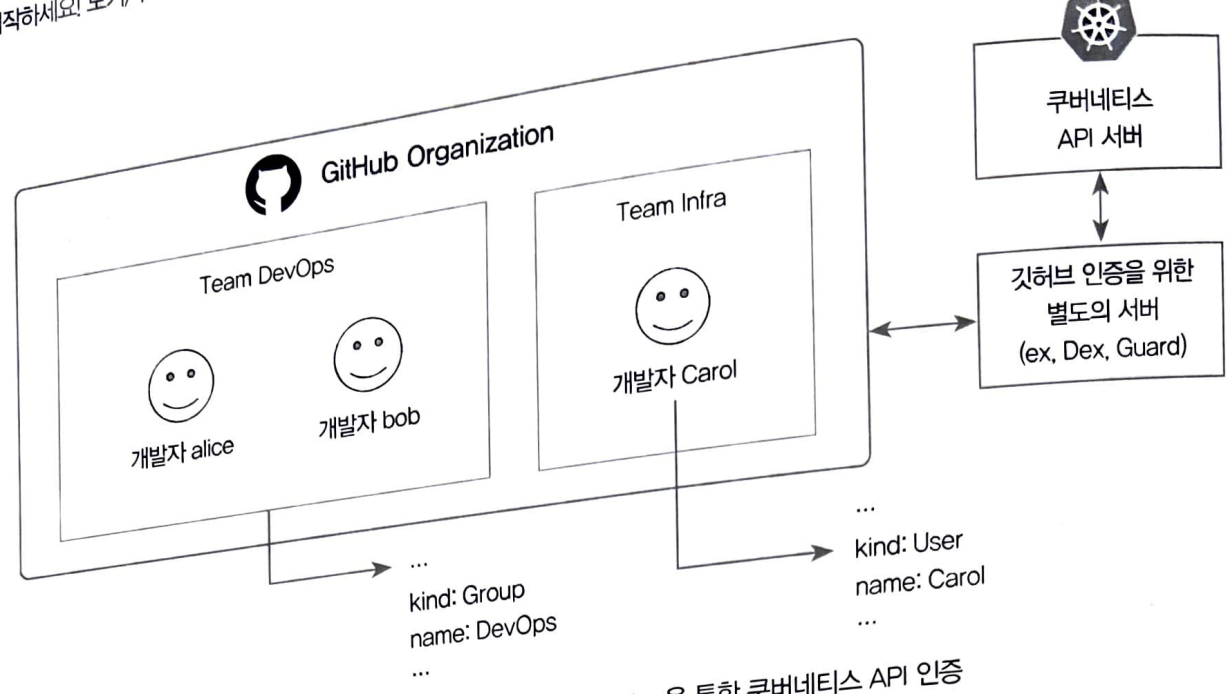

그림 10.8 깃허브의 Team과 Organization을 통한 쿠버네티스 API 인증

위 그림과 같은 상황에서 팀 DevOps의 개발자 전원에게 권한을 부여하는 경우와, 개발자 Carol에게 개인적으로 권한을 부여하는 경우에는 롤 바인딩을 각각 아래와 같이 정의할 수 있습니다.

```
...
subjects:
- kind: Group
  name: DevOps
...
```

```
...
subjects:
- kind: User
  name: Carol
...
```

위 예시에서는 깃허브 조직의 팀을 그룹에, 깃허브 사용자 이름을 유저에 매칭했지만 이는 절대적인 규칙은 아닙니다. 어떤 데이터를 유저, 그룹으로 매칭해 사용할 것인지는 별도로 구축한 인증 서버나 API 서버에서 따로 설정할 수 있기 때문입니다. 예를 들어, 깃허브 사용자 이름이 아닌 깃허브 이메일 이름을 유저로 매칭해 쿠버네티스에서 사용하는 것도 가능합니다.

단, 이러한 데이터는 어떠한 서드 파티로부터 인증 정보를 발급받는지, 어떠한 도구를 별도의 인증 서버로 사용하는지에 따라 조금씩 다를 수 있습니다.

10.7 x509 인증서를 이용한 사용자 인증

쿠버네티스는 보안 연결을 위해 자체적으로 사인(self-signed)한 루트 인증서를 사용합니다. 이 루트 인증서는 쿠버네티스를 설치할 때 자동으로 생성되며, kubeadm의 경우 기본적으로 쿠버네티스 마스터의 /etc/kubernetes/pki 디렉터리에 저장돼 있습니다. kops를 사용하고 있다면 S3 버킷의 ${클러스터 이름}/pki/ 디렉터리에서 확인할 수 있습니다.

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

이 파일 중 ca.crt가 바로 루트 인증서에 해당하며, ca.key는 이 인증서에 대응하는 비밀키입니다. 그 외의 apiserver.crt와 같은 인증서 파일들은 이 루트 인증서로부터 발급된 하위 인증서입니다. 이러한 하위 인증서들은 쿠버네티스 핵심 컴포넌트들이 서로 보안 연결을 수립하는 데 사용됩니다.

쿠버네티스의 루트 인증서로부터 발급된 하위 인증서를 사용하면 쿠버네티스 사용자를 인증할 수 있습니다. 쿠버네티스를 설치하면 기본적으로 설정되는 kubeconfig(~/.kube/config) 파일에 저장돼 있던 인증 정보 또한 x509 인증서를 이용한 인증 방법입니다.

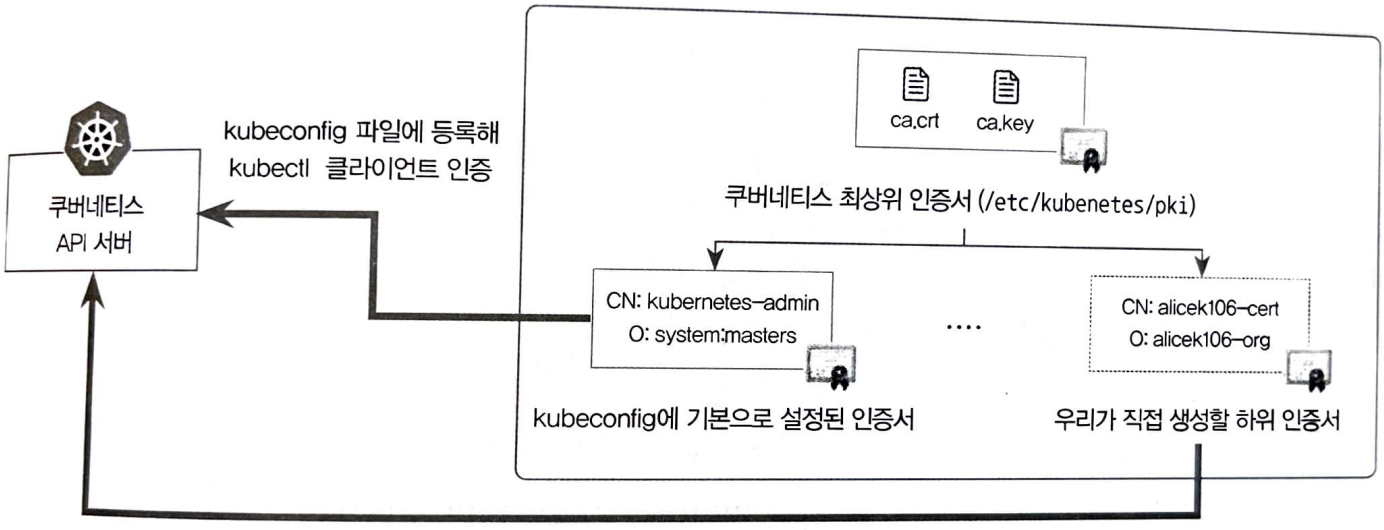

그림 10.9 인증서를 이용한 쿠버네티스 API의 인증 구조

이번에는 kubeconfig에 설정돼 있던 기본 인증서가 아닌, 루트 인증서로부터 하위 인증서를 직접 생성해 API 서버에 인증해 보겠습니다.

가장 먼저 하위 인증서를 위한 비밀키와 인증서 사인 요청 파일(.csr) 파일을 생성합니다.

```
$ openssl genrsa -out alicek106.key 2048
$ openssl req -new -key alicek106.key \
 -out alicek106.csr -subj "/O=alicek106-org/CN=alicek106-cert"
```

위 명령어에서 중요한 부분은 -subj 옵션의 값입니다. 하위 인증서를 사용자 인증에 사용할 때는 **인증서의 CN(Common Name)이 유저(User)로, O(Organization)가 그룹(Group)으로** 취급됩니다. 따라서 위처럼 인증서를 생성하면 롤 바인딩 등에서 alicek106-cert라는 이름의 유저에게 권한을 부여해야 합니다.

kubeconfig 파일에 기본적으로 설정된 인증서는 Organization이 system:masters로 설정돼 있습니다. 쿠버네티스를 설치하면 system:masters 그룹에 cluster-admin 클러스터 롤이 부여되기 때문에 지금까지 아무런 문제 없이 관리자 권한을 사용할 수 있었던 것입니다.

```
$ # kubernetes-admin.crt 파일은 kubeconfig에 기본적으로 설정된 관리자 사용자의
$ # 인증서(client-certificate-data)를 base64로 디코딩한 뒤 저장한 파일입니다.
$ openssl x509 -in kubernetes-admin.crt -noout -text
...
        Issuer: CN=kubernetes
        Validity
            Not Before: Jul 28 12:36:44 2019 GMT
            Not After : Jul 27 12:36:47 2020 GMT
        Subject: O=system:masters, CN=kubernetes-admin
```

system:masters 그룹에 clustet-admin 권한을 부여하는 클러스터 바인딩 롤은 cluster-admin에 설정돼 있습니다.

```
$ kubectl describe clusterrolebinding cluster-admin
...
Subjects:
  Kind   Name            Namespace
  ----   ----            ---------
  Group  system:masters
```

다음은 쿠버네티스의 비밀키로 alicek106.csr 파일에 서명할 차례인데, 이를 위해서 openssl 명령어를 직접 사용해도 됩니다. 하지만 openssl 명령어를 사용하려면 쿠버네티스의 비밀키에 직접 접근해야 하는데, 사실 이는 그다지 바람직한 습관은 아닙니다. 실수로 인증서의 비밀키가 유출되면 쿠버네티스 클러스터의 모든 컴포넌트가 보안에 취약해질 수 있기 때문입니다.

이러한 경우를 위해 쿠버네티스는 인증서 사인 요청(.csr) 파일에 간접적으로 서명하는 기능을 API로 제공합니다. 우선 아래의 내용으로 YAML 파일을 작성해 보겠습니다.

예제 10.11 chapter10/alicek106-csr.yaml

```
apiVersion: certificates.k8s.io/v1beta1
kind: CertificateSigningRequest
metadata:
  name: alicek106-csr
spec:
  groups:
  - system:authenticated
```

```
request: <CSR>
usages:
- digital signature
- key encipherment
- client auth
```

이번에는 **CertificateSigningRequest**라는 처음 보는 오브젝트를 정의했습니다. 이 오브젝트의 spec.request 항목에 .csr 파일의 내용을 base64로 인코딩해 넣은 뒤 CertificateSigningRequest 리소스를 생성하면 쿠버네티스에서 내부적으로 루트 인증서의 비밀키로 서명해 반환합니다. 즉, 간접적으로 쿠버네티스 루트 인증서의 비밀키를 사용할 수 있는 셈입니다.

아래의 명령어를 입력해 alicek106.csr 파일의 내용을 base64로 인코딩한 다음, alicek106-csr.yaml 파일의 〈CSR〉 부분으로 가져옵니다.

```
$ export CSR=$(cat alicek106.csr | base64 | tr -d '\n')
$ sed -i -e "s/<CSR>/$CSR/g" alicek106-csr.yaml
```

작성한 YAML 파일을 이용해 CertificateSigningRequest 리소스를 생성해 보겠습니다.

```
$ kubectl apply -f alicek106-csr.yaml
certificatesigningrequest.certificates.k8s.io/alicek106-csr created

$ kubectl get csr
NAME           AGE   REQUESTOR          CONDITION
alicek106-csr  3s    kubernetes-admin   Pending
```

CertificateSigningRequest의 목록을 출력해 보면 CONDITION 항목이 Pending인 것을 확인할 수 있습니다. 지금은 '쿠버네티스 사용자'의 입장에서 CertificateSigningRequest를 생성함으로써 서명 요청을 제출했으니, 다음 단계는 '쿠버네티스 관리자'의 입장에서 해당 서명 요청을 승인할 차례입니다. kubectl certificate approve 명령어를 사용하면 해당 서명 요청을 승인할 수 있습니다.

```
$ kubectl certificate approve alicek106-csr
certificatesigningrequest.certificates.k8s.io/alicek106-csr approved

$ kubectl get csr
NAME           AGE     REQUESTOR          CONDITION
alicek106-csr  5m48s   kubernetes-admin   Approved,Issued
```

인증서 서명 요청의 상태가 Approved,Issued로 변경됐으며, 정상적으로 하위 인증서가 발급됐습니다. 다음 명령어를 입력해 CertificateSigningRequest 리소스로부터 하위 인증서를 추출합니다.[14]

```
$ kubectl get csr alicek106-csr -o jsonpath='{.status.certificate}' | base64 -d > alicek106.crt
```

이제 x509 인증서로 쿠버네티스에 사용자를 인증하기 위한 준비가 끝났습니다. 새롭게 생성된 하위 인증서 파일인 alicek106.crt와 비밀키 파일인 alicek106.key로 kubeconfig에 새로운 사용자를 등록합니다.

```
$ kubectl config set-credentials alicek106-x509-user \
  --client-certificate=alicek106.crt --client-key=alicek106.key
User "alicek106-x509-user" set.
```

kubectl의 부가 명령어인 --client-certificate와 --client-key 옵션을 사용하면 kubeconfig에 하위 인증서와 비밀키를 등록하지 않아도 임시로 인증을 테스트할 수 있습니다.

```
$ kubectl get svc --client-certificate alicek106.crt --client-key alicek106.key
Error from server (Forbidden): services is forbidden: User "alicek106-cert" cannot list
resource "services" in API group "" in the namespace "default"
```

새롭게 등록한 사용자를 통해 새 컨텍스트도 함께 생성합니다. --cluster 옵션에는 현재 사용하고 있는 클러스터의 이름을 적절히 입력합니다.

```
$ kubectl config get-clusters
NAME
kubernetes

$ kubectl config set-context alicek106-x509-context \
  --cluster kubernetes --user alicek106-x509-user
Context "alicek106-x509-context" created.
```

컨텍스트를 변경한 다음 kubectl로 API를 요청해보면 권한이 없다는 에러가 출력될 것입니다. 이는 인증이 정상적으로 이뤄졌으니 롤이나 클러스터 롤을 통해 권한을 부여하면 된다는 의미와 같습니다.[15]

```
$ kubectl config use-context alicek106-x509-context
Switched to context "alicek106-x509-context".
```

---

14 맥 OS X에서는 base64 명령어에서 -D 옵션을 사용합니다.

15 error: You must be logged in to the server (Unauthorized) 에러가 출력됐다면 인증 자체가 실패했음을 뜻하므로 처음부터 다시 차근차근 시도해보는 것이 좋습니다

```
$ kubectl get svc
Error from server (Forbidden): services is forbidden: User "alicek106-cert" cannot list resource
"services" in API group "" in the namespace "default"
```

지금까지 문제없이 잘 따라 했다면 에러 메시지에서 User "alicek106-cert"라고 출력된 이유를 눈치챌 수 있을 것입니다. 이전에 하위 인증서를 생성하기 위한 비밀키와 서명 요청 파일(alicek106.csr)을 생성할 때 CN(Common Name)을 alicek106-cert로, O(Organization)를 alicek106-org로 설정했다는 점을 되생각해 보겠습니다. 앞서 설명했던 것처럼 x509 인증서를 이용한 쿠버네티스 인증에서는 인증서의 CN이 유저(User)로, O가 그룹(Group)으로 매칭됩니다. 따라서 이 하위 인증서에 권한을 부여하려면 alicek106-cert라는 유저나 alicek106-org라는 그룹에 롤 및 클러스터 롤을 할당하면 됩니다.[16]

예제 10.12 chapter10/x509-cert-rolebinding-user.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: service-reader-rolebinding-user
  namespace: default
subjects:
- kind: User
  name: alicek106-cert
roleRef:
  kind: Role
  name: service-reader
  apiGroup: rbac.authorization.k8s.io
```

위의 YAML 파일로 롤 바인딩을 생성함으로써 하위 인증서에 권한을 부여해 보겠습니다. 그러나 지금은 아무런 권한이 없는 사용자의 컨텍스트(alicek106-x509-context)로 스위칭된 상태이기 때문에 --context 옵션으로 이전에 사용하던 관리자 권한의 컨텍스트를 임시로 사용하겠습니다.

```
$ kubectl config get-contexts
CURRENT   NAME                          CLUSTER      AUTHINFO
*         alicek106-x509-context        kubernetes   alicek106-x509-user
          kubernetes-admin@kubernetes   kubernetes   kubernetes-admin

$ kubectl apply -f x509-cert-rolebinding-user.yaml --context kubernetes-admin@kubernetes
rolebinding.rbac.authorization.k8s.io/service-reader-rolebinding-user created
```

16 이전에 사용했던 service-reader라는 이름의 롤이 이미 존재한다고 가정합니다.

alicek106-cert라는 유저를 위한 롤 바인딩을 생성했기 때문에 alicek106-cert를 CN으로 갖는 하위 인증서로 서비스의 목록을 정상적으로 출력할 수 있습니다.

```
$ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   23h
```

하지만 x509 인증서를 이용한 인증 방법은 몇 가지 한계점이 있어서 실제 환경에서는 사용하기 어려울 수 있습니다. 쿠버네티스에서 인증서가 유출됐을 때 하위 인증서를 파기(revoke)하는 기능이 제공되고 있지 않을뿐더러, 파일로 인증 정보를 관리하는 것은 보안상 바람직하지 않기 때문입니다.

따라서 이러한 인증서 사용법은 개념적으로만 알고 넘어가되, 실제로 클러스터를 운영할 때는 덱스(Dex)나 가드(Guard) 등의 솔루션을 이용해 깃허브, LDAP와 같은 서드 파티에서 인증 정보를 관리하는 것이 더욱 효율적일 수도 있다는 사실을 알아두기 바랍니다.