올인원 패키지 Online.

CKA 쿠버네티스 자격증 과정

PART1 | CKA 자격증 소개
자격증 등록과 Hands-On 실습 환경 만들기

PART2 | Kubernetes 아키텍쳐
Etcd 백업/복구, Kubernetes 업그레이드, RBAC 인증 실습

PART3 | Workloads & Scheduling
Application deploy 후 scale/rollback, pod 스케줄링 실습

PART4 | Services & Networking
Service 운영과 접근제한(NetworkPolicy),Ingress 운영 실습

PART5 | Storage
StorageClass 적용한 PV, PVC 생성하여 pod에 적용하기 실습

PART6 | Troubleshooting
controller component 설정 정보 수정 및 node/pod 문제해결

PART7 | 실전문제풀이
실전 문제를 직접 풀어 보고, 답안 리뷰 확인

Part 4 Services & Networking

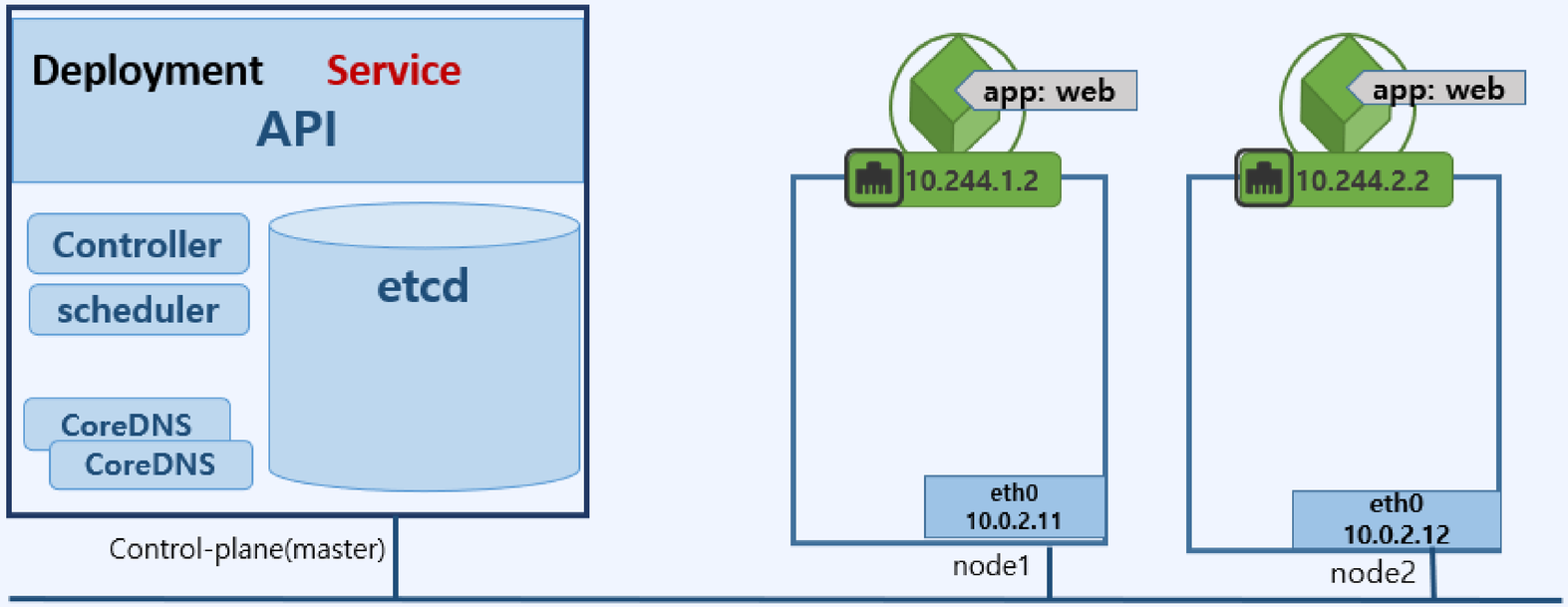
01 Service 동작원리
02 Service Type: ClusterIP
03 Service Type: NodePort
04 Network Policy
05 Ingress
06 kube-dns

Part 4 Services & Networking

01 Service 동작 원리

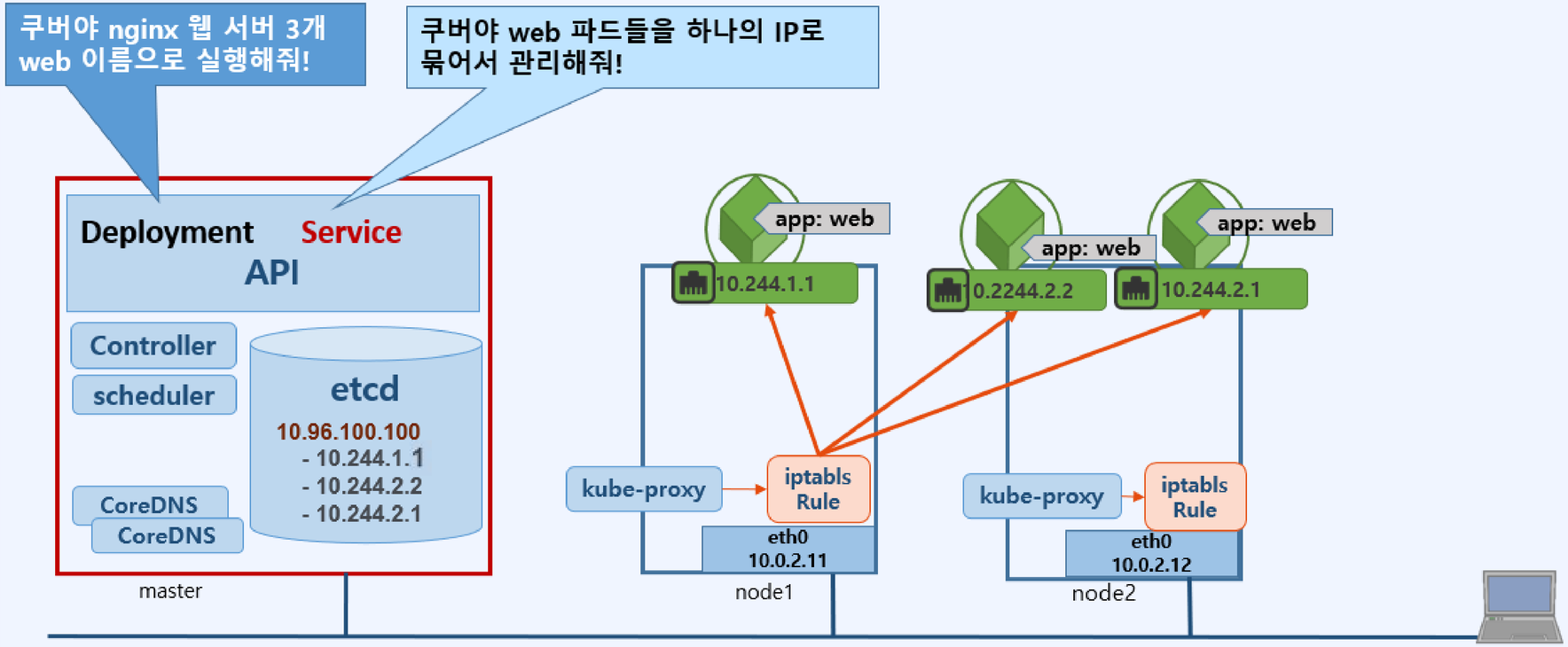
Kubernetes Service 구현

- Pod network
  - CNI(Container Network Interface plugin)에서 관리하는 포드 간 통신에 사용되는 클러스터 전체 네트워크
- Service Network
  - Service discovery를 위해 kube-proxy 가 관리하는 Cluster-wide 범위의 Virtual IP



Kube-proxy

- Kubernetes Network Proxy
- 각각의 Node에서 실행되고, Kubernetes Service API에 정의된 서비스를 각 노드에서 반영
- kube-proxy 역할
  - Iptables rule을 설정하고 외부 네트워크와 Pod를 연결시킨다.



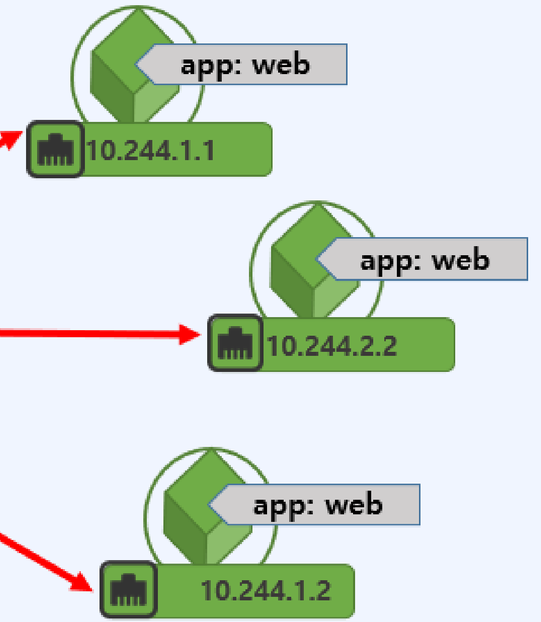
Kubernetes Service Example

Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      name: nginx-pod
      labels:
        app: web
    spec:
      containers:
      - name: nginx-container
        image: nginx:1.14
```

Service

```
apiVersion: v1
kind: Service
metadata:
  name: web-svc
spec:
  clusterIP: 10.96.100.100
  selector:
    app: web
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

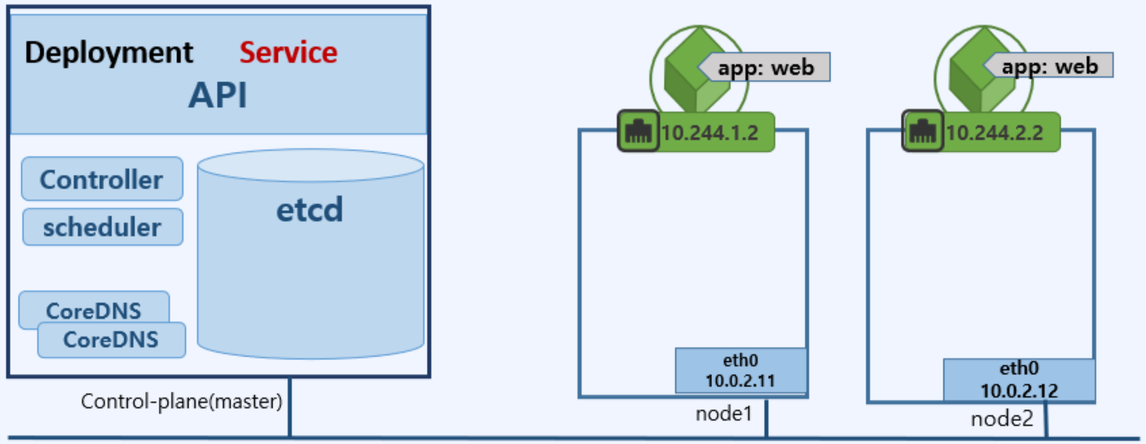


Service Type

ClusterIP(default) : Pod 그룹(동일한 서비스를 지원하는 Pod 모음)의 단일 진입점 (Virtual IP:LB) 생성

NodePort : ClusterIP 가 생성된 후 모든 Worker Node 에 외부에서 접속 가능 한 포트가 예약

LoadBalancer : 클라우드 인프라스트럭처 (AWS, Azure, GCP)에 적용
LoadBalancer를 자동으로 프로 비전하는 기능 지원



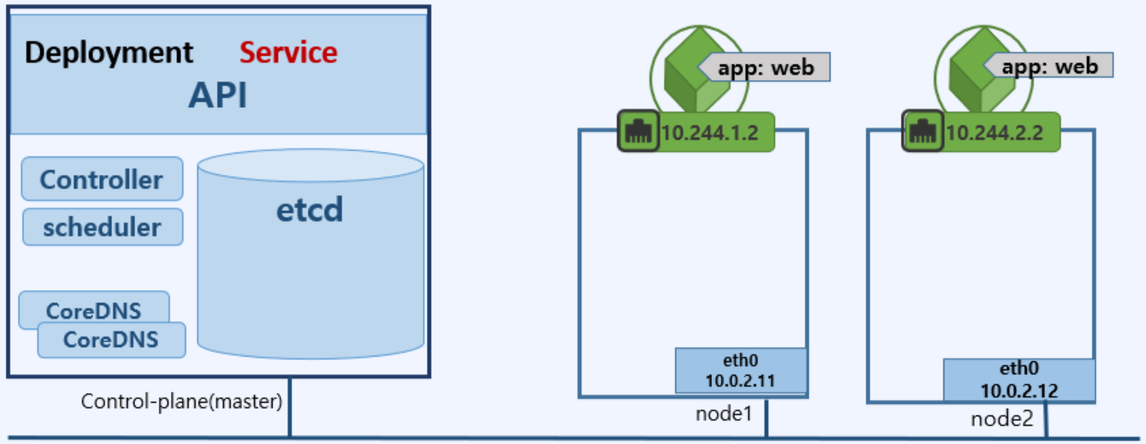
Part 4 Services & Networking

02 Service Type: ClusterIP

ClusterIP(default) : Pod 그룹(동일한 서비스를 지원하는 Pod 모음)의 단일 진입점 (Virtual IP:LB) 생성

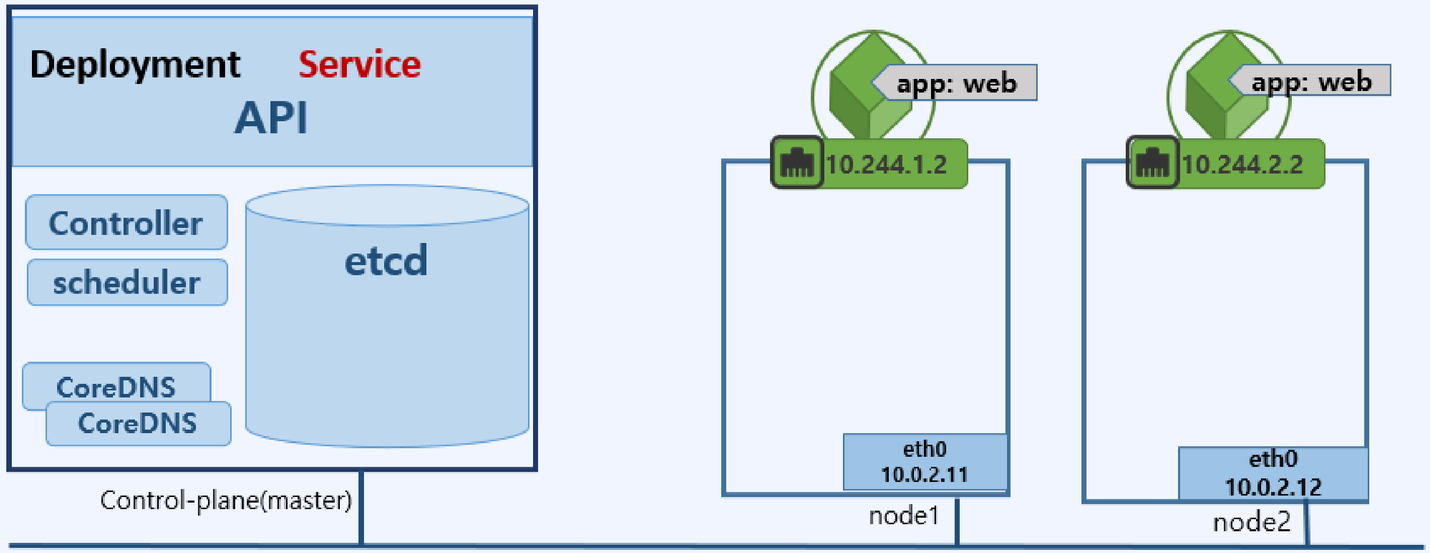
NodePort : ClusterIP 가 생성된 후 모든 Worker Node 에 외부에서 접속 가능 한 포트가 예약

LoadBalancer : 클라우드 인프라스트럭처 (AWS, Azure, GCP)에 적용
LoadBalancer를 자동으로 프로 비전하는 기능 지원



ClusterIP

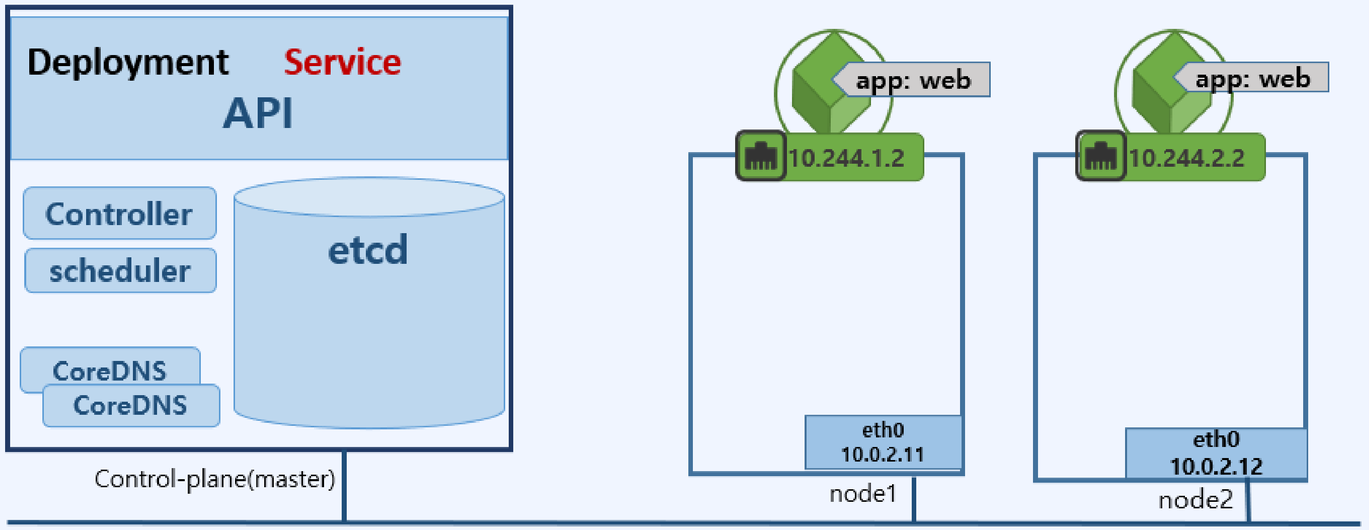
- selector의 label이 동일한 Pod들을 그룹으로 묶어 단일 진입점 (Virtual_IP)을 생성
- 클러스터 내부에서만 사용가능
- Service type 생략 시 default 로 설정
- 10.96.0.0/12 범위에서 할당됨



ClusterIP 서비스 운영

- LAB: 동일한 서비스를 제공하는 Pod 그룹에 ClusterIP 생성하기
  deployment name: web , image:nginx, port:80, replicas:2
  service name: web, type: clusterIP, port: 80

$ kubectl create deployment web --image=nginx --port=80 --replicas=2
$ kubectl expose deployment webserver type=ClusterIP port=80



문제1:
ClusterIP type의 서비스 운영

■ 작업 클러스터 : **k8s**

'devops' namespace에서 운영되고 있는 eshop-order deploymen의 Service를 만드세요.

- Service Name: eshop-order-svc
- Type: ClusterIP
- Port: 80

```
$ kubectl get deployments.apps -n devops

$ kubectl expose deployment eshop-order  --namespace devops --type=ClusterIP --port=80 --target-port=80 --name=eshop-order-svc --dry-run=client -o yaml
$ kubectl expose deployment eshop-order  --namespace devops --type=ClusterIP --port=80 --target-port=80 --name=eshop-order-svc

$ kubectl get svc -n devops eshop-order-svc

$ ssh k8s-worker1
worker1 ~]$ curl 10.XX.XX.XX
worker1 ~]$ exit
```

문제2:
Pod를 이용한 Named Service 구성

02.
Service Type:
ClusterIP

■ 작업 클러스터 : **k8s**

- 미리 배포한 'front-end'에 기존의 nginx 컨테이너의 포트 '80/tcp'를 expose하는 'http'라는 이름을 추가합니다.
- 컨테이너 포트 http를 expose하는 'front-end-svc'라는 새 service를 만듭니다.
- 또한 준비된 node의 'NodePort'를 통해 개별 Pods를 expose되도록 Service를 구성합니다.

```
$ kubectl get deployments.apps front-end
$ kubectl get deployments.apps front-end -o yaml > front-end.yaml

$ vi front-end.yaml
...


$ kubectl apply -f front-end.yaml
$ kubectl get deployments.apps,svc

$ curl k8s-worker1:3XXXX
```

```
apiVersion: apps/v1
kind: Deployment
...
template:
   metadata:
    labels:
      run: nginx
   spec:
    containers:
    - image: nginx
      name: http
      ports:
      - containerPort: 80
        name: http-web-svc
```

```
apiVersion: v1
kind: Service
metadata:
  name: front-end-svc
spec:
  type: NodePort
  selector:
    run: nginx
  ports:
  - name: name-of-service-port
    protocol: TCP
    port: 80
    targetPort: http-web-svc
```

Part 4 Services & Networking

03 Service Type: NodePort

ClusterIP(default) : Pod 그룹(동일한 서비스를 지원하는 Pod 모음)의 단일 진입점 (Virtual IP:LB) 생성

NodePort : ClusterIP 가 생성된 후 모든 Worker Node 에 외부에서 접속 가능 한 포트가 예약

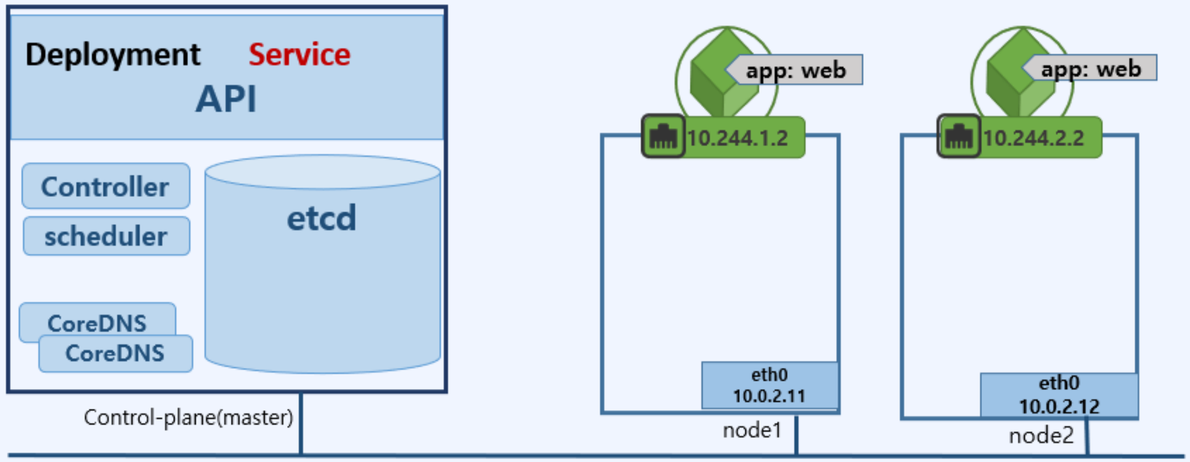
LoadBalancer : 클라우드 인프라스트럭처 (AWS, Azure, GCP)에 적용
LoadBalancer를 자동으로 프로 비전하는 기능 지원

NodePort

- 모든 노드를 대상으로 외부 접속 가능한 포트를 예약
- Default NodePort 범위: 30000-32767
- ClusterIP 를 생성 후 NodePort를 예약

NodePort Service

```
apiVersion: v1
kind: Service
metadata:
  name: nodeport-service
spec:
  type: NodePort
  clusterIP: 10.100.100.200
  selector:
    app: webui
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
    nodePort: 30200
```



03.
Service Type:
NodePort

문제3:
Access the Service from outside the Cluster
via NodePort

■ 작업 클러스터 : **k8s**

- 'front-end' deployment의 nginx 컨테이너를 expose하는 'front-end-nodesvc'라는 새 service를 만듭니다.
- Front-end로 동작중인 Pod에는 node의 30200 포트로 접속되어야 합니다.
- 구성 테스트 curl k8s-worker1:30200 연결 시 nginx 홈페이지가 표시되어야 합니다.

```
$ kubectl config use-context k8s
$ kubectl get deployments.apps front-end
$ kubectl describe deployments.apps front-end | grep -i -e port -e labels

$ kubectl expose deployment front-end --name=front-end-nodesvc --port=80 --
target-port=80  --type=NodePort  --dry-run=client -o yaml
$ kubectl expose deployment front-end --name=front-end-nodesvc --port=80 --
target-port=80  --type=NodePort  --dry-run=client -o yaml > front-end-nodsvc.yaml

$ vi front-end-nodsvc.yaml

$ kubectl apply -f front-end-nodsvc.yaml
$ kubectl get svc

$ curl k8s-worker1:30200
```

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  name: front-end-nodesvc
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
    nodePort: 30200
  selector:
    run: nginx
  type: NodePort
```

Part 4 Services & Networking

04 Network Policy

- Kubernetes가 지원하는 Pod 통신 접근제한.
- 일종의 방화벽으로 Pod로 트래픽이 들어오고(Inbound), 나가는(Outbound)것을 설정하는 정책

- Ingress 트래픽: Inbound 정책. 들어오는 트래픽을 허용할 것 인지를 정의
- Egress 트래픽: Outbound 정책. 트래픽이 나갈 수 있는 허용 범위 정의

- 트래픽 컨트롤 정의
  - ipBlock : CIDR IP 대역으로, 특정 IP 대역에서만 트래픽이 들어오도록 지정할 수 있다.
  - podSelector : label을 이용하여, 특정 label을 가지고 있는 Pod들에서 들어오는 트래픽만 받을 수 있다. 예를 들어 DB Pod의 경우에는 API server로 부터 들어오는 트래픽만 받는 것과 같은 정책 정의가 가능하다.
  - namespaceSelector : 특정 namespace로 부터 들어오는 트래픽 만을 받는 기능이다. 운영 로깅 서버의 경우에는 운영 환경 namespace에서만 들어오는 트래픽을 받거나, 특정 서비스 컴포넌트의 namespace에서의 트래픽만 들어오게 컨트롤이 가능하다. 내부적으로 새로운 서비스 컴포넌트를 오픈했을 때, 베타 서비스를 위해서 특정 서비스나 팀에게만 서비스를 오픈하고자 할 때 유용하게 사용할 수 있다.
  - Protocol & Port : 특정 Protocol 또는 Port로 설정된 트래픽만 허용되는 포트를 정의할 수 있다.

- 적용 예 :
  https://kubernetes.io/docs/concepts/services-networking/network-policies/#networkpolicy-resource

Network Policy:
LAB

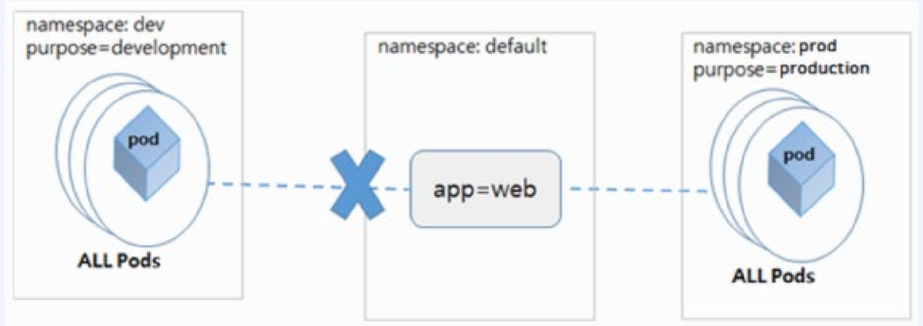
- app: web 레이블을 가진 Pod에 특정 namespace의 Pod들만 접근 허용

```
$ kubectl config use-context hk8s
$ kubectl run webpod --image=nginx --port=80 --labels=app=web
$ kubectl get pods webpod -o wide

$ kubectl create namespace dev
$ kubectl create namespace prod
$ kubectl label namespace prod purpose=production
$ kubectl label namespace dev purpose=development
$ kubectl get namespaces  --show-labels

$ vi web-allow-prod.yaml
$ kubectl apply -f web-allow-prod.yaml
$ kubectl get networkpolicies.networking.k8s.io
$ kubectl run test --namespace=dev --rm -it --image=alpine -- /bin/sh
/ # wget -qO-  --timeout=2 http://web.default.svc.cluster.local
$ kubectl run test --namespace=prod --rm -it --image=alpine -- /bin/sh
/ # wget -qO-  --timeout=2 http://web.default.svc.cluster.local

$ kubectl delete networkpolicy web-allow-prod
$ kubectl delete pod webpod
$ kubectl delete namespace {prod,dev}
```




```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: web-allow-prod
spec:
  podSelector:
    matchLabels:
      app: web
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          purpose: production
```

문제4:
NetworkPolicy

■ 작업 클러스터 : **hk8s**

- default namespace에 다음과 같은 pod를 생성하세요.
  - name: poc
  - image: nginx
  - port: 80
  - label: app=poc
- "partition=customera"를 사용하는 namespace에서만 poc의 80포트로 연결할 수 있도록 default namespace에 'allow-web-from-customera'라는 network Policy를 설정하세요. 보안 정책상 다른 namespace의 접근은 제한합니다.

```
$ kubectl config use-context hk8s
$ kubectl get namespaces -L partition
$ kubectl run poc --image=nginx --port=80 --labels=app=poc
$ kubectl get pod poc -o wide

$ vi allow-web-from-customera.yaml
$ kubectl apply -f allow-web-from-customera.yaml
$ kubectl run testpod -n customerb --image=centos:7 -it --rm -- /bin/bash
/]# curl 192.168.75.100 --실패
/]# exit
$ kubectl run testpod -n customera --image=centos:7 -it --rm -- /bin/bash
/]# curl 192.168.75.100
/]# exit
```

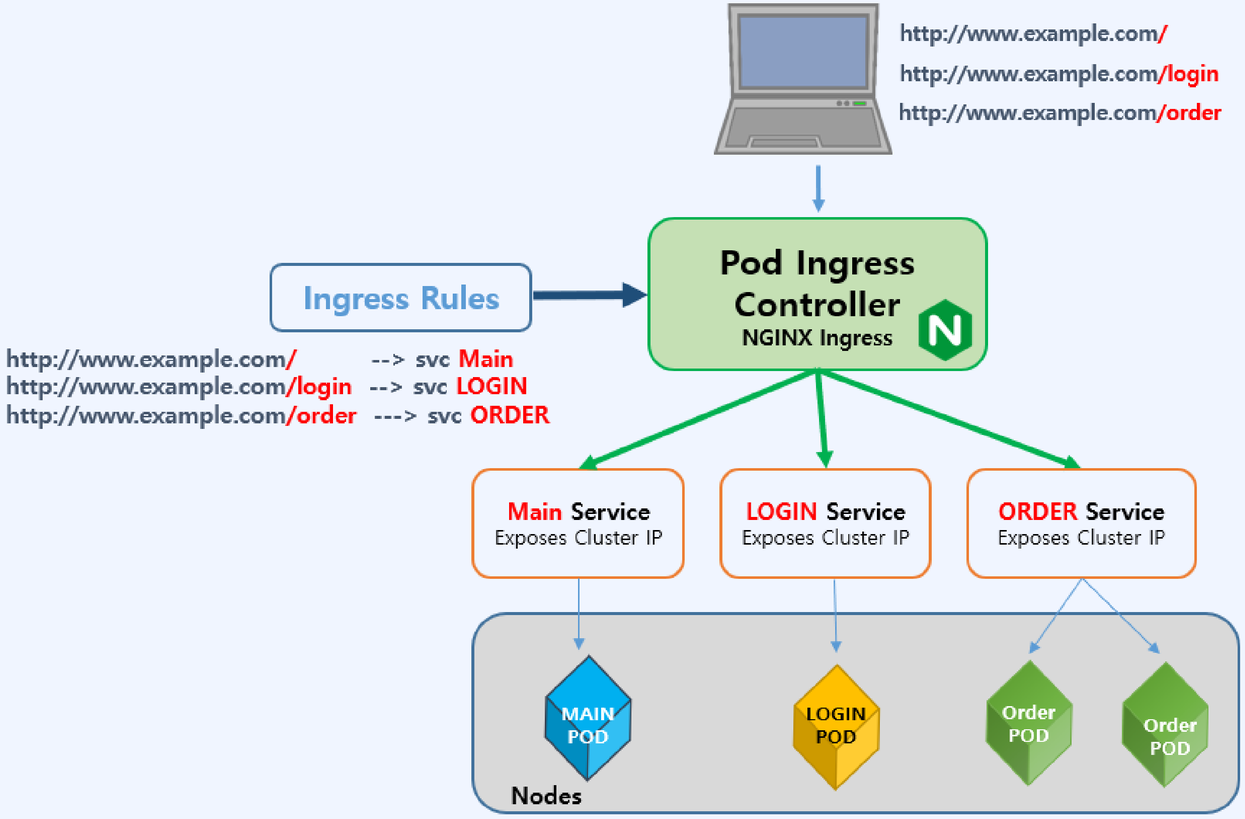
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-web-from-customera
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: poc
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          partition: customera
    ports:
    - protocol: TCP
      port: 80
```

Part 4 Services & Networking

05 Ingress

Kubernetes Ingress

- L7 스위치 역할을 논리적으로 수행
- 클러스터로 접근하는 URL 별로 다른 서비스에 트래픽을 분산
- Kubernetes Ingress의 기능
  - Service에 외부 URL을 제공
  - 트래픽을 로드밸런싱
  - SSL 인증서 처리
  - Virtual hosting을 지정



문제5:
Ingress 구성

■ 작업 클러스터 : **k8s**

- ingress-nginx namespace에 nginx이미지를 app=nginx 레이블을 가지고 실행하는 nginx pod를 구성하세요.
- 현재 appjs-servic와 nginx 서비스는 이미 동작 중입니다. 별도 구성이 필요 없습니다.
- app-ingress.yaml 파일을 생성하고, 다음 조건의 ingress 를 구성하세요.
  - name: app-ingress
  - NODE_PORT:30080/ 접속했을 때 nginx 서비스로 연결
  - NODE_PORT:30080/app 접속했을 때 appjs-service 서비스로 연결
  - Ingress 구성에 다음의 annotations을 포함시키세요.

    annotations:
      kubernetes.io/ingress.class: nginx

```
$ kubectl config use-context k8s
$ kubectl get namespace
$ kubectl run nginx --image=nginx -n ingress-nginx --port 80 --labels=app=nginx
$ kubectl get pod -n ingress-nginx
$ kubectl get svc -n ingress-nginx

$ vi app-ingress.yaml

$ kubectl apply -f app-ingress.yaml
$ kubectl describe ingress -n ingress-nginx
$ curl  k8s-worker1:30080
$ curl  k8s-worker1:30080/app
```

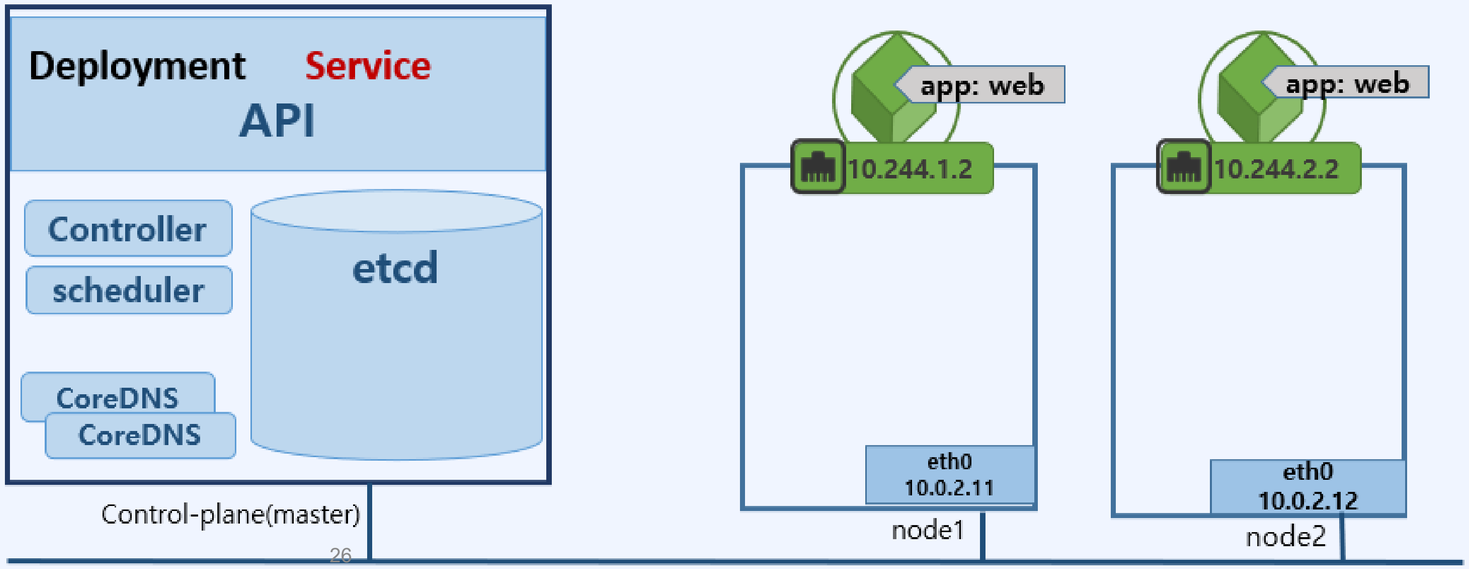
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress
  namespace: ingress-nginx
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: nginx
            port:
              number: 80
  - http:
      paths:
      - pathType: Prefix
        path: "/app"
        backend:
          service:
            name: app-service
            port:
              number: 80
```

Part 4 Services & Networking

06 kube-dns

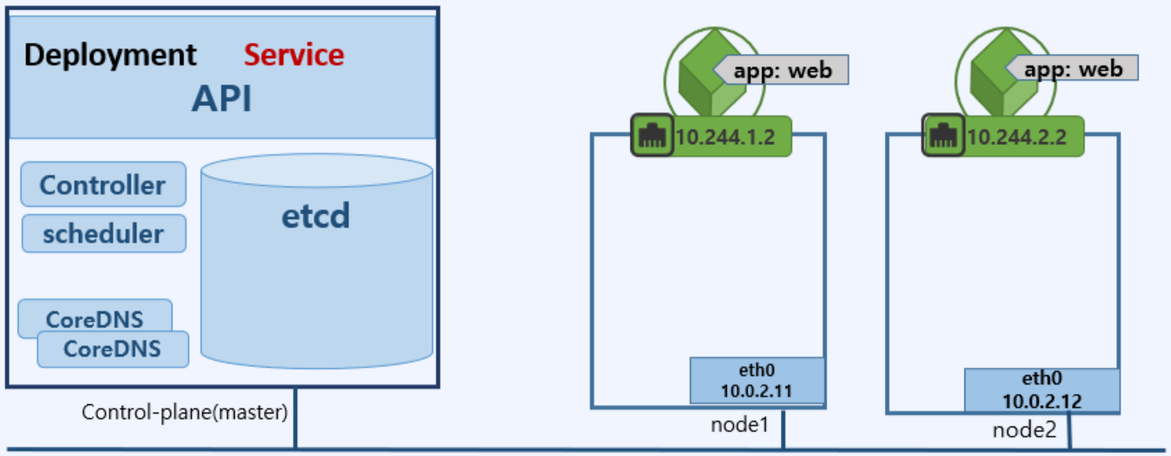
Core DNS

- 쿠버네티스 클러스터에서 사용하는 DNS
- 클러스터 내의 모든 Service에는 DNS 네임이 할당된다
  svc-name.namespace.svc.cluster.local
- 클러스터에서 동작되는 모든 Pod의 /etc/resolv.conf 에는 kube-dns가 namespace로 정의되어 있다.
  cat /etc/resolv.conf
  nameserver 10.32.0.10
  search <namespace>.svc.cluster.local svc.cluster.local cluster.local
- 특정 Pod에서 service name이나 pod name으로 Access 가능
  svc-name.namespace.svc.cluster.local
  pod-ip.namespace.pod.cluster.local



Core DNS:
LAB

- 간단한 web deployment 생성 후 service 구성.
- 다른 Pod에서 name 조회 할 수 있는지 TEST




```
$ kubectl config use-context k8s

$ kubectl create deployment web --image=nginx --port 80 --replicas=2
$ kubectl expose deployment web --port 80
$ kubectl get svc,pod | grep web

$ kubectl run dns-test --image=busybox  -it --rm -- /bin/sh
/ # cat /etc/resolv.conf
/ #  nslookup CLUSTER_IP
/ # exit

$ kubectl delete deployments.apps web
$ kubectl delete deployments.apps web
```

문제6:
Service and DNS Lookup구성

■ 작업 클러스터 : **k8s**

- image nginx를 사용하는 resolver pod를 생성하고 resolver-service라는 service를 구성합니다.
- 클러스터 내에서 service와 pod 이름을 조회할 수 있는지 테스트합니다.
  - dns 조회에 사용하는 pod 이미지는 busybox:1.28이고, service와 pod 이름 조회는 nlsookup을 사용합니다.
  - service 조회 결과는 /var/CKA2022/nginx.svc에 pod name 조회 결과는 /var/CKA2022/nginx.pod 파일에 기록합니다.

```
$ kubectl config use-context k8s
$ kubectl run resolver  --image=nginx --port=80

$ kubectl expose pod resolver --name resolver-service --port=80
$ kubectl get svc resolver-service

$ kubectl run test-nslookup --image=busybox:1.28 -it --restart=Never --rm -- nslookup  10.104.150.11
kubectl run test-nslookup --image=busybox:1.28 -it --restart=Never --rm -- nslookup  nginx-resolver-service > /var/CKA2022/nginx.svc

pod name service
$ kubectl get pod ngnix-resolver -o wide
$ kubectl run test-nslookup --image=busybox:1.28 -it --restart=Never --rm -- nslookup  10-244-1-55.default.pod.cluster.local > /var/CKA2022/nginx.pod

$ cat /var/CKA2022/nginx.svc
$ cat /var/CKA2022/nginx.pod
```