5. 서비스: 클라이언트가 파드를 검색하고 통신을 가능하게 한다.

대다수의 애플리케이션은 외부 요청에 응답하기 위한 것 → MSA 경우, 클러스터 내부의 다른 파드 및 클러스터 외부의 클라이언트에서 오는 HTTP 요청에 응답한다.

- 파드가 다른 파드에게 제공하는 서비스를 사용하려면 다른 파드를 찾는 방법이 필요하다.

그럼, 통신하고자 하는 서버의 정확한 IP주소나 호스트 이름을 지정해 애플리케이션을 구성하면 어떨까?

- 당연히 쿠버네티스에선 정상적으로 작동하기 힘들다.
  - 파드가 일시적이다. 레플리카셋 등 봐서 알겠지만, 파드는 여러 조건에 의해 사라지기도 하고, 새롭게 생겨나기도 한다. 또한 이동도 자유롭다.
  - 쿠버네티스는 파드가 시작되기 바로 전에 파드의 IP 주소를 할당한다. 즉, 클라이언트는 서버 파드의 IP주소를 미리 알 수 없다.
  - 파드의 스케일링을 하면 파드의 IP가 엄청 늘어날텐데, 그 목록을 다 가지고 있을 수 없으며, 클라이언트는 단일 IP주소로 엑세스 할 수 있어야 한다.
- 이러한 문제를 해결하기 위해 서비스를 이용한다.

서비스

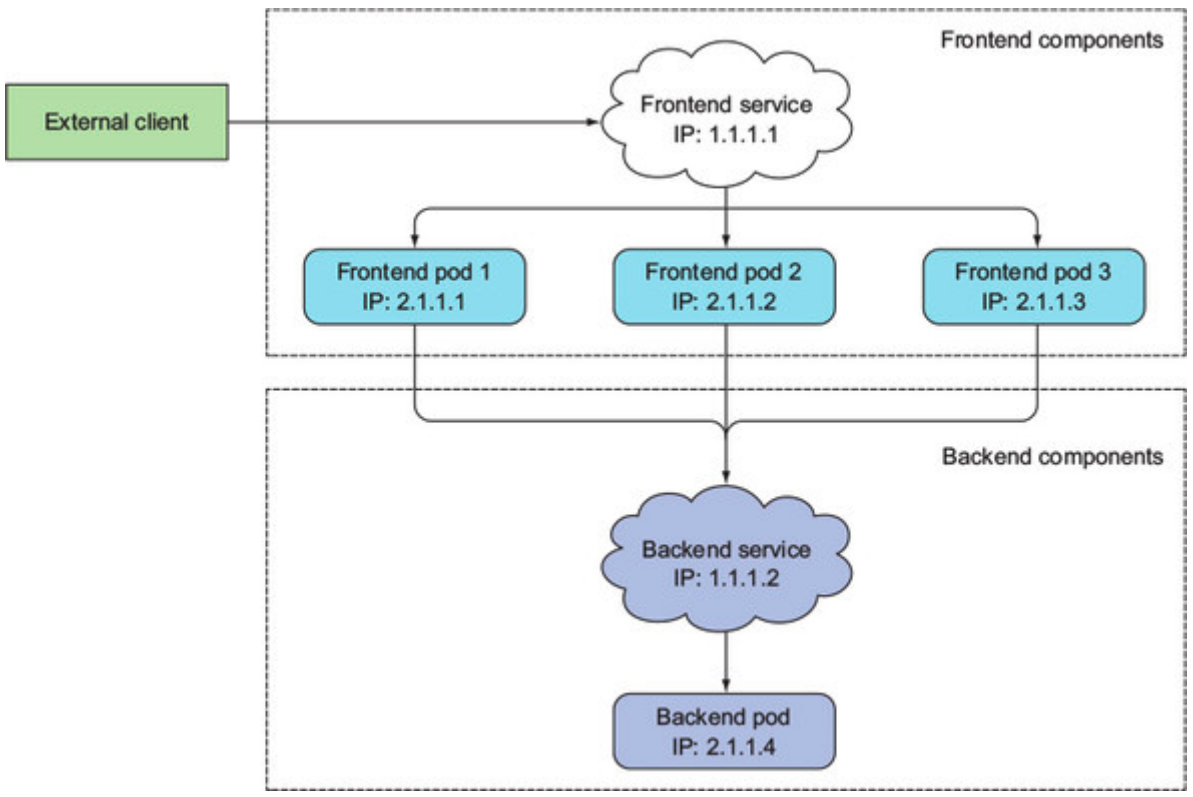
동일한 서비스를 제공하기 위해 파드 그룹에 **지속적인 단일 접점**을 만들려고 할 때 생성하는 리소스

- 서비스가 존재하는 동안 절대 바뀌지 않는 IP주소와 포트가 존재
- 클라이언트는 여기에 접속하면 지원하는 파드 중 하나로 연결됨
- 클라이언트는 서비스를 제공하는 개별 파드의 위치를 알필요가 없어, 파드는 언제든지 클러스터 내에서 이동이 가능.

예제

프론트엔드 웹서버와 백엔드 데이터베이스 서버

- 프론트엔드 서버에 대한 서비스를 만들고 외부에서 엑세스할 수 있도록 구성 → 하나의 고정 IP주소가 노출됨.
- 백엔드 서버에 대한 서비스를 생성해 안정적인 주소 생성 (파드의 IP주소가 바뀌더라도 서비스는 안바뀜)
  - 또한, 서비스를 생성하면 프론트엔드 서버에서 환경변수/DNS 이름으로 쉽게 백엔드 서비스를 찾을 수 있다.



서비스 생성

그림에서 보다시피 서비스를 지원하는 여러개의 파드가 존재할 수 있다. → 서비스를 어떻게 지원하도록 (연결하도록) 할까?

- 레이블 셀렉터 (파드 셀렉터)를 이용하자. ⇒ 동일한 세트에 속하는 파드를 지정

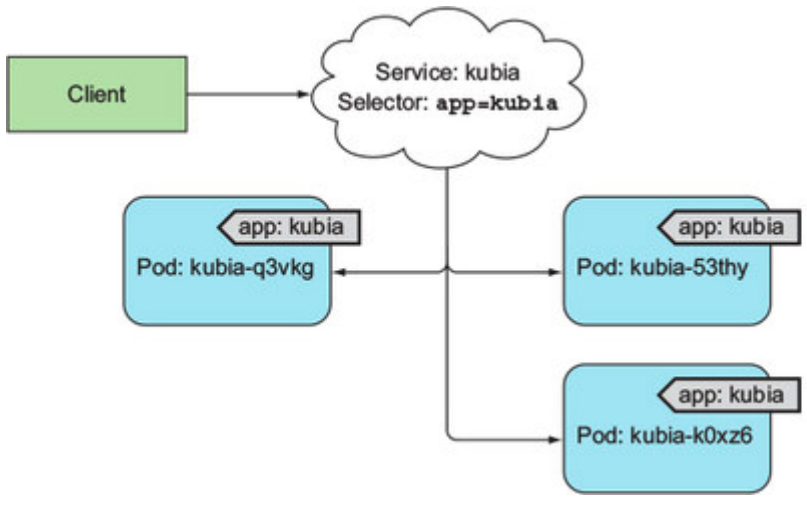


Figure 5.2 Label selectors determine which pods belong to the Service.

YAML을 사용해 서비스를 생성

```
apiVersion: v1
kind: Service
metadata:
  name: kubia
spec:
  ports:
  - port: 80             # 서비스가 사용될 포트 (input)
    targetPort: 8080     # 서비스가 포워드할 컨테이너 포트 (output)
  selector:              # 레이블 셀렉터
    app: kubia
```

`kubectl get svc` 를 통해 상태를 확인할 수 있다.

```
$ kubectl get svc
NAME         CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
kubernetes   10.111.240.1     <none>        443/TCP   30d
kubia        10.111.249.153   <none>        80/TCP    6m
```

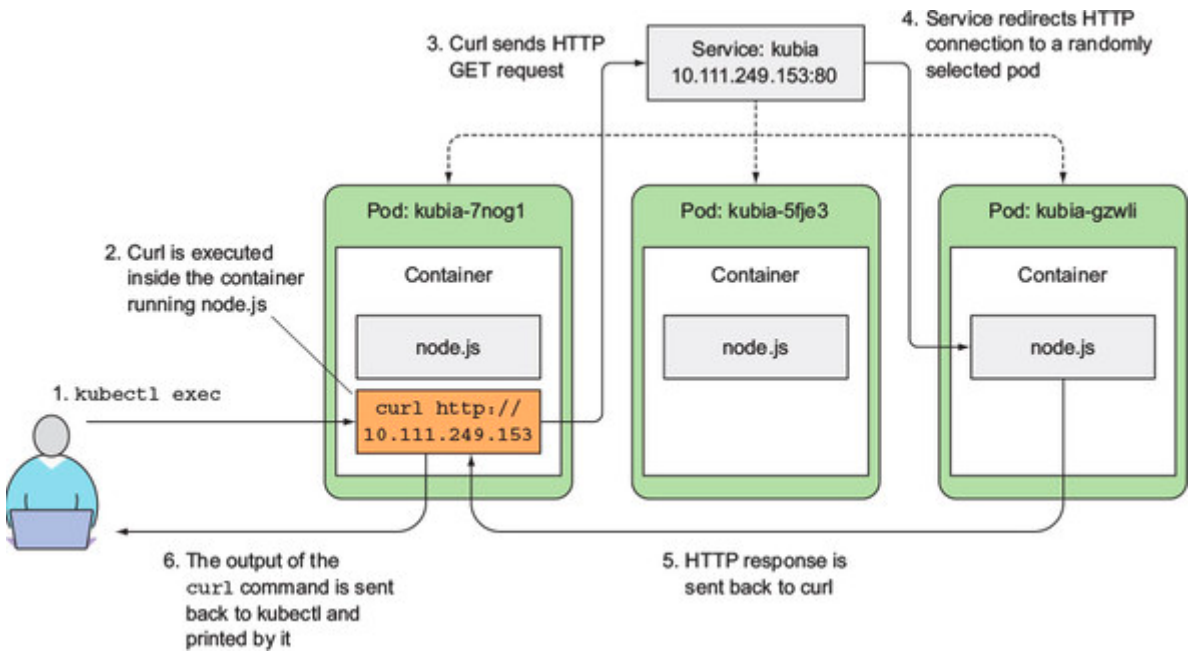
→ 클러스터 IP, 외부 연결 IP, 포트 등의 대한 정보를 확인할 수 있다.

- 클러스터 IP: 클러스터 내부에서만 연결이 가능한 IP 주소
- 외부 연결 IP: 클러스터 외부에서 연결이 가능한 IP 주소

테스트 방법

단순히 위와 같이 생성만 했다면, 테스트 하기 위해선 내부에 들어가 요청을 보내는 방법 밖에 없다. 방법은 대략 2가지

- 쿠버네티스 노드로 ssh 접속해서 curl 명령 실행
- kubectl exec 명령어로 기존 파드에서 curl 명령 실행



세션 어피니티

서비스로 요청이 들어오면 쿠버네티스 서비스 프록시가 3개의 파드 중 임의의 파드로 요청을 전달한다.

- 임의의 파드를 선택해 연결을 전달하기 때문에 요청할 때마다 다른 파드가 선택된다.

매번 같은 파드로 리디렉션하기 위해선, 서비스의 세션 어피니티(`sessionAffinity`) 속성을 기본값(`None`) 대신 `Client IP`로 설정하면 된다.

```
...
spec:
  sessionAffinity: client IP
...
```

쿠버네티스 서비스는 HTTP 수준(L7)에서 작동하지 않는다. 서비스는 TCP, UDP 패킷(L4)을 처리하고, 페이로드는 신경쓰지 않는다.

- 즉, 서비스는 쿠키를 알지 못해 사용하지 못하며, 쿠키 기반의 세션 어피니티를 사용하지 못한다.

여러 개의 포트 노출 (멀티 포트 서비스)

서비스는 여러 포트를 지원 → 파드가 2개의 포트를 수신한다면 하나의 서비스를 사용해 2개의 포트를 매핑해 둘에 들어오는 요청을 따로 보내줄 수 있다.

- 2개의 포트 지원을 위해 2개의 서비스를 만들 필요는 없다.
- 멀티 포트 서비스를 위해선 포트마다 이름을 지정해야 함.

```
...
spec:
  ports:
  - name: http
    port: 80
    targetPort: 8080
  - name: https
    port: 443
    targetPort: 8443
  selector:
    app: kubia
```

파드에선 포트 지정시 위에서 지정한 이름으로 지정하면 된다.

```
kind: Pod
spec:
  containers:
```

```
  - name: kubia
    ports:
    - name: http
      containerPort: 8080
    - name: https
      containerPort: 8443
```

이렇게 사용하면 나중에서 서비스 스펙을 변경하지 않고도 포트 번호를 변경할 수 있다는 큰 장점이 있다.

- 파드에서 포트를 변경하기로 하면, 단순히 파드 스펙에서 받는 포트 번호만 변경하면 된다.

서비스 검색

서비스 → 안정적인 IP 주소와 포트

- 서비스가 유지되는 동안 변경되지 않는다.
- 그럼, 클라이언트 파드는 서비스의 IP와 포트를 어떻게 알 수 있는건가?

환경 변수를 통한 서비스 검색

파드가 시작되면, 쿠버네티스는 해당 시점에 존재하는 **각 서비스를 가리키는 환경변수 세트를 초기화**한다.

- 클라이언트 파드를 생성하기 전에 서비스를 생성하면 해당 파드의 프로세스는 환경변수를 검사해 서비스의 IP주소와 포트를 얻을 수 있다.
- 서비스가 존재하는 상태에서 새로운 파드(셀렉되는)를 생성한다면, 컨테이너 내부로 들어가 env 명령어를 실행하면 환경변수를 확인할 수 있다.
  - `kubectl exec podName env`

backend-database라는 서비스로 백엔드 파드를 노출하면 프론트엔드 파드에서 환경변수 `BACKEND_DATABASE_SERVICE_HOST`, `BACKEND_DATABASE_SERVICE_PORT` 로 주소를 찾을 수 있다.

DNS를 통한 서비스 검색

환경변수를 통한 것보다 인터넷 환경에서 일반적으로 사용하는 DNS의 도메인을 사용하는게 좋지 않을까?

- 그래서 DNS 방법도 지원

파드에서 실행 중인 프로세스에서 수행된 모든 DNS 쿼리는 시스템에서 실행 중인 모든 서비스를 알고 있는 쿠버네티스의 자체 DNS 서버로 처리됨.

- 각 서비스는 내부 DNS 서버에서 DNS 항목을 가져오고 서비스 이름을 알고 있는 클라이언트 파드는 환경변수 대신 FQDN(정규화된 도메인 이름)으로 액세스가 가능

FQDN 통한 서비스 연결

`backend-database.default.svc.cluster.local`

- backend-database: 서비스 이름
- default: 서비스가 속한 네임스페이스
- svc.cluster.local: 클러스터의 로컬 서비스 도메인 접미사

동일한 네임스페이스에 속한 경우, 네임스페이스/도메인 접미사 둘 다 생략이 가능해진다. 즉, `backend-service` 만으로도 연결이 가능해진다.

이것도 마찬가지로 기존 파드내에서 수행해야 함.

- kubia 라는 서비스인 경우, 내부 파드에서 아래가 다 가능해진다.

```
root@kubia-3inly:/# curl http://kubia.default.svc.cluster.local
You've hit kubia-5asi2

root@kubia-3inly:/# curl http://kubia.default
You've hit kubia-3inly

root@kubia-3inly:/# curl http://kubia
You've hit kubia-8awf3
```

클러스터 외부에 있는 서비스 연결

외부 IP와 포트로 연결을 전달하자.

- 클러스터에서 실행 중인 파드는 내부 서비스에 연결하는 것처럼 외부 서비스에 연결할 수 있다.

서비스 엔드포인트

서비스는 파드에 직접 연결 되지 않는다. 대신 엔드포인트 리소스가 그 사이에 있다.

- 서비스로 노출되는 파드의 IP주소와 포트 목록

```
$ kubectl describe svc kubia
Name:                kubia
...
Port:                <unset> 80/TCP
Endpoints:           10.108.1.4:8080,10.108.2.5:8080,10.108.2.6:8080
Session Affinity:    None
```

파드 셀렉터는 연결을 전달할 때, 직접 사용하지는 않는다. IP 주소와 포트 목록을 작성하는데 사용되며, 엔드포인트 리소스에 저장된다. → 서비스에 연결하면 서비스 프록시는 이들 중 하나의 IP주소와 포트를 선택해 연결을 하는 것.

엔드포인트 수동 구성

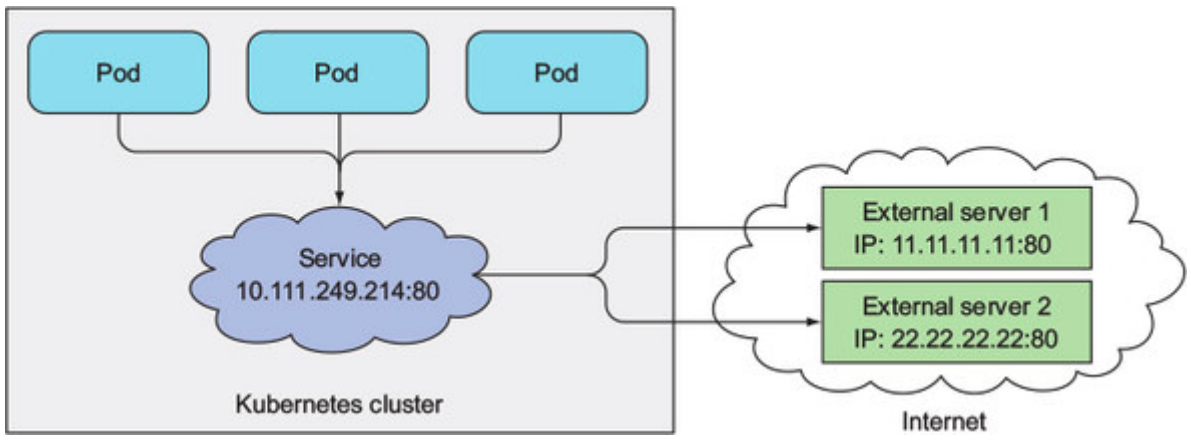
엔드포인트를 서비스와 분리하면 엔드포인트를 수동으로 구성하고 업데이트 할 수 있다.

- 파드 셀렉터 없이 서비스 만들면 쿠버네티스는 엔드포인트 리소스를 만들지 못한다. → 파드 셀렉터 통해서 엔드 포인트를 생성했으니...
- 셀렉터 없이 서비스를 생성하고, 추후 엔드포인트 리소스를 생성하고 연결해서 사용할 수 있다.

엔드포인트 리소스의 이름은 서비스 이름과 같아야 한다.

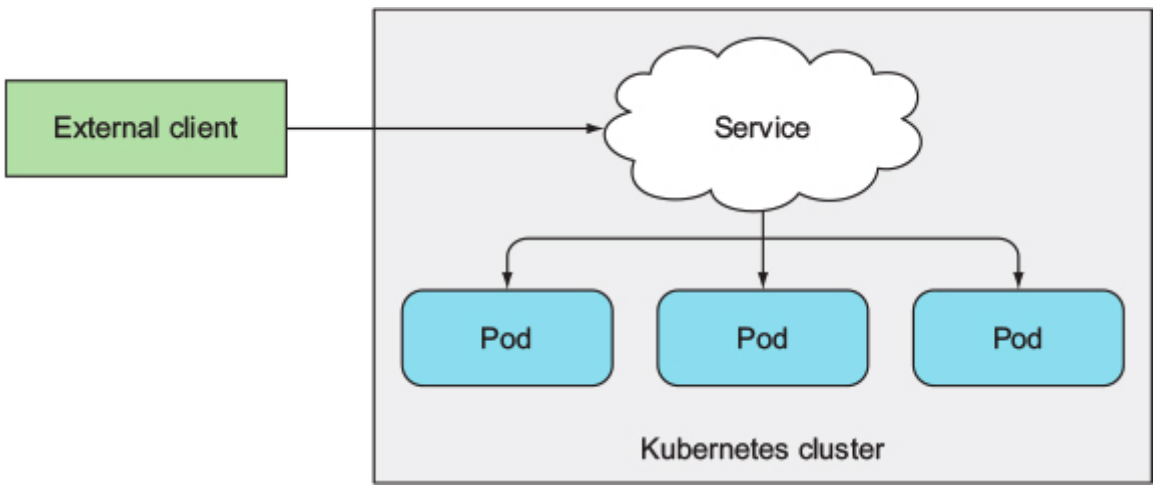
```
# 파드 셀렉터 없는 서비스 생성
apiVersion: v1
kind: Service
metadata:
  name: external-service
spec:
  ports:
  - port: 80
```

```
# 엔드포인트 생성 및 연결
apiVersion: v1
kind: Endpoints
metadata:
  name: external-service
subsets:
  - addresses:
    - ip: 11.11.11.11
    - ip: 22.22.22.22
    ports:
    - port: 80
```



외부 클라이언트에 서비스 노출

특정 서비스를 외부에 노출해 외부 클라이언트가 액세스할 수 있게 하자.



3가지 방법을 제공한다.

- **노드포트 서비스**
  - 클러스터 노드는 노드 자체에서 포트를 열고 해당 포트로 수신된 트래픽을 서비스로 전달
- **로드밸런서 (노드포트 서비스 확장)**
  - 로드밸런서는 트래픽을 모든 노드의 노드포트로 전달, 클라이언트는 IP로 서비스에 액세스
- **인그레스 리소스**
  - 단일 IP주소로 여러 서비스 노출 가능
  - HTTP 레벨에서 작동

노드포트 서비스

모든 노드에 특정 포트를 할당하고, 서비스를 구성하는 파드로 들어오는 연결을 전달한다.

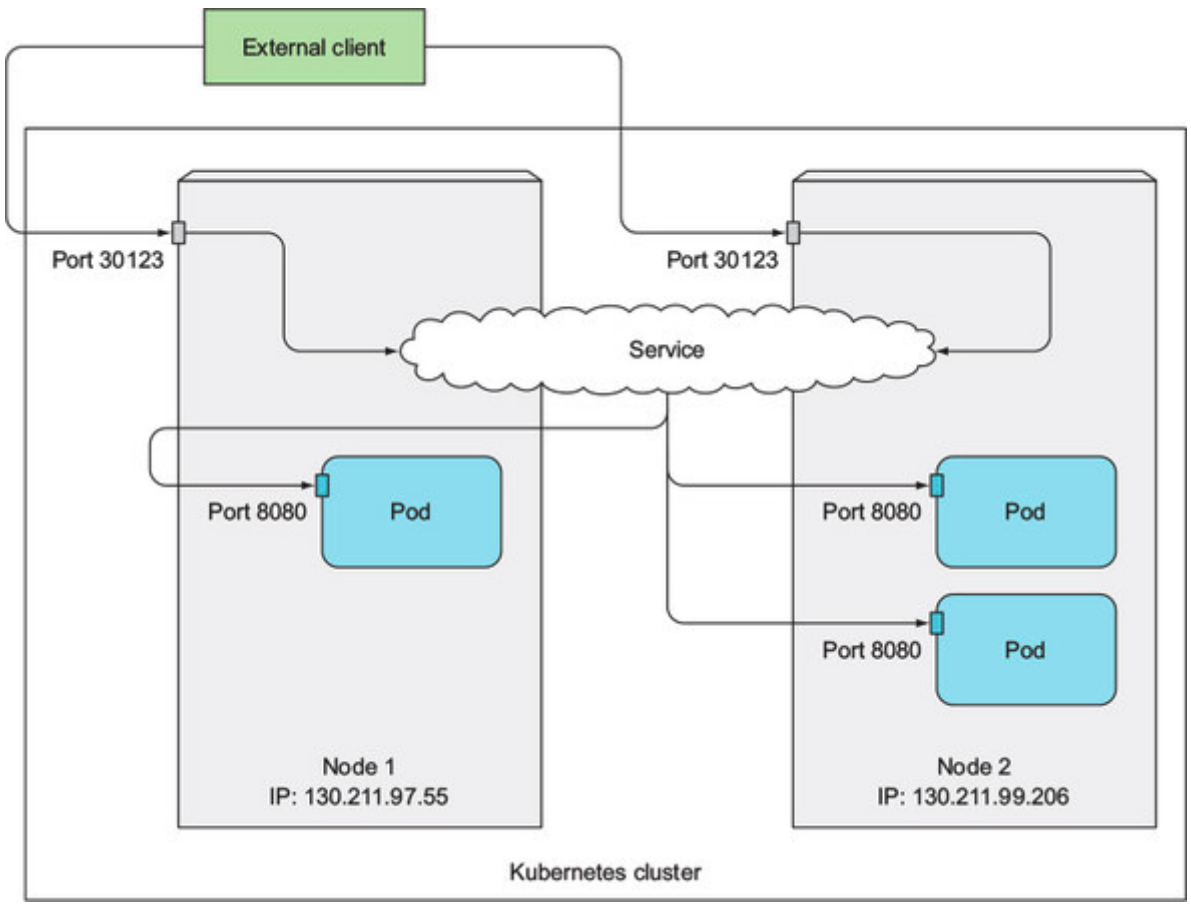
- 내부 클러스터 IP 뿐 아니라 모든 노드의 IP와 할당된 노드포트로 서비스에 엑세스 할 수 있다.

```
apiVersion: v1
kind: Service
metadata:
  name: kubia-nodeport
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 8080
```

```
    nodePort: 30123  # node port 지정
  selector:
    app: kubia
```

```
$ kubectl get svc kubia-nodeport
NAME             CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubia-nodeport   10.111.254.223   <nodes>       80:30123/TCP   2m
```

- External ip → nodes : 노드포트 서비스 이용을 의미
- Port: 노드 포트 30123



포트 30123에서 수신된 연결은 첫 번째 노드에서 실행 중인 파드 또는 두 번째 노드에서 실행 중인 파드로 전달 될 수 있다.

로드밸런서

모든 노드에 요청을 분산시키고, 해당 시점에 오프라인 상태인 노드로 요청을 보내지 않도록 노드 앞에 로드밸랜서를 배치하는 것이 좋다.

쿠버네티스 클러스터는 클라우드 인프라에서 로드밸런서를 자동으로 프로비저닝 하는 기능을 제공한다.

- 로드밸런서는 공개적으로 액세스 가능한 고유한 IP 주소를 가지며 모든 연결을 서비스로 전달

로드밸런서 서비스는 노드포트 서비스의 확장

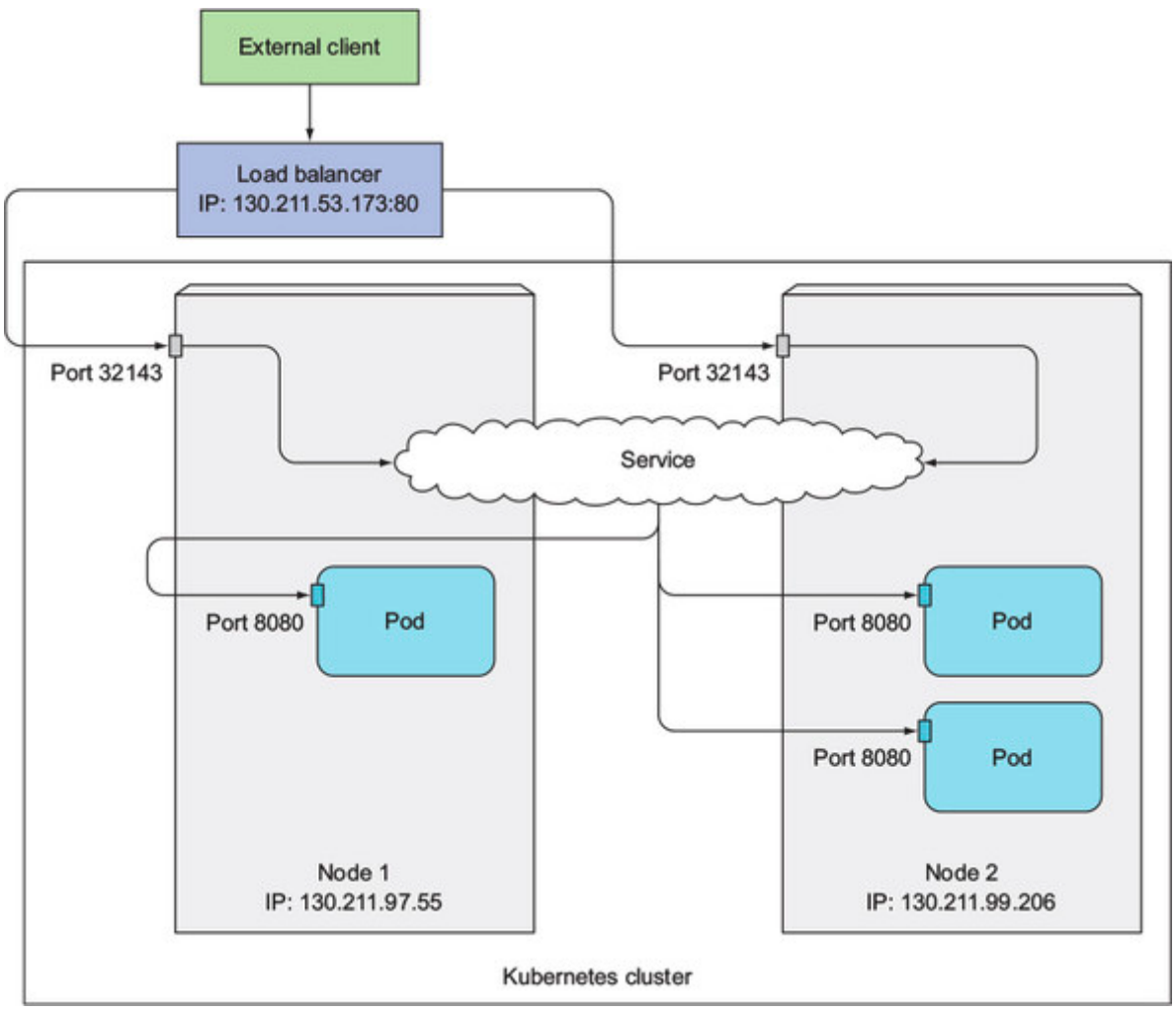
- 서비스는 노드포트 서비스처럼 작동한다.

```
apiVersion: v1
kind: Service
metadata:
  name: kubia-loadbalancer
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: kubia
```

```
$ kubectl get svc kubia-loadbalancer
NAME                 CLUSTER-IP       EXTERNAL-IP      PORT(S)         AGE
kubia-loadbalancer   10.111.241.153   130.211.53.173   80:32143/TCP    1m
```

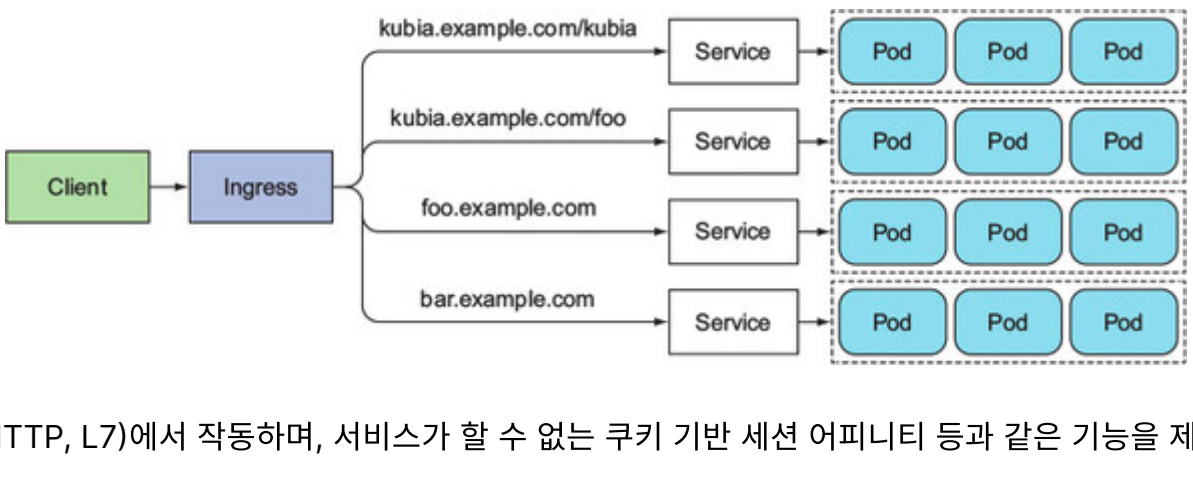
로드 밸런서는 External IP로 직접 외부연결 주소가 표시된다.

- 외부 클라이언트는 로드밸런서 포트 80에 연결하고 노드에 암묵적으로 할당된 노드포트로 라우팅된다. (Port → 32143 참조)



인그레스 리소스

한 IP 주소로 수십 개의 서비스에 접근이 가능하도록 지원



- 애플리케이션 계층(HTTP, L7)에서 작동하며, 서비스가 할 수 없는 쿠키 기반 세션 어피니티 등과 같은 기능을 제공한다.

인그레스 리소스 생성

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: kubia
spec:
  rules:
  - host: kubia.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: kubia-nodeport
          servicePort: 80
```

- Host kubia.example.com 으로 요청되는 인그레스 컨트롤러에 수신된 모든 HTTP 요청을 포트 80의 kubia-nodeport 서비스로 전송하도록 인그레스 규칙을 정의

인그레스 서비스 액세스

IP주소를 먼저 확인하자. → 인그레스 목록 조회

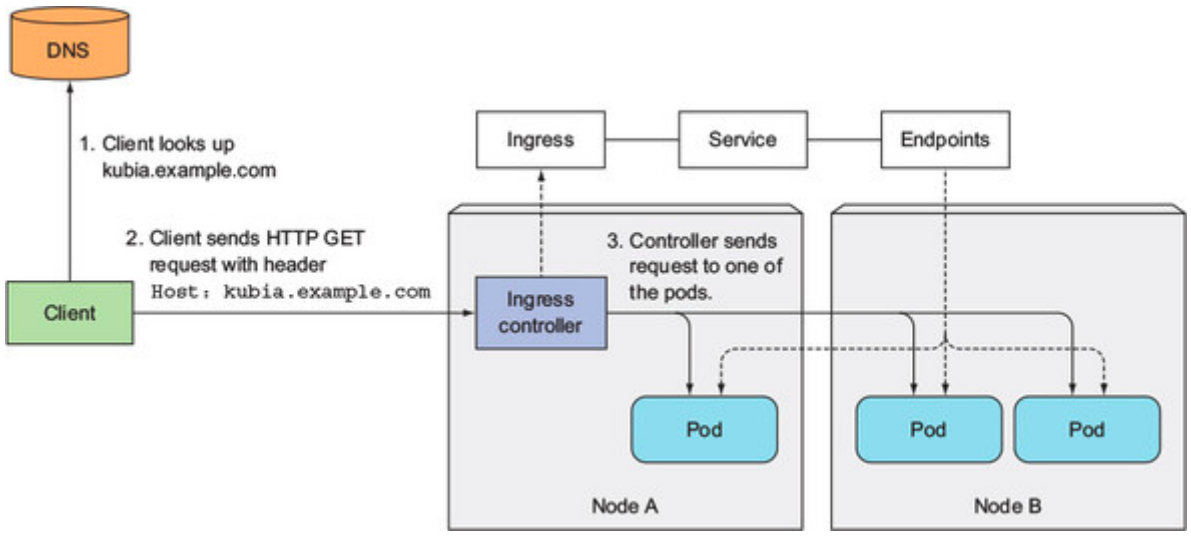
```
$ kubectl get ingresses
NAME      HOSTS               ADDRESS          PORTS     AGE
kubia     kubia.example.com   192.168.99.100   80        29m
```

인그레스 엔드포인트 지정

IP 알고 난 후, DNS를 따로 지정하거나, /etc/hosts에 추가하면 된다.

```
192.168.99.100    kubia.example.com
```

인그레스 동작 방식



- DNS 조회를 수행해 DNS 서버가 인그레스 컨트롤러의 IP를 반환
- 클라이언트는 HTTP 요청을 인그레스 컨트롤러로 전송
- 컨트롤러는 클라이언트가 액세스 하려는 서비스를 결정하고, 관련된 엔드포인트 오브젝트로 파드 IP 주회한 후 클라이언트 요청을 파드에 전송

인그레스 컨트롤러는 요청을 서비스로 전달하지 않는다. → 파드를 선택하는데만 사용한다.

하나의 인그레스로 여러 서비스 노출

여러 호스트, 경로를 여러 서비스에 매핑할 수 있다. → spec에서 host 및 path를 추가하자.

URL 경로에 따라 2개의 다른 서비스로 전송되는 경우

```
...
  - host: kubia.example.com
    http:
      paths:
      - path: /kubia
        backend:
          serviceName: kubia
          servicePort: 80
      - path: /bar
        backend:
```

```
          serviceName: bar
          servicePort: 80
```

호스트 기반으로 서로 다른 서비스를 매핑하는 경우

```
spec:
  rules:
  - host: foo.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: foo
          servicePort: 80
  - host: bar.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: bar
          servicePort: 80
```