Accessing the Cluster as a User

In this lesson, we will impersonate John and try to get the access authenticated.

WE'LL COVER THE FOLLOWING

Authentication

Since John is not around, we'll do some role-playing and impersonate him.

Configuring kubectl

John will first have to set the cluster using the address and the certificate authority we sent him.

```
kubectl config set-cluster jdoe \
    --certificate-authority \
    keys/ca.crt \
    --server $SERVER
```

We created a new cluster called `jdoe`.

Please note that the command we executed created just another config entry that enables to access all resources that currently exist in the cluster. It manipulated local text file, without changing anything in the existing cluster.

Setting the Credentials

Next, he'll have to set the credentials using the certificate and the key we created for him.

created for him.

```
kubectl config set-credentials jdoe \
    --client-certificate keys/jdoe.crt \
    --client-key keys/jdoe.key
```

We created a new set of credentials called `jdoe`.

Creating a New Context

Finally, John will have to create a new context.

```
kubectl config set-context jdoe \
    --cluster jdoe \
    --user jdoe

kubectl config use-context jdoe
```

We created the context `jdoe` that uses the newly created cluster and the user. We also made sure that we're using the newly created context.

Let's take a look at the config.

```
kubectl config view
```

The **output**, limited to John's settings, is as follows.

```
...
clusters:
- cluster:
    certificate-authority: /Users/vfarcic/IdeaProjects/k8s-specs/keys/ca.crt
    server: https://192.168.99.106:8443
  name: jdoe
...
contexts:
- context:
    cluster: jdoe
    user: jdoe
  name: jdoe
...
current-context: jdoe
...
users:
- name: jdoe
  user:
    client-certificate: /Users/vfarcic/IdeaProjects/k8s-specs/keys/jdoe.crt
    client-key: /Users/vfarcic/IdeaProjects/k8s-specs/keys/jdoe.key
...
```

No Authorization

John should be happy thinking that he can access our cluster. Since he's a curious person, he'll want to see the Pods we're running.

```
kubectl get pods
```

The **output** is as follows.

```
Error from server (Forbidden): pods is forbidden: User "jdoe" cannot list resource "pods" in
```

That's frustrating. John can reach our cluster, but he cannot retrieve the list of Pods. Since hope dies last, John might check whether he is forbidden from seeing other types of objects.

```
kubectl get all
```

The **output** is a long list of all the objects he's forbidden from seeing. So, in other words, John is authenticated successfully, but he is not authorized to view objects in the cluster.

John picks up his phone to beg not only that you give him access to the cluster, but also the permissions to "play" with it.

Before we change John's permission, in the next lesson, we should explore the components involved in the RBAC authorization process.