

Anlegen von Authentifizierungs Objekten (User), Rollen, Cluster-Rollen und die Bindings

Der richtig schnelle Weg

```
kubeadm kubeconfig user -h  
sudo kubeadm kubeconfig user \  
  --client-name user \  
> kubeconfig-user.yaml
```

Erstellen eines Authentifizierung Objektes (User) mit Hilfe von OpenSSL Schlüssel und Zertifikat

Schlüssel und Zertifikat erstellen

```
openssl genrsa -out myuser.key 2048  
openssl req -new -key myuser.key \  
  -subj "/C=DK/ST=DK/O=' '/CN=myuser" \  
  -out myuser.csr
```

Extraieren des CSR (certificate signing request)

```
cat myuser.csr | base64 | tr -d "\n"
```

Erstellen eines Kubernetes CSR Objektes

```
apiVersion: certificates.k8s.io/v1  
kind: CertificateSigningRequest  
metadata:  
  name: myuser  
spec:  
  groups:  
    - system:authenticated  
  request: #einfügen der Ausgabe "extraieren des CSR"  
  signerName: kubernetes.io/kube-apiserver-client  
  usages:  
    - client auth
```

Kubernetes CSR approven

```
kubectl get csr  
kubectl certificate approve myuser
```

Extrahieren des von Kubernetes genehmigten signierten Zertifikates

```
kubectl get csr myuser -o jsonpath='{.status.certificate}' | base64 -d >  
myuser.crt
```

RBAC (Role und RoleBinding)

Erstellen einer Rolle für das Authentifizierungs Objekt (User)

!!!achtet auf den Namespace!!!

```
kubectl create role wiki-projekt --verb="*" --namespace wiki \  
  --resource=pod \  
  --resource=service \  
  --resource=configmap \  
  --resource=secret \  
  --resource=ingress \  
  --resource=daemonset \  
  --resource=replicaset \  
  --resource=deployment \  
  --resource=job
```

Verknüpfen der Rolle mit unserem Authentifizierungs Objekt (Verbindung mittels Subjectes) mittels RoleBinding

```
kubectl create rolebinding wiki-projekt-binding --role=wiki-projekt --  
user=myuser --namespace wiki
```

Aufräumen

```
kubectl delete csr myuser
```

extraieren der Konfig in unsere lokalen Konfigurationsdatei

~/.kube/config

```
kubectl config set-credentials myuser --client-key=myuser.key --client-certificate=myuser.crt --embed-certs=true
kubectl config set-context myuser --cluster=kubernetes --user=myuser
kubectl config use-context myuser
```

Keycloak / OAuth mit kubernetes (kube-apiserver)

Einen Client anlegen

Lege ein neues Client Objekt vom Type „OpenID Connect“ in Keycloak an und notiere die Client ID und den Client secret.

Den kube-apiserver konfigurieren

Als nächstes muss der kube-apiserver Konfiguriert werden. Je nachdem wie Eurer Kubernetes deployt wurde passt Ihr folgende Dateien an

kubeadm

/etc/kubernetes/manifests/kube-apiserver.yaml

```
spec:
  containers:
    - command:
      - kube-apiserver
      - --advertise-address=10.6.6.22
      - --allow-privileged=true
      ...
      - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
      - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
      - --oidc-issuer-url=https://<ip(fqdn)keycloak>/realms/<realms>
      - --oidc-client-id=kube-api-server-prod
```

rke2

/etc/rancher/rke2/config.yaml

```
write-kubeconfig-mode: 644
disable:
  - rke2-ingress-nginx
cni: calico
cluster-cidr: "172.20.0.0/16"
service-cidr: "172.30.0.0/16"
kube-apiserver-arg:
  - oidc-issuer-url=https://<ip(fqdn)keycloak>/realms/<realms>
```

```
- oidc-client-id=kube-api-server-test
```

Beim rke2 muss zusätzlich noch der rke2-server.service mittels systemctl neu gestartet werden.

kubectl (Clientseite) konfigurieren

Wir werden uns mittels krew das oidc-login Plugin für kubectl installieren. Dazu muss man sich [krew von github](#) holen.

Im Anschluss installiert man sich oidc-login

```
kubectl krew install oidc-login
```

Für die restliche Konfiguration folgt man einfach den Anweisungen von oidc-login

From:
<https://blog.cooltux.net/> - TuxNet DokuWiki

Permanent link:
https://blog.cooltux.net/doku.php?id=it-wiki:kubernetes:role_rolebinding_authobjects&rev=1709795061

Last update: 2024/03/07 07:04

