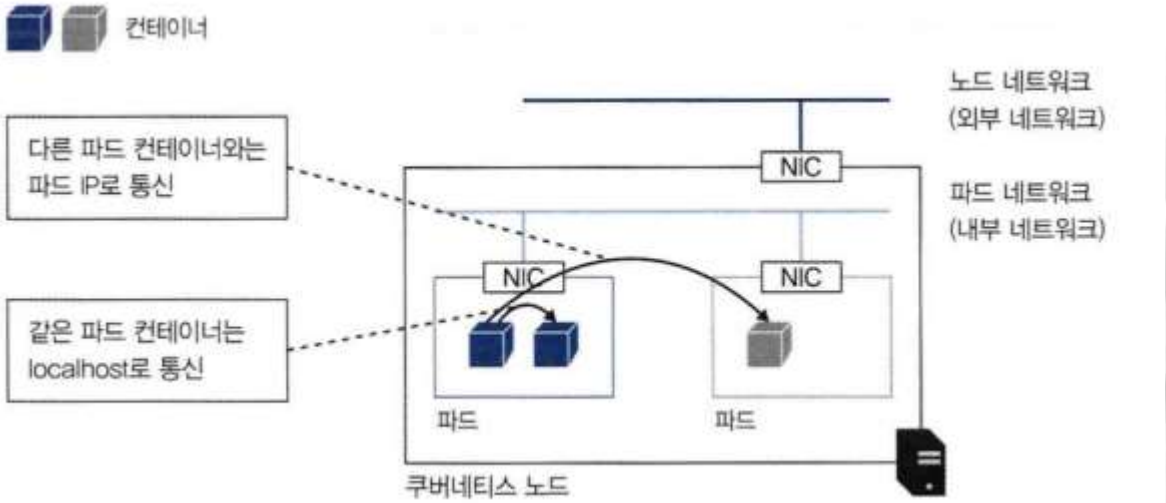
Service API

Service API

- Service API
  - ✓ 개요
    - 서비스 API 카테고리로 분류된 리소스는 클러스터상의 컨테이너에 대한 EndPoint를 제공하거나 레이블과 일치하는 컨테이너의 디스커버리에 사용되는 리소스
    - 내부적으로 사용되는 리소스를 제외하고 사용자가 직접 사용하는 것은 L4 LoadBalancing을 제공하는 서비스 리소스와 L7 LoadBalancing을 제공하는 인그레스 리소스가 있음
    - 서비스 리소스에는 제공 목적에 따라 클러스터 내부나 클러스터 외부에서 접속할 수 있는 접속 창구가 되는 가상 IP 등의 EndPoint를 제공
  - ✓ 종류
    - Service
      - ClusterIP
      - ExternalIP(ClusterIP의 한 종류)
      - NodePort
      - LoadBalancer
      - Headless(None)
      - ExternalName
      - None-Selector
    - Ingress

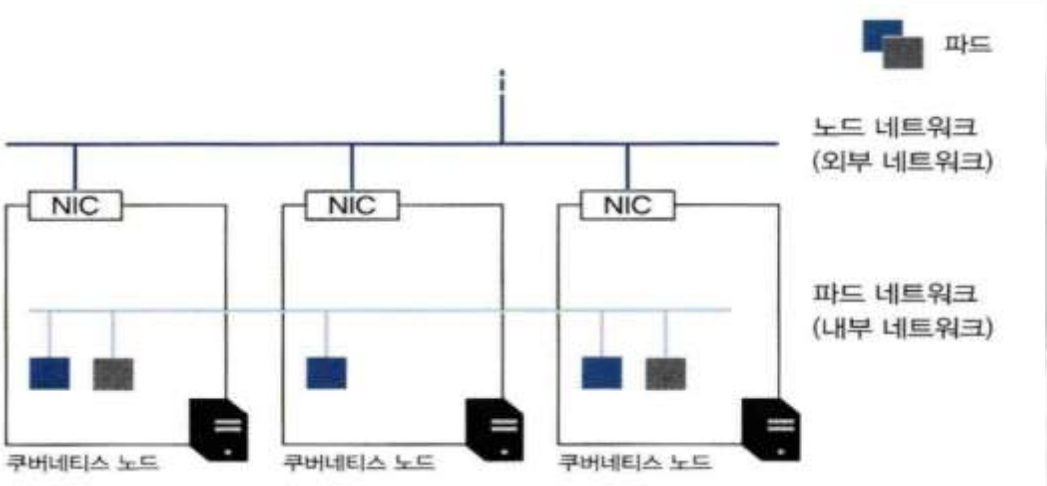
Service API

- 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 동일한 파드에 존재하는 컨테이너 간의 통신: 같은 파드 내에 있는 컨테이너는 동일한 IP 주소를 할당받게 되므로 같은 파드의 컨테이너로 통신하려면 localhost로 통신
  - ✓ 다른 파드의 컨테이너와 통신: 다른 파드에 있는 컨테이너와 통신하려면 파드의 IP 주소로 통신 가능



Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - 쿠버네티스 클러스터는 클러스터를 생성하면 노드 상에 파드를 위한 내부 네트워크가 자동으로 구성됨
    - 내부 네트워크 구성은 사용할 CNI(Container Network Interface)라는 플러그형(Pluggable) 모듈 구현에 따라 다르지만 기본적으로 노드 별로 다른 네트워크 세그먼트를 구성하고 노드 간의 트래픽은 VXLAN이나 L2 Routing 등의 기술을 사용하여 전송함으로써 노드 간 통신이 가능하게 구성
    - 노드 별 네트워크 세그먼트는 쿠버네티스 클러스터 전체에 할당된 네트워크 세그먼트를 자동으로 분할해 할당하므로 사용자가 설정하지 않아도 됨
    - 쿠버네티스 클러스터 내부에 구축된 파드 네트워크



Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드간 통신은 서비스를 사용하지 않고도 통신이 가능하지만 서비스를 사용했을 때 얻을 수 있는 장점
      - ◆ 파드에 대한 트래픽 LoadBalancing
      - ◆ 서비스 디스커버리와 클러스터 내부 DNS

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 Load Balancing
      - ◆ 디플로이먼트를 사용하여 여러 파드를 기동할 수 있는데 파드는 기동 될 때마다 각각 다른 ip 주소를 할당받기 때문에 LoadBalancing하는 구조를 자체적으로 구현하려면 각 파드의 ip 주소를 매번 조회하거나 전송 대상의 목적지를 설정해야 함
      - ◆ 서비스를 사용하면 여러 파드에 대한 LoadBalancing을 자동으로 구성할 수 있으며 서비스는 LoadBalancing의 접속 창구가 되는 EndPoint를 제공
      - ◆ EndPoint는 외부 LoadBalancer가 할당하는 가상 IP 주소(Virtual IP 주소) 나 클러스터 내부에서만 사용할 수 있는 가상 IP 주소(ClusterIP) 등 여러 가지 종류를 제공

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing - 클러스터 IP를 이용한 파드간 통신
      - ◆ nginx 이미지를 이용한 파드를 생성하기 위한 sample-deployment.xml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
      - name: nginx-container
        image: amsy810/echo-nginx:v2.0
```

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing - 클러스터 IP를 이용한 파드간 통신
      - ◆ 배포

        kubectl apply -f sample-deployment.yaml
      - ◆ 파드의 IP 확인

        kubectl get pods -l app=sample-app -o custom-columns="NAME:{metadata.name},IP:{status.podIP}"

        NAME IP

        sample-deployment-7b654b65d9-f98n2 10.244.2.2

        sample-deployment-7b654b65d9-p4x5t 10.244.1.2

        sample-deployment-7b654b65d9-s8hx5 10.244.3.2

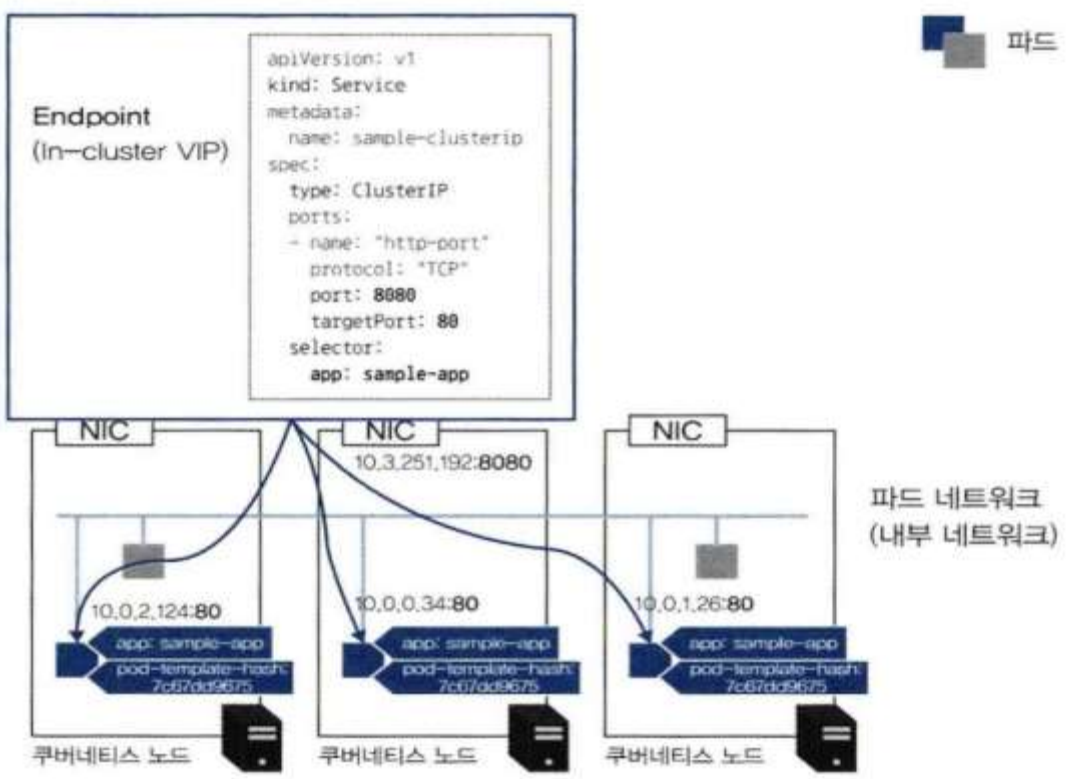
Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing - 클러스터 IP를 이용한 파드간 통신
      - ◆ ClusterIP는 클러스터 내부에서만 사용 가능한 가상 IP를 가진 EndPoint를 제공하는 LoadBalancer를 구성
      - ◆ 서비스는 spec.selector에 정의할 셀렉터 조건에 따라 트래픽을 전송
      - ◆ 파드에 ClusterIP를 할당하는 서비스 작성: sample-clusterip.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: sample-clusterip
spec:
  type: ClusterIP
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 8080
    targetPort: 80
  selector:
    app: sample-app
```

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing - 클러스터 IP를 이용한 파드간 통신
      - ◆ 서비스의 셀렉터와 파드 레이블 관계



Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing - 클러스터 IP를 이용한 파드간 통신
      - ◆ 리소스 생성

        kubectl apply -f sample-clusterip.yaml
      - ◆ 서비스 정보 확인

        kubectl get services sample-clusterip

        NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
        sample-clusterip ClusterIP 10.98.177.240 <none> 8080/TCP 189d

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing - 클러스터 IP를 이용한 파드간 통신
      - ◆ 서비스 상세 정보 확인: kubectl describe services sample-clusterip

```
Name:                     sample-clusterip
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=sample-app
Type:                     ClusterIP
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.98.177.240
IPs:                      10.98.177.240
Port:                     http-port  8080/TCP
TargetPort:               80/TCP
EndPoints:                10.244.205.193:80,10.244.151.2:80,10.244.120.117:80
Session Affinity:         None
Internal Traffic Policy:  Cluster
Events:                   <none>
```

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing - 클러스터 IP를 이용한 파드간 통신
      - ◆ 서비스 상세 정보 확인: kubectl describe services sample-clusterip
        - ▪ EndPoints 항목에 아무것도 표시되지 않을 경우 셀렉터 조건이 맞지 않을 가능성이 있음
        - ▪ 파드에 레이블이 정확히 설정되었는지 레이블 값이 셀렉터 조건과 동일한지를 확인
        - ▪ 생성한 LoadBalancer가 각 파드에 트래픽을 LoadBalancing하여 전송하고 있는지를 확인
        - ▪ EndPoint 서비스 종류에 CiusterIP를 지정했기 때문에 EndPoint 가상 IP는 클러스터 내부에서만 접속 가능한 IP 주소dlam 클러스터 내 별도 컨테이너를 기동하고 그 컨테이너에서 EndPoint로 HTTP 요청을 보내 동작을 확인
        - ▪ HTTP 요청은 LoadBalancing되어 어느 하나의 파드에서 요청을 처리하고 처리한 파드의 호스트명을 포함한 HTTP 응답이 반환되기 때문에 HTTP 응답을 보면 어떤 파드에서 요청을 처리했는지 확인할 수 있음

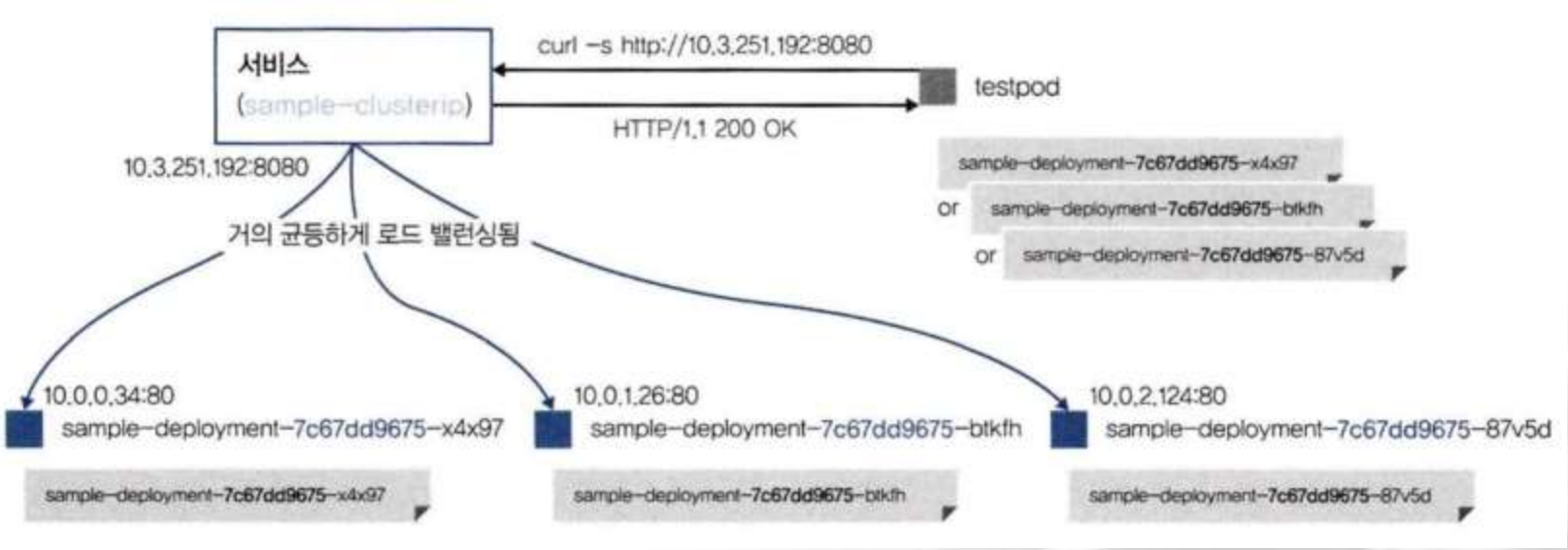
Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing - 클러스터 IP를 이용한 파드간 통신
      - ◆ 통신을 수행해서 IP를 확인

```
$ kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --
command -- curl -s http://10.98.177.240:8080
Host=10.98.177.240   Path=/   From=sample-deployment-598fcf4f7c-r8tkb
ClientIP=10.244.2.7   XFF=
pod "testpod" deleted

kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --
command -- curl -s http://10.98.177.240:8080
Host=10.98.177.240   Path=/   From=sample-deployment-598fcf4f7c-r8tkb
ClientIP=10.244.2.8   XFF=
pod "testpod" deleted

kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --
command -- curl -s http://sample-clusterip:8080
Host=sample-clusterip   Path=/   From=sample-deployment-d68df7d7-7t2zk
ClientIP=10.244.2.9   XFF=
pod "testpod" deleted
```

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing - 클러스터 IP를 이용한 파드간 통신
      - ◆ LoadBalancing



Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing - 여러 포트 할당
      - ◆ 하나의 서비스에 여러 포트를 할당할 수도 있음
      - ◆ http와 https에서 ClusterIP가 다르면 불편한 경우가 많으므로 하나의 서비스에서 여러 포트를 가질 수 있도록 설정하는 것이 바람직
      - ◆ ClusterIP의 8080/TCP 포트로의 요청은 파드 80/TCP 포트로 LoadBalancing하고, ClusterIP의 8443/TCP 포트로의 요청은 파드 443/TCP 포트로 LoadBalancing
      - ◆ 사용하고 있는 디플로이먼트의 파드에서는 443번 포트가 Listen 상태가 아니므로 https-port로는 통신할 수 없음

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing – 여러 포트 할당
      - ◆ sample-cluster-multi.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: sample-clusterip-multi
spec:
  type: ClusterIP
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 8080
    targetPort: 80
  - name: "https-port"
    protocol: "TCP"
    port: 8443
    targetPort: 443
  selector:
    app: sample-app
```

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing – name을 이용한 파드간 통신
      - ◆ 파드 생성: sample-named-port-pods.yaml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: sample-named-port-pod-80
  labels:
    app: sample-app
spec:
  containers:
  - name: nginx-container
    image: amsy810/echo-nginx:v2.0
    ports:
    - name: http # 포트에 이름 지정
      containerPort: 80
```

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing – name을 이용한 파드간 통신
      - ◆ 파드 생성: sample-named-port-pods.yaml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: sample-named-port-pod-81
  labels:
    app: sample-app
spec:
  containers:
  - name: nginx-container
    image: amsy810/echo-nginx:v2.0
    env:
    - name: NGINX_PORT
      value: "81"
    ports:
    - name: http # 포트에 이름 지정
      containerPort: 81
```

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing – name을 이용한 파드간 통신
      - ◆ 서비스 생성: sample-named-port-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: sample-named-port-service
spec:
  type: ClusterIP
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 8080
    targetPort: http # 포트 이름 참조
  selector:
    app: sample-app
```

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 쿠버네티스 클러스터의 내부 네트워크
    - ● 파드에 대한 트래픽 LoadBalancing – name을 이용한 파드간 통신
      - ◆ Pod 와 Service 생성 및 확인

        kubectl apply -f sample-named-port-pods.yaml

        kubectl apply -f sample-named-port-service.yaml

        kubectl get pods -o wide

        ```
        NAME                      READY   STATUS    RESTARTS   AGE    IP
        NODE                  NOMINATED NODE   READINESS GATES
        sample-named-port-pod-80  1/1     Running   0          8m1s   10.244.1.5
        minikubecluster-m02   <none>           <none>
        sample-named-port-pod-81  1/1     Running   0          8m1s   10.244.3.4
        minikubecluster-m04   <none>           <none>
        ```

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 클러스터 내부 DNS와 서비스 디스커버리
    - 쿠버네티스에서는 서비스 디스커버리 기능을 서비스(Service)가 제공
    - 서비스 디스커버리는 간단히 말하면 특정 조건의 대상이 되는 멤버를 보여주거나 이름에서 EndPoint를 판별하는 기능
    - 쿠버네티스에서 서비스 디스커버리란 서비스에 속해 있는 파드를 보여주거나 서비스명에서 EndPoint 정보를 반환하는 것
    - 서비스 디스커버리 방법은 세 가지가 있는데 DNS를 사용한 서비스 디스커버리는 클러스터안에 있는 DNS 서버에 자동으로 등록되는 서비스 EndPoint 정보를 사용
      - ◆ 환경 변수를 사용한 서비스 디스커버리
      - ◆ DNS A 레코드를 사용한 서비스 디스커버리

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 클러스터 내부 DNS와 서비스 디스커버리
    - ● 환경 변수를 사용한 서비스 디스커버리
      - ◆ 파드 내부에서는 환경 변수에서도 같은 네임스페이스 서비스를 확인할 수 있는데 -이 포함된 서비스명은 _ 로 변경된 후 대문자로 변환
      - ◆ docker -links ...로 실행했을 때와 같은 형식으로 환경 변수가 저장되기 때문에 도커 자체에서 사용하던 환경에서 마이그레이션할 때도 사용하기 편함
      - ◆ 파드가 생성된 후 서비스 생성이나 삭제에 따라 변경된 환경 변수가 먼저 기동한 파드 환경에는 자동으로 다시 등록되지 않기 때문에 예기치 못한 장애로 이어질 수 있는데 그런 경우에는 먼저 생성한 파드를 다시 생성

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 클러스터 내부 DNS와 서비스 디스커버리
    - ● 환경 변수를 사용한 서비스 디스커버리
      - ◆ 환경 변수에 등록된 서비스 정보 확인: kubectl exec -it sample-deployment-5d8757b5ff-8pgtb -- env | grep -i kubernetes

        KUBERNETES_PORT_443_TCP_PORT=443

        KUBERNETES_PORT=tcp://10.96.0.1:443

        KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443

        KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1

        KUBERNETES_SERVICE_HOST=10.96.0.1

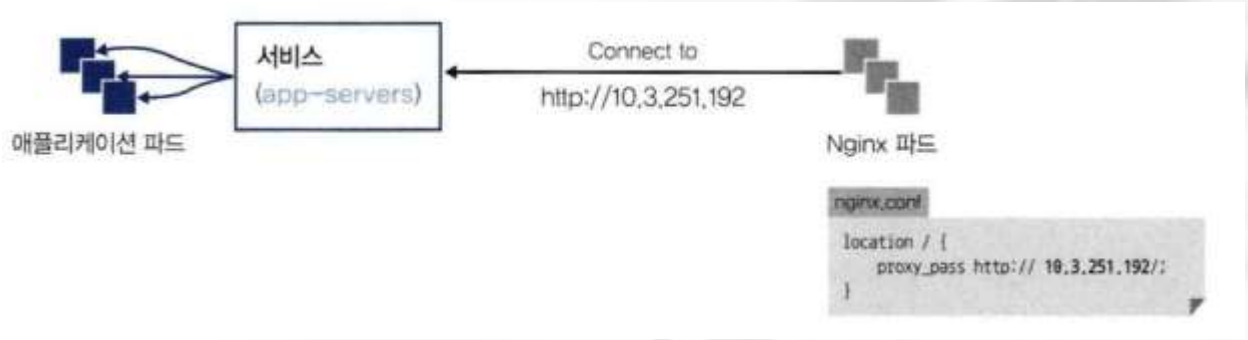
        KUBERNETES_SERVICE_PORT=443

        KUBERNETES_SERVICE_PORT_HTTPS=443

        KUBERNETES_PORT_443_TCP_PROTO=tcp
      - ◆ 파드의 spec.enableServiceLinks를 false로 설정하면 환경 변수 추가를 비활성화 할 수 있는데 이 파라미터의 기본값은 true로 설정되어 있음

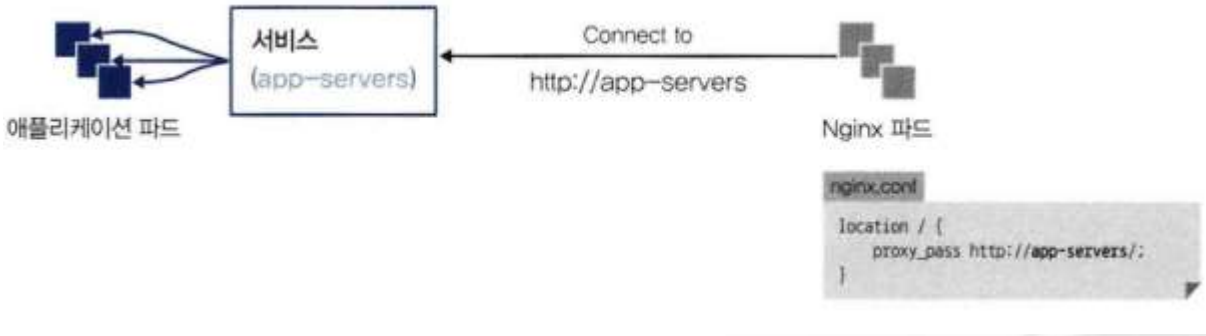
Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 클러스터 내부 DNS와 서비스 디스커버리
    - ● DNS A 레코드를 사용한 서비스 디스커버리
      - ◆ 개요
        - ▪ 다른 파드에서 서비스로 할당되는 EndPoint에 접속하려면 당연히 목적지가 필요하지만 할당된 IP 주소를 사용하는 방법 외에도 자동 등록된 DNS 레코드를 사용할 수 있음
        - ▪ 쿠버네티스에서 IP 주소를 편하게 관리하려면 기본적으로 자동 할당된 IP 주소에 연결된 DNS명을 사용하는 것이 바람직
        - ▪ 할당되는 IP 주소는 서비스를 생성할 때마다 변경되므로 IP 주소를 송신 측 컨테이너 설정 파일 등에서 명시적으로 설정하면 변경될 때마다 설정 파일 등을 변경해야 하기 때문에 컨테이너를 변경 불가능한(immutable) 상태로 유지할 수 없어 추천하지 않음
        - ▪ 서비스에 대한 요청을 IP 주소로 지정한 경우



Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 클러스터 내부 DNS와 서비스 디스커버리
    - ● DNS A 레코드를 사용한 서비스 디스커버리
      - ◆ 개요
        - ▪ 서비스에 대한 요청을 DNS 이름으로 지정한 경우



Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 클러스터 내부 DNS와 서비스 디스커버리
    - ● DNS A 레코드를 사용한 서비스 디스커버리
      - ◆ 서비스 이름을 사용한 요청 전송

kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command -- curl -s http://sample-clusterip:8080

Host=sample-clusterip Path=/ From=sample-deployment-5d9fcfcbb6-f5gf2
ClientIP=10.244.1.4 XFF=
pod "testpod" deleted

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 클러스터 내부 DNS와 서비스 디스커버리
    - ● DNS A 레코드를 사용한 서비스 디스커버리
      - ◆ FQDN
        - ▪ 정식 명칭은 서비스명.네임스페이스명.svc.cluster.local
        - ▪ FQDN 확인

          kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command -- dig sample-clusterip.default.svc.cluster.local

          ;; QUESTION SECTION:
          ;sample-clusterip.default.svc.cluster.local. IN A

          ;; ANSWER SECTION:
          sample-clusterip.default.svc.cluster.local. 30 IN A 10.101.106.107

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 클러스터 내부 DNS와 서비스 디스커버리
    - ● DNS A 레코드를 사용한 서비스 디스커버리
      - ◆ FQDN
        - ▪ FQDN에는 서비스가 충돌하지 않도록 네임스페이스 등이 포함되어 있는데, 컨테이너 내부의 /etc/resolve.conf 에 다음과 같은 내용(search로 시작하는 행)이 있기 때문에 sample-clusterip.default 나 sample-clusterip 만으로 이름을 해석할 수 있음
        - ▪ 이 설정은 기본 네임스페이스라면 default.svc.cluster.local이 들어가 있듯이 네임스페이스별로 다르며 동일한 네임스페이스인 경우에는 sample-clusterip 와 같이 서비스명만으로 이름 해석이 가능하지만 다른 네임스페이스의 경우에는 sample-clusterip.default처럼 네임스페이스명을 붙여 이름을 해석
        - ▪ /etc/resolve.conf 확인

          kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command -- cat /etc/resolv.conf

          search default.svc.cluster.local svc.cluster.local cluster.local ap-northeast-2.compute.internal
          nameserver 10.96.0.10
          options ndots:5
          pod "testpod" deleted

Service API

- ❖ 쿠버네티스 클러스터 네트워크 와 서비스
  - ✓ 클러스터 내부 DNS와 서비스 디스커버리
    - ● DNS A 레코드를 사용한 서비스 디스커버리
      - ◆ FQDN
        - ▪ 역방향 질의

          kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command -- dig -x 10.101.106.107

          ;; QUESTION SECTION:

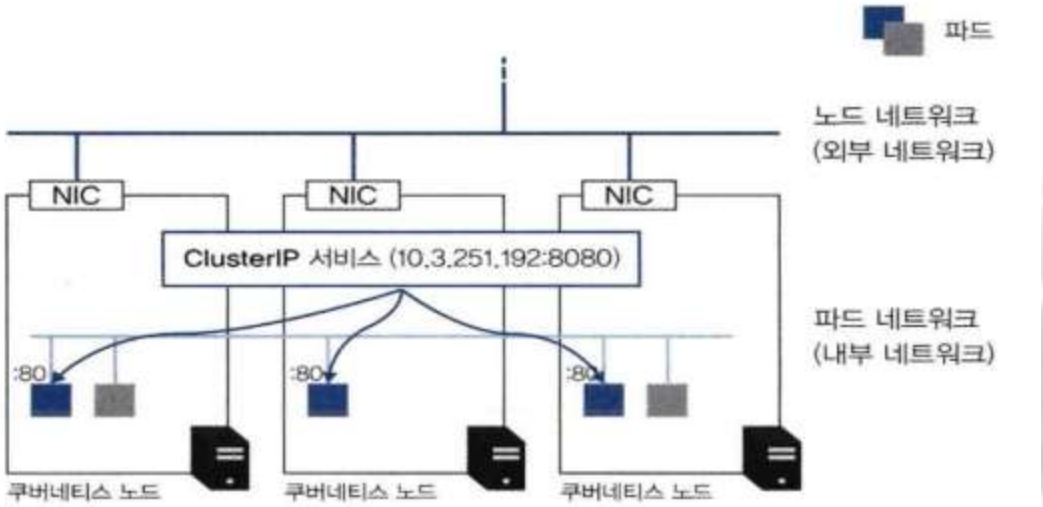
          ;107.106.101.10.in-addr.arpa. IN PTR

          ;; ANSWER SECTION:

          107.106.101.10.in-addr.arpa. 30 IN PTR sample-clusterip.default.svc.cluster.local.
        - ▪ 이름 해석을 사용하지 않을 경우 고정 IP로 설정하여 기존과 같이 IP 주소를 수동으로 관리하거나 서비스 생성 시에 자동으로 할당된 EndPoint의 IP 주소를 확인하고 애플리케이션 등의 설정 파일에 포함시켜야 하므로 많은 시간이 소요되므로 쿠버네티스의 이식성을 유지하기 위해 IP 주소는 최대한 서비스명을 사용하여 이름 해석을 하도록 권장

ClusterIP Service

- 개요
  - ✓ ClusterIP 서비스는 쿠버네티스의 가장 기본이 되는 서비스
  - ✓ type: ClusterIP 서비스를 생성하면 쿠버네티스 클러스터 내부에서만 통신 가능한 Internal Network에 생성되는 가상 IP가 할당됨
  - ✓ C;usterIP와 통신은 각 노드 상에서 실행 중인 시스템 구성 요소 kube-proxy가 파드로 전송
  - ✓ CiusterIP는 쿠버네티스 클러스터 외부에서 트래픽을 수신할 필요가 없는 환경에서 클러스터 내부 LoadBalancer로 사용



ClusterIP Service

- 개요
  - ✓ 기본적으로는 쿠버네티스 API에 접속하기 위한 Kubernetes 서비스가 생성되어 있고 ClusterIP가 발급되어 있음

kubectl get services

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	**ClusterIP**	**10.96.0.1**	**<none>**	**443/TCP**	**23h**
sample-clusterip	ClusterIP	10.101.106.107	<none>	8080/TCP	22h
sample-named-port-service	ClusterIP	10.104.250.170	<none>	8080/TCP	

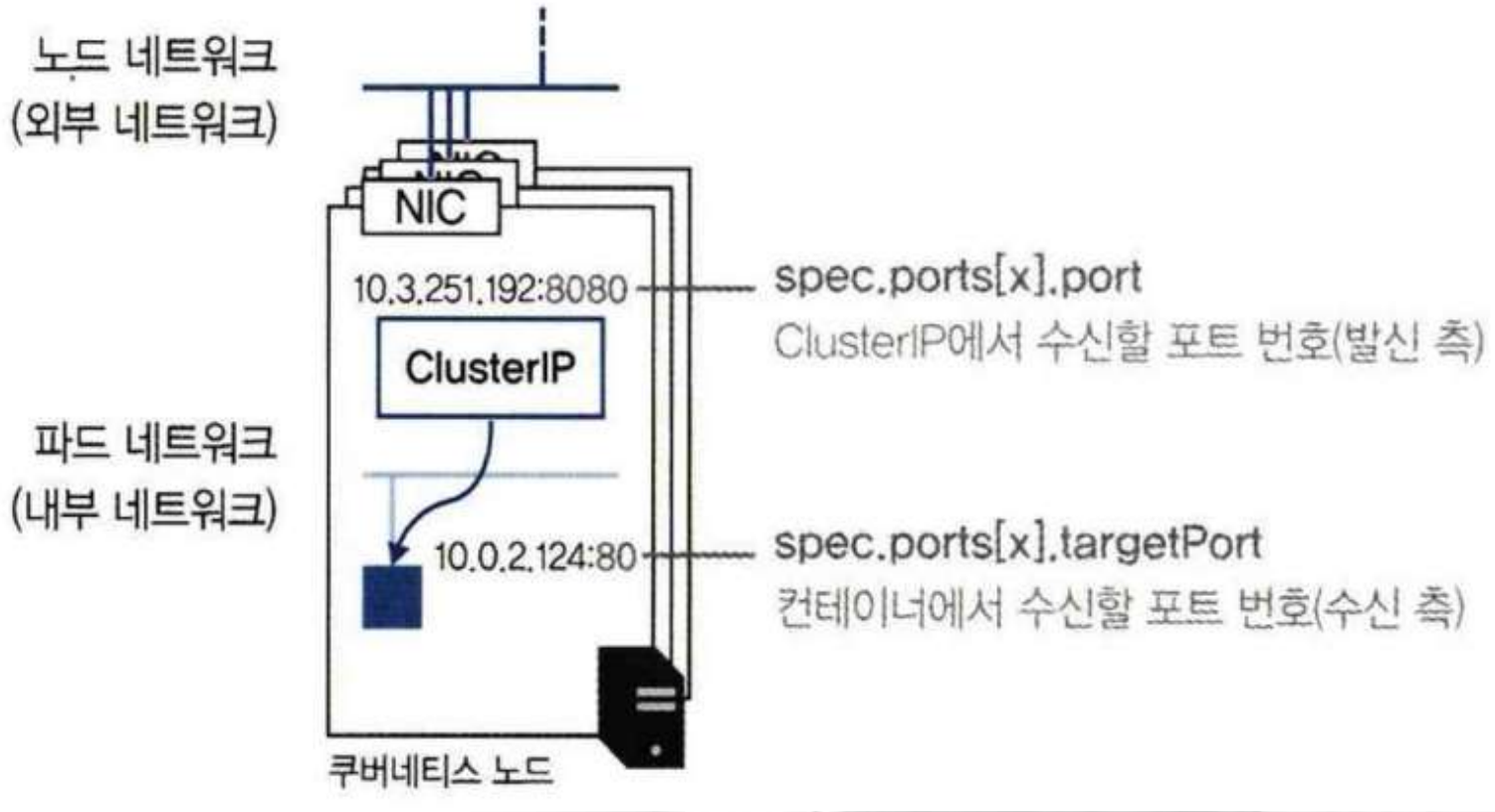
ClusterIP Service

- 생성
  - ✓ type: ClusterIP를 지정하면 ClusterIP 서비스를 생성할 수 있음
  - ✓ spec.ports[ ].port에는 ClusterIP에서 수신할 포트 번호를 지정하고 spec.ports[ ].targetPort는 목적지 컨테이너 포트 번호를 지정
  - ✓ sample-clusterip.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: sample-clusterip
spec:
  type: ClusterIP
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 8080
    targetPort: 80
  selector:
    app: sample-app
```

ClusterIP Service

❖ 생성



ClusterIP Service

- 생성
  - ✓ LoadBalancing 확인

```
$ kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command --
curl -s http://sample-clusterip:8080
Host=sample-clusterip  Path=/  From=sample-deployment-5d9fcfcbb6-wp7t4
ClientIP=10.244.2.8  XFF=
pod "testpod" deleted

$ kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command --
curl -s http://sample-clusterip:8080
Host=sample-clusterip  Path=/  From=sample-deployment-5d9fcfcbb6-578xf
ClientIP=10.244.2.9  XFF=
pod "testpod" deleted

$ kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command --
curl -s http://sample-clusterip:8080
Host=sample-clusterip  Path=/  From=sample-deployment-5d9fcfcbb6-sv4bv
ClientIP=10.244.2.10  XFF=
pod "testpod" deleted
```

ClusterIP Service

- 가상 IP 정적 지정
  - ✓ 애플리케이션에서 데이터베이스 서버를 지정하는 경우에는 기본적으로 쿠버네티스 서비스에 등록된 클러스터 내부 DNS 레코드를 사용하여 호스트를 지정하는 것이 바람직
  - ✓ IP 주소로 지정하고자 하는 경우 ClusterIP를 정적으로 지정할 수도 있음
  - ✓ ClusterIP를 자동 할당이 아닌 수동으로 설정할 경우 spec.ClusterIP를 지정

ClusterIP Service

- 가상 IP 정적 지정
  - ✓ 실습
    - yaml 파일 작성: sample-clusterip-vip

```
apiVersion: v1
kind: Service
metadata:
  name: sample-clusterip-vip
spec:
  type: ClusterIP
  clusterIP: 10.101.106.108
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 8080
    targetPort: 80
  selector:
    app: sample-app
```

ClusterIP Service

- 가상 IP 정적 지정
  - ✓ 실습
    - yaml 파일 실행

      kubectl apply -f sample-clusterip-vip.yaml
    - 통신 확인

      kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command -- curl -s http://sample-clusterip-vip:8080

      Host=sample-clusterip-vip Path=/ From=sample-deployment-d68df7d7-7t2zk ClientIP=10.244.2.16 XFF=

      pod "testpod" deleted

      kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command -- curl -s http://10.101.106.108:8080

      Host=10.96.253.81 Path=/ From=sample-deployment-d68df7d7-cd64g ClientIP=10.244.2.17 XFF=

      pod "testpod" deleted

ClusterIP Service

- ClusterIP 변경
  - ✓ 서비스가 이미 생성된 후에는 ClusterIP를 변경할 수 없음
  - ✓ 쿠버네티스에서는 kubectl apply를 사용하여 설정 값을 변경할 수 있다고 설명하지만 쿠버네티스 특성상 ClusterIP 값 등과 같은 일부 필드는 변경할 수 없게(immutable) 되어 있음
  - ✓ ClusterIP를 지정하려면 삭제하고 다시 생성해야 함한다.
  - ✓ 실습
    - yaml 파일 수정 후 실행

      sed -i -e 's|10.101.106.108|10.101.106.109|g' sample-clusterip-vip.yaml

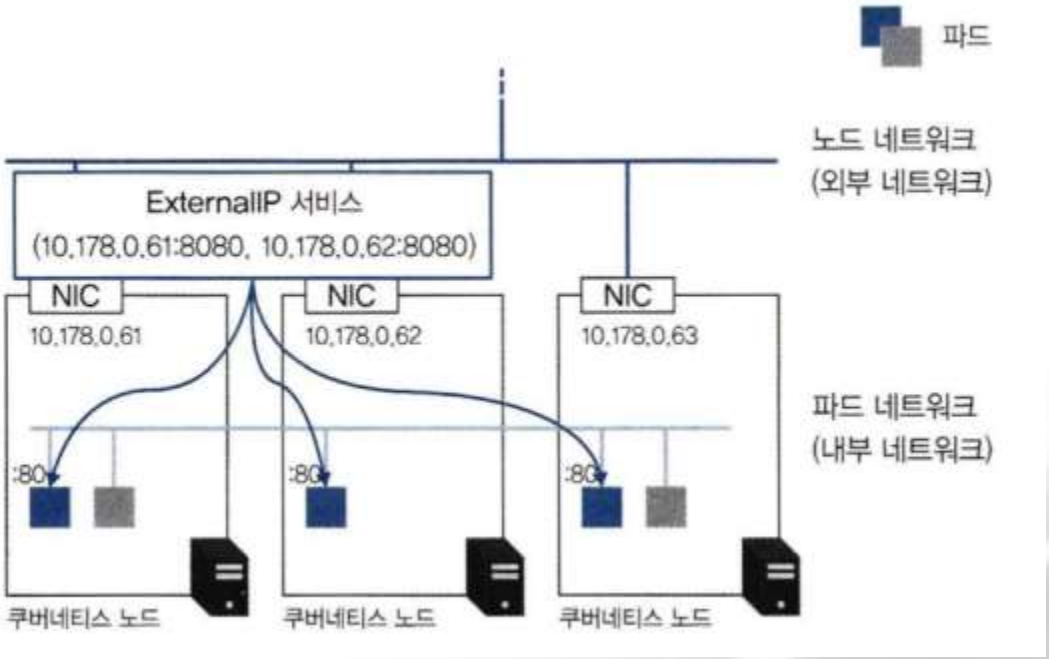
      kubectl apply -f sample-clusterip-vip.yaml

      The Service "sample-clusterip-vip" is invalid: spec.clusterIPs[0]: Invalid value: []string{"10.96.253.82"}: may not change once set

ExternalIP Service

❖ 개요

✓ ExternalIP 서비스는 특정 쿠버네티스 노드 IP 주소:포트에서 수신한 트래픽을 컨테이너로 전송하는 형태로 외부와 통신할 수 있도록 하는 서비스



ExternalIP Service

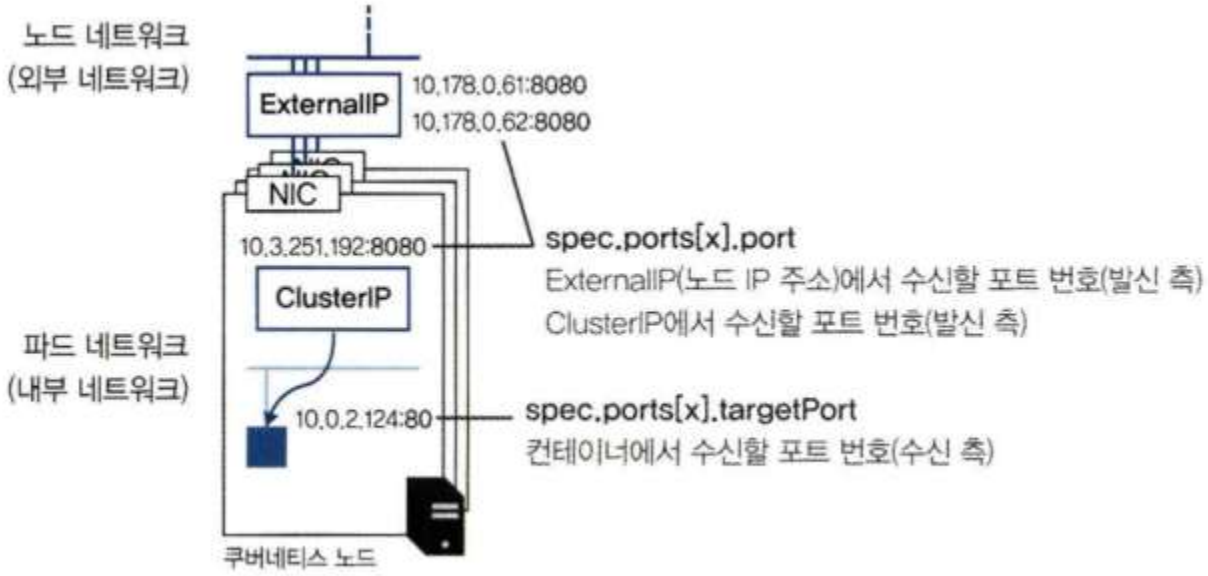
- 워커 노드의 IP 확인

```
kubectl get nodes -o custom-
columns="NAME:{metadata.name},IP:{status.addresses[].address}"
```

```
NAME                IP
ip-172-31-40-1      172.31.40.1
ip-172-31-40-216    172.31.40.216
ip-172-31-42-129    172.31.42.129
```

ExternalIP Service

- ExternalIP 서비스에 설정하는 포트 번호
  - ✓ spec.externalIPs: 쿠버네티스 노드 IP 주소(ExternalIP)
  - ✓ spec.ports[].port: ExternalIP 와 ClusterIP에서 수신할 포트 번호
  - ✓ spec.ports[].targetPort: 목적지 컨테이너 포트 번호



ExternalIP Service

- 실습
  - ✓ externalIP 설정을 위한 yaml 파일 생성: sample-externalip.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: sample-externalip
spec:
  type: ClusterIP
  externalIPs:
  - 172.31.40.216
  - 172.31.42.129
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 8080
    targetPort: 80
  selector:
    app: sample-app
```

ExternalIP Service

- 실습
  - ✓ sample-externalip.yaml 실행

    kubectl apply -f sample-externalip.yaml
  - ✓ 서비스 확인

    kubectl get svc

```
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP                 PORT(S)    AGE
SELECTOR
kubernetes          ClusterIP   10.96.0.1       <none>                      443/TCP    16h
<none>
sample-externalip   ClusterIP   10.110.41.145   172.31.15.241,172.31.4.96   8080/TCP   15h
app=sample-app
```

ExternalIP Service

- 실습
  - ✓ 외부에서 접근(Public IP로 접근 가능)

    curl 172.31.40.216:8080

    Host=10.110.41.145 Path=/ From=sample-deployment-56968fc67b-nw2nv
    ClientIP=10.244.0.0 XFF=

    curl 172.31.42.129:8080

    Host=10.110.41.145 Path=/ From=sample-deployment-56968fc67b-pwvlh
    ClientIP=10.244.0.0 XFF=

ExternalIP Service

- 실습
  - ✓ 서비스 확인

    kubectl get service

```
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP                 PORT(S)
AGE
kubernetes          ClusterIP   10.96.0.1       <none>                      443/TCP
23h
sample-externalip   ClusterIP   10.104.248.61   172.31.40.216,172.31.42.129
8080/TCP   16m
```

    - 컨테이너 내부에서의 통신은 ClusterIP를 사용하기 위해 ClusterIP도 자동으로 확보

ExternalIP Service

- 실습
  - ✓ 컨테이너 내부에서 확인하면 클러스터 내부 DNS에서 반환하는 서비스 디스커버리의 IP 주소는 ExternalIP가 아닌 ClusterIP

    kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command -- dig sample-externalip.default.svc.cluster.local

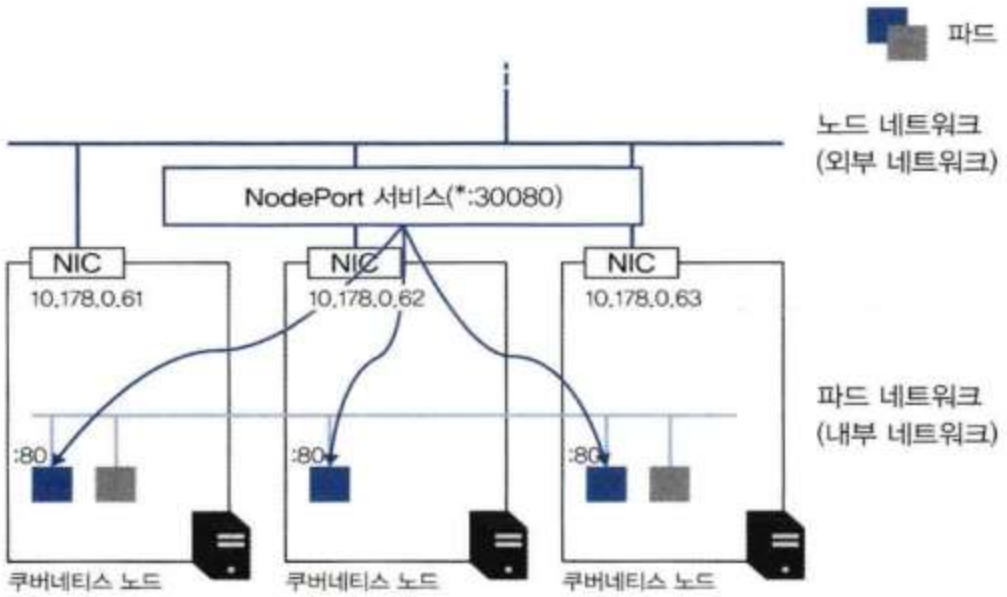
    ;; ANSWER SECTION:

    sample-externalip.default.svc.cluster.local. 30 IN A 10.104.248.61

NodePort Service

❖ 개요

- ✓ NodePort 서비스는 ExternalIP 서비스와 유사
- ✓ ExternalIP는 지정한 쿠버네티스 노드의 IP 주소:포트에서 수신한 트래픽을 컨테이너로 전송하는 형태로 외부와 통신할 수 있는데 NodePort는 모든 쿠버네티스 노드의 IP 주소:포트에서 수신한 트래픽을 컨테이너에 전송하는 형태로 외부와 통신
- ✓ ExternalIP 서비스에서 전체 노드의 트래픽을 수신할 수 있도록 설정하는 기능과 비슷하지만 Listen할 때 0.0.0.0:포트를 사용하여 모든 IP 주소로 바인드하는 형태



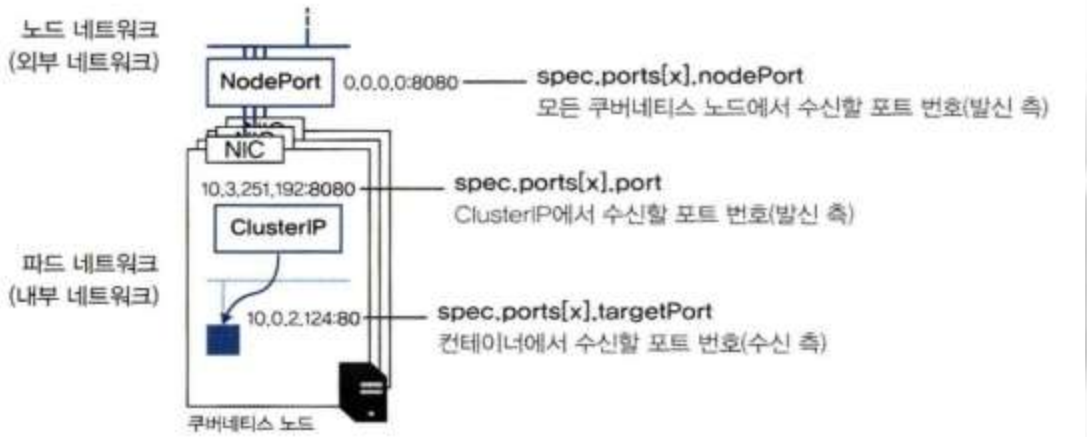
NodePort Service

- 실습
  - ✓ 서비스 작성: sample-nodeport.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: sample-nodeport
spec:
  type: NodePort
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 8080
    targetPort: 80
    nodePort: 30080
  selector:
    app: sample-app
```

NodePort Service

- 실습
  - ✓ 서비스 작성: sample-nodeport.yaml
    - spec.ports[ ].port에는 ClusterIP에서 수신할 포트 번호를 spec.ports[].targetPort에는 목적지 컨테이너 포트 번호를 spec.ports[ ].nodePort에는 모든 쿠버네티스 노드에서 수신할 포트 번호를 지정
    - 모든 쿠버네티스 노드의 IP 주소에서 spec.ports[].nodePort에 지정한 포트를 Listen하기 때문에 포트 충돌이 없도록 주의해야 하고 30000-32767 까지만 사용 가능
    - 할당된 포트 번호를 지정할 필요가 없을 경우에는 spec.ports[ ].nodePort 항목에 명시적으로 포트를 지정하지 않으면 빈 포트 번호가 자동으로 선택



NodePort Service

- 실습
  - ✓ 서비스 생성
    - kubectl apply -f sample-nodeport.yaml
  - ✓ 서비스 확인 – ClusterIP 도 자동 할당
    - kubectl get svc

```
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP          113m
sample-clusterip    ClusterIP   10.105.145.150   <none>        8080/TCP         63m
sample-nodeport     NodePort    10.105.89.234    <none>        8080:30080/TCP   8m49s
```

NodePort Service

- 실습
  - ✓ 서비스 디스커버리는 ClusterIP
    - kubectl apply -f sample-nodeport.yaml
  - ✓ 서비스 확인 – ClusterIP 도 자동 할당
    - kubectl get svc

```
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP          113m
sample-clusterip    ClusterIP   10.105.145.150   <none>        8080/TCP         63m
sample-nodeport     NodePort    10.105.89.234    <none>        8080:30080/TCP   8m49s
```

NodePort Service

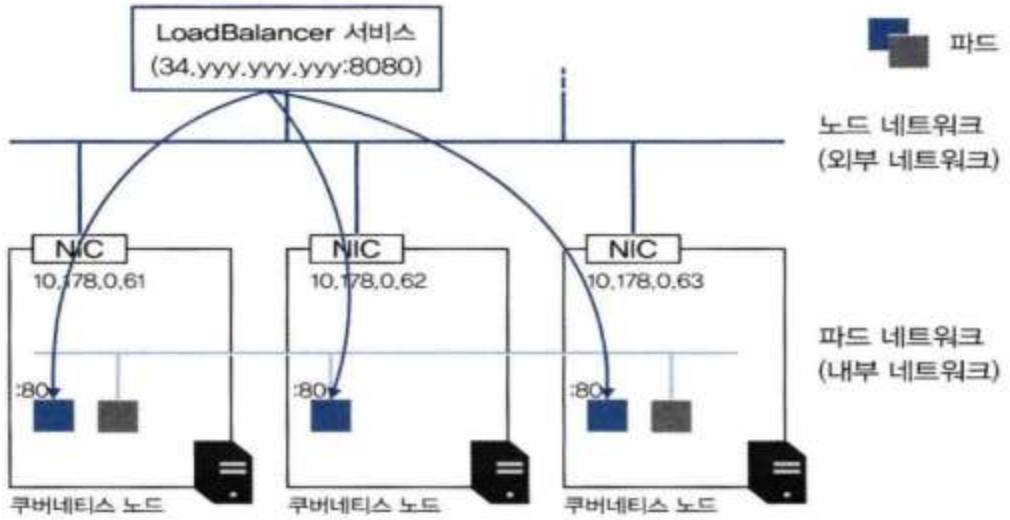
- ❖ 실습
  - ✓ 접속
    - ● 요청 전송: LoadBalancing

```
$ curl -s 172.31.40.216:30080
Host=172.31.40.216  Path=/  From=sample-deployment-5d9fcfcbb6-578xf
ClientIP=10.244.1.1  XFF=

$ curl -s 172.31.40.216:30080
Host=172.31.40.216  Path=/  From=sample-deployment-5d9fcfcbb6-wp7t4
ClientIP=10.244.1.1  XFF=

$ curl -s 172.31.40.216:30080
Host=172.31.40.216  Path=/  From=sample-deployment-5d9fcfcbb6-wp7t4
ClientIP=10.244.1.1  XFF=
```

LoadBalancer Service

❖ 개요

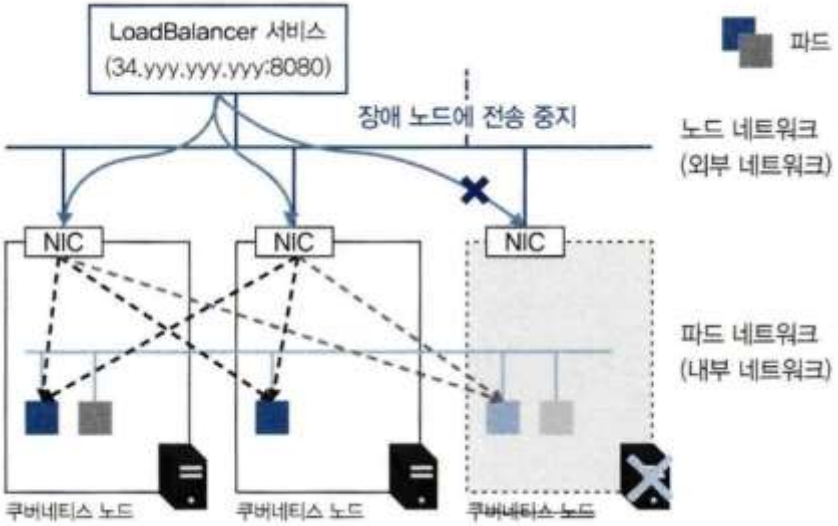
- ✓ LoadBalancer 서비스는 서비스 환경에서 클러스터 외부로부터 트래픽을 수신할 때 사용하는 서비스
- ✓ LoadBalancer 서비스는 클러스터로 트래픽을 전달해 주는 외부 LoadBalancer와 함께 동작하며 레이블 셀렉터와 일치하는 파드로 트래픽을 전달하는 서비스
- ✓ 외부 LoadBalancer를 사용하려면 쿠버네티스 클러스터가 구축된 인프라가 이 구조에 맞도록 설계되어 있어야 함
- ✓ 현재는 GCP/AWS/애저(Azure)/OpenStack을 비롯한 클라우드 프로바이더가 LoadBalancer 서비스를 사용할 수 있는 환경을 제공(MetalLB 등도 출시되어 클라우드 프로바이더 이외의 환경을 사용할 수 있는 선택의 폭도 넓어지고 있음)



LoadBalancer Service

❖ 개요

- ✓ NodePort나 ExternalIP에서는 결국 하나의 쿠버네티스 노드에 할당된 IP 주소로 통신을 하기 때문에 그 노드가 단일 장애점(Single Point of Failure SPoF)이 되지만 type: LoadBalancer에서는 쿠버네티스 노드와 별도로 외부 LoadBalancer를 사용하기 때문에 노드 장애가 발생해도 크게 문제가 되지 않음
- ✓ 구조는 NodePort 서비스를 생성하고 클러스터 외부의 LoadBalancer에서 쿠버네티스 노드로 밸런싱을 하는 형태
- ✓ 쿠버네티스 노드에 장애가 발생한 경우에는 그 노드에 트래픽을 전송하지 않음으로써 자동으로 복구하게 되어 있지만 LoadBalancer가 노드 장애를 감지하고 목적지에서 제외 처리하기까지의 시간이 필요하므로 그동안 일시적으로 서비스 중단 현상이 발생



LoadBalancer Service

- 실습
  - ✓ 서비스 생성: sample-lb.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: sample-lb
spec:
  type: LoadBalancer
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 8080
    targetPort: 80
    nodePort: 30082
  selector:
    app: sample-app
```

LoadBalancer Service

❖ 실습

✓ 서비스 확인

kubectl get svc

```
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP      10.96.0.1       <none>        443/TCP          24m
sample-lb    LoadBalancer   10.100.160.84   <pending>     8080:30082/TCP   24m
```

◆ 관리형 쿠버네티스가 아니므로 LoadBalancer의 외부 IP가 생성되지 않음

LoadBalancer Service

❖ 실습

✓ minikube에서 터널링을 이용해서 LoadBalancer 생성

minikube service sample-lb

```
|-----------|-----------|---------------|-------------------------|
| NAMESPACE |   NAME    |  TARGET PORT  |           URL           |
|-----------|-----------|---------------|-------------------------|
| default   | sample-lb | http-port/8080 | http://192.168.49.2:30082 |
|-----------|-----------|---------------|-------------------------|
```

🎉 Opening service default/sample-lb in default browser...

👉 http://192.168.49.2:30082

curl 192.168.49.2:30082

Host=192.168.49.2 Path=/ From=sample-deployment-d68df7d7-4pj56
ClientIP=192.168.49.2 XFF=

curl 192.168.49.2:30082

Host=192.168.49.2 Path=/ From=sample-deployment-d68df7d7-b9bdx
ClientIP=10.244.0.1 XFF=

LoadBalancer Service

- MetalLB
  - ✓ 개요
    - MetalLB는 BareMetal LoadBalancer의 약자로 온프레미스 환경에서도 클라우드 LoadBalancer처럼 사용할 수 있는 서비스(LoadBalancer 타입)
    - 퍼블릭 클라우드 환경 AWS의 경우 LoadBalancer 타입의 서비스를 사용할 때 클라우드 제공자의 LoadBalancer(ALB)가 자동으로 할당되지만 온프레미스 환경에서는 이러한 LoadBalancer가 기본적으로 제공되지 않기 때문에 MetalLB를 사용하여 클러스터 외부로 서비스를 노출할 수 있음
    - MetalLB는 AWS, GCP 등 대부분의 퍼블릭 클라우드 환경을 지원하지는 않음
    - MetalLB는 로컬 Kubernetes 환경(Bare-metal, VM) 에서 LoadBalancer 타입의 서비스를 사용할 수 있게 해주는 네트워크 Load Balancer
    - 클라우드 환경이 아닌 Ubuntu + kubeadm 클러스터에 매우 적합하며 VMware, VirtualBox, UTM, Proxmox 등의 가상 환경에서도 많이 사용

LoadBalancer Service

- MetalLB
  - ✓ 지원 여부

Cloud platform	Supported
AWS	No, use EKS
Azure	No, use AKS
DigitalOcean	No, use DigitalOcean Kubernetes
Equinix Metal	Yes, see Equinix Metal notes
Google Cloud	No, use GKE
Hetzner	Yes, see Hetzner notes
OVH	Yes, when using a vRack
OpenShift OCP	Yes, see OpenShift notes
OpenStack	Yes, see OpenStack notes
Proxmox	Yes
VMware	Yes
Vultr	Yes

LoadBalancer Service

- MetalLB
  - ✓ 특징
    - DaemonSet으로 speaker 파드를 생성하여 External IP를 전파
      - speaker 파드는 호스트 네트워크 사용(NetworkMode: host)
    - 서비스(LoadBalancer)의 External IP 전파를 위해 표준 프로토콜 ARP(IPv4), NDP(IPv6), BGP를 사용
    - MetalLB는 두 가지 모두를 지원하며 환경에 맞게 구성하여 사용할 수 있음
      - L2 mode ARP: 같은 서브넷 내에서 ARP/NDP를 통해 IP를 광고
      - BGP mode: BGP를 사용하여 외부 네트워크 라우터와 경로 정보를 동적으로 공유
  - ✓ MetalLB 사용 절차 요약
    - MetalLB를 Kubernetes에 설치 (Manifest 방식)
    - LoadBalancer 서비스에 할당할 IP 주소 범위 지정
    - type: LoadBalancer 서비스 생성 시 자동으로 외부 IP 부여

LoadBalancer Service

- MetalLB
  - ✓ 설치
    - kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.13.10/config/manifests/metallb-native.yaml
  - ✓ 확인(controller 와 speaker 모두 Running 상태 이어야 함)
    - kubectl get all -n metallb-system

```
NAME                                  READY   STATUS    RESTARTS   AGE
pod/controller-679855f7d7-6hp5n   2/2     Running   0          30m
pod/speaker-7xx4w                 2/2     Running   0          30m
pod/speaker-f8pdq                 2/2     Running   0          30m
pod/speaker-nc8dn                 2/2     Running   0          30m

NAME                                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/controller-monitor-service    ClusterIP   None            <none>
9120/TCP   30m
service/metallb-webhook-service       ClusterIP   10.105.71.20    <none>
443/TCP    30m
service/speaker-monitor-service       ClusterIP   None            <none>
9120/TCP   30m
```

LoadBalancer Service

- MetalLB
  - ✓ 모드 및 서비스 대역 지정
    - IPAddressPool 생성: IP 대역은 외부에서 사용 가능한 대역으로 설정

```
cat <<EOF | kubectl apply -f -
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: my-ippool
  namespace: metallb-system
spec:
  addresses:
  - 192.168.49.2-192.168.49.50
EOF
```

LoadBalancer Service

- MetalLB
  - ✓ 모드 및 서비스 대역 지정
    - l2advertisements 생성: spec.ipAddressPools - 설정한 IPpool을 기반으로 Layer2 모드로 LoadBalancer IP 사용 허용

```
# 생성
cat <<EOF | kubectl apply -f -
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: my-l2-advertise
  namespace: metallb-system
spec:
  ipAddressPools:
  - my-ippool
EOF
```

LoadBalancer Service

- MetalLB
  - ✓ 모드 및 서비스 대역 지정
    - yaml 파일로 작성해서 설정 가능

```
# metallb-config.yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: my-ip-pool
  namespace: metallb-system
spec:
  addresses:
    - 192.168.56.240-192.168.56.250
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2adv
  namespace: metallb-system
```

LoadBalancer Service

- MetalLB
  - ✓ 모드 및 서비스 대역 지정
    - 확인

      kubectl get l2advertisements -n metallb-system

      NAME IPADDRESSPOOLS IPADDRESSPOOL SELECTORS INTERFACES
      my-l2-advertise ["my-ippool"]

LoadBalancer Service

- MetalLB
  - ✓ 서비스 생성
    - sample-lb.yaml 파일 작성

```
apiVersion: v1
kind: Service
metadata:
  name: sample-lb
spec:
  type: LoadBalancer
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 8080
    targetPort: 80
    nodePort: 30082
  selector:
    app: sample-app
```

LoadBalancer Service

- MetalLB
  - ✓ 서비스 생성
    - 실행

      kubectl apply -f sample-lb.yaml
    - 확인

      kubectl get svc

      ```
      NAME          TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
      kubernetes    ClusterIP      10.96.0.1       <none>          443/TCP          115m
      sample-lb     LoadBalancer   10.110.139.29   192.168.49.2    8080:30082/TCP   5s
      ```

      curl 192.168.49.2:8080

      ```
      Host=192.168.49.2  Path=/  From=sample-deployment-d68df7d7-jjzs2
      ClientIP=192.168.49.2  XFF=
      ```

      curl 192.168.49.2:8080

      ```
      Host=192.168.49.2  Path=/  From=sample-deployment-d68df7d7-4pj56
      ClientIP=192.168.49.2  XFF=
      ```

LoadBalancer Service

- LoadBalancer에 할당되는 가상 IP 정적 지정
  - ✓ 서비스 운영 환경에서는 외부로부터 요청을 수신하는 IP 주소에 대해 DNS 설정 등의 이유로 고정 IP를 사용하려는 경우가 있는데 이런 경우 spec.loadBalancerIP로 외부 LoadBalancer에서 사용하는 IP 주소를 지정할 수 있음
  - ✓ sample-lb-fixip.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: sample-lb-fixip
spec:
  type: LoadBalancer
  loadBalancerIP: xxx.xxx.xxx.xxx
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 8080
    targetPort: 80
  selector:
    app: sample-app
```

LoadBalancer Service

- LoadBalancer 방화벽 정책 설정
  - ✓ LoadBalancer 서비스를 생성하면 기본적으로 전 세계(글로벌)로 공개됨
  - ✓ Google의 GKE와 AWS의 EKS에서는 LoadBalancer 서비스의 spec.loadBalancerSourceRanges에 접속을 허가하는 발신 측 네트워크를 지정하면 클러스터 밖의 외부 LoadBalancer에 클라우드 프로바이더가 제공하는 방화벽 기능을 사용하여 접속 제한을 설정할 수 있는데 지정하지 않을 경우 기본값으로 0.0.0.0/0이 지정되어 전 세계에 공개됨

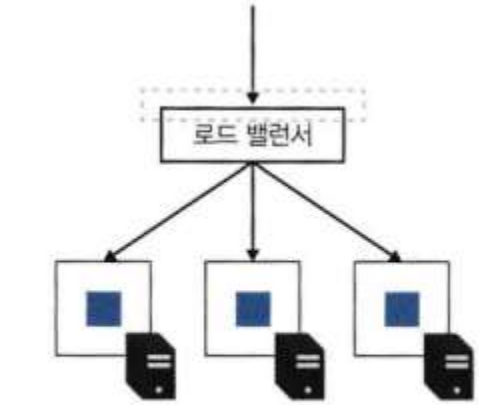
LoadBalancer Service

- LoadBalancer 방화벽 정책 설정
  - ✓ sample-lb-fw.yaml

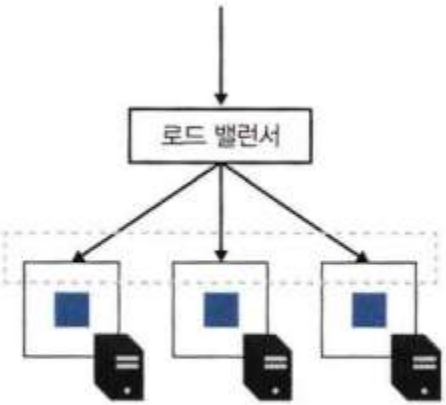
```
apiVersion: v1
kind: Service
metadata:
  name: sample-lb-fw
spec:
  type: LoadBalancer
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 8080
    targetPort: 80
  selector:
    app: sample-app
  loadBalancerSourceRanges:
  - 10.0.0.0/8
```

LoadBalancer Service

- LoadBalancer 방화벽 정책 설정
  - ✓ 외부 LoadBalancer에서 접속 제어가 구현되지 않은 쿠버네티스 환경에서 loadBalancerSourceRanges를 설정하면 쿠버네티스 노드 측의 iptables를 사용하여 접속 제어 처리가 이루어 짐
  - ✓ 세밀하게 접속 제어를 해야 하는 경우에는 NetworkPolicy 리소스를 사용하면 되는데 NetworkPolicy 리소스도 사용할 수 있는 환경이 한정되어 있지만 NetworkPolicy를 사용해도 쿠버네티스 노드의 iptables를 사용하여 접속을 제한할 수 있는데 확장성 저하나 레이턴시에 영향을 미치기 쉬우므로 가능하면 LoadBalancer 쪽에서 제한하는 것을 권장
  - ✓ LoadBalancer에서의 접속 제어와 NetworkPolicy에서의 접속 제어



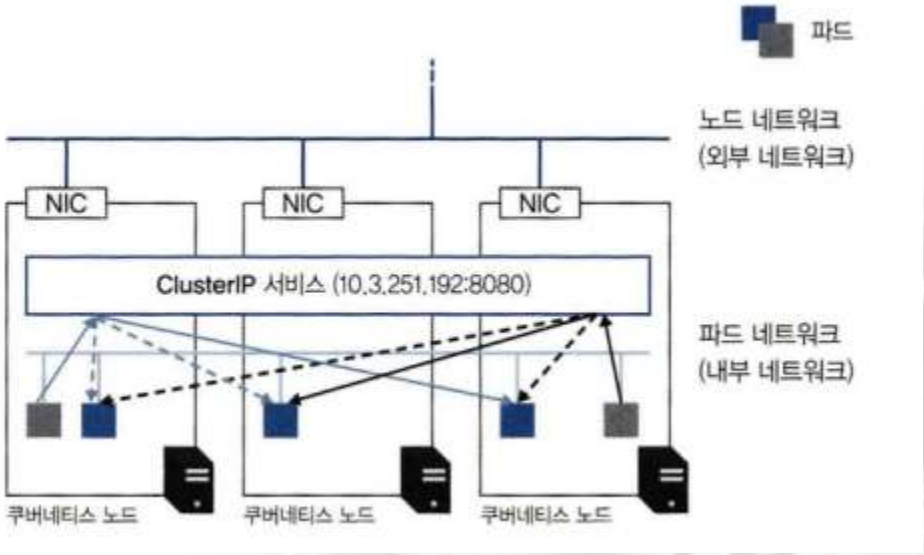
type: LoadBalancer 서비스의
loadBalancerSourceRanges를 사용한 접속 제어



NetworkPolicy를 사용한 접속 제어

그 외 서비스 기능

- ❖ Session Affinity
  - ✓ 개요
    - ClusterIP 서비스에서 활성화한 경우 파드에서 ClusterIP로 보내진 트래픽은 서비스에 연결된 어느 하나의 파드에 전송된 후 다음 트래픽도 계속 같은 파드에 보내짐
    - 서비스에 대한 설정은 spec.sessionAffinity 와 spec.sessionAffinityConfig를 사용
    - 세션 어피니티의 기능은 각 쿠버네티스 노드에 iptables로 구현되어 있으며 최대 세션 고정 시간(sessionAffinityConfig.clientIP.timeoutSeconds)을 설정할 수 있음
    - sessionAffinity 기본값은 None



그 외 서비스 기능

- Session Affinity
  - ✓ 실습
    - 서비스를 위한 yaml 파일 생성: sample-session-affinity.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: sample-session-affinity
spec:
  type: LoadBalancer
  selector:
    app: sample-app
  ports:
  - name: http-port
    protocol: TCP
    port: 8080
    targetPort: 80
```

그 외 서비스 기능

- Session Affinity
  - ✓ 실습
    - 테스트를 위한 파드 생성을 위한 yaml 파일 작성: sample-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
spec:
  containers:
  - name: tools-container
    image: amsy810/tools:v2.0
```

그 외 서비스 기능

- Session Affinity
  - ✓ 실습
    - yaml 파일 실행

      kubectl apply -f sample-session-affinity.yaml

      kubectl apply -f sample-pod.yaml
    - 확인

      kubectl exec -it sample-pod -- /bin/bash

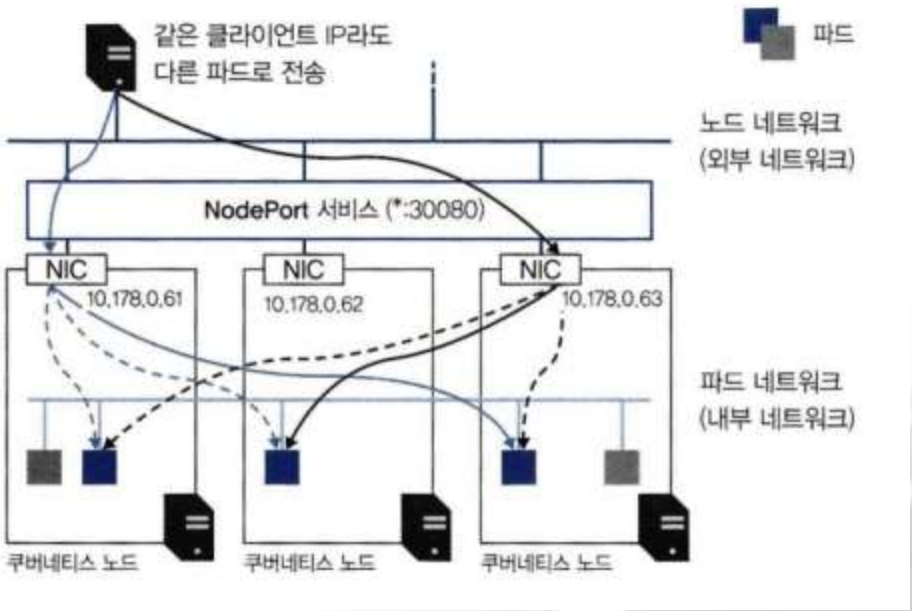
      root@sample-pod:/# curl http://sample-session-affinity.default.svc.cluster.local:8080
      Host=sample-session-affinity.default.svc.cluster.local Path=/ From=sample-deployment-d68df7d7-jjzs2 ClientIP=10.244.1.3 XFF=

      root@sample-pod:/# curl http://sample-session-affinity.default.svc.cluster.local:8080
      Host=sample-session-affinity.default.svc.cluster.local Path=/ From=sample-deployment-d68df7d7-jjzs2 ClientIP=10.244.1.3 XFF=

      root@sample-pod:/# curl http://sample-session-affinity.default.svc.cluster.local:8080
      Host=sample-session-affinity.default.svc.cluster.local Path=/ From=sample-deployment-d68df7d7-4pj56 ClientIP=10.244.1.3 XFF=

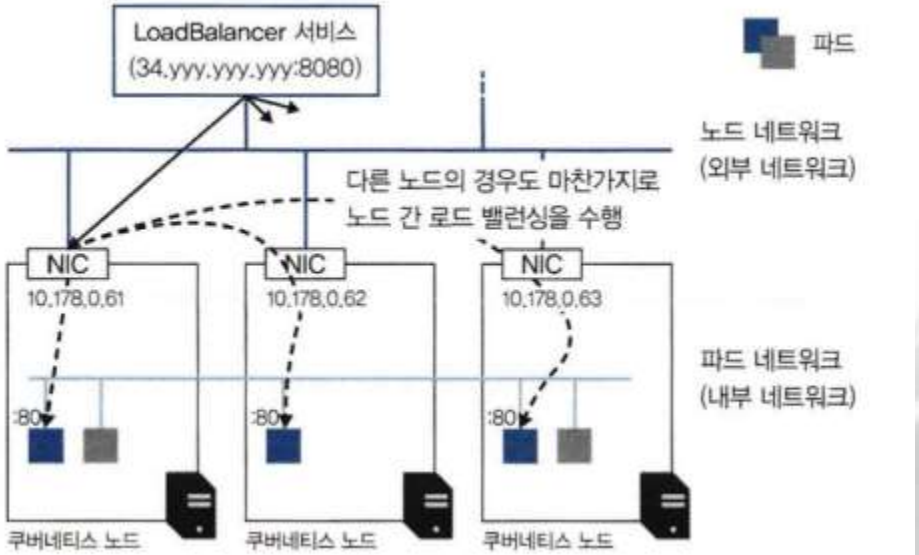
그 외 서비스 기능

- Session Affinity
  - ✓ NodePort 서비스에도 세션 어피니티를 활성화할 수 있지만 어느 쿠버네티스 노드에 전송하는지에 따라 같은 클라이언트 IP라도 같은 파드에 전송된다고 단정할 수 없으므로 주의
  - ✓ NodePort에서는 세션 어피니티 기능을 사용하는 경우가 제한적



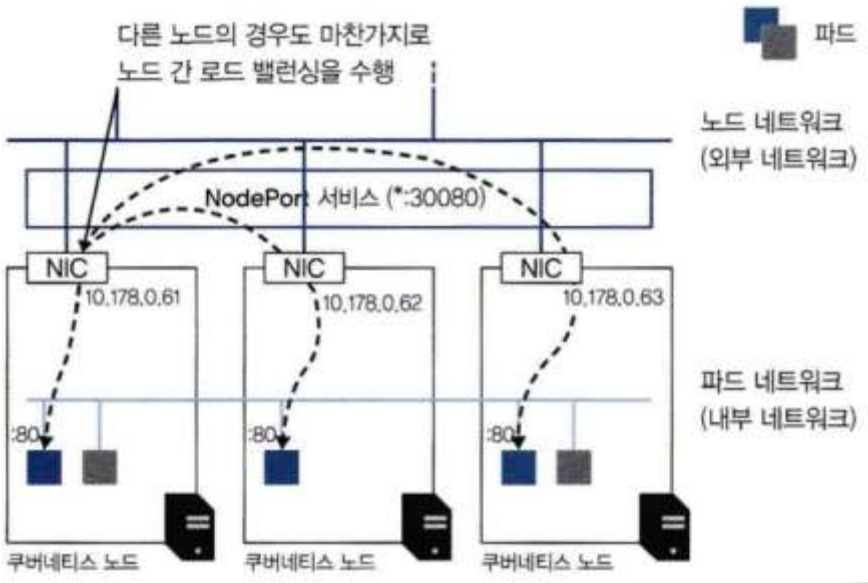
그 외 서비스 기능

- ❖ 노드 간 통신 제외와 발신 측 IP 주소 유지
  - ✓ NodePort 서비스와 LoadBalancer 서비스에서 쿠버네티스 노드에 도착한 요청은 노드를 통해 파드에도 LoadBalancing하게 되어 있으므로 불필요한 2단계 LoadBalancing이 이루어지는데 LoadBalancer 서비스에서 LoadBalancer가 LoadBalancing하여 노드에 도착한 요청은 노드를 통해 파드에도 LoadBalancing 되는 형태
  - ✓ LoadBalancer 서비스에서 2단계 LoadBalancing



그 외 서비스 기능

- ❖ 노드 간 통신 제외와 발신 측 IP 주소 유지
  - ✓ NodePort 서비스에서 2단계 LoadBalancing



  - ✓ 2단계 LoadBalancing은 균일하게 요청을 분산하기 쉽지만 불필요한 레이턴시 오버헤드가 발생하거나 밸런싱을 수행할 때 네트워크 주소 변환(Nerwork Address Translation, NAT)이 이루어져 발신 측 IP 주소가 유실되는 특징도 있음

그 외 서비스 기능

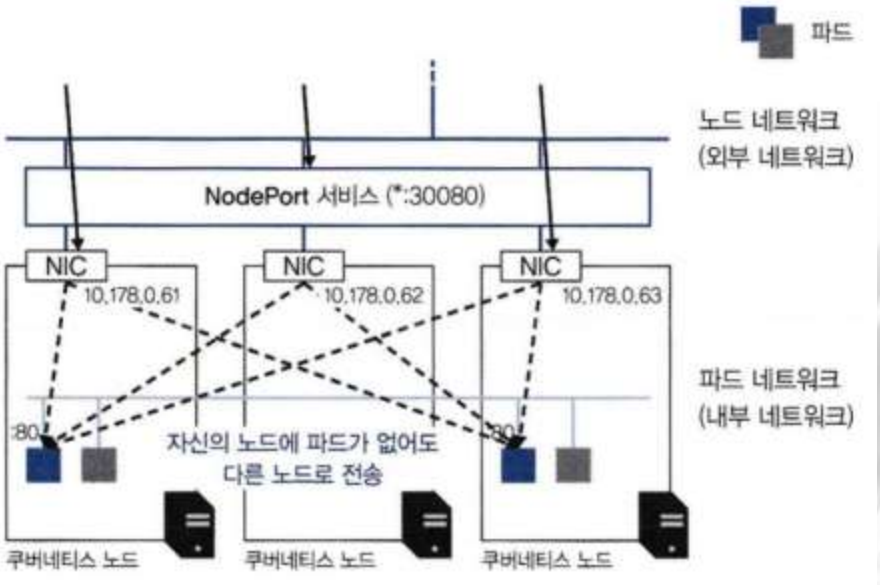
- 노드 간 통신 제외와 발신 측 IP 주소 유지
  - ✓ 파드가 가동 중인 노드 와 IP 주소 확인

```
kubectl get pods -o custom-columns="NAME:{metadata.name},Node:{spec.nodeName},NodeIP:{status.hostIP}"
```

```
NAME                                Node                NodeIP
nginx-bf5d5cf98-svrvl               ip-172-31-40-216    172.31.40.216
sample-deployment-5d9fcfcbb6-578xf  ip-172-31-40-216    172.31.40.216
sample-deployment-5d9fcfcbb6-sv4bv  ip-172-31-42-129    172.31.42.129
sample-deployment-5d9fcfcbb6-wp7t4  ip-172-31-40-216    172.31.40.216
sample-pod                          ip-172-31-42-129    172.31.42.129
```

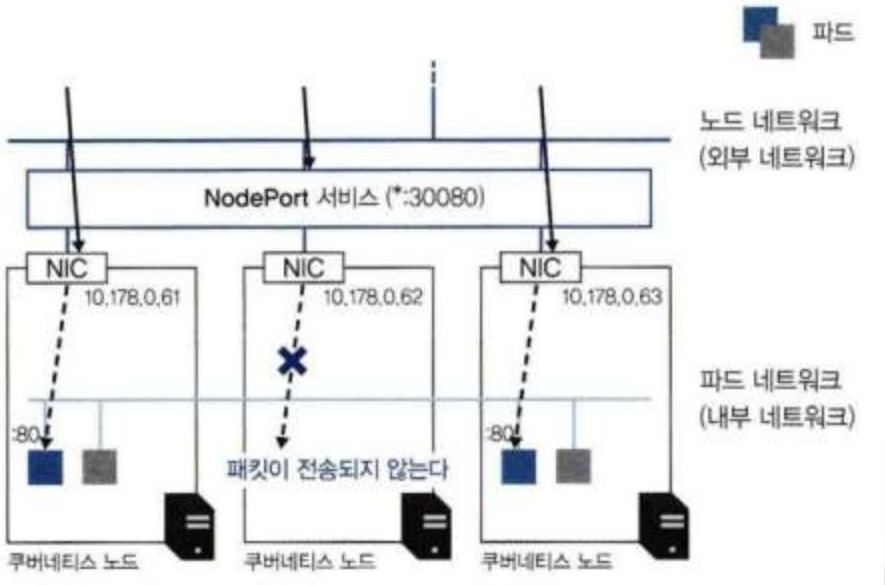
그 외 서비스 기능

- 노드 간 통신 제외와 발신 측 IP 주소 유지
  - ✓ 데몬셋은 하나의 노드에 하나의 파드가 배치되기 때문에 굳이 다른 노드의 파드에 전송하지 않고 같은 노드에만 통신하고 싶은 경우가 있는데 그런 경우 spec.externalTrafficPolicy를 사용할 수 있는데 ClusterIP 서비스 등에는 사용할 수 없음
  - ✓ ExternalTrafficPolicy 설정값
    - Cluster(기본값): 노드에 트래픽이 도착한 후 다른 노드에 있는 파드를 포함하여 다시 LoadBalancing 함으로써 파드 부하를 균등하게 분산
    - Local: 노드에 트래픽이 도착한 후 노드 간 LoadBalancing을 하지 않음
  - ✓ NodePort 서비스의 externalTrafficPolicy: Cluster 동작



그 외 서비스 기능

- ❖ 노드 간 통신 제외와 발신 측 IP 주소 유지
  - ✓ NodePort 서비스의 externalTrafficPolicy: Local 동작



그 외 서비스 기능

- ❖ 노드 간 통신 제외와 발신 측 IP 주소 유지
  - ✓ 실습
    - 리소스 파일 작성: sample-nodeport-local.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: sample-nodeport-local
spec:
  type: NodePort
  externalTrafficPolicy: Local
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 8080
    targetPort: 80
    nodePort: 30081
  selector:
    app: sample-app
```

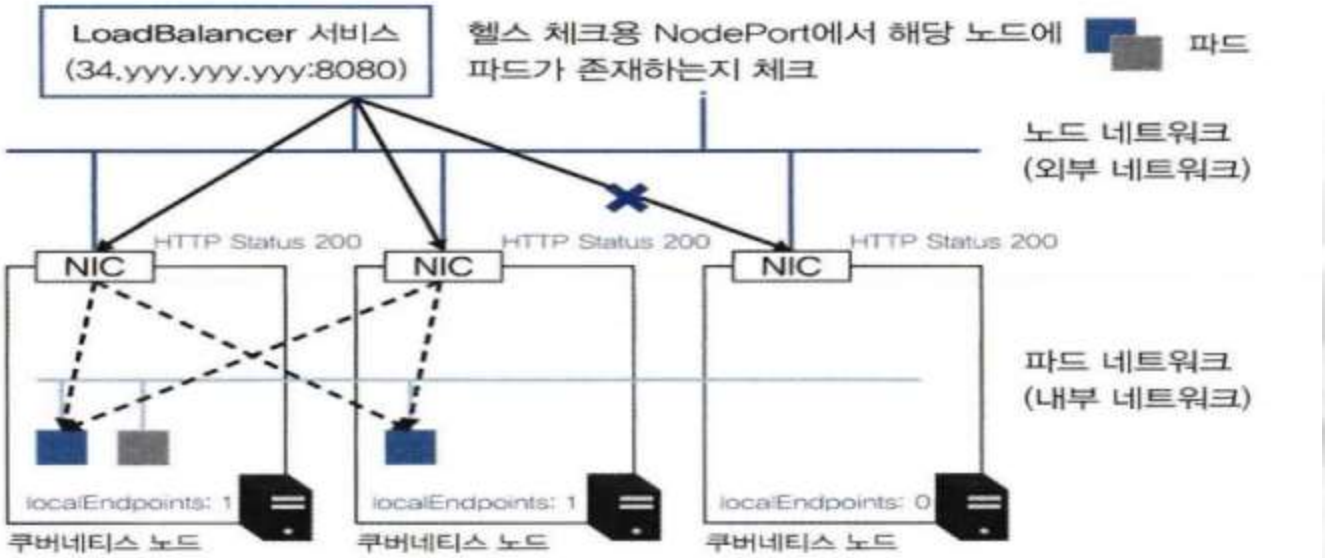
그 외 서비스 기능

- ❖ 노드 간 통신 제외와 발신 측 IP 주소 유지
  - ✓ 실습
    - 리소스 생성: kubectl apply -f sample-nodeport-local.yaml
    - 확인

```
$ curl -s http://172.31.42.129:30081
Host=172.31.42.129  Path=/  From=sample-deployment-5d9fcfcbb6-5nf2m
ClientIP=172.31.40.1  XFF=

$ curl -s http://172.31.42.129:30081
Host=172.31.42.129  Path=/  From=sample-deployment-5d9fcfcbb6-5nf2m
ClientIP=172.31.40.1  XFF=

$ curl -s http://172.31.42.129:30081
Host=172.31.42.129  Path=/  From=sample-deployment-5d9fcfcbb6-5nf2m
ClientIP=172.31.40.1  XFF=

$ curl -s http://172.31.42.129:30081
Host=172.31.42.129  Path=/  From=sample-deployment-5d9fcfcbb6-5nf2m
ClientIP=172.31.40.1  XFF=
```

그 외 서비스 기능

- 노드 간 통신 제외와 발신 측 IP 주소 유지
  - ✓ NodePort 서비스의 경우 해당 노드상에 파드가 없다면 요청에 응답할 수 없지만 LoadBalancer 서비스의 경우에는 별도로 헬스 체크용 NodePort가 할당되기 때문에 파드가 존재하지 않는 노드에는 LoadBalancer에서 요청이 전송되지 않음
  - ✓ 헬스 체크용 NodePort의 포트 번호는 spec.healthCheckNodePort에서 설정
  - ✓ 지정하지 않으면 NodePort 서비스처럼 임의의 포트 번호가 할당되고 healthCheckNodePort는 LoadBalancer 서비스 와 externalTrafficPolicy가 Local인 경우에만 설정할 수 있는 항목



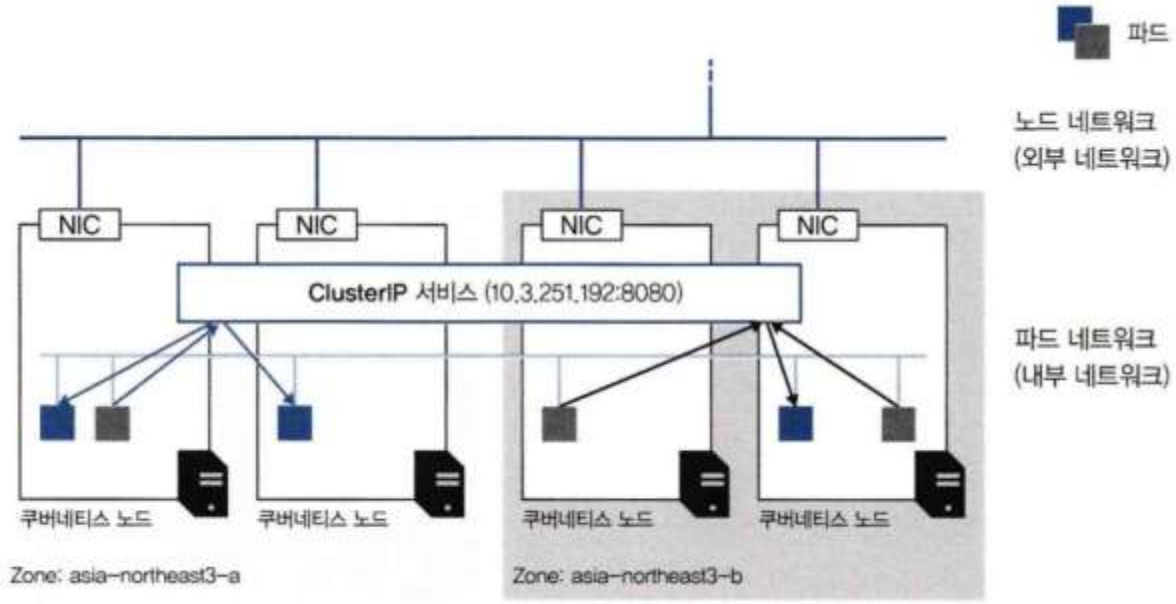
그 외 서비스 기능

- 노드 간 통신 제외와 발신 측 IP 주소 유지
  - ✓ externalTrafficPolicy를 Loca로 설정한 LoadBalancer 서비스
  - ✓ 리소스 매니페스트 파일: sample-lb-local.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: sample-lb-local
spec:
  type: LoadBalancer
  externalTrafficPolicy: Local
  healthCheckNodePort: 30086
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 8080
    targetPort: 80
    nodePort: 30085
  selector:
    app: sample-app
```

그 외 서비스 기능

- 토폴리지를 고려한 서비스 전송
  - ✓ 일반적으로 서비스 전송 대상은 리전(region)이나 가용성 영역 등의 토폴로지를 고려하지 않고 전송되는데 노드가 같은 리전이나 영역에 있으면 레이턴시는 별로 높지 않지만 멀리 떨어져 있는 경우 성능이 저하되고 노드 수가 너무 많으면 East-West 트래픽이 증가되어 결과적으로 성능이 저하되는데 이러한 문제를 해결하는 것이 Topology-aware Service Routing
  - ✓ 동일한 가용성 영역에만 트래픽을 전송하는 서비스

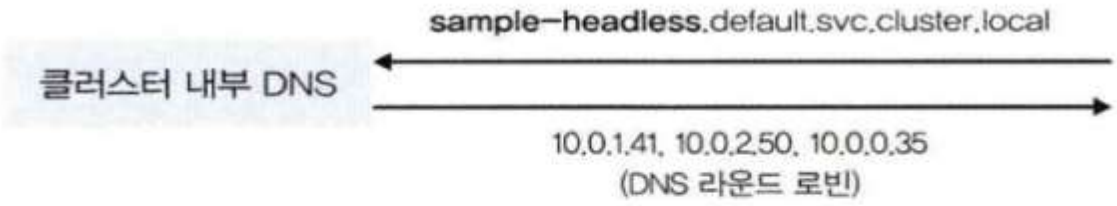


Headless Service(None)

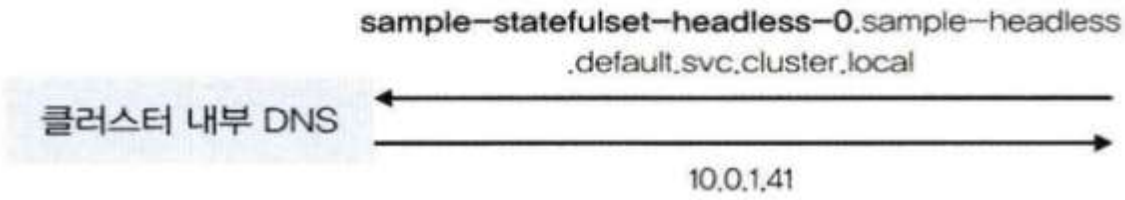
- 개요
  - ✓ 헤드리스(Headless) 서비스는 대상이 되는 개별 파드의 IP 주소가 직접 반환되는 서비스
  - ✓ 다른 서비스에서 부하 분산을 위해 제공되는 EndPoint는 가상 IP로 LoadBalancing 동작을 하기 위해 여러 파드로 전송되는 IP EndPoint
  - ✓ 헤드리스 서비스는 LoadBalancing을 하기 위한 IP 주소는 제공되지 않고 DNS 라운드 로빈(DNS RR)을 사용한 EndPoint를 제공
  - ✓ 헤드리스 서비스의 DNS 라운드 로빈에서는 목적지 파드 IP 주소가 클러스터 내부 DNS에서 반환되는 형태로 부하 분산을 하기 때문에 클라이언트에서 DNS 캐시에 주의해야 함
  - ✓ 스테이트풀셋이 헤드리스 서비스를 사용하는 경우에만 파드명으로 IP 주소를 디스커버리할 수 있는데 sample-statefulset-0 등의 파드명으로 IP 주소를 가져올 수 있음
  - ✓ 기본적으로 쿠버네티스는 파드의 IP 주소를 의식할 필요가 없게 만들어졌으므로 디플로이먼트 등과 같은 그 외 리소스의 경우에는 파드명으로 IP 주소를 가져올 수 없으며 꼭 필요한 경우에는 다른 쿠버네티스 API에서 정보를 가져와야 함

Headless Service(None)

- 개요
  - ✓ 헤드리스(Headless) 서비스 와 DNS 라운드 로빈: 헤드리스 서비스에서는 DNS 라운드 로빈으로 파드의 IP 주소가 직접 반환됨



스테이트풀셋의 경우에만 파드명으로도 이름 해석이 가능



Headless Service(None)

- 생성
  - ✓ 만족해야 하는 조건
    - 서비스의 spec.type이 ClusterIP일 것
    - 서비스의 spec.ClusterIP가 None일 것
    - 옵션
      - 서비스의 metadata.name이 스테이트풀셋의 spec.serviceName과 같을 것
      - 스테이트풀셋으로 생성된 파드명으로 디스커버리하는 경우

Headless Service(None)

- ❖ 생성
  - ✓ 실습
    - ● 헤드리스 서비스 매니페스트(sample-headless.yaml)

```
apiVersion: v1
kind: Service
metadata:
  name: sample-headless
spec:
  type: ClusterIP
  clusterIP: None
  ports:
  - name: "http-port"
    protocol: "TCP"
    port: 80
    targetPort: 80
  selector:
    app: sample-app
```

Headless Service(None)

- ❖ 생성
  - ✓ 실습
    - ● 헤드리스 서비스를 사용하기 위한 매니페스트(sample-statefulset-headless.yaml)

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: sample-statefulset-headless
spec:
  serviceName: sample-headless
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
      - name: nginx-container
        image: amsy810/echo-nginx:v2.0
```

Headless Service(None)

- ❖ 생성
  - ✓ 실습
    - 리소스 생성
      - ◆ kubectl apply -f sample-statefulset-headless.yaml
      - ◆ kubectl apply -f sample-headless.yaml
    - 서비스 이름 해석은 서비스명.네임스페이스명.svc.cluster.local로 질의
    - 헤드리스 서비스의 경우는 FQDN에 대해 정방향으로 질의하면 ClusterIP 대신 DNS 라운드 로빈에서 여러 파드의 IP 주소가 반환되기 때문에 부하 분산 때는 클라이언트 캐시 등을 고려해야 함
    - 레플리카셋 등 의 리소스에도 DNS 라운드 로빈에서 IP 주소가 반환되도록 할 수 있음

Headless Service(None)

- ❖ 생성
  - ✓ 실습
    - 파드의 IP 주소 확인: kubectl get pods -o wide

```
sample-statefulset-headless-0   1/1   Running   0   2m21s
10.244.2.18   ip-172-31-42-129   <none>   <none>
sample-statefulset-headless-1   1/1   Running   0   2m20s
10.244.2.19   ip-172-31-42-129   <none>   <none>
sample-statefulset-headless-2   1/1   Running   0   2m19s
10.244.1.30   ip-172-31-40-216   <none>   <none>
```

    - 서비스 디스커버리: kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command -- dig sample-headless.default.svc.cluster.local

```
;; ANSWER SECTION:
sample-headless.default.svc.cluster.local. 30 IN A 10.244.2.17
sample-headless.default.svc.cluster.local. 30 IN A 10.244.2.19
sample-headless.default.svc.cluster.local. 30 IN A 10.244.1.28
sample-headless.default.svc.cluster.local. 30 IN A 10.244.1.27
sample-headless.default.svc.cluster.local. 30 IN A 10.244.1.30
sample-headless.default.svc.cluster.local. 30 IN A 10.244.2.18
```

Headless Service(None)

- ❖ 생성
  - ✓ 실습
    - 파드명으로 서비스 디스커버리: kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command -- dig sample-statefulset-headless-0.sample-headless.default.svc.cluster.local

      ;; ANSWER SECTION:

      sample-statefulset-headless-0.sample-headless.default.svc.cluster.local. 30 IN A 10.244.2.18
    - 컨테이너 내부의 resolv.conf 확인: kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command -- cat /etc/resolv.conf

      search default.svc.cluster.local svc.cluster.local cluster.local ap-northeast-2.compute.internal

      nameserver 10.96.0.10

      options ndots:5

      pod "testpod" deleted

Headless Service(None)

- 스테이트풀셋 외의 파드명으로 이름 해석
  - ✓ 파드에 설정을 추가하여 파드명으로 이름 해석을 할 수도 있음
  - ✓ 파드에서 이름을 해석하려면 파드에 spec.hostname과 헤드리스 서비스명과 동일한 spec.subdomain 설정을 추가하는데 spec.hostname은 파드명이 아니어도 됨

Headless Service(None)

- ❖ 스테이트풀셋 외의 파드명으로 이름 해석
  - ✓ 실습
    - 파드명으로 이름을 해석하는 리소스 매니페스트(sample-subdomain.yaml)

```
---
apiVersion: v1
kind: Pod
metadata:
  name: sample-subdomain
  labels:
    app: sample-app
spec:
  hostname: sample-hostname
  subdomain: sample-subdomain
  containers:
  - name: nginx-container
    image: amsy810/tools:v2.0
```

Headless Service(None)

- ❖ 스테이트풀셋 외의 파드명으로 이름 해석
  - ✓ 실습
    - 리소스 생성: kubectl apply -f sample-subdomain.yaml

      pod/sample-subdomain created

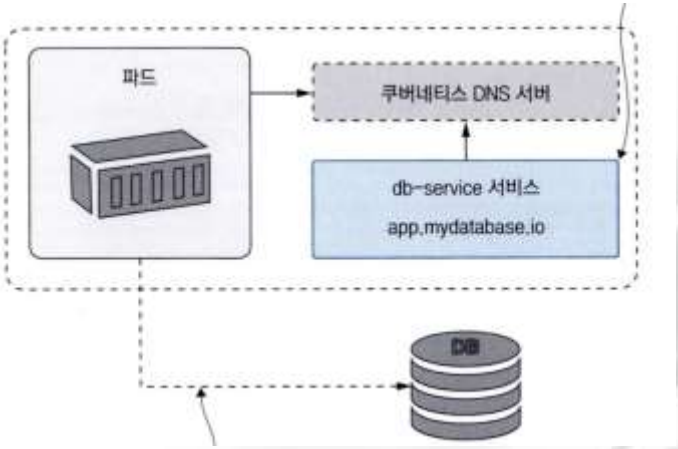
      service/sample-subdomain created
    - 파드의 IP 주소 확인: kubectl get pods -o wide

      NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
      sample-subdomain 1/1 Running 0 8s 10.244.1.33 ip-172-31-40-216 <none> <none>
    - 파드명으로 서비스 디스커버리: kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command -- dig sample-hostname.sample-subdomain.default.svc.cluster.local

      ;; ANSWER SECTION:
      sample-hostname.sample-subdomain.default.svc.cluster.local. 30 IN A 10.244.1.33

ExternalName Service

- 개요
  - ✓ 쿠버네티스는 거의 모든 서버용 소프트웨어를 실행할 수 있는데 모든 서버용 소프트웨어를 꼭 쿠버네티스에서 실행하는 것은 아님
  - ✓ 데이터베이스 같은 스토리지 컴포넌트 등이 대표적으로 쿠버네티스 외부에서 동작하는 소프트웨어
  - ✓ 매니지드 데이터베이스 서비스를 활용하거나 쿠버네티스와 통합되지 않은 시스템과 연동할 필요가 있을 수도 있음
  - ✓ 애플리케이션 아키텍처와 무관하게 클러스터 외부를 가리키는 도메인 네임 해소에도 쿠버네티스 서비스 리소스를 활용할 수 있음
  - ✓ ExternalName 서비스는 도메인 네임에 대한 별명
  - ✓ ExternalName 서비스는 애플리케이션 파드에서 로컬 네임을 사용하고 쿠버네티스 DNS 서버에 이 로컬 네임을 조회하면 외부 도메인으로 변경해 주는 방식



ExternalName Service

- 실습
  - ✓ 서비스를 위한 yaml 파일: sample-externalname.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: sample-externalname #쿠버네티스 클러스터 내부에서 사용할 도메인
  namespace: default
spec:
  type: ExternalName
  externalName: external.example.com #로컬 도메인 네임을 해소할 외부 도메인
```

ExternalName Service

- 실습
  - ✓ 서비스 생성: kubectl apply -f sample-externalname.yaml
  - ✓ 서비스 확인: kubectl get svc

    NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

    kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 19s

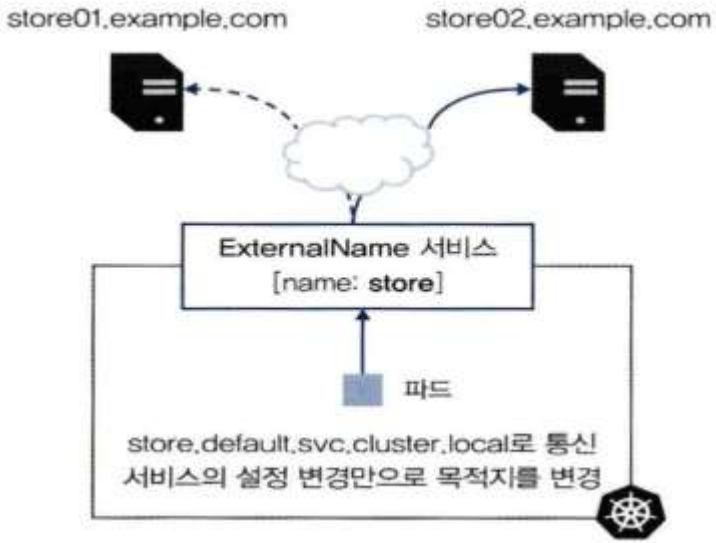
    sample-externalname ExternalName <none> external.example.com <none> 6s
  - ✓ CNAME 확인: kubectl run --image=amsy810/tools:v2.0 --restart=Never --rm -i testpod --command -- nslookup sample-externalname

    sample-externalname.default.svc.cluster.local canonical name = external.example.com.

    ** server can't find external.example.com: NXDOMAIN

ExternalName Service

- 느슨한 결합
  - ✓ 클러스터 내부에서는 파드와 통신에 서비스 이름 해석을 사용하여 느슨한 결합(loose coupling)을 유지하고 있고 SaaS나 IaaS 등 외부에 있는 서비스를 사용할 때도 가능하면 느슨한 결합으로 구성해야 함
  - ✓ 애플리케이션 등에 외부의 EndPoint를 등록해 두면 목적지가 변경되었을 때 애플리케이션 측의 설정 변경이 필요한데 ExternalName을 사용하면 목적지가 변경되어도 ExternalName 서비스를 변경하는 것만으로 가능
  - ✓ 목적지 변경에 대한 대응을 쿠버네티스 내부에서 끝낼 수 있으므로 외부 서비스와 느슨한 결합을 유지할 수 있음



ExternalName Service

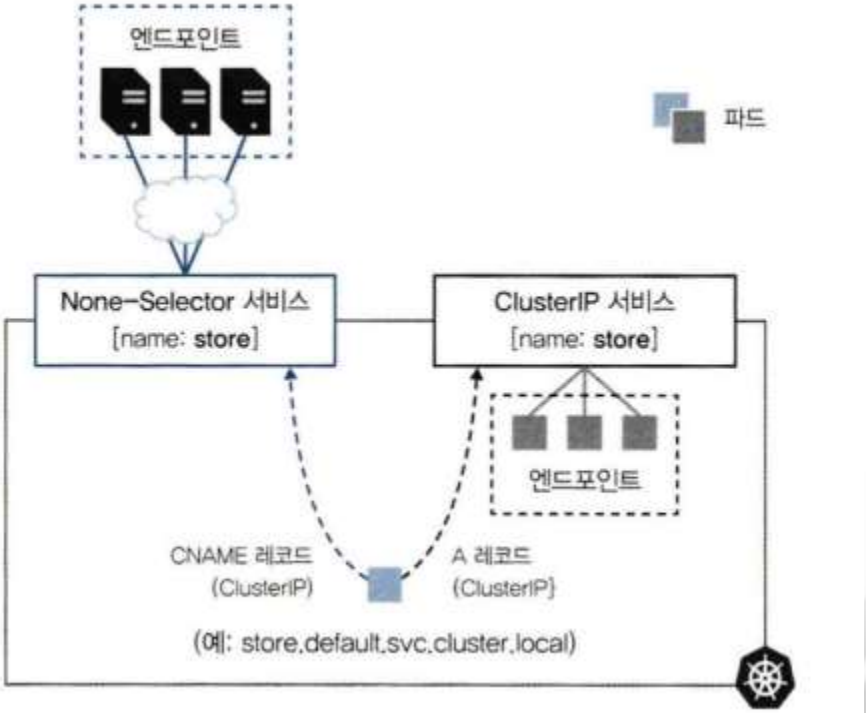
- 외부 서비스 와 내부 서비스 간의 전환
  - ✓ ExtemalName을 사용하면 외부 서비스와의 느슨한 결합을 확보하고 외부 서비스와 쿠버네티스에 배포된 클러스터 내부 서비스와의 전환도 유연하게 할 수 있음
  - ✓ 애플리케이션 측은 서비스의 FQDN(store.default.svc.cluster.local)을 지정해 두고 이름 해석이 되면 ExtemalName의 CNAME 레코드 또는 ClusterIP의 A 레코드가 반환되는 형태가 되어 애플리케이션 측은 변경 없이 내부 서비스와 외부 서비스를 전환할 수 있음
  - ✓ ClusterIP 서비스에서 ExtemalName 서비스로 전환할 경우 spec.clusterIP를 명시적으로 공란으로 만들어 두어야 함

None-Selector Service

- 개요
  - ✓ ExternalName 서비스에서는 서비스명으로 이름 해석을 하면 외부 도메인에 전송되는데 반해 None-Selector 서비스에서는 서비스명으로 이름 해석을 하면 자신이 설정한 멤버에 대해 LoadBalancing을 수행
  - ✓ 쿠버네티스 내에 원하는 목적지로 LoadBalancer를 생성할 수 있는 기능
  - ✓ 클라이언트 사이드 LoadBalancing용 EndPoint를 제공하는 서비스라고 할 수 있음
  - ✓ externalName을 지정하지 않고 셀렉터가 존재하지 않는 서비스를 생성한 후 EndPoint 리소스를 수동으로 만들면 유연한 서비스를 만들 수 있는데 쿠버네티스 클러스터 외부의 애플리케이션 서버에 대해 요청을 분산하는 경우에도 쿠버네티스 서비스를 사용할 수 있음
  - ✓ 서비스 환경과 스테이징 환경에서 클러스터 외부와 클러스터 내부 서비스를 분리했다 해도 애플리케이션에서 같은 ClusterIP로 보여줄 수 있음
  - ✓ 외부로 요청을 보내기 위한 서비스로 ExternalName 서비스와 거의 비슷하지 ExternalName 서비스는 CNAME이 반환되는 반면 None-Selector 서비스는 ClusterIP의 A 레코드가 반환되기 때문에 쿠버네티스 서비스를 사용하는 것처럼 외부용 LoadBalancing을 할 수 있음

None-Selector Service

❖ 개요



None-Selector Service

- 서비스 생성
  - ✓ sample-none-selector.yaml

```
---
apiVersion: v1
kind: Service
metadata:
  name: sample-none-selector
spec:
  type: ClusterIP
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 80
---
apiVersion: v1
kind: EndPoints
metadata:
  name: sample-none-selector
```

  - ✓ kubectl apply -f sample-none-selector.yaml

None-Selector Service

- 서비스 확인
  - ✓ kubectl describe svc sample-none-selector

```
Name:              sample-none-selector
Namespace:         default
Labels:            <none>
Annotations:       <none>
Selector:          <none>
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.104.157.206
IPs:               10.104.157.206
Port:              <unset>  8080/TCP
TargetPort:        80/TCP
EndPoints:         192.168.1.1:80,192.168.1.2:80
Session Affinity:  None
Events:            <none>
```

None-Selector Service

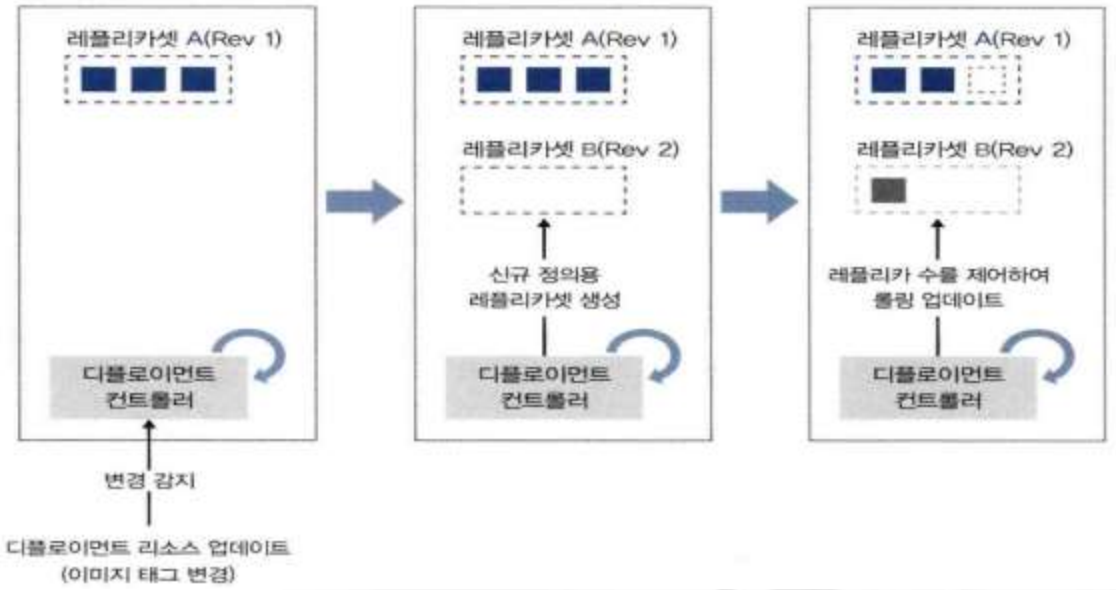
- 대체
  - ✓ 쿠버네티스 1.18부터는 IPv6 대응/토폴로지를 고려한 라우팅/대규모 환경에서의 확장성 등을 구현하기 위해 EndPoint 리소스 대신 EndPointSlice 리소스가 도입되기 시작
  - ✓ 당분간은 EndPoint 리소스와 EndPointSlice 리소스가 공존하겠지만 앞으로는 EndPointSlice로 대체될 것

Ingress

- 개요
  - ✓ 서비스는 외부에서 들어오는 트래픽을 쿠버네티스 안으로 인도하는 역할을 수행
  - ✓ 애플리케이션에 외부 사용자가 접근할 수 있도록 LoadBalancer 서비스를 사용했는데 이 방법은 애플리케이션마다 IP 주소를 따로 부여하고 부여된 주소와 애플리케이션 관계를 DNS 서버에서 제공해야 하기 때문에 관리적인 문제가 발생하기 쉬움
  - ✓ 인입되는 트래픽을 올바른 애플리케이션으로 연결하는 일은 라우팅과 관련된 문제이지만 인그레스(Ingress)를 사용하면 이를 쿠버네티스 안에서도 다룰 수 있음
  - ✓ 인그레스는 일련의 규칙에 따라 도메인 네임과 애플리케이션의 요청 경로를 매핑해 주는 역할을 수행
  - ✓ 하나의 공인 IP만으로도 전체 클러스터에서 동작하는 모든 애플리케이션까지 트래픽을 정확히 라우팅할 수 있음
  - ✓ 도메인 네임을 이용한 라우팅과 관련된 문제는 리버스 프록시(reverse proxy)로 거의 해결할 수 있는데 쿠버네티스의 인그레스는 플러그인이 가능한 아키텍처가 적용되어 있음
  - ✓ 라우팅 규칙을 일반적인 쿠버네티스 리소스로 정의하고 리버스 프록시로 사용할 컴포넌트에 이 규칙에 따라 트래픽을 처리하도록 하는 구조

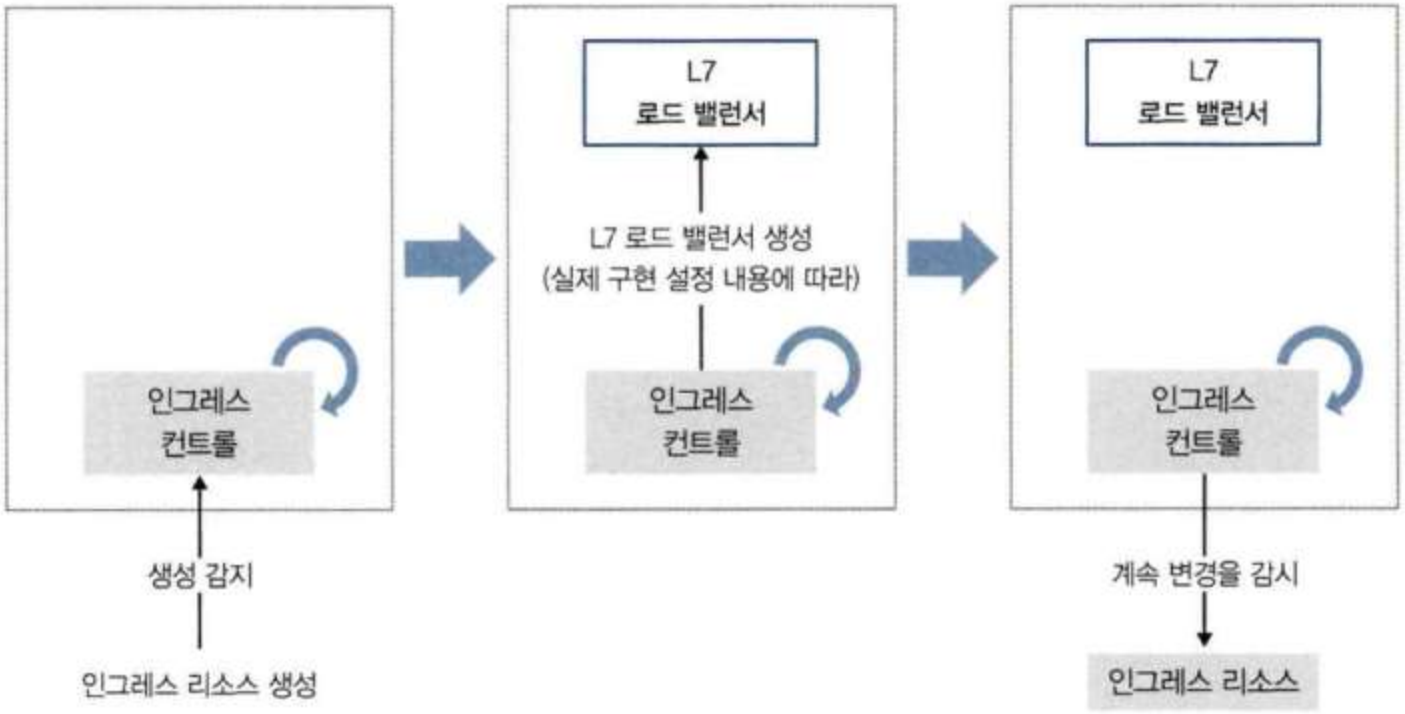
Ingress

- Resource 와 Controller
  - ✓ 쿠버네티스는 분산 시스템이며 매니페스트로 정의한 리소스를 쿠버네티스에 등록하는 것으로 시작되는데 등록만으로는 아무런 처리가 이루어지지 않으며 실제 처리를 하는 컨트롤러라는 시스템 구성 요소가 필요
  - ✓ 디플로이먼트에서는 레플리카셋을 생성하거나 레플리카 수를 변경해 가면서 롤링 업데이트를 하는 디플로이먼트 컨트롤러라고 하는 시스템 구성 요소가 쿠버네티스 클러스터에서 동작
  - ✓ 디플로이먼트 컨트롤러가 없는 경우 매니페스트로 디플로이먼트 리소스를 생성해도 레플리카셋은 생성되지 않는데 이처럼 각 리소스는 등록된 후 여러 컨트롤러가 실제로 처리를 함으로써 시스템이 동작하게 되어 있음



Ingress

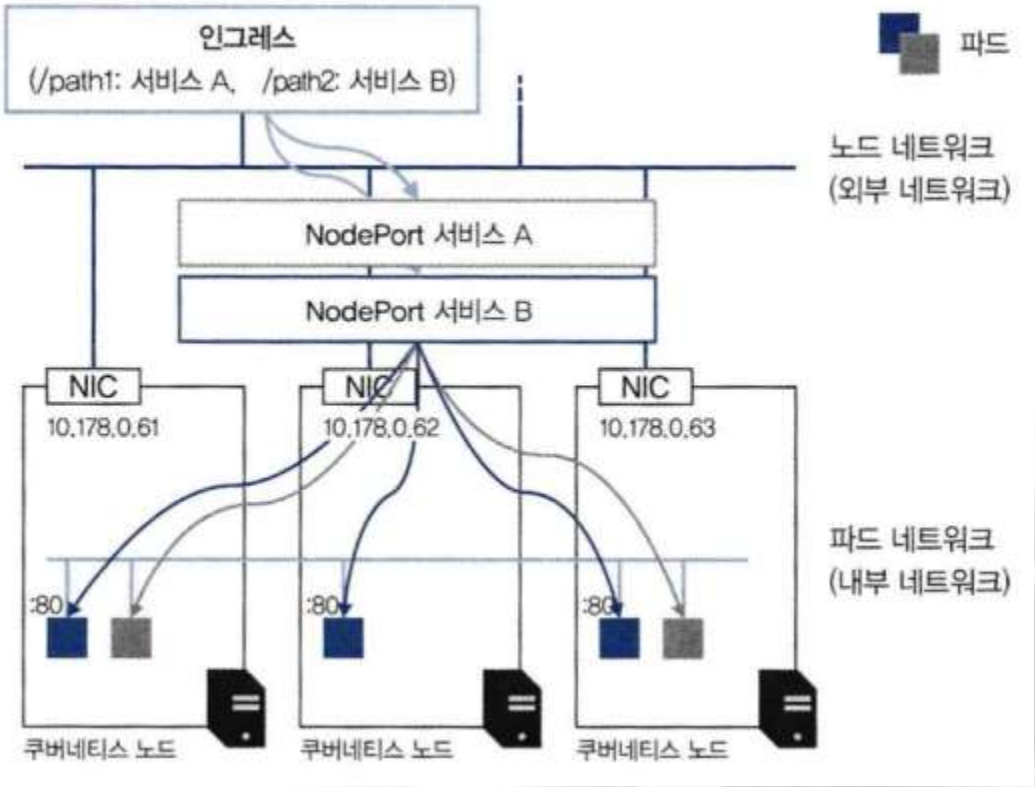
- Ingress Resource 와 Ingress Controller
  - ✓ 인그레스가 실제로 가리키는 것은 다양한 개념의 집합
  - ✓ 인그레스 리소스란 매니페스트에 등록된 API 리소스를 의미하고 인그레스 컨트롤러는 인그레스 리소스가 쿠버네티스에 등록되었을 때 어떤 처리를 수행하는 것
  - ✓ 처리의 예로는 GCP의 GCLB를 조작하여 L7 로드 밸런서 설정을 하거나 Nginx 설정을 변경하여 리로드를 하는 등의 것



Ingress

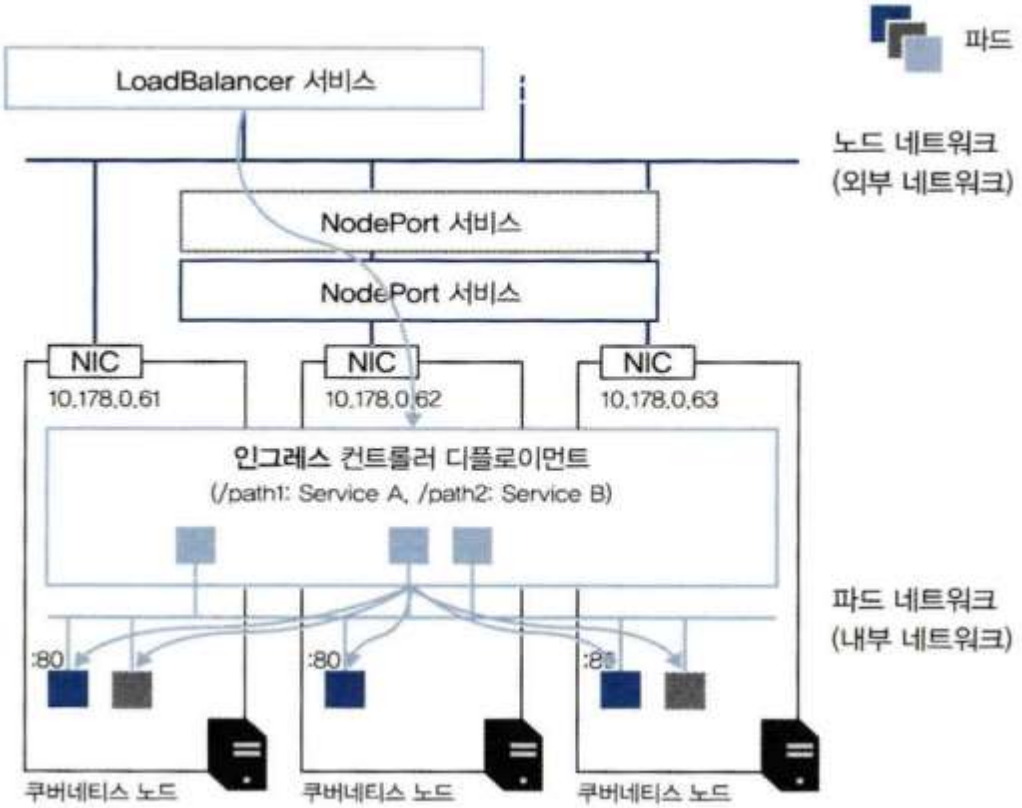
❖ 종류

✓ 클러스터 외부 LoadBalancer를 사용한 인그레스



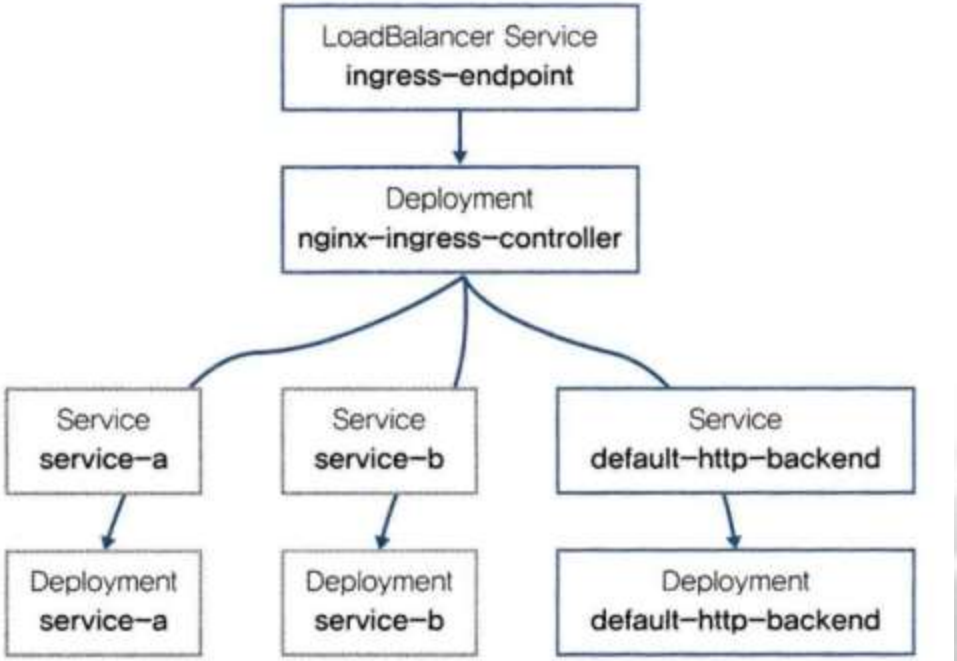
Ingress

- 종류
  - ✓ 클러스터 내부에 인그레스용 파드를 배포하는 인그레스
    - 내부 인그레스를 사용하는 경우에는 내부 인그레스 컨트롤러를 배포해야 하는데 내부 인그레스에서는 인그레스 컨트롤러 자체가 L7 수준의 처리를 하는 파드이기도 하므로 이름은 컨트롤러이지만 실제 처리도 수행



Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포
  - ✓ Nginx 인그레스 컨트롤러 구성



Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 클러스터 내부에 인그레스용 파드를 배포하는 인그레스 설치: kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.6.4/deploy/static/provider/cloud/deploy.yaml
    - nginx ingress controller가 설치되면서 k8s object들이 배포됨(pod, service, delpoyment, replicaset, job.batch)

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 설치 확인: kubectl get all -n ingress-nginx

```
NAME                                        READY   STATUS      RESTARTS   AGE
pod/ingress-nginx-admission-create-8bf8j    0/1     Completed   0          16h
pod/ingress-nginx-admission-patch-v7cmz     0/1     Completed   0          16h
pod/ingress-nginx-controller-68949dbd99-q6fn7   1/1     Running     0          16h

NAME                                         TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)
AGE
service/ingress-nginx-controller             LoadBalancer   10.96.16.210    192.168.49.2
80:30572/TCP,443:32205/TCP   16h
service/ingress-nginx-controller-admission   ClusterIP      10.106.54.237   <none>
443/TCP                      16h

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ingress-nginx-controller   1/1     1            1           16h
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 설치 확인: kubectl get all -n ingress-nginx

```
NAME                                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/ingress-nginx-controller-68949dbd99   1         1         1       16h

NAME                                       STATUS     COMPLETIONS   DURATION   AGE
job.batch/ingress-nginx-admission-create   Complete   1/1           8s         16h
job.batch/ingress-nginx-admission-patch    Complete   1/1           24s        16h
```

    - ingress-nginx-controller service가 생성되고 Port가 할당되며 ingress-nginx-controller pod가 running 상태이어야 함

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ Ingress 생성 및 확인
    - Deployment 생성(deploy-test.yaml)

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: service-test
spec:
  replicas: 3
  selector:
    matchLabels:
      app: service_test_pod
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ Ingress 생성 및 확인
    - Deployment 생성(deploy-test.yaml)

```
template:
  metadata:
    labels:
      app: service_test_pod
  spec:
    containers:
    - name: simple-http
      image: python:2.7
      imagePullPolicy: IfNotPresent
      command: ["/bin/bash"]
      args: ["-c", "echo '<p>Hello from $(hostname)</p>' > index.html; python -m SimpleHTTPServer 8080"]
      ports:
      - name: http
        containerPort: 8080
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ Ingress 생성
    - Service 생성(svc-test.yaml)

```
apiVersion: v1
kind: Service
metadata:
  name: service-test
spec:
  selector:
    app: service_test_pod
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ Ingress 생성
    - Ingress 생성(ingress-test.yaml)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-nginx
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    kubernetes.io/ingress.class: "nginx"
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ Ingress 생성
    - Ingress 생성(ingress-test.yaml)

```
spec:
  rules:
  - host: "ingress.test.com"
    http:
      paths:
      - pathType: Prefix
        path: /test
        backend:
          service:
            name: service-test
            port:
              number: 80
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ Ingress 생성
    - Ingress 생성(ingress-test.yaml)
      - ◆ ingress.test.com/test로 요청이오면 -> service-test라는 서비스에서 요청을 받고 Service에 작성된 Pod로 요청이 되어 해당 컨테이너에서 응답을 해주는 형태
      - ◆ nginx-ingress-controller를 설치할 경우 기본적으로 오브젝트들의 라벨명이 ingress-nginx로 되어있기 때문에 사용자가 배포할 Ingress의 이름을 기본적으로 ingress-nginx로 해주어야 매핑됨
      - ◆ 다른 이름으로 작성할 경우 nginx-ingress-controller 설치 시 배포될 오브젝트의 yaml파일에 Label을 전부 해당 이름에 맞게 수정해주어야 함

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ Ingress 생성
    - Ingress 확인
      - local PC에서 /etc/hosts파일에 wokernode의 ip와 ingress에 명시한 host명을 작성

        # /etc/hosts

        10.0.1.82 ingress.test.com
      - 포트 매핑 확인

        kubectl get svc -n ingress-nginx

        NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

        ingress-nginx-controller LoadBalancer 10.96.16.210 192.168.49.2 80:30572/TCP,443:32205/TCP 21h

        ingress-nginx-controller-admission ClusterIP 10.106.54.237 <none> 443/TCP 21h

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ Ingress 생성
    - Ingress 확인
      - ◆ 요청: curl -s http://ingress.test.com:30572/test

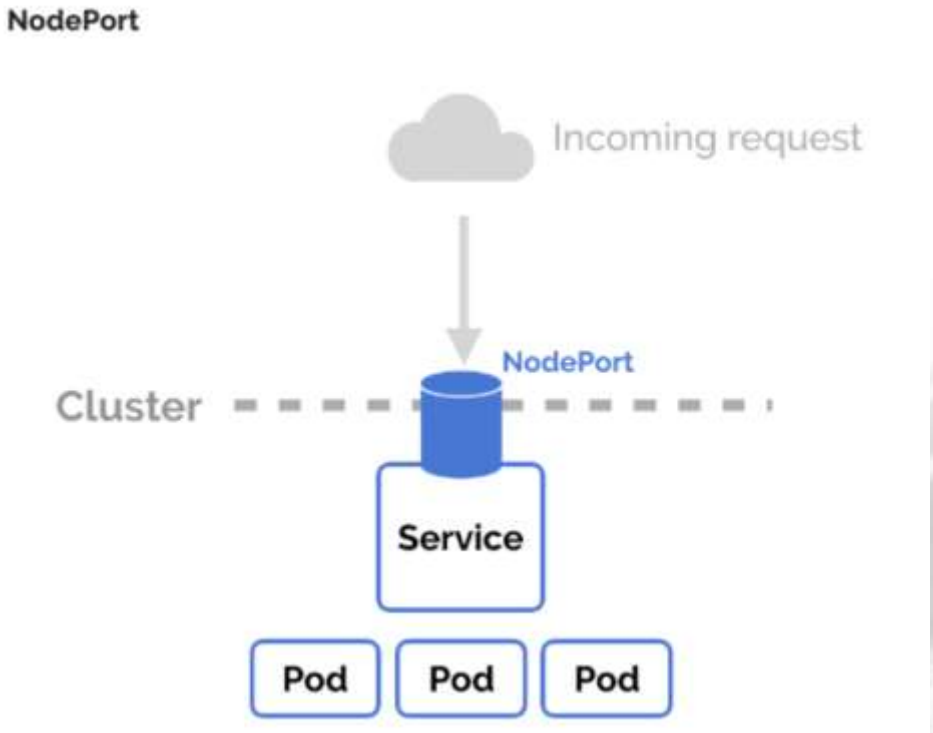
        Hello from $(hostname)

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 정리
    - 기본적으로 k8s-cluster에서 외부 접근을 허용하려면 pod에 연결된 service를 NodePort 타입으로 생성해주면 해당 서비스에 생성된 NodePort를 통해 워커노드의 ip와 해당 port로 컨테이너에 접근이 가능해지지만 Ingress를 사용하면 단일 지점 요청을 통한 외부 접근 허용 및 라우팅이 가능해지기 때문에 현업에선 Ingress를 주로 사용한다.

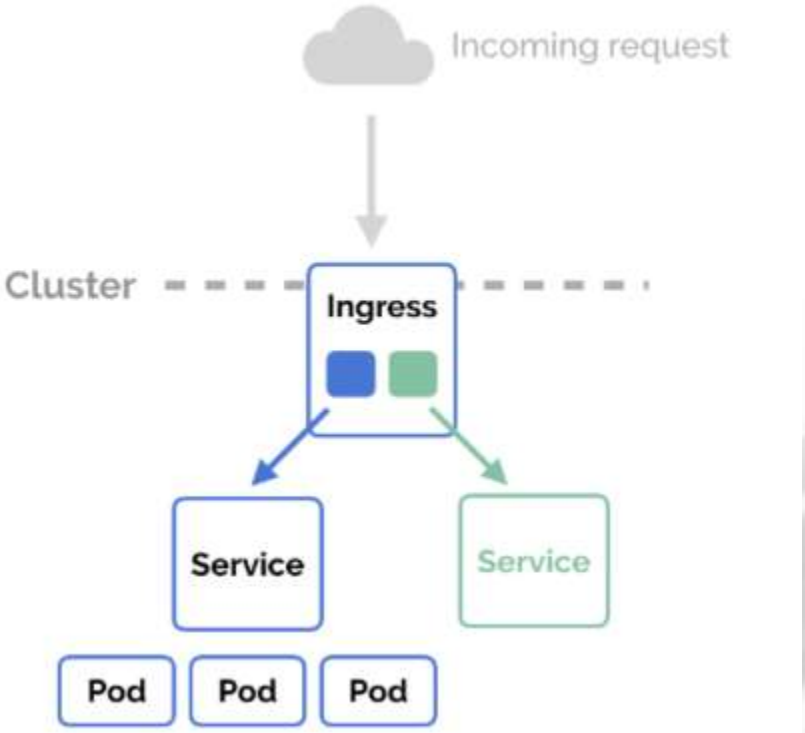
Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 정리
    - NodePort를 이용한 외부 접근 흐름



Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 정리
    - Ingress를 이용한 외부 접근 흐름



Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - Ingress 실습을 위한 리소스 파일(ingress-resource.yaml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend1
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend1
  template:
    metadata:
      labels:
        app: backend1
    spec:
      containers:
      - name: backend1
        image: nginxdemos/nginx-hello:plain-text
        ports:
        - containerPort: 8080
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - Ingress 실습을 위한 리소스 파일(ingress-resource.yaml)

```
---
apiVersion: v1
kind: Service
metadata:
  name: backend1-svc
spec:
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
    name: http
  selector:
    app: backend1
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - Ingress 실습을 위한 리소스 파일(ingress-resource.yaml)

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend2
  template:
    metadata:
      labels:
        app: backend2
    spec:
      containers:
      - name: backend2
        image: nginxdemos/nginx-hello:plain-text
        ports:
        - containerPort: 8080
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - Ingress 실습을 위한 리소스 파일(ingress-resource.yaml)

```
---
apiVersion: v1
kind: Service
metadata:
  name: backend2-svc
  labels:
spec:
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
    name: http
  selector:
    app: backend2
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - 리소스 생성: kubectl apply -f ingress-resource.yaml
    - 리소스 확인: kubectl get pod,service

```
NAME                                READY   STATUS    RESTARTS   AGE
pod/backend1-656f7c6885-9brnv       1/1     Running   0          20m
pod/backend1-656f7c6885-pb9ps       1/1     Running   0          20m
pod/backend2-6c84bb867-5q2dw        1/1     Running   0          20m

NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
service/backend1-svc    ClusterIP   10.111.78.75     <none>        80/TCP
20m
service/backend2-svc    ClusterIP   10.110.165.49    <none>        80/TCP
20m
service/kubernetes      ClusterIP   10.96.0.1        <none>        443/TCP    32h
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - Host 기반 NGINX Ingress Controller 라우팅 설정(ingress-test.yaml)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-nginx
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    kubernetes.io/ingress.class: "nginx"
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - Host 기반 NGINX Ingress Controller 라우팅 설정(ingress-test.yaml)

```
spec:
  rules:
  - host: example1.com
    http:
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: backend1-svc
              port:
                number: 80
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - Host 기반 NGINX Ingress Controller 라우팅 설정(ingress-test.yaml)

```
- host: example2.com
  http:
    paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: backend2-svc
            port:
              number: 80
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - 배포: kubectl apply -f ingress-test.yaml
    - 리소스 배포 확인: kubectl get ingress

      NAME CLASS HOSTS ADDRESS PORTS AGE

      ingress-nginx <none> example1.com,example2.com 192.168.49.2 80 9h
    - 노드 포트 확인: kubectl get svc -n ingress-nginx

      NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

      ingress-nginx-controller LoadBalancer 10.96.16.210 192.168.49.2 80:30572/TCP,443:32205/TCP 25h

      ingress-nginx-controller-admission ClusterIP 10.106.54.237 <none> 443/TCP 25h

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - 도메인 과 IP 주소 매핑(/etc/hosts)

      172.31.40.216 example1.com

      172.31.40.216 example2.com

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - 요청 수행

curl example2.com:30572

Server address: 10.244.1.40:8080

Server name: backend2-6c84bb867-5q2dw

Date: 15/Jul/2025:08:32:00 +0000

URI: /

Request ID: f76ec40e9c7d66788e4a6cd6eeef070a

curl example1.com:30572

Server address: 10.244.2.25:8080

Server name: backend1-656f7c6885-9brnv

Date: 15/Jul/2025:08:32:04 +0000

URI: /

Request ID: de7a814a7c2d9f3d168e54bcd9046314

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - Path 기반 NGINX Ingress Controller 라우팅 설정(ingress-test.yaml)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-nginx
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    kubernetes.io/ingress.class: "nginx"
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - Path 기반 NGINX Ingress Controller 라우팅 설정(ingress-test.yaml)

```
spec:
  rules:
  - host: example.com
    http:
      paths:
        - path: /backend1
          pathType: Prefix
          backend:
            service:
              name: backend1-svc
              port:
                number: 80
        - path: /backend2
          pathType: Prefix
          backend:
            service:
              name: backend2-svc
              port:
                number: 80
```

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - 배포: kubectl apply -f ingress-test.yaml
    - 리소스 배포 확인: kubectl get ingress

      NAME CLASS HOSTS ADDRESS PORTS AGE

      ingress-nginx <none> example.com 192.168.49.2 80 9h
    - 노드 포트 확인: kubectl get svc -n ingress-nginx

      NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

      ingress-nginx-controller LoadBalancer 10.96.16.210 192.168.49.2 80:30572/TCP,443:32205/TCP 25h

      ingress-nginx-controller-admission ClusterIP 10.106.54.237 <none> 443/TCP 25h

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - 도메인 과 IP 주소 매핑(/etc/hosts)

      172.31.40.216 example.com

Ingress

- Nginx 인그레스 컨트롤러를 이용한 배포(https://kubernetes.io/docs/concepts/services-networking/ingress/)
  - ✓ 실습
    - 요청 수행

curl example.com:30572/backend1

Server address: 10.244.1.40:8080

Server name: backend2-6c84bb867-5q2dw

Date: 15/Jul/2025:08:32:00 +0000

URI: /

Request ID: f76ec40e9c7d66788e4a6cd6eeef070a

curl example.com:30572/backend2

Server address: 10.244.2.25:8080

Server name: backend1-656f7c6885-9brnv

Date: 15/Jul/2025:08:32:04 +0000

URI: /

Request ID: de7a814a7c2d9f3d168e54bcd9046314