

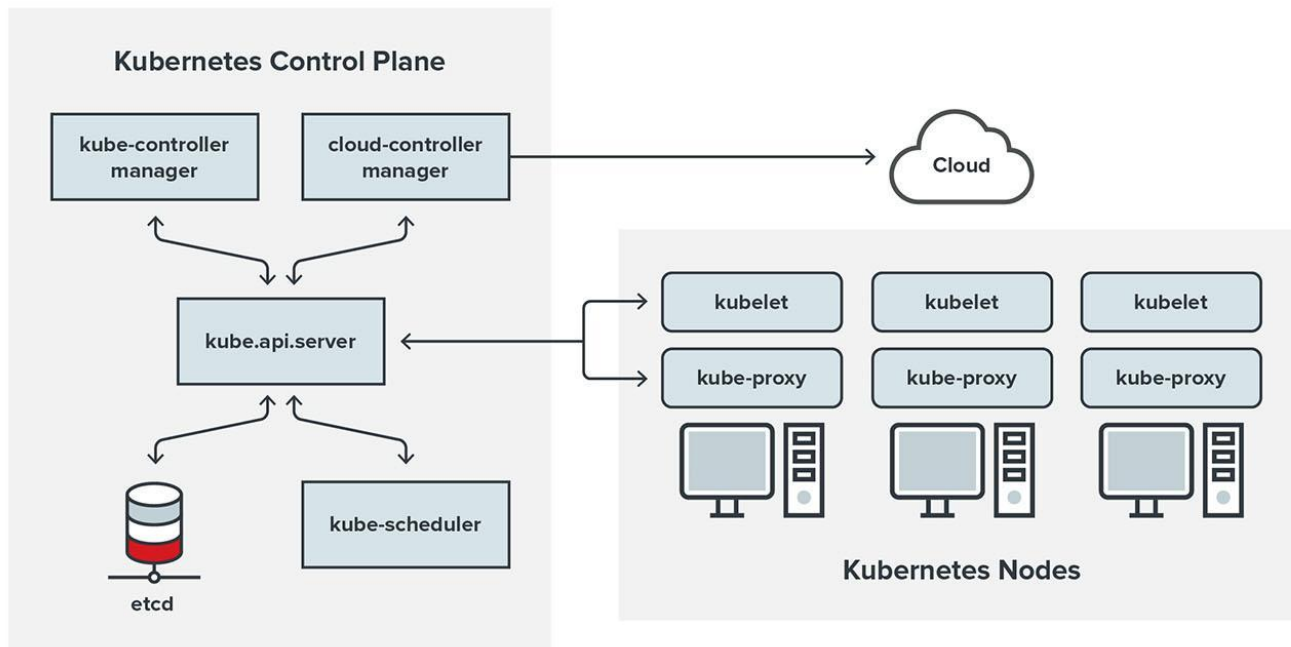
# Orquestração de Containers

**Segurança no Kubernetes**

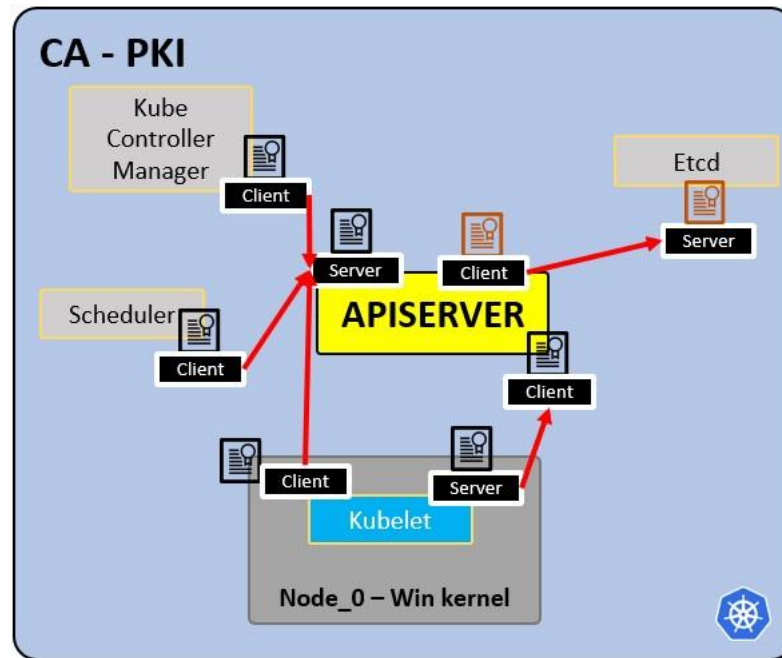
# Módulo 6 - Segurança no Kubernetes

- Comunicação TLS no cluster Kubernetes
- Role-Based Access Control (RBAC)
- kubeconfig
- Network Policies
- Private registry

# Comunicação entre componentes no k8s



## TLS entre componentes no k8s



## Onde os certificados são armazenados?

**/etc/kubernetes/pki**  
**/etc/kubernetes/pki/etcd**

```
-rw-r--r-- 1 root root 1289 Oct 24 2022 apiserver.crt
-rw-r--r-- 1 root root 1155 Oct 24 2022 apiserver-etcd-client.crt
-rw----- 1 root root 1675 Oct 24 2022 apiserver-etcd-client.key
-rw----- 1 root root 1679 Oct 24 2022 apiserver.key
-rw-r--r-- 1 root root 1164 Oct 24 2022 apiserver-kubelet-client.crt
-rw----- 1 root root 1679 Oct 24 2022 apiserver-kubelet-client.key
-rw-r--r-- 1 root root 1099 Oct 24 2022 ca.crt
-rw----- 1 root root 1675 Oct 24 2022 ca.key
-rw-r--r-- 1 root root  41 Jul  4 00:11 ca.srl
drwxr-xr-x 2 root root 4096 Oct 24 2022 etcd
-rw-r--r-- 1 root root 1115 Oct 24 2022 front-proxy-ca.crt
-rw----- 1 root root 1675 Oct 24 2022 front-proxy-ca.key
-rw-r--r-- 1 root root 1119 Oct 24 2022 front-proxy-client.crt
-rw----- 1 root root 1679 Oct 24 2022 front-proxy-client.key
-rw----- 1 root root 1675 Oct 24 2022 sa.key
-rw----- 1 root root  451 Oct 24 2022 sa.pub
```

/etc/kubernetes/pki/etcd:

total 32

```
-rw-r--r-- 1 root root 1086 Oct 24 2022 ca.crt
-rw----- 1 root root 1679 Oct 24 2022 ca.key
-rw-r--r-- 1 root root 1159 Oct 24 2022 healthcheck-client.crt
-rw----- 1 root root 1679 Oct 24 2022 healthcheck-client.key
-rw-r--r-- 1 root root 1204 Oct 24 2022 peer.crt
-rw----- 1 root root 1679 Oct 24 2022 peer.key
-rw-r--r-- 1 root root 1204 Oct 24 2022 server.crt
-rw----- 1 root root 1675 Oct 24 2022 server.key
```

## Visualizando informações de certificados

```
root@s2-master-1:~# openssl x509 -noout -text -in /etc/kubernetes/pki/apiserver.crt | head -n20
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 687183064776542058 (0x9895d5b47ab876a)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = kubernetes
        Validity
            Not Before: Jul 12 12:51:11 2023 GMT
            Not After : Jul 11 12:51:11 2024 GMT
        Subject: CN = kube-apiserver
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public-Key: (2048 bit)
            Modulus:
                00:b5:6a:fc:6a:dd:e0:8e:9e:9f:81:25:c3:46:8a:
                78:4a:aa:41:65:34:82:2c:91:d2:27:38:5d:35:22:
                a3:b0:f7:5a:d8:22:fb:5c:f8:03:07:8c:3b:b8:cf:
                07:31:47:ce:21:97:c0:a4:72:04:e8:8d:60:52:58:
                83:8e:55:78:ab:f4:fd:74:0d:26:e8:a8:56:ba:10:
                75:ac:15:b9:6b:d0:e0:4e:30:de:8c:02:67:7d:d0:
```

## Na dúvida, vamos direto à fonte

Kubernetes PKI certificates and requirements

<https://kubernetes.io/docs/setup/best-practices/certificates/>

## Gerência de Certificados com kubeadm (FELIZMENTE)

- Gera todos os certificados necessário para comunicação entre os componentes do cluster
- Verifica certificados expirados => `kubeadm certs check-expiration`
- Renova automaticamente os certificados durante o upgrade do cluster => `kubeadm upgrade apply`
- Renova manualmente os certificados (geralmente tem validade de 1 ano) => `kubeadm certs renew`

<https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-certs/>



## Processo de criação de usuários

1

Para criar um novo usuário no *cluster*, o primeiro passo é gerar sua **chave privada** e a requisição de certificado (**CSR**)

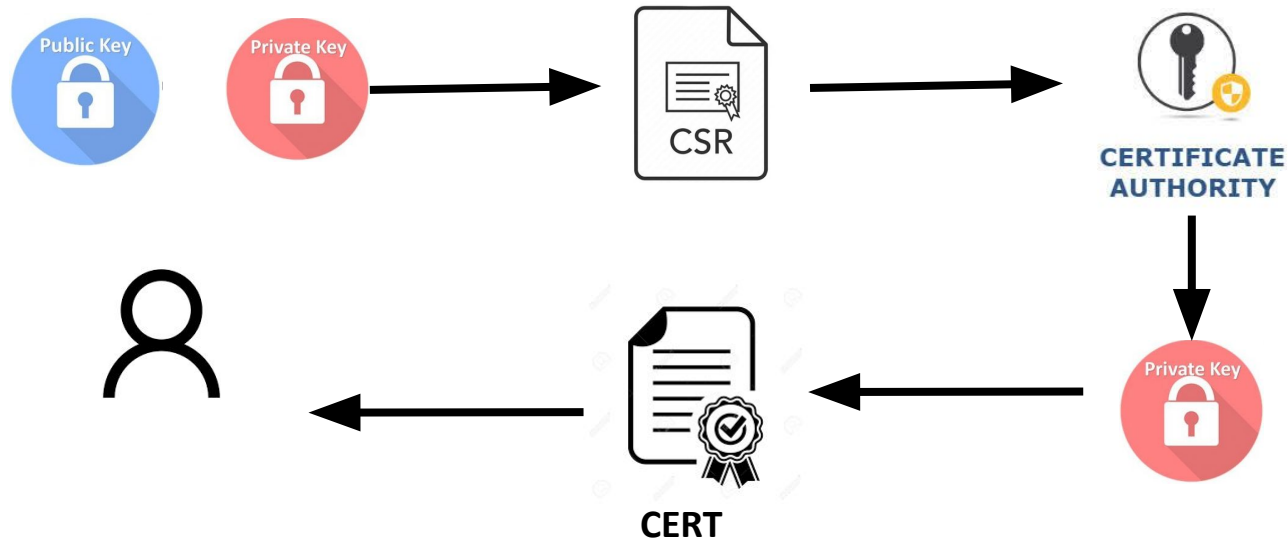
2

A seguir, a autoridade certificadora (CA) do apiserver deve criar e assinar o certificado

3

Por último, deve-se autorizar acesso ao cluster através de **roles** e **rolebindings**

## Processo de certificação digital



## Processo de criação de usuários

### Criando a chave privada e CSR via OpenSSL

1

```
mkdir -p /root/users/user1 && cd /root/users/user1  
openssl genrsa -out user1.key 2048  
openssl req -new -key user1.key -out user1.csr -subj "/CN=user1"
```

Importante configurar corretamente os parâmetros CN (username) e O (groupname)



Atenção

**O=system:masters**  
dá privilégio total ao cluster



“O” é usado para  
definição de grupos

## Processo de criação de usuários - Assinando o certificado

2

Identificar certificado e chave privada do CA do cluster

```
ls -l /etc/kubernetes/pki
```

```
apiserver.crt  
apiserver-etcd-client.crt  
apiserver-etcd-client.key  
apiserver.key  
apiserver-kubelet-client.crt  
apiserver-kubelet-client.key  
ca.crt  
ca.key  
ca.srl  
etcd  
front-proxy-ca.crt  
front-proxy-ca.key  
front-proxy-client.crt  
front-proxy-client.key  
sa.key  
sa.pub
```



## Processo de criação de usuários - Assinando o certificado

3

Gerar certificado assinado com ca.crt e ca.key do cluster k8s

```
openssl x509 -req -in user1.csr -CA /etc/kubernetes/pki/ca.crt \
-CAkey /etc/kubernetes/pki/ca.key -CAcreateserial \
-out user1.crt -days 365
```

```
root@s2-master-1:~/users/user1# openssl x509 -req -in user1.csr -CA /etc/kubernetes/pki/ca.crt \
> -CAkey /etc/kubernetes/pki/ca.key -CAcreateserial \
> -out user1.crt -days 364
Signature ok
subject=CN = user1
Getting CA Private Key
root@s2-master-1:~/users/user1# ls
user1.crt  user1.csr  user1.key
```

### Processo de criação de usuários - Assinando o certificado

Pronto! agora você já tem um usuário **user1** com certificado assinado (**user1.crt**) pela CA do cluster k8s.

```
root@s2-master-1:~/users/user1# openssl x509 -req -in user1.csr -CA /etc/kubernetes/pki/ca.crt \
> -CAkey /etc/kubernetes/pki/ca.key -CAcreateserial \
> -out user1.crt -days 364
Signature ok
subject=CN = user1
Getting CA Private Key
root@s2-master-1:~/users/user1# ls
user1.crt  user1.csr  user1.key
```

## Processo de criação de usuários

### Associar usuário ao cluster

```
kubectl config set-credentials user1 \  
--client-certificate=user1.crt \  
--client-key=user1.key --embed-certs=true
```

```
root@s2-master-1:~/users/user1# kubectl config set-credentials user1 \  
--client-certificate=user1.crt \  
--client-key=user1.key --embed-certs=true  
User "user1" set.
```

```
- name: user1
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURStLS0tCk1JSUNyRENDQVpRQ0ZBZTV
EUUVCQ3dVQU1CVXgKRxpBUKJnTLZCQU1UQ210MVltVnlibVYwWlhNd0hoY05Nak13T0RBMk1qRXd0VEF5V2hjTkl
jMLZ5TVRDQ0FTSXdEUVlKS29aSWWh2Y05BUUVCQlFBRGdnRVB BRENDCKFRb0NnZ0VCQUppqSFUvOTBkV2s0MkRJaFJ
TUGEKRUXyM0s0LzZyZ292Q1JlMLZ1bG1HMUdqanpvNy9GOGVKTm41e1FsOFVhQmFOL0pwc1B2ZWkwQnc2STZBZUd
0KzkyeFVKewliOG9lQk9JQjdNQ1UwUUUYTldhdGFUCjllLT2dEbWtndW11SmNMNnQwR0JKdEtoRE1QdzMrYU9IRGV
OYXdLaTFiNFerc3drTnBSZHVhTnd4VnJnTVBJVEFEblhhaJY1anhGMG56cUI2cXVIRTJ1cEJTMknZVApSWWFhMno
rcWhraUc5dzBCQVFzRkFBT0NBuUVBCK160FpsbGhLTG14CHBUWHoyQmZpM3hTbWk2RkZHaJvVHWFZKeHQ4bkV4cWt
ne1M0by9GOEJ6NHBxZGtZQ2dFbTFFbHovMndESnUzcWlZeXdwU3BwODgxQUJUzkhubApLUXc0djRMchVIVTlacUF
zciszbdDNZnmNFZms0CnVKSg1vZERkK3hOUgVkaE5jTDBLdzYzOXhoa05EQy83YU9zYWFuYU80L1NYb3BXVFhOVGL
1K2RGUmXkZVFjbl1EaDRoMUM4M0FswmVFZlFsc0oxNEl4T2NaOVFjawovM1RDY2JXdGg4NFJpTThnbE5tNGxBPT0
    client-key-data: LS0tLS1CRUdJTiBSU0EgUUFJJVkfURSBURVktLS0tLQpNSU1Fb2dJQkFBS0NBuUVBbU1
VbVlNMW9TSG5Wck5ETmhJOW9RdXZjcmovcXVDaThKRjdaVzZXSWJvYU9QT2p2OFh4NGsyZm50Q1h4Um9GbzM4bWw1
YektWRW5XMjhmQkRML2o3M2JGUW5LSnZ5aDRFNGdIc3dKVfJBUpjMVpxMXBQMG82QU9hU0M2YTRsd3ZxM1FZRW0
ZCmlONllPZUZtSUV6TTFyQXFMVnZorDZ6Q1EyBEYyNW8zREZXdUF3OGhNQU9kZHFQcm1QRVhTZk9vSHFNGNUYTY
RMVfJREFRQUJBb01COUZYMY29XRkc3N3ROR21zd0pBMDcvZ2VseXFMczJweXFnTHNETHo0ZX1SVHR4ak9McU9VbWw0
```



## Processo de criação de usuários

### Atribuir um contexto

```
kubectl create ns dev (OPCIONAL)
```

```
kubectl config set-context user1-context \  
--cluster=kubernetes --namespace=dev --user=user1
```

```
root@s2-master-1:~# kubectl config set-context user1-context \  
> --cluster=kubernetes --namespace=dev --user=user1  
Context "user1-context" created.
```

## Processo de criação de usuários

Context *user1-context* adicionado ao `$HOME/.kube/config`

```
contexts:
- context:
  cluster: kubernetes
  namespace: lab
  user: kubernetes-admin
  name: kubernetes-admin@kubernetes
- context:
  cluster: kubernetes
  namespace: dev
  user: user1
  name: user1-context
current-context: kubernetes-admin@kubernetes
```

## Processo de criação de usuários

### Acessando o novo contexto

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

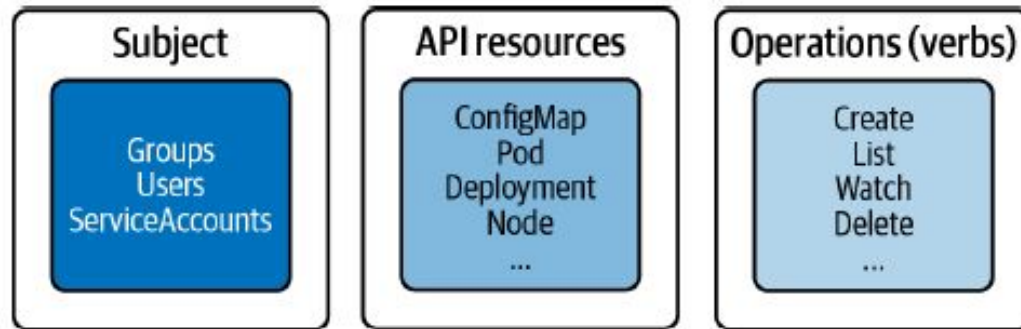
```
kubectl get pods Error from server (Forbidden): pods is forbidden: User "user1" cannot list resource "pods" in API group "" in the namespace "user1"
```

```
kubectl config use-context kubernetes-admin@kubernetes (volta para o context admin)
```

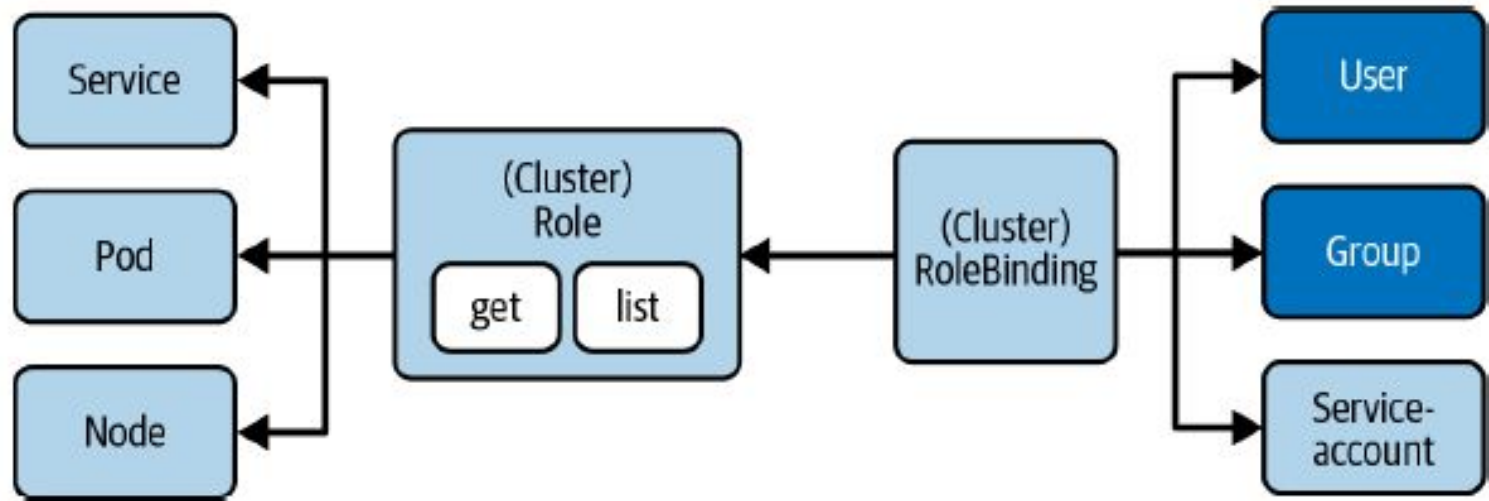


É preciso atribuir uma *role* para o novo usuário

## RBAC API no Kubernetes



## RBAC API no Kubernetes



## Exemplo de role

```
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: Role
```

```
metadata:
```

```
  namespace: default
```

```
  name: pod-reader
```

```
rules:
```

```
- apiGroups: [""], # "" indicates the core API group
```

```
  resources: ["pods"]
```

```
  verbs: ["get", "watch", "list"]
```

kubectl api-resource

kubectl -v6 ...

declarativo

**kubectl -n default create role pod-reader --verb=get,list,watch \**  
**--resource=pods**

imperativo

## Determinando *verbs* utilizados

### HTTP

#### verb

#### request verb

POST

create

GET,  
HEAD

get (for individual resources), list (for collections, including full object content), watch  
(for watching an individual resource or collection of resources)

PUT

update

PATCH

patch

DELETE

delete (for individual resources), deletecollection (for collections)

```
root@s2-master-1:~# k get pod -v6
I0807 01:26:40.198000 519276 loader.go:373] Config loaded from file: /root/.kube/config
I0807 01:26:40.228002 519276 round_trippers.go:553] GET https://192.168.68.20:6443/api/v1/namespaces/lab/pods?limit=500
OK in 11 milliseconds
```

## Processo de criação de usuários

### Criando uma role para o novo usuário

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

```
kubectl get roles -n user1
```

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

```
kubectl get pods
```

```
Error from server (Forbidden): pods is forbidden: User  
"user1" cannot list resource "pods" in API group "" in  
the namespace "user1"
```

```
kubectl config use-context \
```

```
kubernetes-admin@kubernetes
```



É preciso criar um *rolebinding* que associa a **role** ao novo usuário



## Processo de criação de usuários

### Criando uma rolebinding

```
kubectl apply -f user1-rolebind.yaml
```

```
kubectl get rolebindings -n user1
```

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

```
kubectl get pods
```

```
No resources found in user1 namespace.
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: user1-rolebind
  namespace: dev
subjects:
- kind: User
  name: user1
  apiGroup: ""
roleRef:
  kind: Role
  name: user1-role
  apiGroup: ""
```

## Escopos

Namespace

- Roles
- RoleBindings

Cluster

- ClusterRoles
- ClusterRoleBindings

Podem haver diferentes *clusters*, usuários, *namespaces* e autenticações num ambiente complexo

Para gerenciar isso, do ponto de vista do usuário, é interessante o uso dos arquivos *kubeconfig*

Por padrão, este arquivo é procurado com o nome *config* em *\$HOME/.kube*

Também é possível especificar via variável *\$KUBECONFIG* ou *flag* *--kubeconfig*

<https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/>

## Contextos no *kubeconfig*

Um contexto é definido em um arquivo kubeconfig como sendo:

Um cluster-alvo

Um namespace

Um usuário para acesso

O contexto padrão usado pelo comando kubectl é denominado current context

## Visualizando o *kubeconfig*

**kubectl config view**

**vim \$HOME/.kube/config**

```
root@s2-master-1:~# k config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://192.168.68.20:6443
    name: kubernetes
contexts:
- context:
    cluster: kubernetes
    namespace: lab
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
- context:
    cluster: kubernetes
    namespace: dev
    user: user1
  name: user1-context
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: DATA+OMITTED
    client-key-data: DATA+OMITTED
- name: user1
  user:
    client-certificate-data: DATA+OMITTED
    client-key-data: DATA+OMITTED
```

## Adicionando um usuário ao *kubeconfig*

```
kubectl config set-credentials user1 \  
--client-certificate=user1.crt \  
--client-key=user1.key --embed-certs=true
```

```
kubectl config set-context user1-context \  
--cluster=kubernetes --namespace=user1  
--user=user1
```

```
kubectl config view
```

```
root@s2-master-1:~# k config view  
apiVersion: v1  
clusters:  
- cluster:  
  certificate-authority-data: DATA+OMITTED  
  server: https://192.168.68.20:6443  
  name: kubernetes  
contexts:  
- context:  
  cluster: kubernetes  
  namespace: lab  
  user: kubernetes-admin  
  name: kubernetes-admin@kubernetes  
- context:  
  cluster: kubernetes  
  namespace: dev  
  user: user1  
  name: user1-context  
current-context: kubernetes-admin@kubernetes  
kind: Config  
preferences: {}  
users:  
- name: kubernetes-admin  
  user:  
    client-certificate-data: DATA+OMITTED  
    client-key-data: DATA+OMITTED  
- name: user1  
  user:  
    client-certificate-data: DATA+OMITTED  
    client-key-data: DATA+OMITTED
```

## Acesso remoto via kubeconfig

```
cp /root/.kube/config /root/users/user1/kubeconfig
```

\* retirar do arquivo as partes relacionadas ao usuário admin do cluster

```
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data:
  server: https://192.168.68.20:6443
  name: kubernetes
contexts:
- context:
  cluster: kubernetes
  namespace: user1
  user: user1
  name: user1-context
current-context: user1-context
kind: Config
preferences: {}
users:
- name: user1
  user:
  client-certificate-data:
  client-key-data:
```

# Acesso remoto via kubeconfig

```
kubectl --kubeconfig=kubeconfig get pods  
No resources found in user1 namespace.
```



Usuário *user1* não tem permissão para *criar* PODs  
Vamos resolver esse problema adicionando o verb  
create na role do usuário

```
apiVersion: rbac.authorization.k8s.io/v1  
kind: Role  
metadata:  
  name: user1-role  
  namespace: user1  
rules:  
- apiGroups: [""]  
  resources: ["pods", "services"]  
  verbs: ["get", "update", "list", "create", "delete"]
```



O serviceaccount é usado no cluster para que aplicativos que rodam nos pods acessem recursos do cluster k8s

Vamos criar um serviceaccount reader-sa que terá permissão para listar pods no namespace teste

## 1 serviceaccount

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: reader-sa
  namespace: teste
```

## 2 role

```
apiVersion:
rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: teste
  name: pod-reader
rules:
- apiGroups: ["" ]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
```

Vamos criar um serviceaccount reader-sa que terá permissão para listar pods no namespace teste

3

**apiVersion:** rbac.authorization.k8s.io/v1

**kind:** RoleBinding

**metadata:**

**name:** read-pods

**namespace:** test

**subjects:**

- **kind:** ServiceAccount

**name:** reader-sa

**namespace:** teste

**roleRef:**

**kind:** Role

**name:** pod-reader

**apiGroup:** rbac.authorization.k8s.io

Vamos criar um serviceaccount reader-sa que terá permissão para listar pods no namespace teste

4

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: my-pod**

**namespace: teste**

**spec:**

**serviceAccountName: reader-sa**

**containers:**

**- name: my-container**

**image: nginx**

Permitem controlar o  
fluxo de tráfego em  
camadas 3 e 4

Permitem definir como um  
*POD* pode interagir com  
outras entidades de rede

Essas entidades são: outros  
*PODs*, *namespaces* e  
endereços IP

Seletores são utilizados para  
escolher os alvos das políticas

*netpols* são implementadas  
pelo *plugin* de rede do *cluster*

Nem todos os *plugins*  
possuem essa capacidade –  
verificar antes!

# Network Policies

```
{  
  apiVersion: networking.k8s.io/v1  
  kind: NetworkPolicy  
  metadata:  
    name: test-network-policy  
    namespace: default  
  spec:  
    podSelector:  
      matchLabels:  
        role: db  
    policyTypes:  
      - Ingress  
      - Egress  
    ingress:  
      - from:  
        - ipBlock:  
            cidr: 172.17.0.0/16  
            except:  
              - 172.17.1.0/24
```

Campos mandatórios

Inclui todos os PODs com o selector definido

**podSelector:** {} -> inclui todos os PODs

Toda netpol tem uma **policyType** que pode incluir ambos **Ingress** e **Egress**

A definição da regra

Para **Ingress** deve-se apenas utilizar a chave - **from**

Para **Egress** deve-se apenas utilizar a chave - **to**

Tráfego  
Entrada



Ingress

Tráfego  
Saída



Egress

POD

## IPTABLES

Chain INPUT => Ingress

Chain OUTPUT => Egress

## KUBERNETES

```
- namespaceSelector:  
  matchLabels:  
    project: myproject  
- podSelector:  
  matchLabels:  
    role: frontend  
ports:  
- protocol: TCP  
  port: 6379  
egress:  
- to:  
  - ipBlock:  
    cidr: 10.0.0.0/24  
ports:  
- protocol: TCP  
  port: 5978
```

Existem 3 tipos de seletores  
que podem ser usados aqui

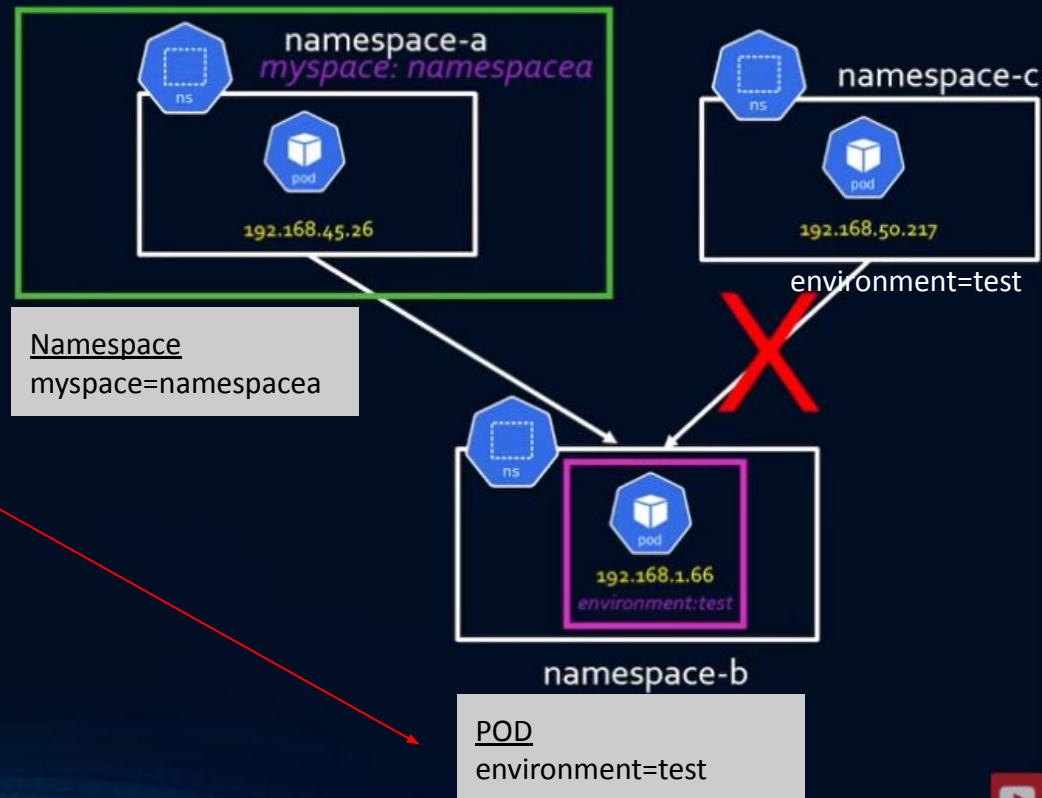


- podSelector
- namespaceSelector
- ipBlock



# Network Policies

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: namespace-b
spec:
  podSelector:
    matchLabels:
      environment: test
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          myspace: namespacea
```



# Network Policies

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: namespace-b
spec:
  podSelector:
    matchLabels:
      environment: test
  policyTypes:
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          myspace: namespacea
    - podSelector:
        matchLabels:
          role: frontend
```

OR Condition

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: namespace-b
spec:
  podSelector:
    matchLabels:
      environment: test
  policyTypes:
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          myspace: namespacea
    - podSelector:
        matchLabels:
          role: frontend
```

AND Condition

## *default allow all*

Por padrão, se nenhuma *netpol* existe num *namespace*, o comportamento é permitir todo o tráfego *ingress* e *egress*

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-allow-all
spec:
  podSelector: {}
  ingress:
  - {}
  egress:
  - {}
  policyTypes:
  - Ingress
  - Egress
```

## *default deny all*

```
---  
apiVersion: networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
  name: default-deny-all  
spec:  
  podSelector: {}  
  policyTypes:  
    - Ingress  
    - Egress
```

# A ordem de aplicação das regras importa?

Não importa a ordem de execução das network policies.

Ou seja, você pode aplicar uma regra que permite acesso ao conjunto de PODs através de seletores e em seguida aplicar a regra deny-all, e mesmo assim a primeira regra continuará funcionando.

Na dúvida, vamos direto à fonte

Kubernetes Network Policies

<https://kubernetes.io/docs/concepts/services-networking/network-policies/>

Permitem definir privilégios e controle de acesso para um *pod* ou container. Configurações incluem:

Controle de acesso discricionário (baseado em UID ou GID)

*Linux capabilities*

## Contexto de segurança: Controle de acesso discricionário (DAC)

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
    volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
    securityContext:
      allowPrivilegeEscalation: false
```



## Contexto de segurança: *capabilities*

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo-4
spec:
  containers:
    - name: sec-ctx-4
      image: gcr.io/google-samples/node-hello:1.0
      securityContext:
        capabilities:
          add: ["NET_ADMIN", "SYS_TIME"]
```

# Contexto de segurança: *capabilities*

## 7.2) Habilitando *capabilities*

- a. Tente alterar a data e hora correntes do pod `aurora` para 21 de outubro de 2015, às 7:28 da manhã. O que ocorre?
  - ▶ Visualizar resposta
- b. Adicione a *capability* `SYS_TIME` ao pod. Recrie-o se necessário.
  - ▶ Visualizar resposta
- c. Tente novamente alterar a data com o comando utilizado no passo (a) desta atividade.
  - ▶ Visualizar resposta
- d. A lista de *capabilities* disponíveis em um sistema Linux é bastante extensa. Qual recurso pode ser consultado para visualizar a lista completa, bem como explicações sobre cada um desses *capabilities*?
  - ▶ Visualizar resposta

## Contexto de segurança: *DAC*

### 7) Contextos de segurança

#### 7.1) Alterando usuário e grupo efetivos

a. Crie o pod `aurora` com a imagem `busybox`, executando o comando `sleep 600`. Feito isso, responda: qual é o usuário utilizado para rodar esse processo?

► Visualizar resposta

b. Edite as configurações do pod, de forma que o processo execute com o *user ID* 1000. O que ocorre?

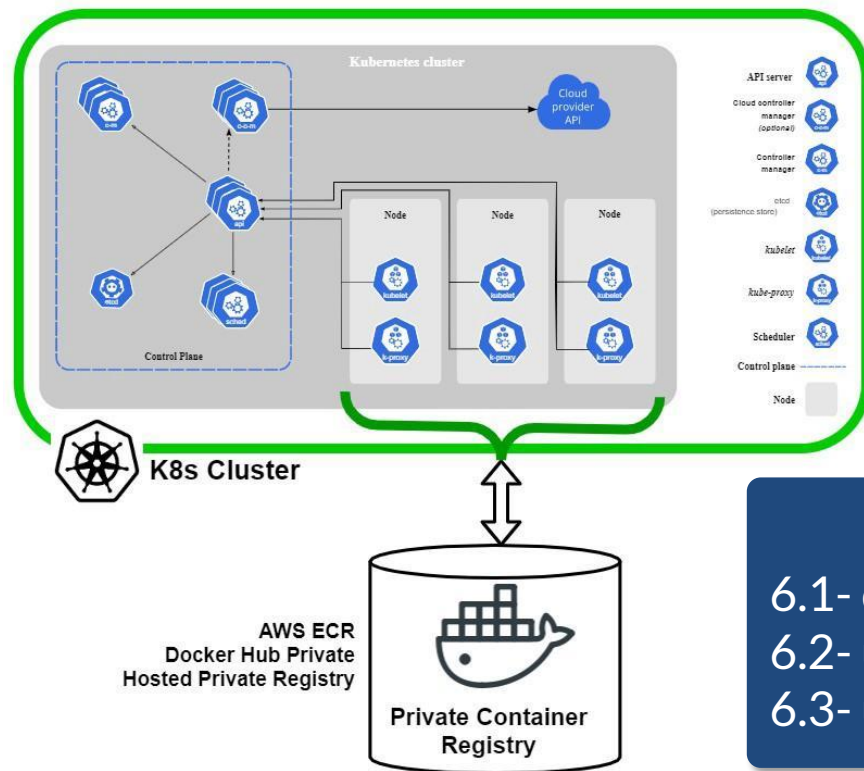
Em caso de erro, exporte e edite a configuração do pod, e em seguida recrie-o.

► Visualizar resposta

c. Por que o *user ID* do processo é mostrado em formato numérico?

► Visualizar resposta

# Private registry



## Atividade 6

- 6.1- deploy registry privado
- 6.2- upload de imagens
- 6.3- utilizando o registry privado

## Pull image de um registry privado

### Criando o secret

```
docker login  
cat ~/.docker/config.json
```

```
kubectl create secret generic regcred \  
--from-file=.dockerconfigjson=<~/.docker/config.json> \  
--type=kubernetes.io/dockerconfigjson
```

```
kubectl create secret docker-registry regcred \  
--docker-server=<your-registry-server> \  
--docker-username=<your-name> \  
--docker-password=<your-password> \  
--docker-email=<your-email>
```

## Pull image de um registry privado

### Criando o POD

```
apiVersion: v1
kind: Pod
metadata:
  name: private-reg
spec:
  containers:
    - name: private-reg-container
      image: <your-private-image>
      imagePullSecrets:
        - name: regcred
```

secret criado no passo anterior



ESCOLA  
SUPERIOR  
DE REDES

# Segurança no Kubernetes



**RNP**

ORGANIZAÇÃO SOCIAL DO MCTI

MINISTÉRIO DO  
TURISMO

MINISTÉRIO DA  
DEFESA

MINISTÉRIO DA  
SAÚDE

MINISTÉRIO DA  
EDUCAÇÃO

MINISTÉRIO DA  
CIÊNCIA, TECNOLOGIA  
E INOVAÇÕES



PÁTRIA AMADA  
**BRASIL**  
GOVERNO FEDERAL