Section 7: Security

164. Certificates API

인증서 API 동작 과정

1. 인증서 서명 요청을 받을 때 마다 CertificateSigningRequest Object라는 쿠버네티스 API 객체를 만든다.
2. 모든 인증서 서명 요청은 클러스터 관리자들이 보게 된다.
3. 이 요청들은 kubectl 명령어를 통해 쉽게 검토하고 승인할 수 있다. 또한, 이 인증서는 추출되어 사용자와 공유될 수 있다.

인증서 생성 과정

- 개인 키 생성: `openssl genrsa -out jane.key 2048`
- CSR 생성: `openssl req -new -key jane.key -subj "/CN=jane" -out jane.csr`
- CSR을 base64로 인코딩: `cat jane.csr | base64`
  - 한 줄로 출력 옵션 주기 : `-w 0`
- CertificateSigningRequest 객체 생성(yaml)
- CSR 생성 확인: `k get csr`
- 요청 승인: `kubectl certificate approve jane`
- 요청 거부: `kubectl certificate deny jane`
  - 요청 거부 후 CSR object 제거 (`k delete csr jane`

167. KubeConfig

KubeConfig 기본 구조

- kubectl 명령어 인증 정보 저장(KubeConfig 파일을 사용할지 지정하지 않으면 다음 경로의 파일을 기본으로 사용: `$HOME/.kube/config`)
- **Clusters**: 연결할 클러스터 정보
- **Users**: 클러스터에 access 권한이 있는 사용자 계정
- **Contexts**: 어떤 User 계정이 어떤 클러스터에 접근하기 위해 사용될지 정의

KubeConfig yaml

```
apiVersion: v1
kind: Config
current-context: my-kube-admin@my-kube-playground
clusters:
- name: my-kube-playground
  cluster:
```

```
    certificate-authority: ca.crt
    server: https://my-kube-playground:6443
contexts:
- name: my-kube-admin@my-kube-playground
  context:
    cluster: my-kube-playground
    user: my-kube-admin
    namespace: finance  # optional
users:
- name: my-kube-admin
  user:
    client-certificate: admin.crt
    client-key: admin.key
```

KubeConfig 명령어

- 현재 적용된 config 확인: `kubectl config view`
- 특정 config 파일 확인: `kubectl config view --kubeconfig=my-custom-config`
- current-context 변경: `kubectl config --kubeconfig=my-custom-config use-context prod-user@production`
- 현재 적용된 context 확인: yaml 파일의 `current-context`
- 도움말: `k config --help`

Default KubeConfig file 변경

- 매번 `--kubeconfig=my-kube-config` 옵션 붙이고 싶지 않을 때, 두 가지 방법
  - 기본 경로에 있는 파일을 `my-kube-config` 파일로 대체
    - `mv /root/my-kube-config /root/.kube/config`
  - `my-kube-config` 파일을 KUBECONFIG 환경변수에 추가한다. - **기존 파일을 유지하는 방법**
    - `vi ~/.bashrc` : shell configuration file 을 연다.
    - `export KUBECONFIG=/root/my-kube-config` : 변수를 export
    - `source ~/.bashrc` : 현재 세션에 변경사항을 적용하기 위해, shell configuration을 reload한다

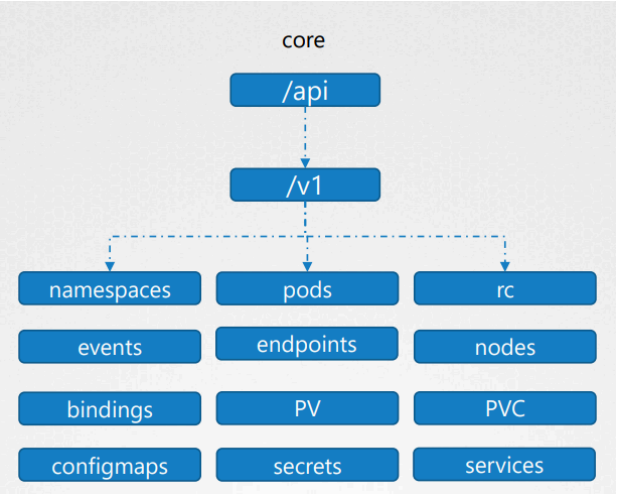
171. API Groups

API Server 접근

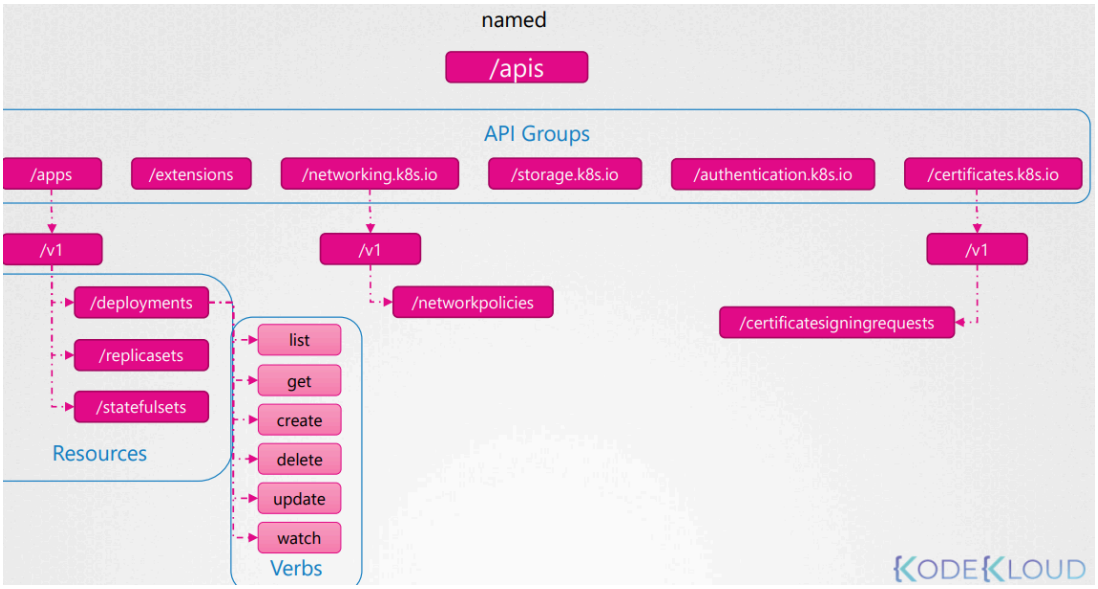
- Master Node Adress로 직접 접근(인증서 파일 필요): `curl https://kube-master:6443/api/v1/pods --key admin.key --cert admin.crt --cacert ca.crt`
  - 기본 port = 6443
- kubectl proxy를 통한 접근:
  1. `kubectl proxy` 실행
  2. `curl http://localhost:8001 -k`

API 구조

- **/api**: Core Gruop(v1)
  - pods, services, nodes, configmaps 등 기본 resources



- **/apis**: Named Group
  - /apps, /extensions, /networking.k8s.io 등 resources
  - /v1 - deployments, replicasets, statefulsets 등



172. Authorization

- ABAC(Attribute-Based Access Control): 정책 기반으로 사용자/그룹에 권한 부여 (관리 어렵)
- RBAC(Role-Based Access Control): 역할 기반으로 권한 부여
- Node: 노드 간 통신을 위한 인증
- Webhook: 외부 서비스를 통한 권한 검증(예: Open Policy Agent)

- **Authorization Mode**:
  - AlwaysAllow: 모든 요청 허용(기본값)
  - AlwaysDeny: 모든 요청 거부
  - Node, RBAC, Webhook 등을 조합하여 사용 가능
  - kube-apiserver 설정에서 `-authorization-mode` 로 지정, 쉼표로 구분, 순서대로 권한 부여 (앞 순서의 모듈이 요청 거부 시 다음 순서의 모듈로 계속 넘어가서 승인)
    - ex) `--authorization-mode=Node,RBAC,Webhook`

173. RBAC (Role Based Access Controls)

- **Role**: 특정 namespace 내에서 리소스에 대한 권한 정의
  - apiVersion: rbac.authorization.k8s.io/v1
  - kind: Role
  - rules
  - apiGroups: API 그룹 지정
    - deployment resource - `- apiGroups: ["apps"]`
  - resources: 접근할 리소스(pods, configmaps 등)
  - verbs: 수행 가능한 작업(get, list, create, update, delete 등)
  - resourceNames: 특정 namespace, 특정 pods에만 권한 제한 가능 ex) [”blue”, “orange”]
- **RoleBinding**: 사용자/그룹을 앞서 만든 Role에 연결
  - apiVersion: rbac.authorization.k8s.io/v1
  - kind: RoleBinding
  - subjects: 권한을 부여할 사용자/그룹에 대한 세부 정보 지정
  - roleRef: Role의 세부사항 제공

- `kubectl get roles` : Role 생성 확인
- `kubectl get rolebindings` : RoleBindings 생성 확인

- **클러스터의 특정 리소스에 대한 사용자의 권한 확인 명령어**:

```
kubectl auth can-i <verb> <resource>
kubectl auth can-i <verb> <resource> --as <user>   # 특정 사용자에 대한 권한 확인
```

176. Cluster Roles and Role Bindings

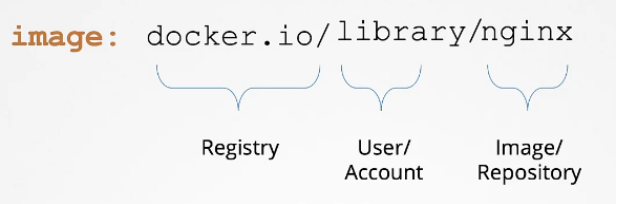
- **Namespaced vs Cluster Scoped Resource**:

    - **Namespaced**: 지정하는 네임스페이스에서 생성되는 리소스
        - pods, deployments, services, configmaps, secrets 등
    - **Cluster Scoped**: 생성 시 네임스페이스 지정 X
        - **nodes**, PV, PVC, namespaces, clusterroles, clusterrolebindings 등
- **ClusterRole**
    - 클러스터 범위에서 리소스에 대한 권한 부여
    - 여러 namespace에 걸친 리소스 권한 관리
- **ClusterRoleBinding**: 사용자/그룹을 ClusterRole에 연결
    - 클러스터 전체 범위의 권한 부여

179. Service Accounts

- Service Accounts : 쿠버네티스와 상호작용하기 위해 앱이 사용하는 계정
    - ex) 프로메테우스와 같은 모니터링 Application
- `k create serviceaccount <계정이름>`
- `k get serviceaccount`
- 서비스 계정이 생성되면 자동으로 토큰도 생성된다. 토큰은 secret object로 저장됨
- 각각의 namespace는 default service account를 가지고 있다.
- 쿠버네티스는 기본적으로 pod 생성 시 default service account와 token을 자동으로 마운트한다.
    - 버전 1.24 - service account 생성 시 자동으로 secret이나 token을 생성하지 않음. 따라서, 토큰을 생성하려면 `kubectl create token dashboard-sa` 명령어 실행
        - 토큰을 얻기 위한 API를 사용할수 없을 때에만 service account token secret을 만들어야 한다.
- 기존 pod의 service account는 수정할 수 없다. 따라서, 삭제 후 재생성해야함.
    - 단, deployment의 경우 service account 수정 가능 (pod 정의 파일이 변하면 deployment를 위한 새 roll out이 자동으로 트리거되기 때문)

182. Image Security



- **이미지 참조 형식**: `{registry}/{user/account}/{image/repository}` ex) `docker.io/library/nginx`
    - registry: 이미지 저장소(docker.io, gcr.io 등)
    - user: user 혹은 account

    - image: image 혹은 repository
- **Private Repository 접근**:
    1. Docker 로그인: `docker login private-registry.io`
    2. Docker 시크릿 생성:

```
kubectl create secret docker-registry regcred \
--docker-server=private-registry.io \   # 레지스트리 서버 이름 지정
--docker-username=registry-user \       # 레지스트리에 엑세스할 사용자 이름
--docker-password=registry-password \   # 암호
--docker-email=registry-user@org.com    # 사용자의 이메일 주소
```

    3. Pod 정의 yaml에 imagePullSecrets 지정:

```
spec:
  containers:
  - name: ngins
    image: ~
  imagePullSecrets:
  - name: regcred
```

185. Pre-requisite - Security in Docker

- 컨테이너 안의 루트 사용자 ≠ 호스트의 루트 사용자
- 기본적으로 docker는 unprevileged로 컨테이너를 실행함
- `docker run --cap-add MAC_ADMIN ubuntu` : `--cap-add` flag로 특정 권한을 부여 가능
- `docker run --cap-drop KILL ubuntu` : KILL 권한을 제거
- `docker run --privileged ubuntu` : 모든 권한이 활성화된 컨테이너 실행

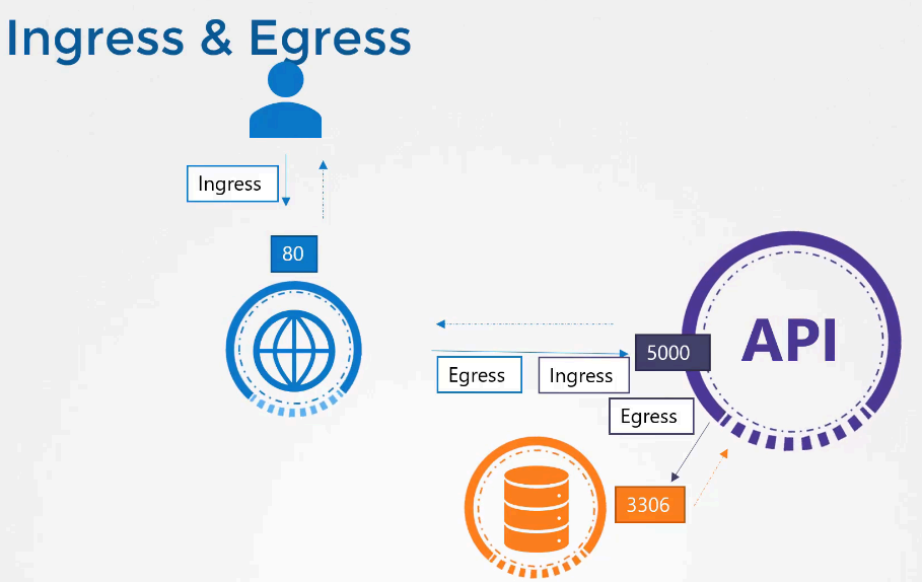
186. Security Contexts

- container level이나 pod level에서 security context 구성 가능
    - **둘 다 구성하면 컨테이너의 설정이 파드의 설정을 덮어씀**

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    supplementalGroups: [4000]
```

```
  containers:
  - name: sec-ctx-demo
    image: registry.k8s.io/e2e-test-images/agnhost:2.45
    command: [ "sh", "-c", "sleep 1h" ]
    securityContext:
      runAsUser: 1000
      capabilities:
        add: ["MAC_ADMIN"]
```

189. Network Policy

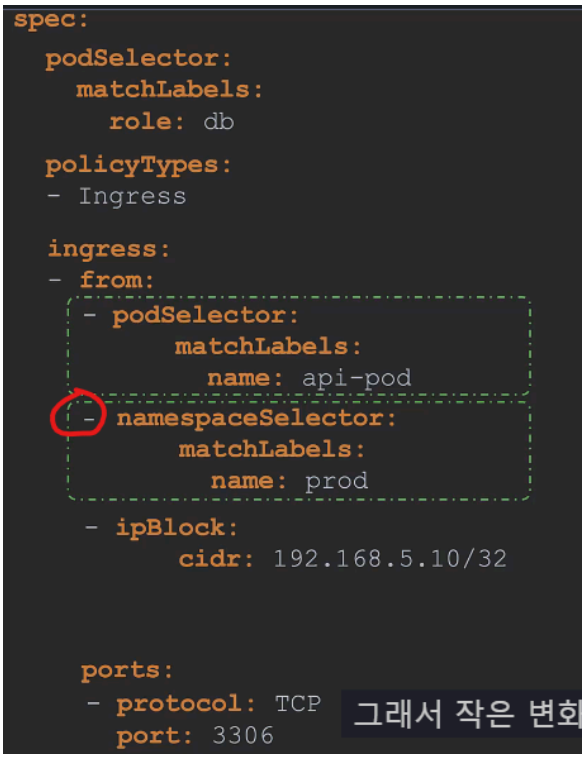


- 웹서버 기준으로 하면 사용자로부터 오는 트래픽은 Ingress, 앱 서버로 가는 요청은 Egress
- 모든 CNI(Container Network Interface) 플러그인이 NetworkPolicy를 지원하는 것은 아님
  - 지원하는 네트워크 솔루션: Kube-router, Calico, Romana 등
  - 지원 X : Flannel

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: db-policy
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  ingress:
  - from:
```

```
  - podSelector:
      matchLabels:
        name: api-pod
    ports:
    - protocol: TCP
      port: 3306
```

190. Developing network policies



- 표시한 부분 `-` 가 없으면 and 연산처럼 작용, 있으면 or

192. Kubectx and Kubens - Command line Utilities

- **Kubectx**
  - 컨텍스트 간 전환을 위해 긴 "kubectl config" 명령을 사용할 필요가 없음
  - 다중 클러스터 환경에서 클러스터 간 컨텍스트를 전환하는 데 특히 유용
  - `kubectx` : 모든 context 조회
  - `kubectx <context_name>` : 새로운 context로 전환
  - `kubectx -` : 이전 context로 돌아가기
  - `kubectx -c` : 현재 context 확인
- Kubens
  - namespace 간 전환 빠르게 가능

- `kubens <new_namespace>` : 새 namespace로 전환
- `kubens -` : 이전 namespace로 돌아가기

194. (2025 Updates) Custom Resource Definition (CRD)

- 사용자 정의 resource 유형
- `kubectl` 명령어로 관리 가능
- 사용자 지정 리소스 정의 (CRD)
  - `apiVersion`: `apiextensions.k8s.io/v1` 사용
  - `kind`: `CustomResourceDefinition`
  - `spec.scope`: 리소스 범위 (Namespaced or Cluster)
  - `spec.names.singular/plural/shortnames` : 단수, 복수, 약어 설정 가능

195. (2025 Updates) Custom Controllers

- CRD 생성만으로는 ETCD에 저장만 되고, controller가 없기 때문에 리소스에 대한 처리 불가
- Custom Controller는 사용자 정의 리소스의 상태를 관리
- custom controller 구축에 대한 시험 문제는 출제되지 않을 것으로 예상, CRD 작성하거나 만들어져있는 controller로 작업하는 정도의 문제는 나올 수 있겠다고함

196. (2025 Updates) Operator Framework

- CRD와 Custom Controller를 패키지화하여 단일 엔티티로 배포할 수 있음
- 따라서 operator를 배포하면, 내부적으로 CRD와 CUstom Controllers를 deployment로 배포
- 시험엔 출제되지 않을 것으로 예상