cluster
- Managing RBAC simply and effectively

Authorization is

Challenging

Authorization systems often feel either **too simple** or **too complex**

Authorization is only really noticeable when it's **getting in the way**

Even the best intentions can still end in failure

- Organizations start with highly granular policies, doing everything by the book
- At some point, something doesn’t work, and a “temporary” solution emerges
- Temporary solutions are rarely temporary

Kubernetes has unique challenges

- Users and Groups are not actually managed by Kubernetes
- Kubernetes RBAC configuration quickly becomes difficult to manage at scale

A Quick Recap of RBAC Basics

Roles and **Cluster Roles** define specific sets of actions allowed

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: list-deployments
  namespace: dev
rules:
  - apiGroups: [ apps ]
    resources: [ deployments ]
    verbs: [ get, list ]
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: list-deployments
rules:
  - apiGroups: [ apps ]
    resources: [ deployments ]
    verbs: [ get, list ]
```

Default Roles

- **view**: read only access, excludes secrets
- **edit:** above + ability to edit most resources, excludes roles and role bindings
- **admin:** above + ability to manage roles and role bindings at a namespace level
- **cluster-admin:** everything

Role Bindings and **Cluster Role Bindings** connect accounts to roles

A Simple Example

Avery should be able to **edit** the **web** namespace and **view** the **api** namespace

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: avery
  namespace: web
subjects:
- kind: User
  name: avery@example.com
roleRef:
  kind: ClusterRole
  name: edit
  apiGroup: rbac.authorization.k8s.io
```

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: avery
  namespace: api
subjects:
- kind: User
  name: avery@example.com
roleRef:
  kind: ClusterRole
  name: view
  apiGroup: rbac.authorization.k8s.io
```

Understanding

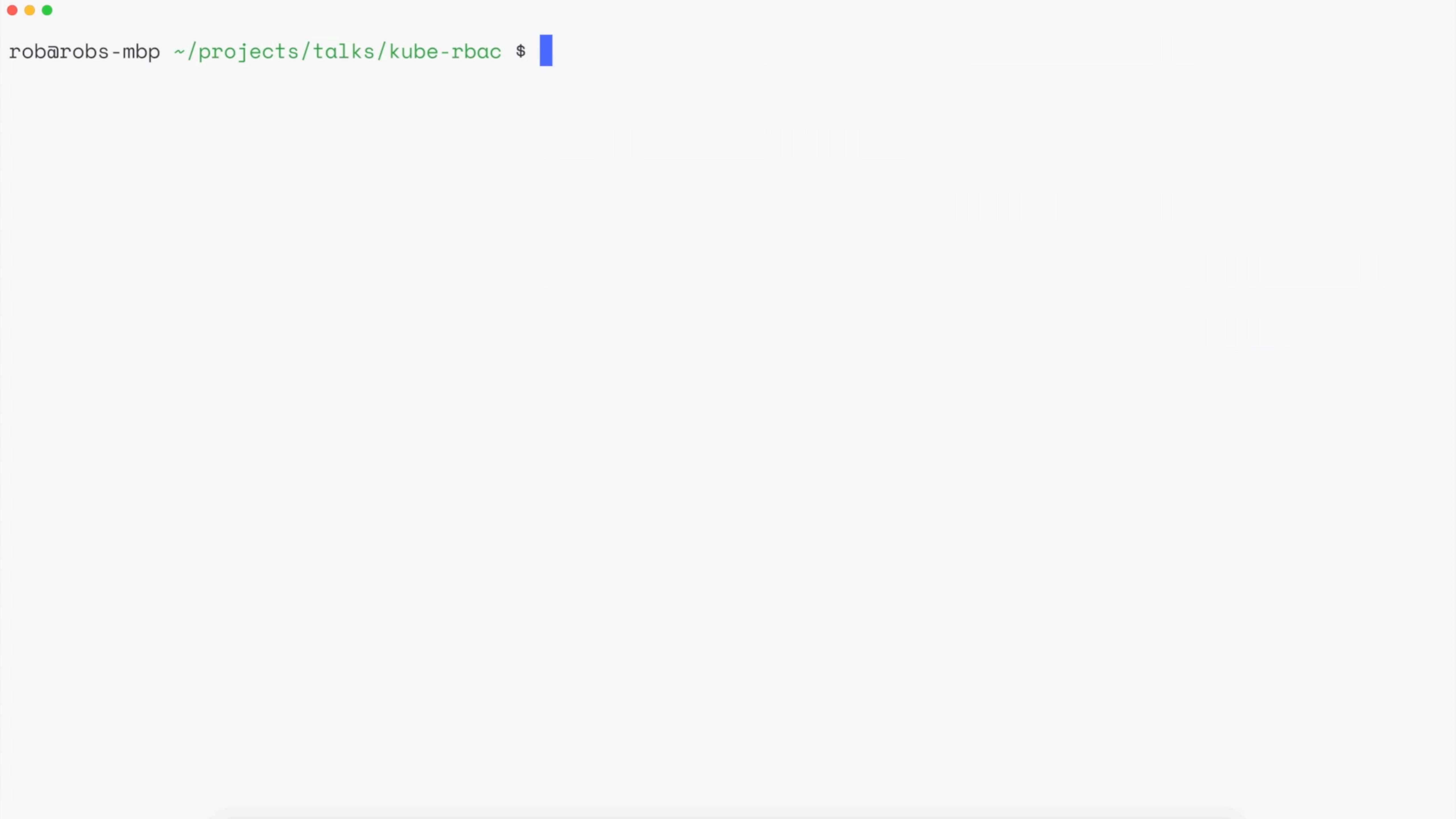
Kubernetes Authorization

Can Avery list pods? If so, why?

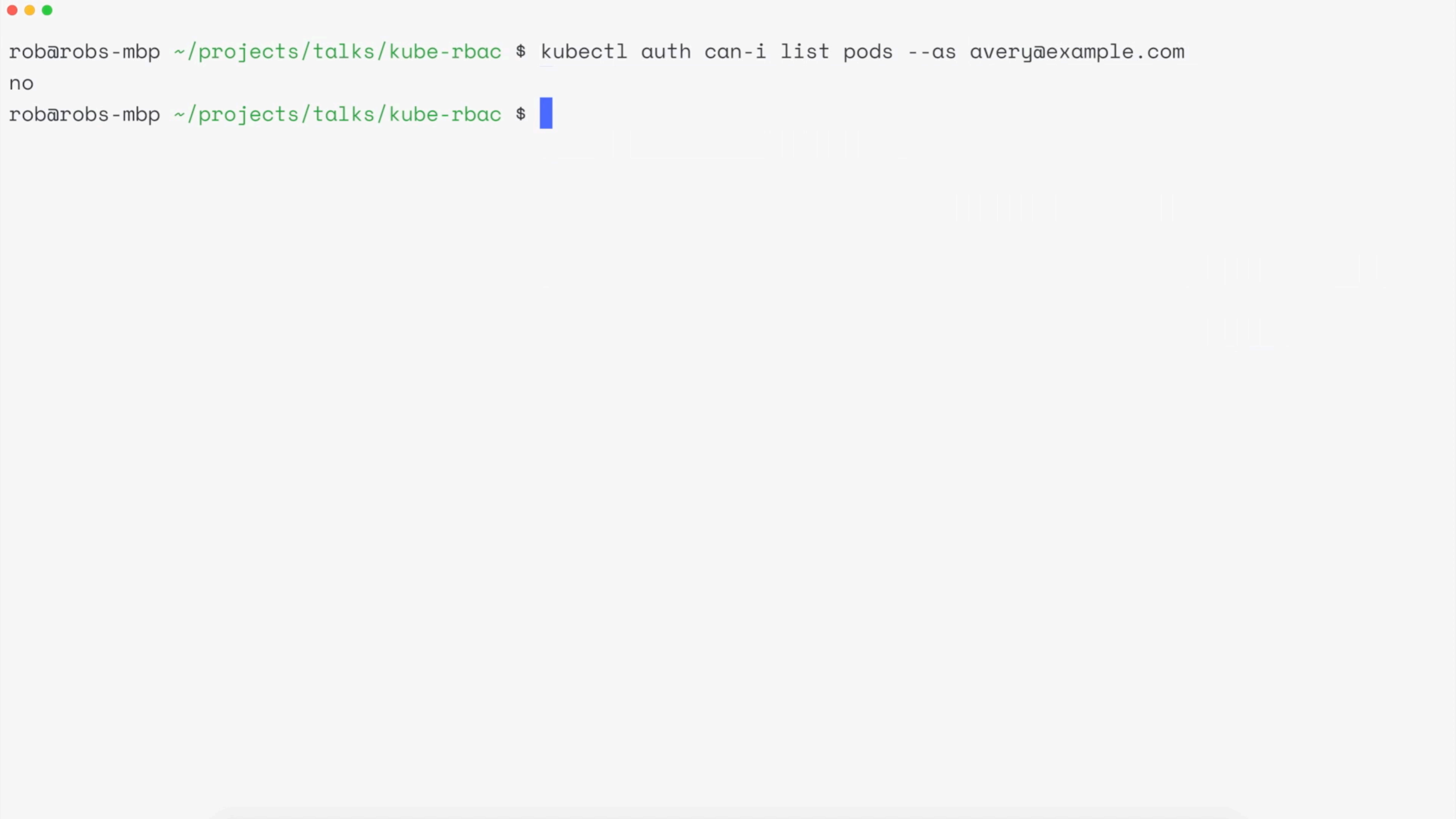
SUBJECT ACTION RESOURCE

```
kubectl auth can-i list pods --as avery
```

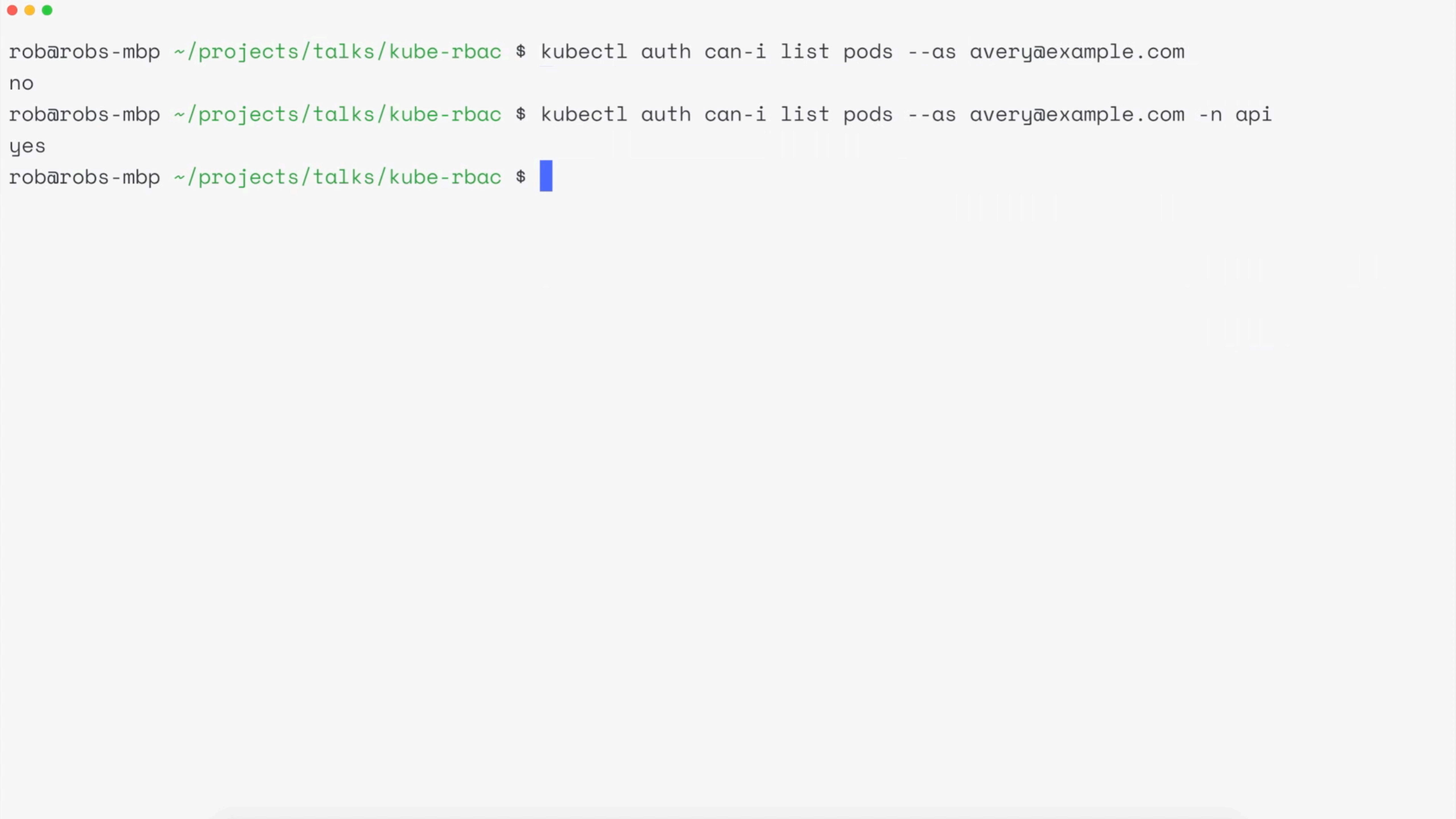
ACTION RESOURCE SUBJECT



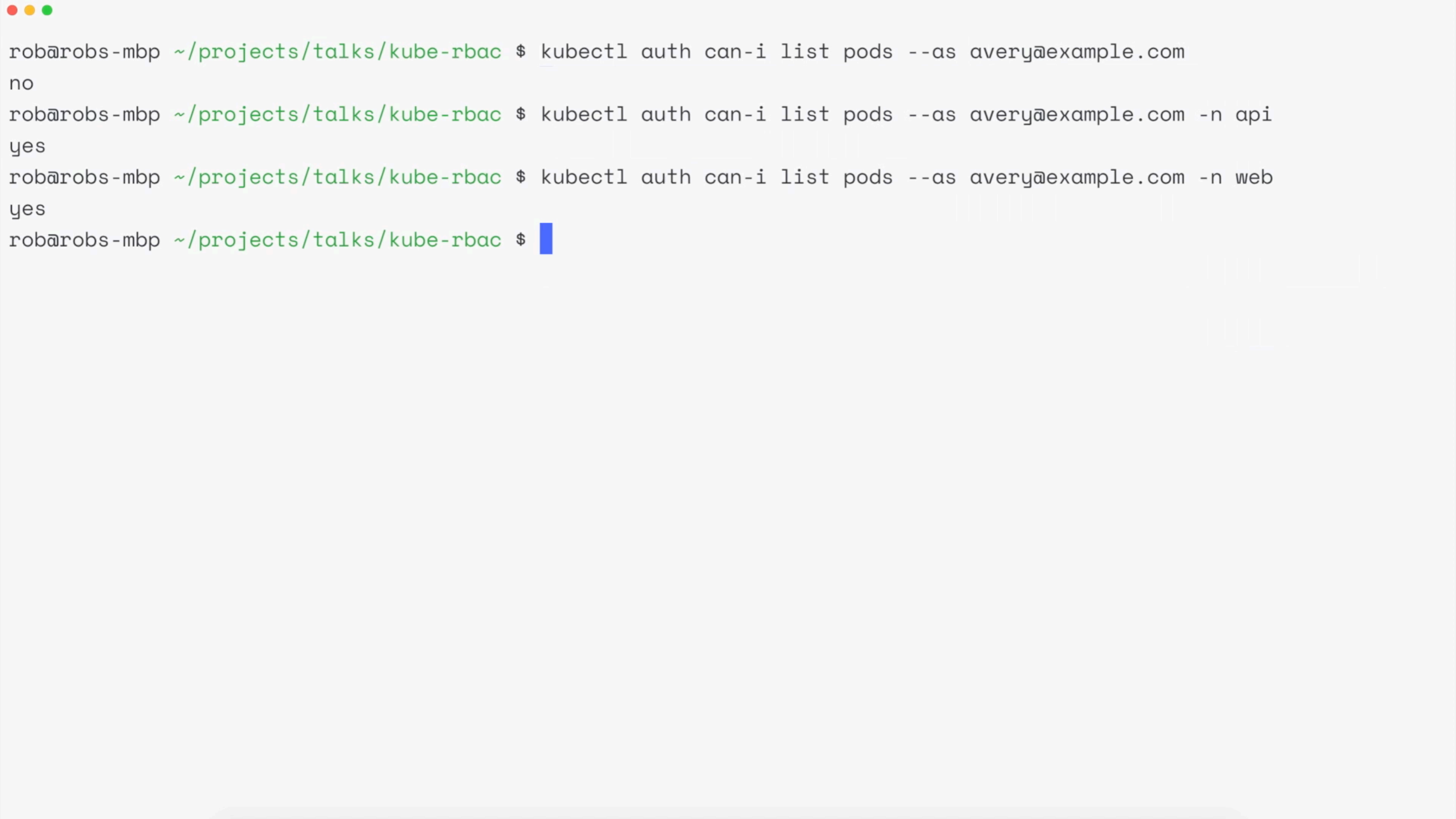
```
rob@robs-mbp ~/projects/talks/kube-rbac $ 
```



```
rob@robs-mbp ~/projects/talks/kube-rbac $ kubectl auth can-i list pods --as avery@example.com
no
rob@robs-mbp ~/projects/talks/kube-rbac $ 
```

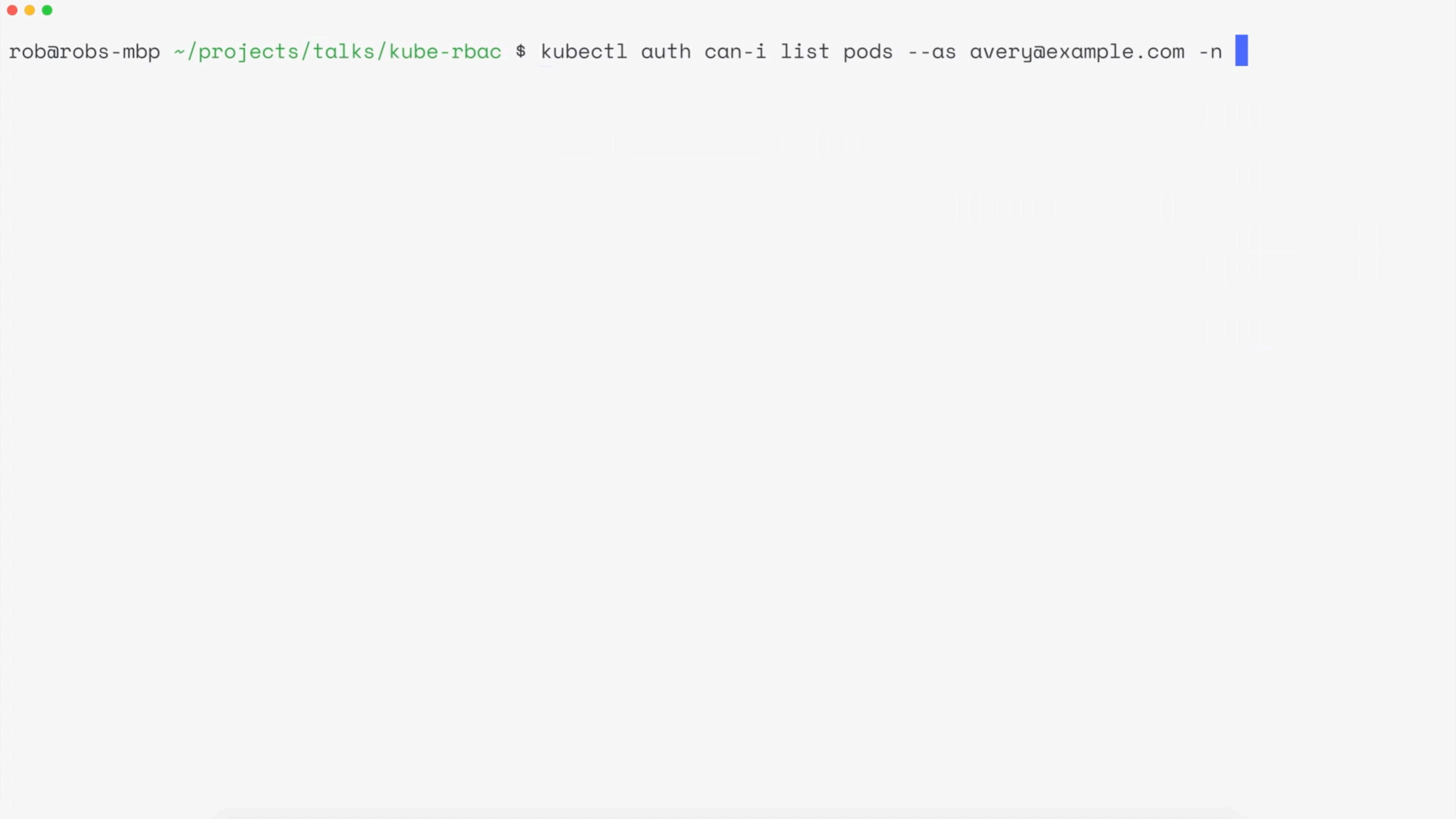


```
rob@robs-mbp ~/projects/talks/kube-rbac $ kubectl auth can-i list pods --as avery@example.com
no
rob@robs-mbp ~/projects/talks/kube-rbac $ kubectl auth can-i list pods --as avery@example.com -n api
yes
rob@robs-mbp ~/projects/talks/kube-rbac $ 
```



```
rob@robs-mbp ~/projects/talks/kube-rbac $ kubectl auth can-i list pods --as avery@example.com
no
rob@robs-mbp ~/projects/talks/kube-rbac $ kubectl auth can-i list pods --as avery@example.com -n api
yes
rob@robs-mbp ~/projects/talks/kube-rbac $ kubectl auth can-i list pods --as avery@example.com -n web
yes
rob@robs-mbp ~/projects/talks/kube-rbac $ 
```

How do you know why?



```
rob@robs-mbp ~/projects/talks/kube-rbac $ kubectl auth can-i list pods --as avery@example.com -n
```

```
{
  "kind": "SelfSubjectAccessReview",
  "apiVersion": "authorization.k8s.io/v1",
  "spec": {
    "resourceAttributes": {
      "namespace": "web",
      "verb": "list",
      "resource": "pods"
    }
  },
  "status": {
    "allowed": true,
    "reason": "RBAC: allowed by RoleBinding \"avery/web\" of
               ClusterRole \"edit\" to User \"avery@example.com\""
  }
}
```

```
RBAC: allowed by RoleBinding
"avery/web" of ClusterRole "edit"
to User "avery@example.com"
```

What can Avery do?

ACTION SUBJECT

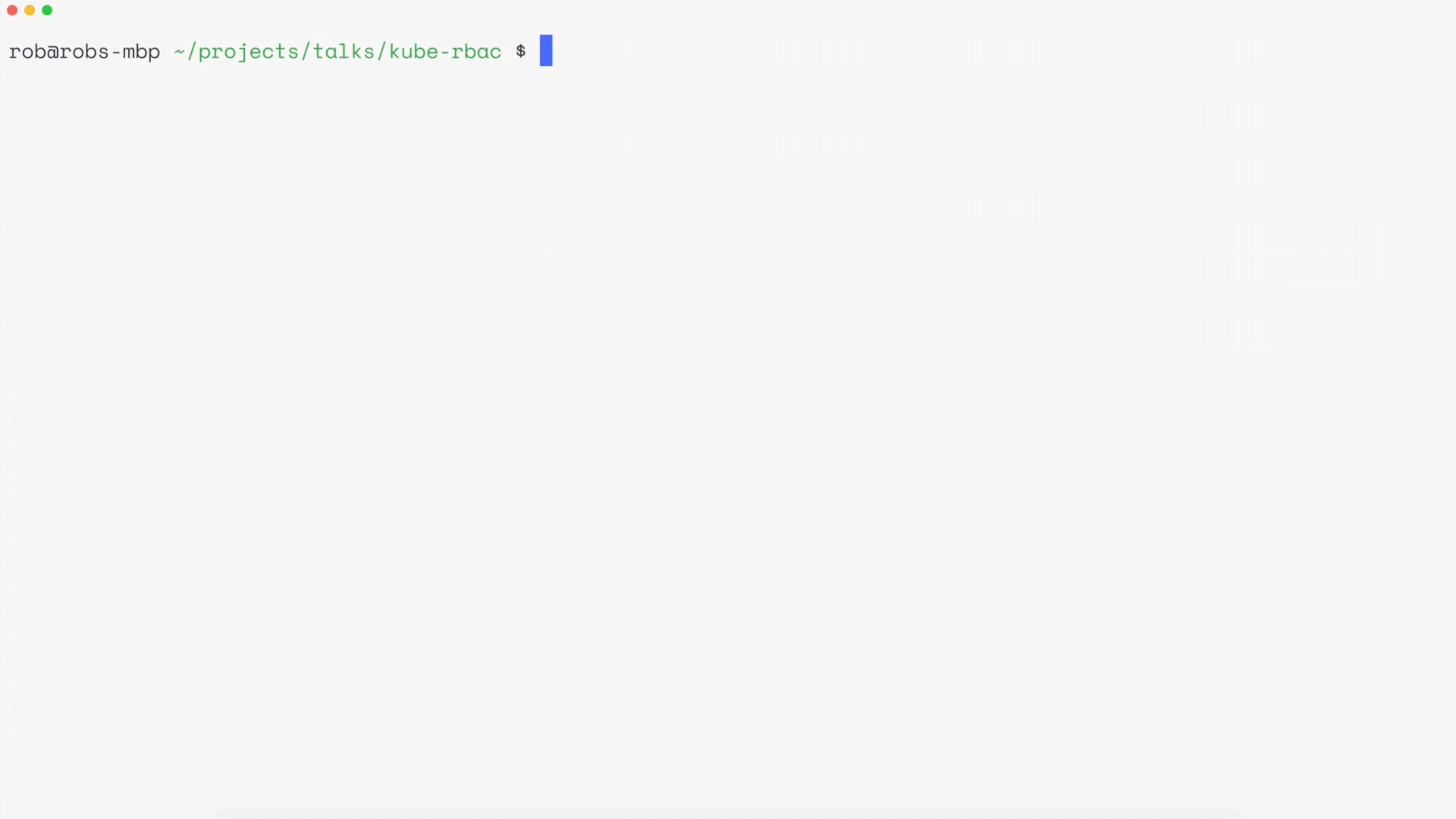
List **everything** Avery can do **cluster wide**

```
> rakkess --as avery
```

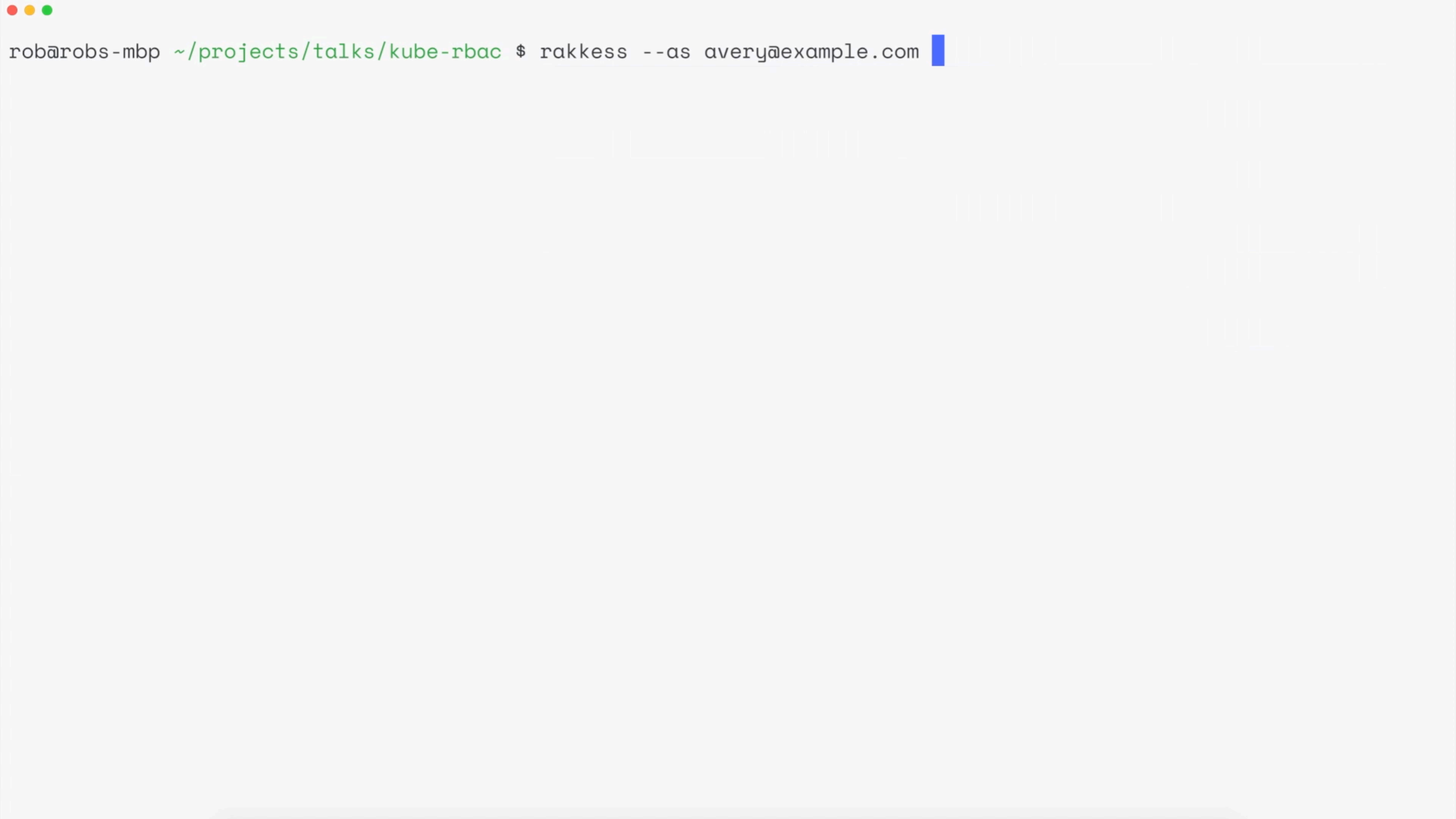
List **everything** Avery can do in **dev namespace**

```
> rakkess --as avery --namespace dev
```

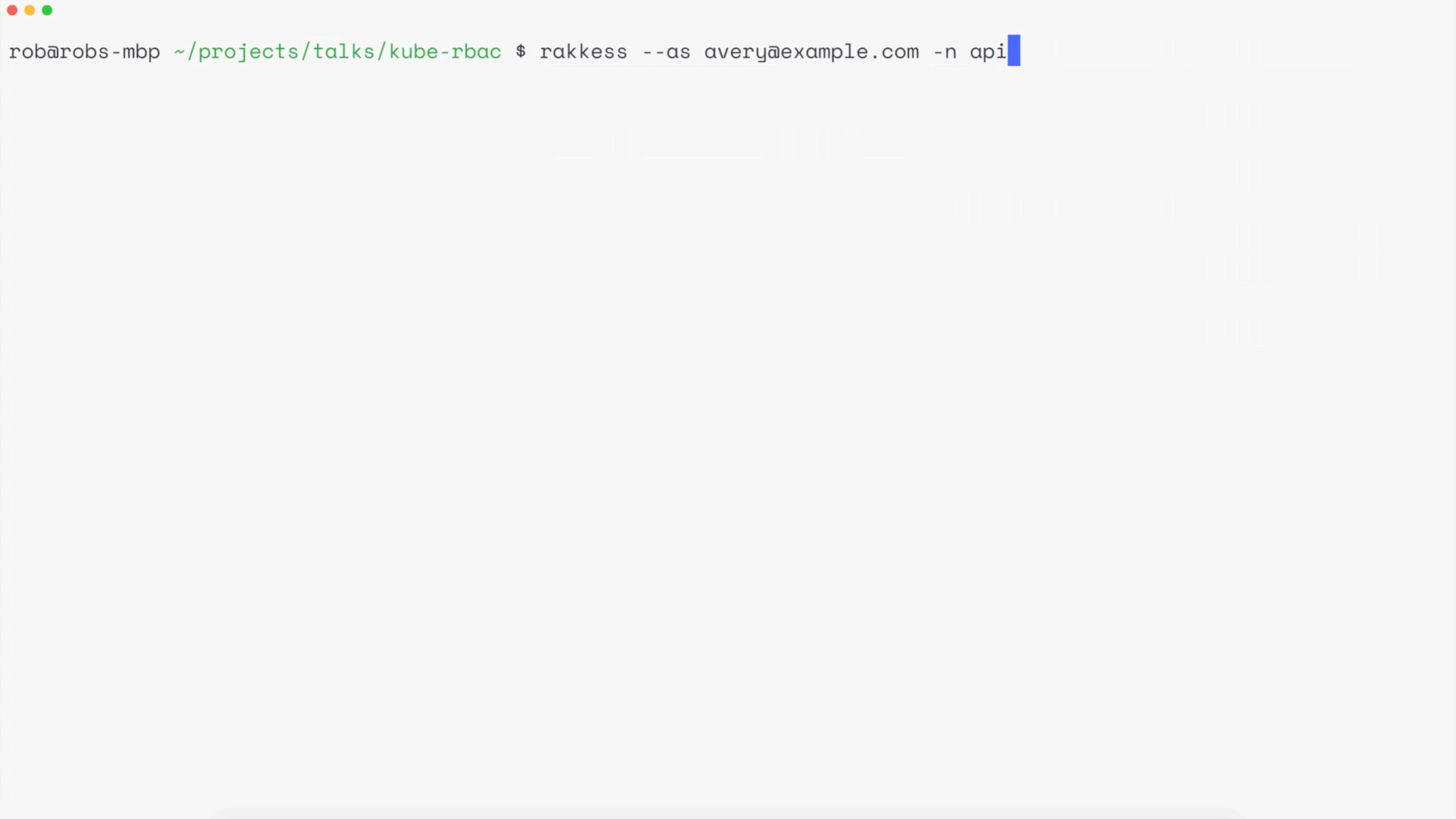
github.com/corneliusweig/**rakkess**



```
rob@robs-mbp ~/projects/talks/kube-rbac $ 
```



```
rob@robs-mbp ~/projects/talks/kube-rbac $ rakkess --as avery@example.com
```



```
rob@robs-mbp ~/projects/talks/kube-rbac $ rakkess --as avery@example.com -n api
```

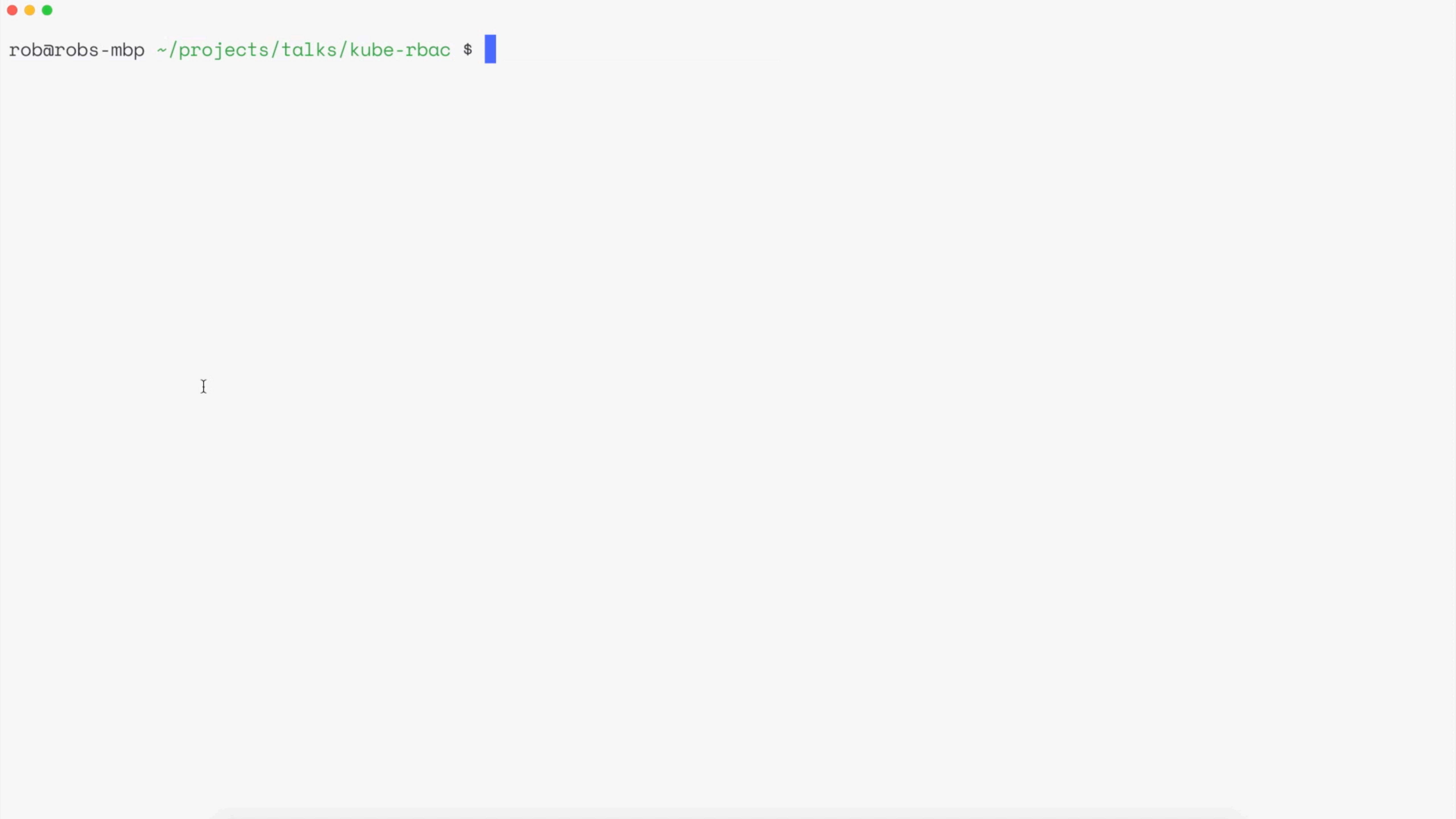
Who can list pods?

SUBJECT ACTION RESOURCE

List **everyone** who can list pods **cluster wide**

```
> kubectl-who-can list pods
```

github.com/aquasecurity/**kubectl-who-can**



```
rob@robs-mbp ~/projects/talks/kube-rbac $ 
```

Can I see a top level overview?

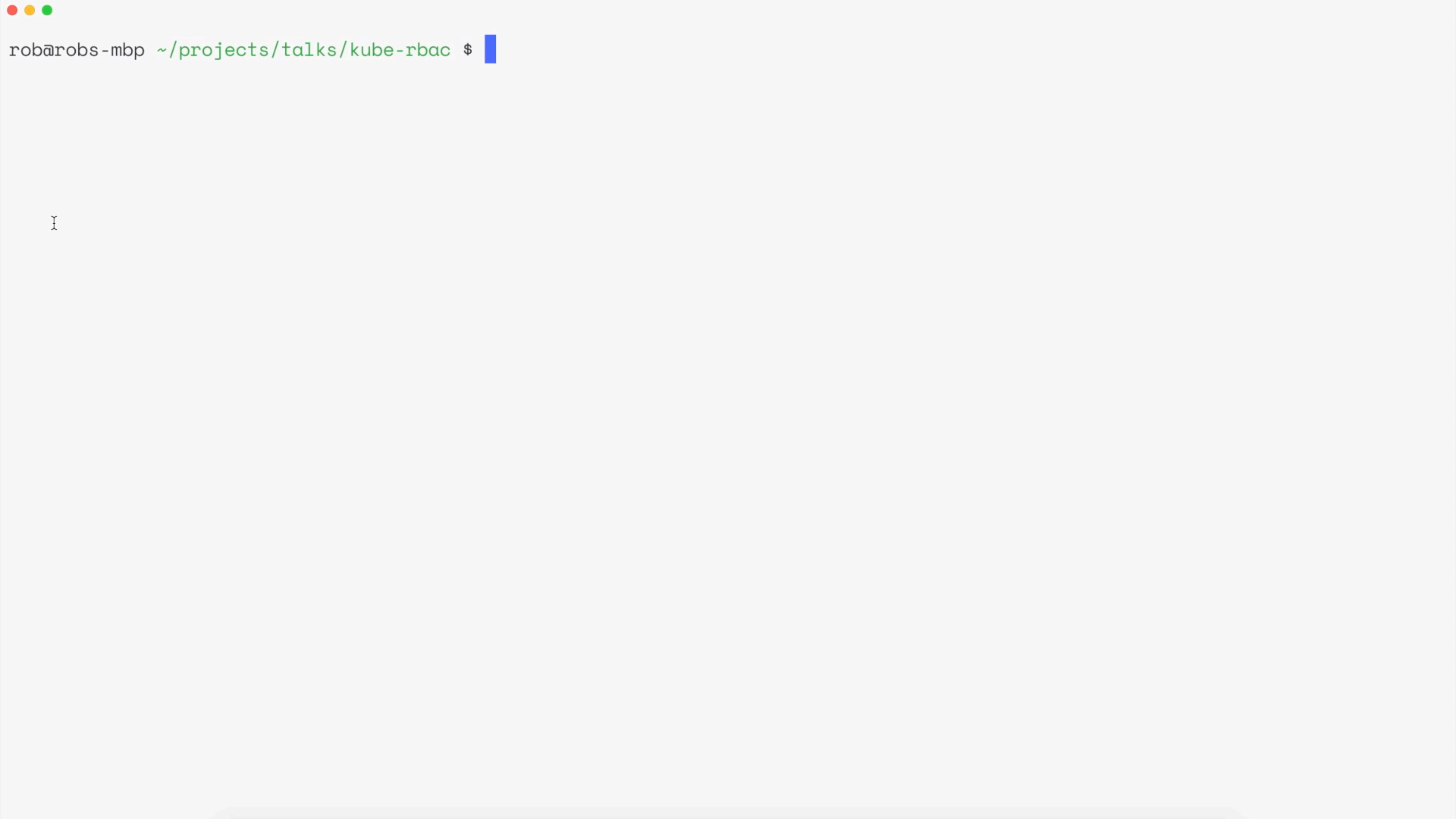
List **everyone’s** access within the cluster

```
> rbac-lookup
```

List access for **matching subjects** within the cluster

```
> rbac-lookup avery
```

github.com/reactiveops/**rbac-lookup**

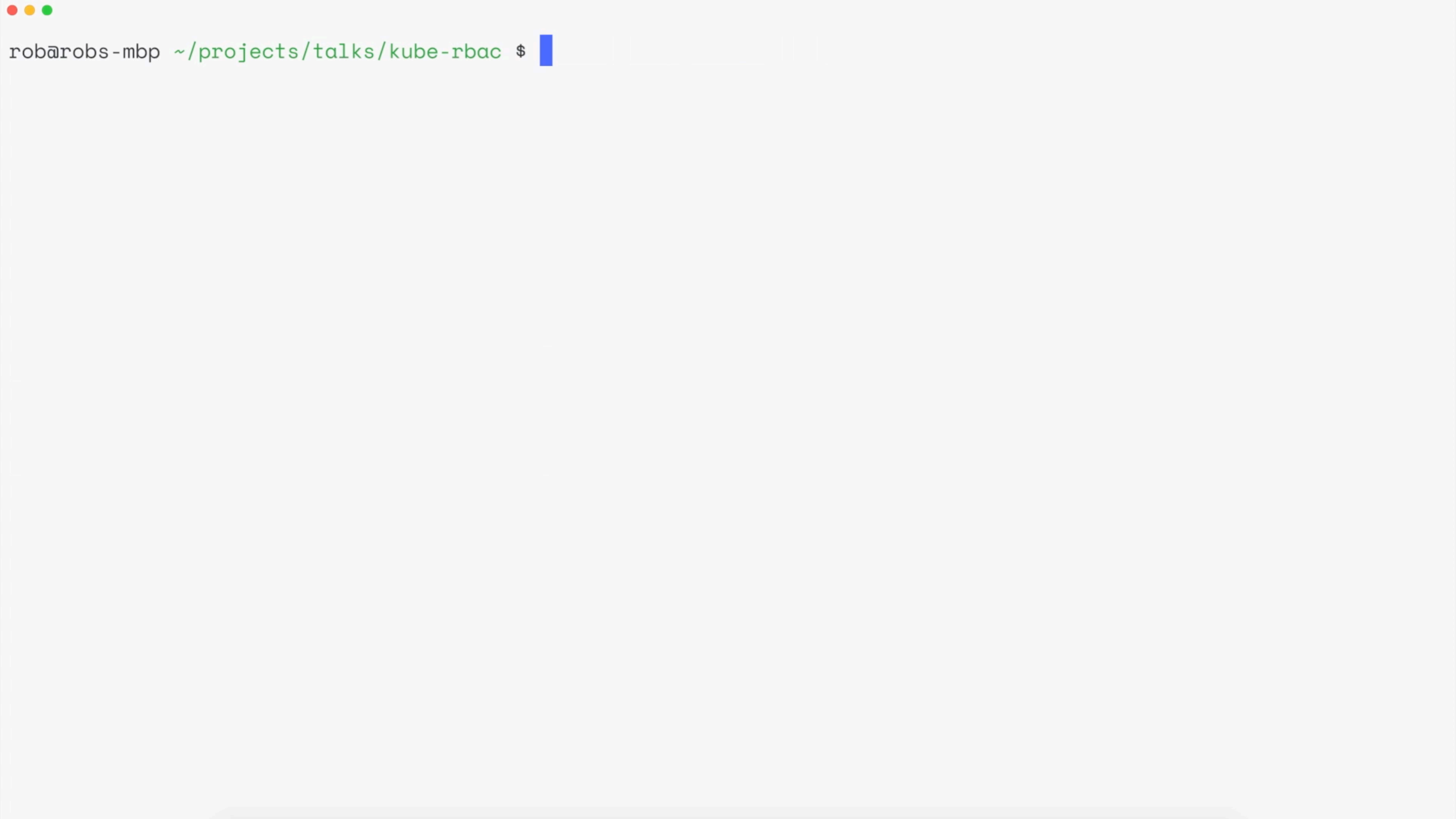


```
rob@robs-mbp ~/projects/talks/kube-rbac $ 
```

```
rob@robs-mbp ~/projects/talks/kube-rbac $
```

```
rob@robs-mbp ~/projects/talks/kube-rbac $
```

Sometimes RBAC isn’t all there is



```
rob@robs-mbp ~/projects/talks/kube-rbac $ 
```

Tools to help Understand RBAC

- **kubectl auth can-i** - see if a user can perform a specific action, and if so, why
- **rakkess** - get that same information for a specific user across all potential actions
- **kubectl-who-can** - list who can perform a specific action in a cluster
- **rbac-lookup** - get an RBAC (and GKE IAM) overview

Managing Kubernetes Authorization

Simply and Effectively

Effective RBAC

- **Principle of Least Privilege:** Don't grant any more access than user's actually need
- **Use Namespaces Effectively:** These need to be granular enough for your auth strategy
- **Have a Clear Update Process:** Ideally this should include automation with CI

Simpler RBAC

- **Centralize config:** Group your RBAC configuration together into one central place per cluster
- **Give less people access:** In many cases, engineers don’t need direct access to a Kubernetes cluster
- **Use default roles:** For user authorization, the default roles can cover most use cases

RBAC Manager

github.com/reactiveops/**rbac-manager**

RBAC Manager

- Use more concise configuration by grouping resources together
- Automate RBAC changes
- Support ephemeral namespaces and more with label selectors

Deployments simplify managing **pods**

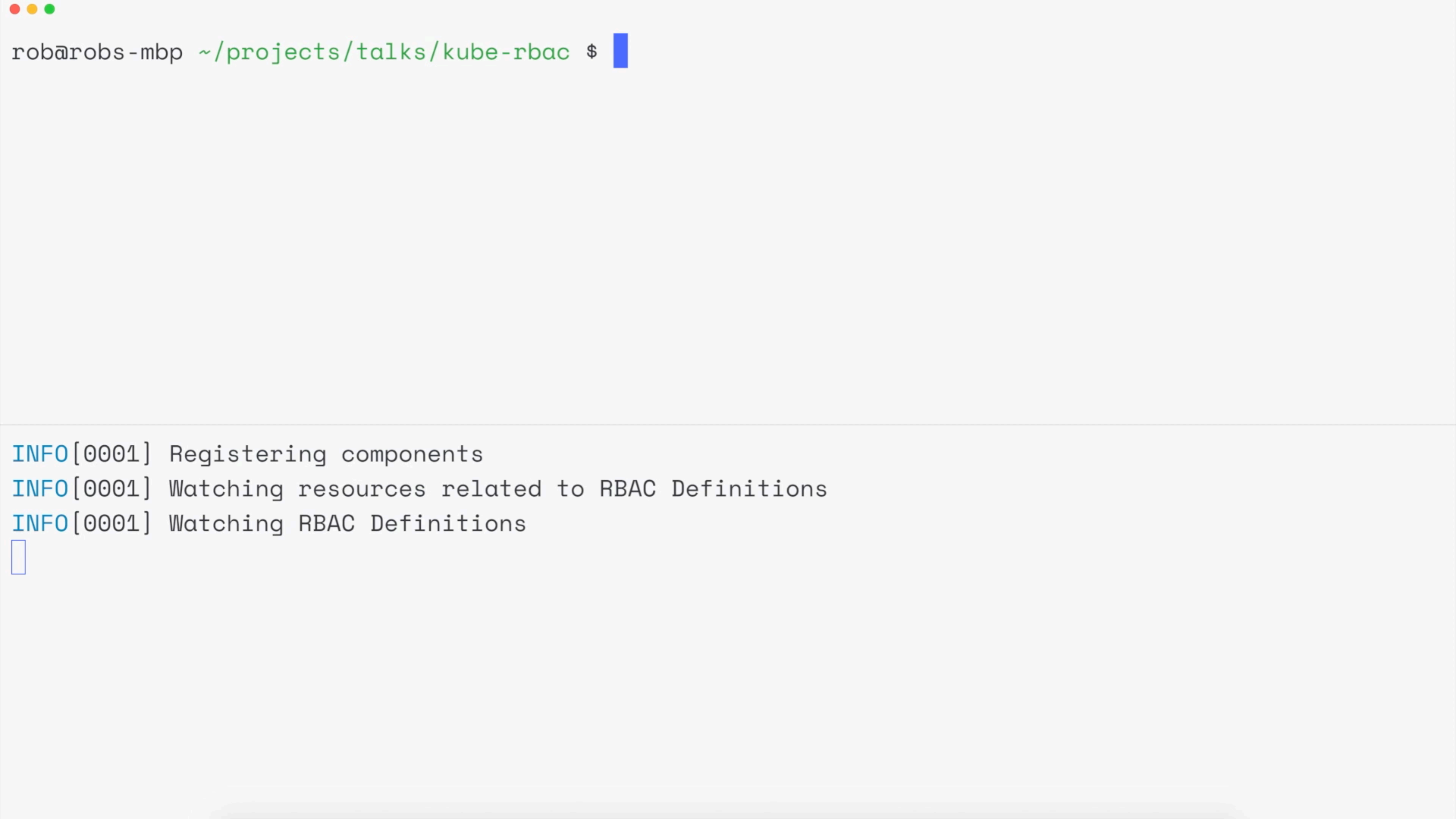
RBAC Definitions simplify managing **role bindings**

More Concise

Advantage #1

Representing our simple example from before with an **RBAC Definition**

```yaml
apiVersion: rbacmanager.reactiveops.io/v1beta1
kind: RBACDefinition
metadata:
  name: demo
rbacBindings:
  - name: avery
    subjects:
      - kind: User
        name: avery@example.com
    roleBindings:
      - namespace: api
        clusterRole: view
      - namespace: web
        clusterRole: edit
```



```
rob@robs-mbp ~/projects/talks/kube-rbac $
```

```
INFO[0001] Registering components
INFO[0001] Watching resources related to RBAC Definitions
INFO[0001] Watching RBAC Definitions
```

Path to Automation

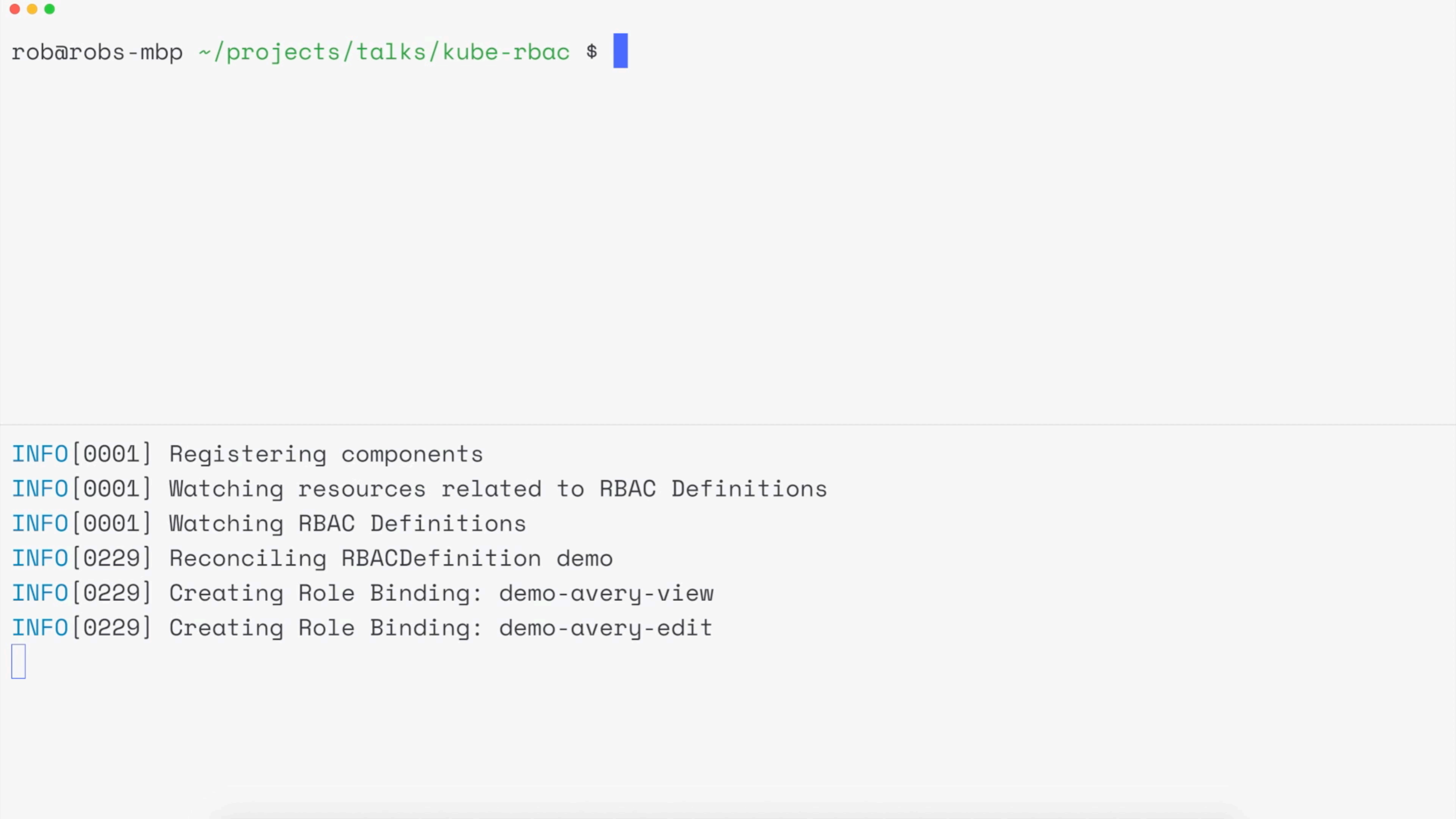
Advantage #2

RBAC Automation Requires

- Newly defined role bindings are reliably created
- Role bindings that require changes are updated or replaced, even where attributes are considered immutable (role refs)
- Role bindings that are no longer referenced are deleted

```
rbacBindings:
  - name: avery
    subjects:
      - kind: User
        name: avery@example.com
    roleBindings:
      - namespace: api
        clusterRole: view
      - namespace: web
        clusterRole: edit
```

```yaml
rbacBindings:
  - name: avery
    subjects:
      - kind: User
        name: avery@example.com
    roleBindings:
      - namespace: api
        clusterRole: admin
      - namespace: web
        clusterRole: edit
```

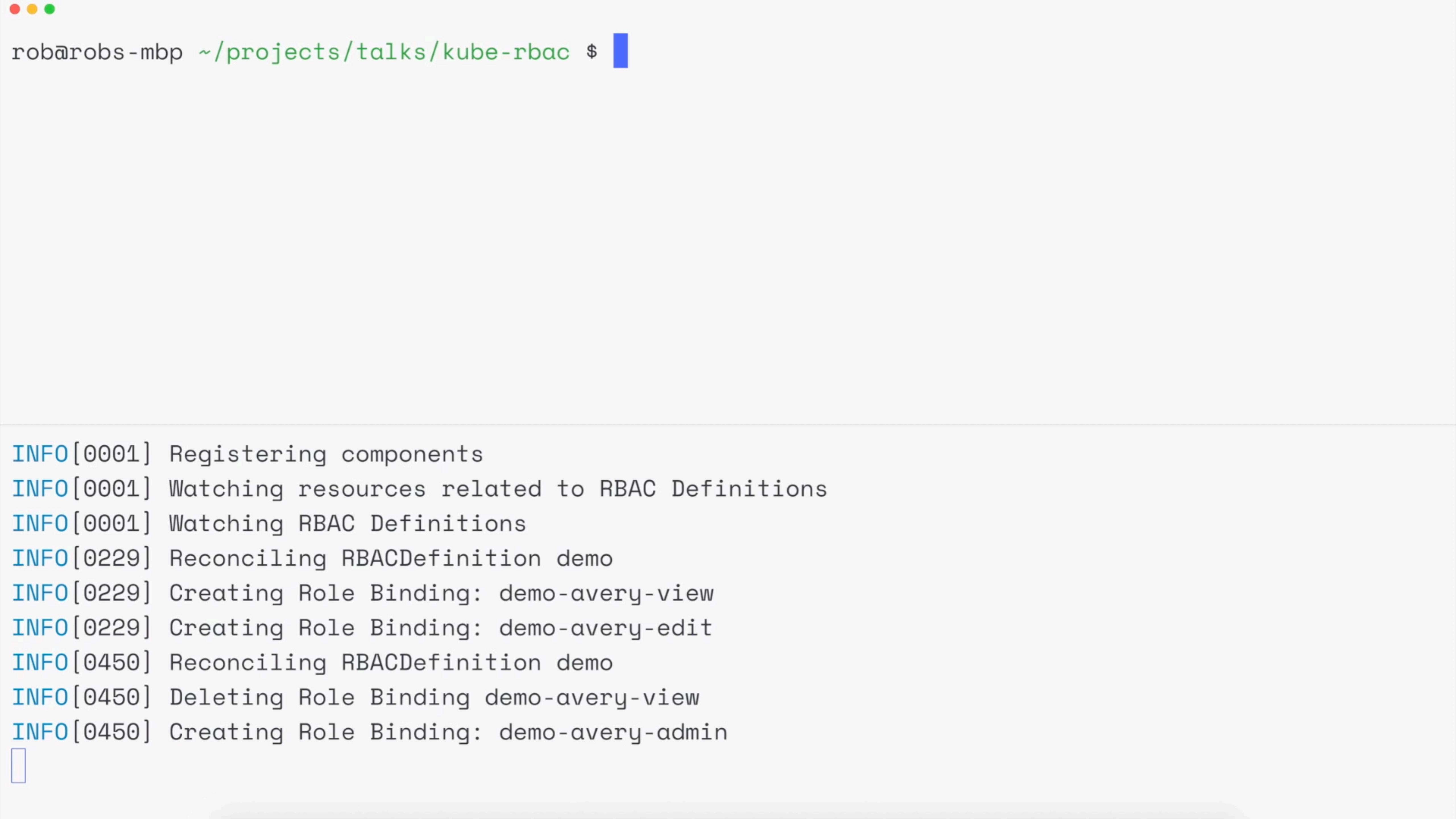


```
rob@robs-mbp ~/projects/talks/kube-rbac $
```

```
INFO[0001] Registering components
INFO[0001] Watching resources related to RBAC Definitions
INFO[0001] Watching RBAC Definitions
INFO[0229] Reconciling RBACDefinition demo
INFO[0229] Creating Role Binding: demo-avery-view
INFO[0229] Creating Role Binding: demo-avery-edit
```

```yaml
rbacBindings:
  - name: avery
    subjects:
      - kind: User
        name: avery@example.com
    roleBindings:
      - ~~namespace: api~~
        ~~clusterRole: admin~~
      - namespace: web
        clusterRole: edit
```

```yaml
rbacBindings:
  - name: avery
    subjects:
      - kind: User
        name: avery@example.com
    roleBindings:
     - namespace: web
       clusterRole: edit
```



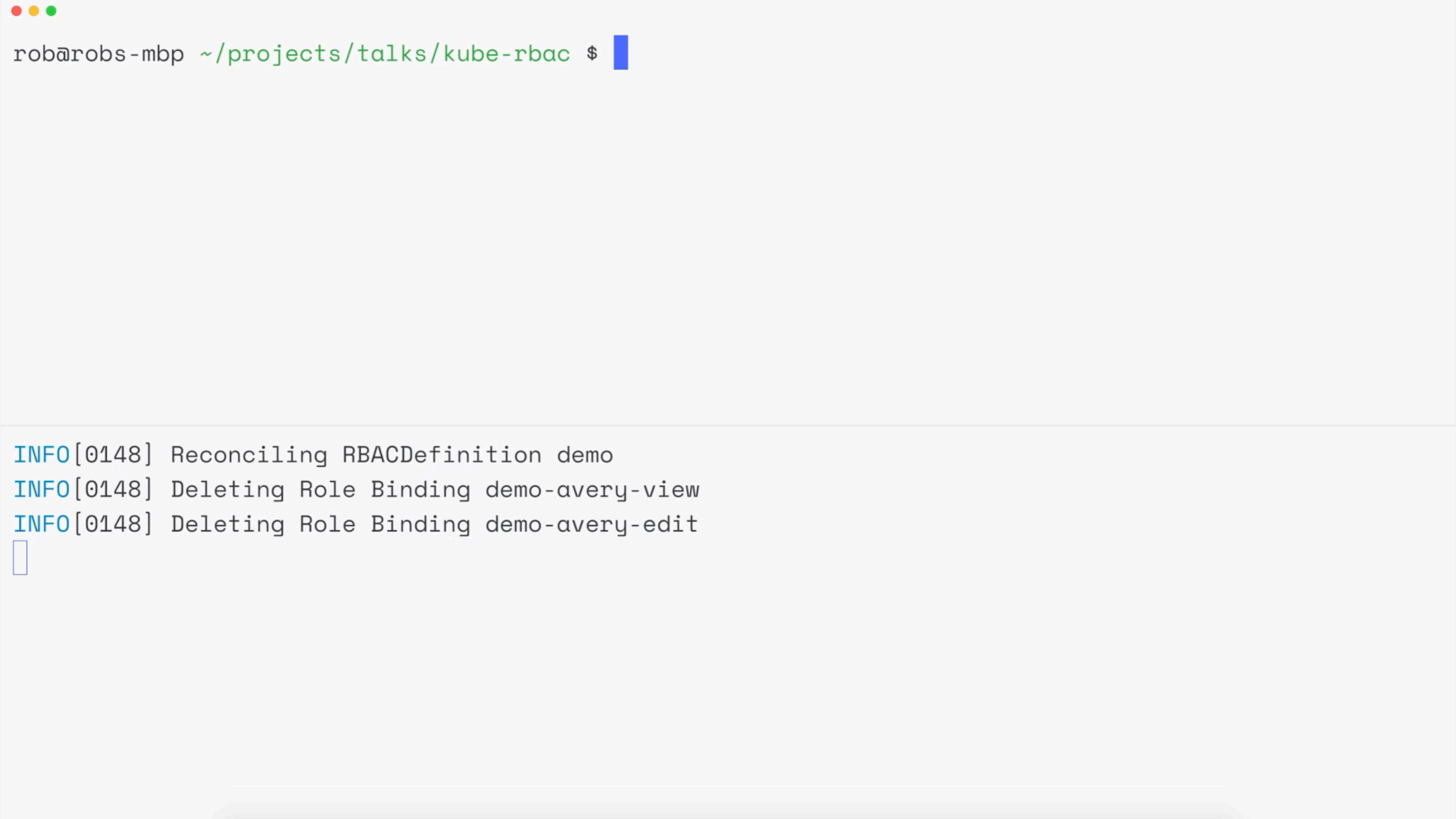
```
rob@robs-mbp ~/projects/talks/kube-rbac $
```

```
INFO[0001] Registering components
INFO[0001] Watching resources related to RBAC Definitions
INFO[0001] Watching RBAC Definitions
INFO[0229] Reconciling RBACDefinition demo
INFO[0229] Creating Role Binding: demo-avery-view
INFO[0229] Creating Role Binding: demo-avery-edit
INFO[0450] Reconciling RBACDefinition demo
INFO[0450] Deleting Role Binding demo-avery-view
INFO[0450] Creating Role Binding: demo-avery-admin
```

Label Selectors

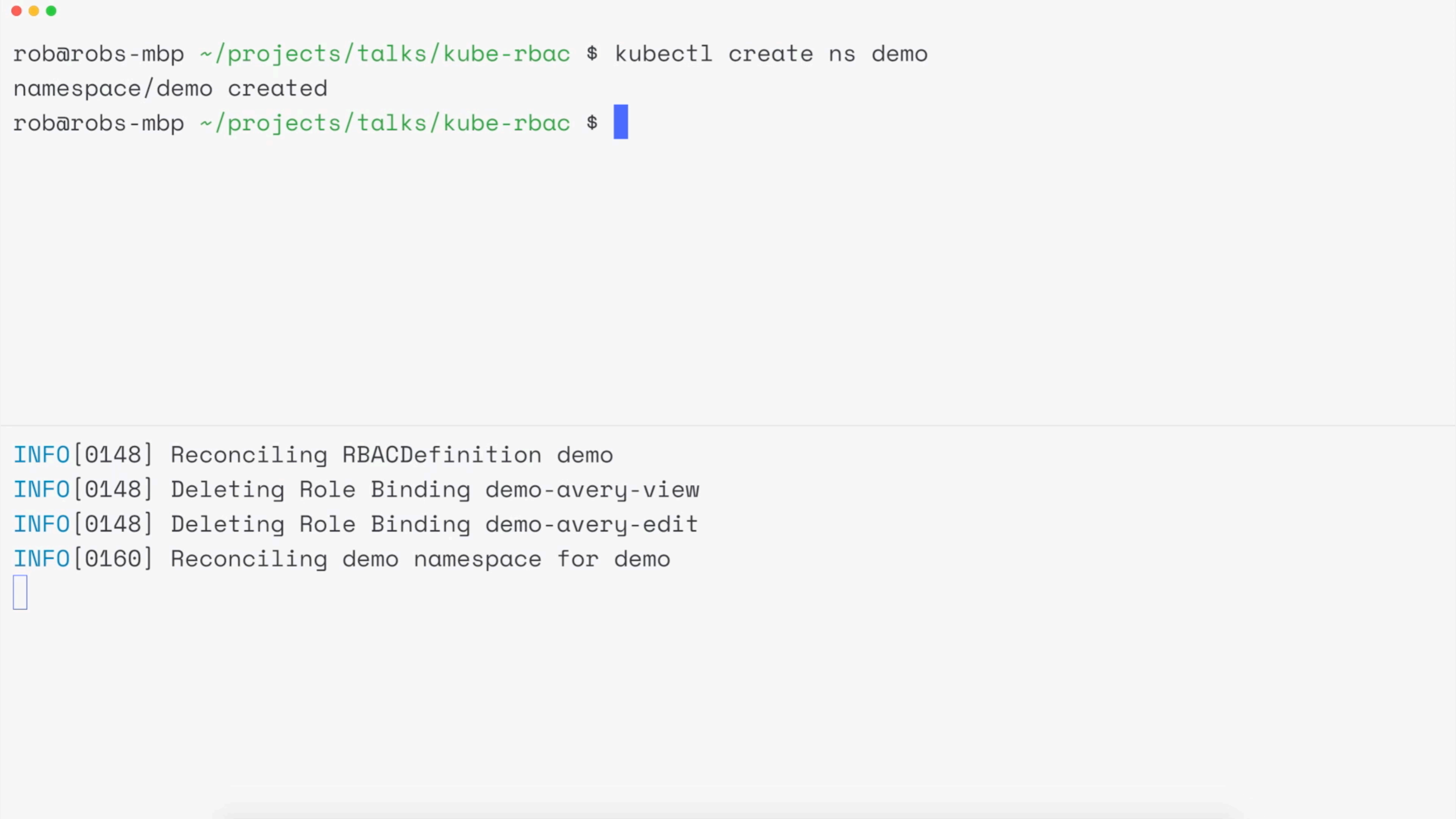
Advantage #3

```yaml
rbacBindings:
  - name: avery
    subjects:
      - kind: User
        name: avery@example.com
    roleBindings:
      - clusterRole: edit
        namespaceSelector:
          matchLabels:
            team: api
```



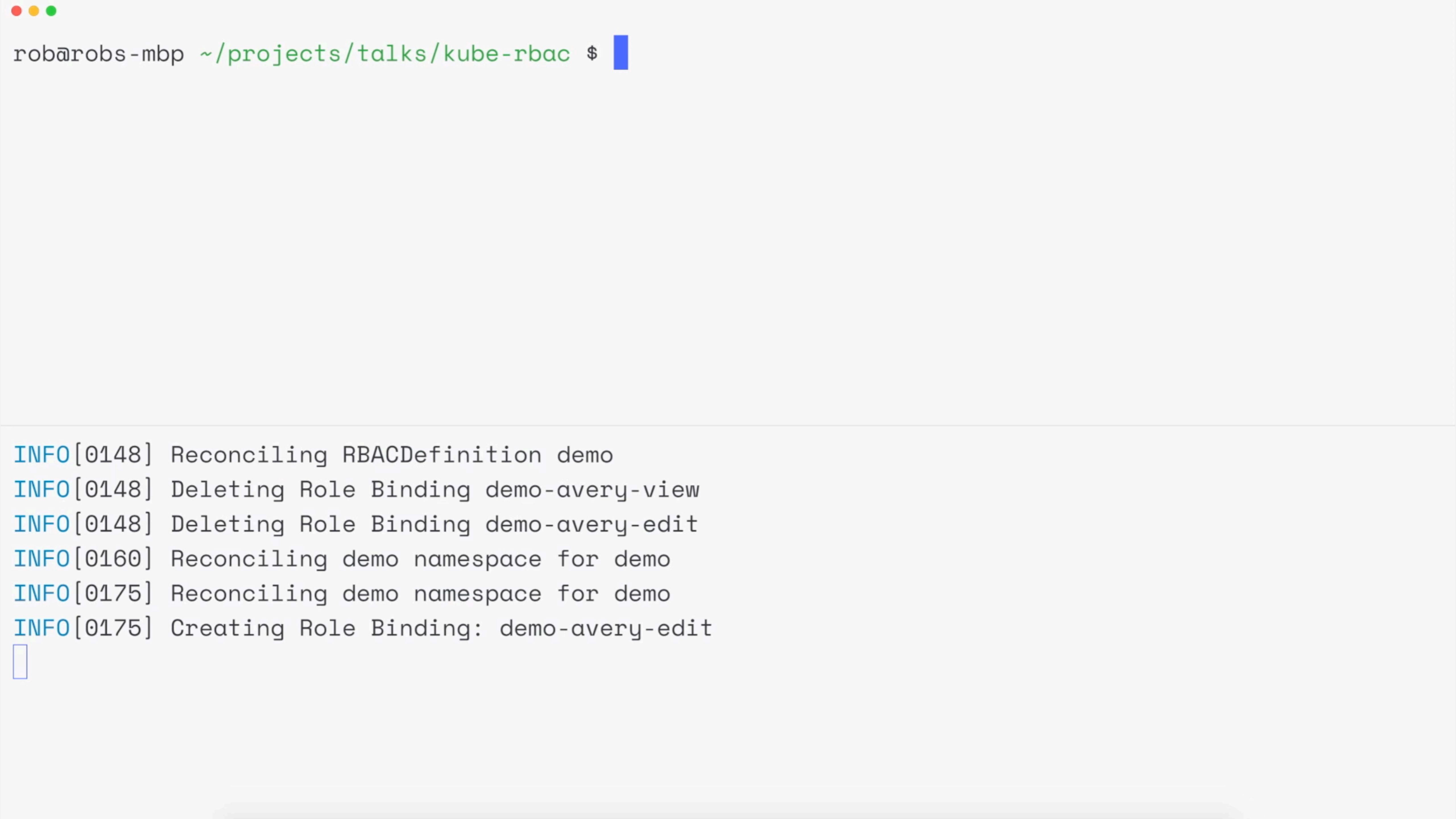
```
rob@robs-mbp ~/projects/talks/kube-rbac $
```

```
INFO[0148] Reconciling RBACDefinition demo
INFO[0148] Deleting Role Binding demo-avery-view
INFO[0148] Deleting Role Binding demo-avery-edit
```



```
rob@robs-mbp ~/projects/talks/kube-rbac $ kubectl create ns demo
namespace/demo created
rob@robs-mbp ~/projects/talks/kube-rbac $
```

```
INFO[0148] Reconciling RBACDefinition demo
INFO[0148] Deleting Role Binding demo-avery-view
INFO[0148] Deleting Role Binding demo-avery-edit
INFO[0160] Reconciling demo namespace for demo
```



```
rob@robs-mbp ~/projects/talks/kube-rbac $
```

```
INFO[0148] Reconciling RBACDefinition demo
INFO[0148] Deleting Role Binding demo-avery-view
INFO[0148] Deleting Role Binding demo-avery-edit
INFO[0160] Reconciling demo namespace for demo
INFO[0175] Reconciling demo namespace for demo
INFO[0175] Creating Role Binding: demo-avery-edit
```

RBAC Manager Recap

- More concise and simpler configuration that groups role bindings together
- RBAC changes are now easy to automate
- Label selectors simplify RBAC for ephemeral environments

Thanks!

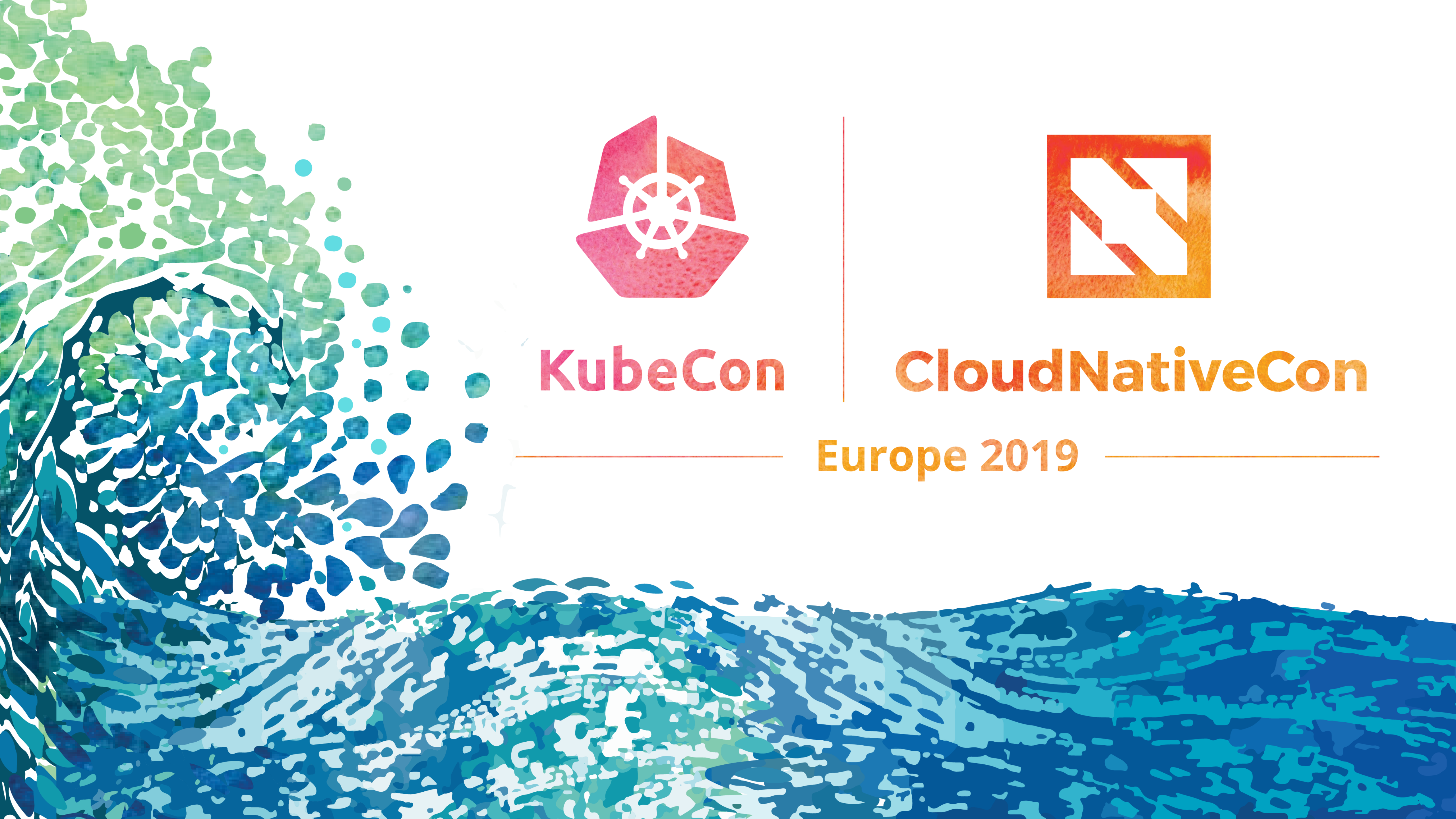
github.com/corneliusweig/**rakkess**

github.com/aquasecurity/**kubectl-who-can**

github.com/reactiveops/**rbac-lookup**

github.com/reactiveops/**rbac-manager**

@robertjscott



KubeCon | CloudNativeCon

Europe 2019