Steps To Create a User and Restrict Secrets in Minikube (wrt ndb-operator-system namespace)

1) Generate client certificates for the users, in this case it is user1 and user2

```
# User1
openssl genpkey -algorithm RSA -out user1-key.pem
openssl req -new -key user1-key.pem -out user1.csr -subj
"/CN=user1"

# User2
openssl genpkey -algorithm RSA -out user2-key.pem
openssl req -new -key user2-key.pem -out user2.csr -subj
"/CN=user2"
```

2) Approve the certificate signing requests

Unset

```
# User1
openssl x509 -req -in user1.csr -CA
/Users/<your-mac-username>/.minikube/ca.crt -CAkey
/Users/<your-mac-username>/.minikube/ca.key -CAcreateserial
-out user1.crt

# User2
openssl x509 -req -in user2.csr -CA
/Users/<your-mac-username>/.minikube/ca.crt -CAkey
/Users/<your-mac-username>/.minikube/ca.key -CAcreateserial
-out user2.crt
```

3) Create the Kubernetes Configurations

Unset

```
#User1
kubectl config set-credentials user1
--client-certificate=user1.crt --client-key=user1-key.pem

kubectl config set-context user1-context --cluster=minikube
--namespace=ndb-operator-system --user=user1
```

```
# User2
kubectl config set-credentials user2
--client-certificate=user2.crt --client-key=user2-key.pem

kubectl config set-context user2-context --cluster=minikube
--namespace=ndb-operator-system --user=user2
```

4) Create a role for the namespace, save it in a file called ndb-operator-system-role.yaml

Unset

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 namespace: ndb-operator-system
 name: ndb-operator-system-role
rules:
- apiGroups: [""]
 resources: ["pods", "services", "configmaps"] # Add more
resources as needed
 verbs: ["get", "list", "watch", "create", "update", "delete"]
```

5) Apply the role to the cluster

Unset

```
kubectl apply -f ndb-operator-system-role.yaml
```

6) Create a role binding for the users, create a yaml file for each user and change appropriately (user1-role-binding.yaml, user2-role-binding.yaml)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: user1-role-binding
```

```
  namespace: ndb-operator-system
subjects:
- kind: User
  name: user1
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: ndb-operator-system-role
  apiGroup: rbac.authorization.k8s.io
```

7) Apply the rolebindings

Unset

```
kubectl apply -f user1-role-binding.yaml
kubectl apply -f user2-role-binding.yaml
```

Part 2: Creating secrets for authentication, and restricting a secret only meant for a particular user

1) Create a RBAC role for each user, name them as user1-role.yaml and user2-role.yaml

Unset

```
# user1-role.yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 namespace: ndb-operator-system
 name: user1-role
rules:
- apiGroups: [""]
 resources: ["secrets"]
 verbs: ["get"]
 resourceNames: ["user1"] #The names of user1's secrets
should be here. No other user can add their secrets (add
names of db secrets over here as well once ready)

#Replace user1 with user2 for user2's role.yaml
```

2) Apply the roles

```
Unset
kubectl apply -f user1-role.yaml
kubectl apply -f user2-role.yaml
```

3) Create the role bindings for each user

```
Unset
# user1-rolebinding.yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: user1-rolebinding
 namespace: ndb-operator-system
subjects:
- kind: User
 name: user1
 apiGroup: ""
roleRef:
 kind: Role
 name: user1-role
 apiGroup: rbac.authorization.k8s.io

# user2-rolebinding.yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: user2-rolebinding
 namespace: ndb-operator-system
subjects:
- kind: User
 name: user2
 apiGroup: ""
roleRef:
```

```
  kind: Role
  name: user2-role
  apiGroup: rbac.authorization.k8s.io
```

4) Apply the role bindings

```
Unset
kubectl apply -f user1-rolebinding.yaml
kubectl apply -f user2-rolebinding.yaml
```

5) Create secrets for each user, with their username and password and optional CA certificate. These will be used by the ndb operator to authenticate into the NDB databases. Note: This example only demonstrates creating a secret for the ndb, a separate secret needs to be created with appropriate fields for the db. Refer to the git for how a db secret is created.

```
Unset
kubectl create secret generic user1
--namespace=ndb-operator-system \
 --from-literal=username=user1-username-for-ndb-server \
 --from-literal=password=user1-password-for-ndb-server
```

```
Unset
kubectl create secret generic user2
--namespace=ndb-operator-system \
 --from-literal=username=user2-username-for-ndb-server \
 --from-literal=password=user2-password-for-ndb-server
```

6) With the secrets created, now switch to user1 context

```
Unset
kubectl config use-context user1-context
```

7) In user1 context, issue the command - kubectl get secret user1. You should be able to view the secret created for user1. Now if you issue the command kubectl get secret user2, it should throw an error as expected. This ensures that the secrets have been secured for each user.