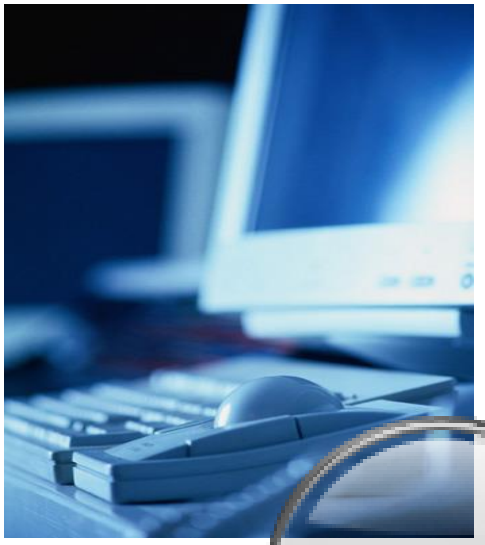
쿠버네티스 유저 관리

- 인증을 위한 다양한 리소스
- 유저와 서비스어카운트
- TLS 인증서를 활용한 통신 이해
- TLS 인증서를 활용한 유저 생성
- kube config 파일을 사용한 인증
- RBAC 기반 권한 관리



인증을 위한 다양한 리소스

인증을 위한 다양한 리소스

쿠버네티스 인증 체계

- 모든 통신은 TLS로 대부분의 엑세스는 kube-apiserver를 통해서 통신해야 함

- 엑세스 가능한 유저
  - X509 Client Certs
  - Static Token File
  - Putting a Bearer Token in a Request
  - Bootstrap Tokens
  - Service Account Tokens
  - OpenID Connect Tokens: Azure Active Directory, Salesforce 및 Google에서 지원

인증을 위한 다양한 리소스

쿠버네티스 인증 체계

- **무엇을 할 수 있는가?**
  - RBAC Authorization: 조직 내의 개별 사용자의 역할에 따라 컴퓨터 또는 네트워크 리소스에 대한 액세스를 규제
  - ABAC Authorization(속성 기반 액세스 제어): 속성을 결합하는 정책을 사용하여 사용자에게 액세스 권한을 부여
  - Node Authorization: kubelets에서 만든 API 요청을 특별히 승인하는 특수 목적 권한 부여 모드
  - WebHook Mode: HTTP 콜백, 특정 일이 발생할 때 URL에 메시지를 전달

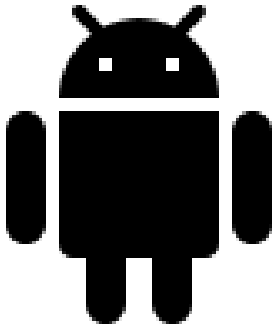
유저와 서비스어카운트

유저와 서비스어카운트

어카운츠에는 두 가지의 타입이 존재

- 유저: 일반 사용자를 위한 계정
- 서비스어카운트: 애플리케이션(파드 등)을 위한 계정

user



Service account

유저와 서비스어카운트

Static Token File

- Apiserver 서비스를 실행할 때 --token-auth-file=SOMEFILE.csv 전달 (kube-apiserver 수정 필요)
- API 서버를 다시 시작해야 적용됨

- 토큰, 사용자 이름, 사용자 uid, 선택적으로 그룹 이름 다음에 최소 3 열의 csv 파일

SOMEFILE.csv 파일 내용

```
password1,user1,uid001,"group1"
password2,user2,uid002
password3,user3,uid003
password4,user4,uid004
```

유저와 서비스어카운트

Static Token File 을 적용했을 때 사용 방법

- HTTP 요청을 진행할 때 다음과 같은 내용을 헤더에 포함해야함
- Authorization: Bearer 31ada4fd-adec-460c-809a-9e56ceb75269

```
$ TOKEN=password1
$ APISERVER=https://127.0.0.1:6443
$ curl -X GET $APISERVER/api --header "Authorization: Bearer $TOKEN" --insecure
```

- Kubectl에 등록하고 사용하는 방법

```
$ kubectl config set-credentials user1 --token=password1
$ kubectl config set-context user1-context --cluster=kubernetes \
                                           --namespace=frontend --user=user1
$ kubectl get pod --user user1
```

유저와 서비스어카운트

연습문제

- 다음 csv 파일을 생성하고 apiserver에 토큰을 등록하여 재시작하자.

```
password1,user1,uid001,"group1"
password2,user2,uid002
password3,user3,uid003
password4,user4,uid004
```

- 마스터 노드에 문제가 발생하는가? 어떤 서비스가 문제가 발생하는가?
- 문제가 발생하는 서비스의 컨테이너를 확인하기 위해 docker logs를 사용하고 그 해결 방법을 찾으라.
- 정상적으로 서비스가 시작됐다면 kubectl에 유저 정보를 등록하고 등록한 유저 권한으로 kubectl get pod 요청을 수행하라.
  - ✓ 반드시 Fobidden이라는 결과가 나와야 함 (유저는 생성됐으나 권한은 없다)

유저와 서비스어카운트

서비스 어카운트 만들기

- 명령어를 사용하여 serviceaccount sa1를 생성

```
kubectl create serviceaccount sa1
```

- sa와 함께 시크릿이 생성, 시크릿에는 토큰 값을 포함

```
$ kubectl get sa,secret
NAME                     SECRETS   AGE
serviceaccount/default   1         14m
serviceaccount/sa1       1         26s

NAME                           TYPE                                  DATA   AGE
secret/default-token-n7nvk     kubernetes.io/service-account-token   3      14m
secret/sa1-token-zgv85         kubernetes.io/service-account-token   3      26s
```

- 파드에 별도의 설정을 주지 않으면 default sa를 사용

유저와 서비스어카운트

파드에서 서비스 어카운트 사용하기

- Pod 에 spec.serviceAccount: <service-account-name>과 같은 형식으로 지정

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: nx
spec:
  serviceAccountName: sa1
  containers:
  - image: nginx
    name: nx
EOF
```

유저와 서비스어카운트

파드에서 서비스 어카운트 사용하기

- 생성된 파드에 sa1에 대한 시크릿에 잘 전달되었는지 확인 (자동 마운트됨)

```
$ kubectl get pod nx -o yaml
...
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: nx
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: sa1-token-zgv85
      readOnly: true
  volumes:
  - name: sa1-token-zgv85
    secret:
      defaultMode: 420
      secretName: sa1-token-zgv85
```

유저와 서비스어카운트

파드에서 서비스 어카운트 사용하기

- 생성된 파드에 sa1에 대한 시크릿에 잘 전달되었는지 확인 (자동 마운트됨)

```
$ kubectl exec -it nx -- bash
# ls /var/run/secrets/kubernetes.io/serviceaccount
ca.crt  namespace  token
```

- 토큰 값을 변수에 전달하고 curl 명령을 실행

```
TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
curl -X GET https://$KUBERNETES_SERVICE_HOST/api --header "Authorization: Bearer $TOKEN"  - insecure
```

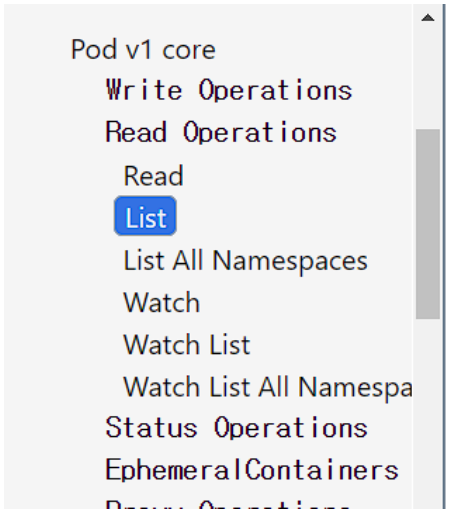
- 실행 결과

```
{
  "kind": "APIVersions",
  "versions": [
    "v1"
  ],
  "serverAddressByClientCIDRs": [
    {
      "clientCIDR": "0.0.0.0/0",
      "serverAddress": "172.17.0.49:6443"
    }
  ]
}
```

유저와 서비스어카운트

파드에서 서비스 어카운트 사용하기

- 파드에서 특정 네임스페이스에 리스트 파드 조회하기
- https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.22/#list-pod-v1-core



HTTP Request

GET /api/v1/namespaces/{namespace}/pods

Path Parameters

Parameter	Description
namespace	object name and auth scope, such as for teams and projects

```
curl -X GET https://$KUBERNETES_SERVICE_HOST/api/v1/namespaces/default/pods --header "Authorization: Bearer $TOKEN" -k
```

유저와 서비스어카운트

연습문제

- http-go라는 이름을 가진 ServiceAccount를 생성하고 http-go 파드를 생성해 http-go 서비스어카운트를 사용하도록 설정하라.

TLS 인증서를 활용한 통신 이해

TLS 인증서를 활용한 통신 이해

SSL 통신 과정 이해

- 응용계층인 HTTP와 TCP계층사이에서 작동,
- 어플리케이션에 독립적 -> HTTP제어를 통한 유연성
- 데이터의 암호화 (기밀성), 데이터 무결성, 서버인증기능, 클라이언트 인증기능

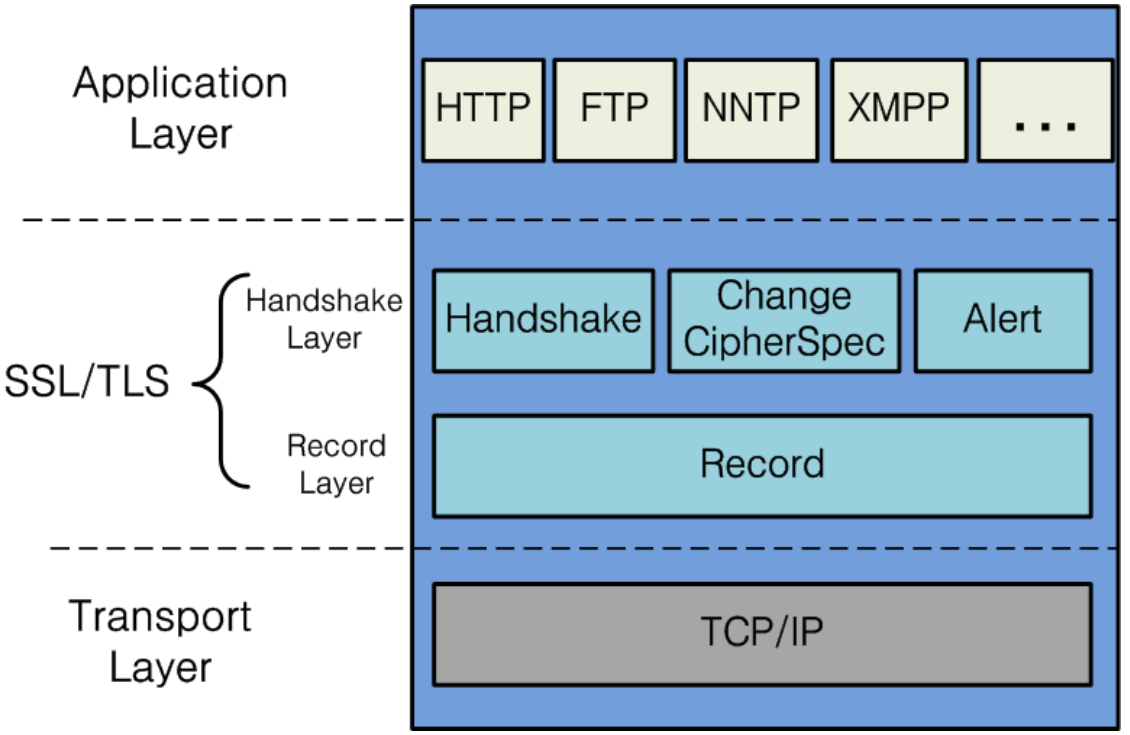


그림 출처: https://soul0.tistory.com/510

TLS 인증서를 활용한 통신 이해

Certificate를 보장하는 방법! 인증기관(CA)

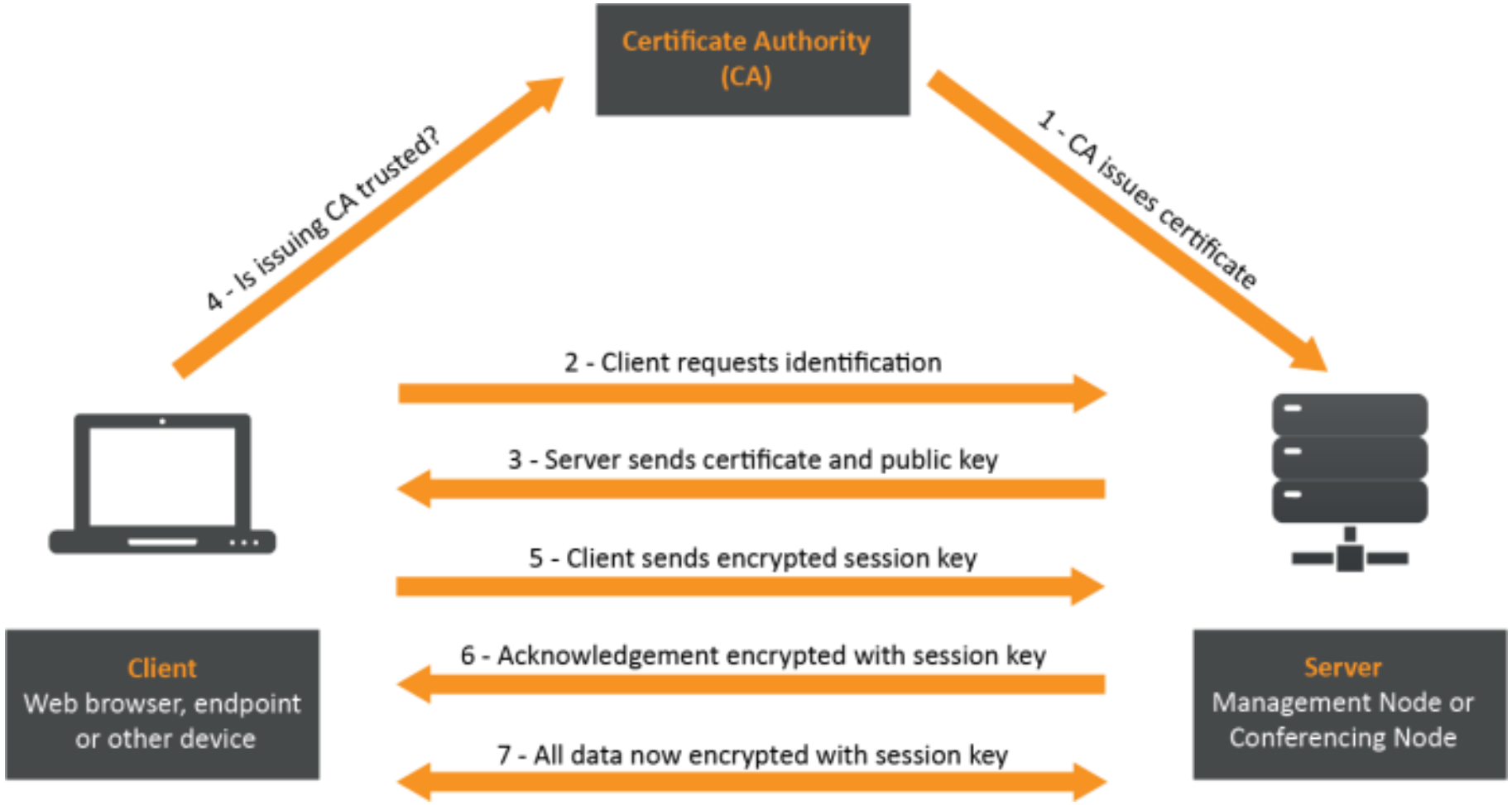


그림 출처: https://docs.pexip.com/admin/certificate_management.htm

TLS 인증서를 활용한 통신 이해

kubernete의 인증서 위치

```
$ sudo ls /etc/kubernetes/pki
apiserver-etcd-client.crt     apiserver.key    front-proxy-ca.key
apiserver-etcd-client.key     ca.crt                    front-proxy-client.crt
apiserver-kubelet-client.crt  ca.key                    front-proxy-client.key
apiserver-kubelet-client.key  etcd             sa.key
apiserver.crt                    front-proxy-ca.crt  sa.pub

$ sudo ls /etc/kubernetes/pki/etcd
ca.crt   healthcheck-client.crt     peer.crt  server.crt
ca.key   healthcheck-client.key     peer.key  server.key
```

TLS 인증서를 활용한 통신 이해

정확한 TLS 인증서 사용 점검

- 적절한 키를 사용하는지 확인하려면 manifests 파일에서 실행하는 certificate 확인 필요
- 적절한 키를 사용하지 않으면 에러가 발생

```
$ sudo ls /etc/kubernetes/manifests/
etcd.yaml                 kube-controller-manager.yaml
kube-apiserver.yaml       kube-scheduler.yaml
```

TLS 인증서를 활용한 통신 이해

정확한 TLS 인증서 사용 점검

- Kubelet의 위치는 조금 다름

```
server1@MASTER:~$ sudo ls /var/lib/kubelet/pki
kubelet-client-2019-07-09-12-44-22.pem       kubelet.crt
kubelet-client-2019-07-09-12-44-54.pem       kubelet.key
kubelet-client-current.pem
server1@MASTER:~$ sudo ls /var/lib/kubelet/
config.yaml          kubeadm-flags.env  plugins                 pods
cpu_manager_state     pki               plugins_registry
device-plugins       plugin-containers  pod-resources
server1@MASTER:~$ sudo cat /var/lib/kubelet/config.yaml
address: 0.0.0.0
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 2m0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
```

TLS 인증서를 활용한 통신 이해

TLS 인증서를 올바르지 않을 때 발생하는 현상 확인하기

- 먼저 현재 MASTER 노드의 상태를 스냅샷으로 저장하라.
- Kube-apiserver.yaml을 찾아 특정 경로를 수정하는 방식으로 certificate를 찾을 수 없도록 만들어라.

TLS 인증서를 활용한 통신 이해

인증서 정보 확인하기

```
$ sudo openssl x509 -in <certificate> -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            c4:70:38:32:90:74:e1:37:8a:77:36:5b:f4:39:06:a0
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=8bc6e46e-289a-4e7a-a182-a9cd4cfa5ab8
        Validity
            Not Before: Jul  6 05:09:38 2019 GMT
            Not After : Jul  4 06:09:38 2024 GMT
        Subject: CN=8bc6e46e-289a-4e7a-a182-a9cd4cfa5ab8
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b4:da:76:9f:0d:ef:08:60:32:d5:39:a6:99:51:
                    8a:97:5b:7f:11:98:20:d9:b6:55:af:95:9d:aa:a7:
                    fc:e1:68:a0:0d:31:64:69:d2:7f:cb:20:eb:8b:38:
                    22:92:c7:e3:92:5d:b9:67:60:16:fe:2a:35:02:4d:
                    ...
```

TLS 인증서를 활용한 통신 이해

인증서 목록

CERTIFICATE	RESIDUAL TIME	Certificate Path	CERTIFICATE AUTHORITY
admin.conf	365d	/etc/kubernetes	
apiserver	365d	/etc/kubernetes/pki	ca
apiserver-etcd-client	365d	/etc/kubernetes/pki/etcd	etcd-ca
apiserver-kubelet-client	365d	/etc/kubernetes/pki	ca
controller-manager.conf	365d	/etc/kubernetes/pki	
etcd-healthcheck-client	365d	/etc/kubernetes/pki/etcd	etcd-ca
etcd-peer	365d	/etc/kubernetes/pki/etcd	etcd-ca
etcd-server	365d	/etc/kubernetes/pki/etcd	etcd-ca
front-proxy-client	365d	/etc/kubernetes/pki	front-proxy-ca
scheduler.conf	365d	/etc/kubernetes	

CA 목록

CERTIFICATE AUTHORITY	RESIDUAL TIME	Certificate Path
ca	10y	/etc/kubernetes/pki
etcd-ca	10y	/etc/kubernetes/pki/etcd
front-proxy-ca	10y	/etc/kubernetes/pki

참고: front-proxy 인증서는 kube-proxy에서 API 서버 확장을 지원할 때만 kube-proxy에서 필요하다.

TLS 인증서를 활용한 통신 이해

모든 인증서 갱신하기

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

- Check certificate expiration
  - kubeadm certs check-expiration

- Automatic certificate renewal
  - kubeadm은 컨트롤 플레인을 업그레이드하면 모든 인증서를 자동으로 갱신

- Manual certificate renewal
  - kubeadm certs renew all

TLS 인증서를 활용한 통신 이해

연습문제

- Apiserver가 사용하는 ca.crt 정보를 텍스트 형태로 출력하라.
- ca.crt의 CN은 무엇인가?
- ca.crt를 이슈화한 CN은 무엇인가?

TLS 인증서를 활용한 유저 생성하기

TLS 인증서를 활용한 유저 생성하기

1. ca를 사용하여 직접 csr 승인하기

- **개인 키 생성하기**
  - 길이 2048 만큼의 개인 키 생성

```
$ openssl genrsa -out gasbugs.key 2048
```

- **private 키를 기반으로 인증서 서명 요청하기**
  - CN: 사용자 이름
  - O: 그룹 이름
  - CA에게 csr 파일로 인증을 요청할 수 있음!

```
$ openssl req -new -key gasbugs.key -out gasbugs.csr -subj
"/CN=gasbugs/O=boanproject"
```

TLS 인증서를 활용한 유저 생성하기

1. ca를 사용하여 직접 csr 승인하기

- Kubernetes 클러스터 인증 기관(CA) 사용이 요청을 승인해야 함
- 내부에서 직접 승인하는 경우 pki 디렉토리에 있는 ca.key와 ca.crt를 사용하여 승인 가능
- gasbugs.csr을 승인하여 최종 인증서인 gasbugs.crt를 생성
- -days 옵션을 사용해 며칠간 인증서가 유효할 수 있는지 설정(예제에서는 500일)

```
$ openssl x509 -req -in gasbugs.csr -CA /etc/kubernetes/pki/ca.crt -CAkey
/etc/kubernetes/pki/ca.key -CAcreateserial -out gasbugs.crt -days 500

Signature ok
subject=CN = gasbugs, O = boanproject
Getting CA Private Key
```

TLS 인증서를 활용한 유저 생성하기

2. 인증 사용을 위해 쿠버네티스에 crt를 등록

- crt를 사용할 수 있도록 kubectl을 사용하여 등록

```
$ kubectl config set-credentials gasbugs --client-certificate=.certs/gasbugs.crt  --
client-key=.certs/gasbugs.key
$ kubectl config set-context gasbugs-context --cluster=kubernetes --namespace=office -
-user=gasbugs
```

- 다음 명령을 사용하여 사용자 권한으로 실행 가능(지금은 사용자에게 권한을 할당하지 않아 실행되지는 않는다)

```
$ kubectl --context=gasbugs-context get pods
Error from server (Forbidden): pods is forbidden: User "gasbugs" cannot list
resource "pods" in API group "" in the namespace "office"
```

TLS 인증서를 활용한 유저 생성하기

연습문제

- dev1팀에 john이 참여했다. John을 위한 인증서를 만들고 승인해보자.

kube config 파일을 사용한 인증

kube config 파일을 사용한 인증

kube config 파일 확인하기

- $ kubectl config view
- $ kubectl config view --kube config=<config file>

```
$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://10.0.2.10:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes # 현재 kubectl이 사용중인 쿠버네티스 계정
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

kube config 파일을 사용한 인증

kube config의 구성

- ~/.kube/config 파일을 확인하면 세 가지 부분으로 작성됨
- clusters: 연결할 쿠버네티스 클러스터의 정보 입력
- users: 사용할 권한을 가진 사용자 입력
- contexts: cluster와 user를 함께 입력하여 권한 할당

```
$ curl https://kube-api-server:6443/api/v1/pods\
--key user.key \
--cert user.crt \
--cacert ca.crt \
```

Clusters

```
name: kube-cluster
cluster:
  certificate-authority: ca.crt
  server: https://cluster-ip:6443
```

Contexts

```
name: user-name@kube-cluster
context:
  cluster: kube-cluster
  user: user-name
```

Users

```
name: user-name
user:
  client-certificate: user.crt
  client-key: user.key
```

kube config 파일을 사용한 인증

인증 사용자 바꾸기

- kubectl config use-context user@kube-cluster

인증 테스트

- kubectl get pod
- forbidden이 나오면 성공!

kube config 파일을 사용한 인증

연습문제

- john을 유저로 사용할 수 있도록 세팅하라.

RBAC 기반 권한 관리

RBAC 기반 권한 관리

역할 기반 액세스 제어 (RBAC)

- 기업 내에서 개별 사용자의 역할을 기반으로 컴퓨터, 네트워크 리소스에 대한 액세스를 제어
- rbac.authorization.k8s.io API를 사용하여 정의
- 권한 결정을 내리고 관리자가 Kubernetes API를 통해 정책을 동적으로 구성
- RBAC를 사용하여 룰을 정의하려면 apiserver에 --authorization-mode=RBAC 옵션이 필요

- RBAC를 다루는 이 API는 총 4가지의 리소스를 컨트롤
  - Role
  - RoleBinding
  - ClusterRole
  - ClusterRoleBinding

RBAC 기반 권한 관리

rbac.authorization.k8s.ioAPI

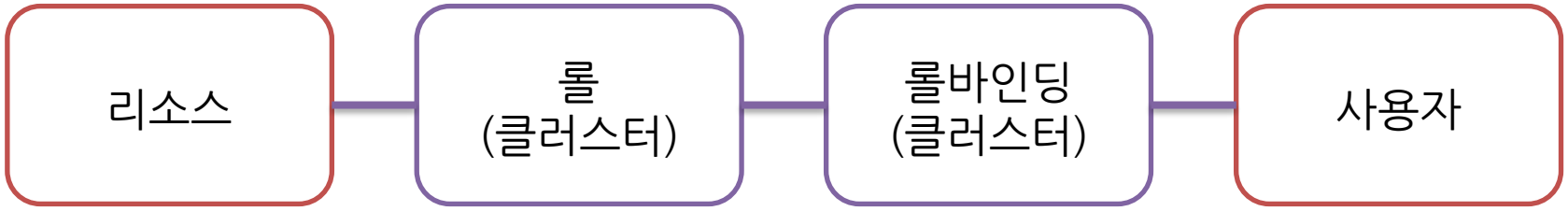
● 롤과 롤바인딩의 차이

➢ 롤

- ✓ “누가”하는 것인지는 정하지 않고 룰만을 정의
- ✓ 일반 롤의 경우에는 네임스페이스 단위로 역할을 관리
- ✓ 클러스터롤은 네임스페이스 구애 받지 않고 특정 자원을 전체 클러스터에서 자원을 관리할 롤을 정의

➢ 롤 바인딩

- ✓ “누가”하는 것인지만 정하고 롤은 정의하지 않음
- ✓ 롤을 정의하는 대신에 참조할 롤을 정의 (roleRef)
- ✓ 어떤 사용자에게 어떤 권한을 부여할지 정하는(바인딩) 리소스
- ✓ 클러스터롤에는 클러스터롤바인딩, 일반 롤에는 롤바인딩을 필요



RBAC 기반 권한 관리

rbac.authorization.k8s.ioAPI

- 룰(Rule) 작성 요령

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: office
  name: deployment-manager
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["deployments", "replicasets", "pods"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

- 룰에는 api 그룹, 리소스, 작업 가능한 동작을 작성
  - API를 사용하게 할 그룹 정의

RBAC 기반 권한 관리

사용자 권한 할당

- 롤바인딩을 사용하여 office 네임스페이스 권한 할당
- kubectl create -f 를 사용하여 생성
- 해당 네임 스페이스에 모든 권한을 생성

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: deployment-manager-binding
  namespace: office
subjects:
- kind: User
  name: gasbugs
  apiGroup: ""
roleRef:
  kind: Role
  name: deployment-manager
  apiGroup: ""
```

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: office
  name: deployment-manager
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["deployments", "replicasets", "pods"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

RBAC 기반 권한 관리

사용자 권한 테스트

- 실행돼야 하는 명령어

```
$ kubectl --context=gasbugs-context get pods -n office
$ kubectl --context=gasbugs-context run --generator=pod-run/v1 nginx --
image=nginx -n office
```

- 실행되면 안되는 명령어

```
$ kubectl --context=gasbugs-context get pods
$ kubectl --context=gasbugs-context run --generator=pod-run/v1 nginx --
image=nginx
```

RBAC 기반 권한 관리

API 도큐먼트

- https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.19/