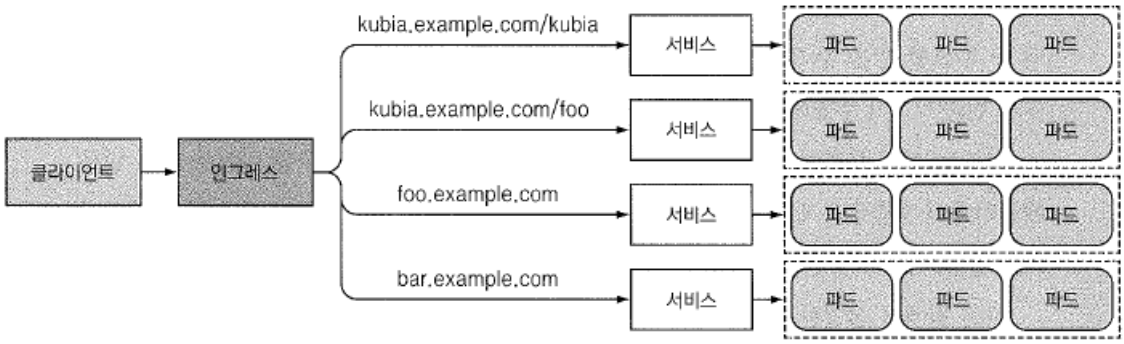
- 서비스를 위한 metal lb 설치 https://cla9.tistory.com/m/94?category=814452

> 환경
>
> Vmware
>
> Masternode, Workernode : CentOS 7 + kubeadm

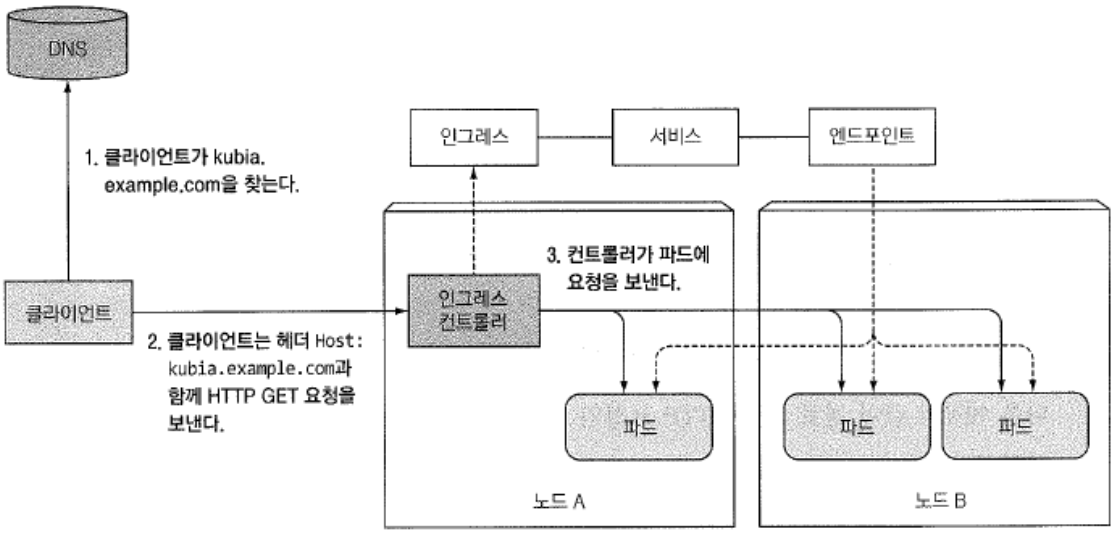
Ingress 이론

서비스의 경우에는 TLS (SSL)이나, VirtualHost와 같이 여러 호스트명을 사용하거나 호스트명에 대한 라우팅이 불가능하고, URL Path에 따른 서비스간 라우팅이 불가능하다.



URL 기반의 경로(host, path) 를 통해 클러스터 내의 서비스에 대한 외부 접근을 관리하는 역할.

로드밸런싱 / 인증서 처리(TLS/SSL) / 도메인 기반 가상 호스팅 등 처리와 Ingress 규칙에 따른 경로를 찾는 역할은 Ingress Controller가 담당한다.



▲ 그림 5.10 인그레스로 파드 액세스

Ingress 실습

1. 인그레스 컨트롤러 설치

인그레스 리소스가 작동하려면, 클러스터는 실행 중인 인그레스 컨트롤러가 반드시 필요하다. 인그레스 컨트롤러에 여러가지 종류(AWS, GCE, NGINX)가 있는데 그 중 NGINX Ingress Controller를 활용한다.

해당 링크로 들어가보면 컨트롤러를 생성하는 yaml파일을 확인할 수 있다. 해당 yaml파일을 이용하여 인그레스 컨트롤러를 설치한다.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-
nginx/controller-v0.45.0/deploy/static/provider/baremetal/deploy.yaml
```

```
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
configmap/ingress-nginx-controller created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
service/ingress-nginx-controller-admission created
service/ingress-nginx-controller created
deployment.apps/ingress-nginx-controller created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created
serviceaccount/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
```

```
[root@master ~]# kubectl get namespace
NAME              STATUS   AGE
default           Active   4d15h
ingress-nginx     Active   14m
kube-node-lease   Active   4d15h
kube-public       Active   4d15h
kube-system       Active   4d15h
```

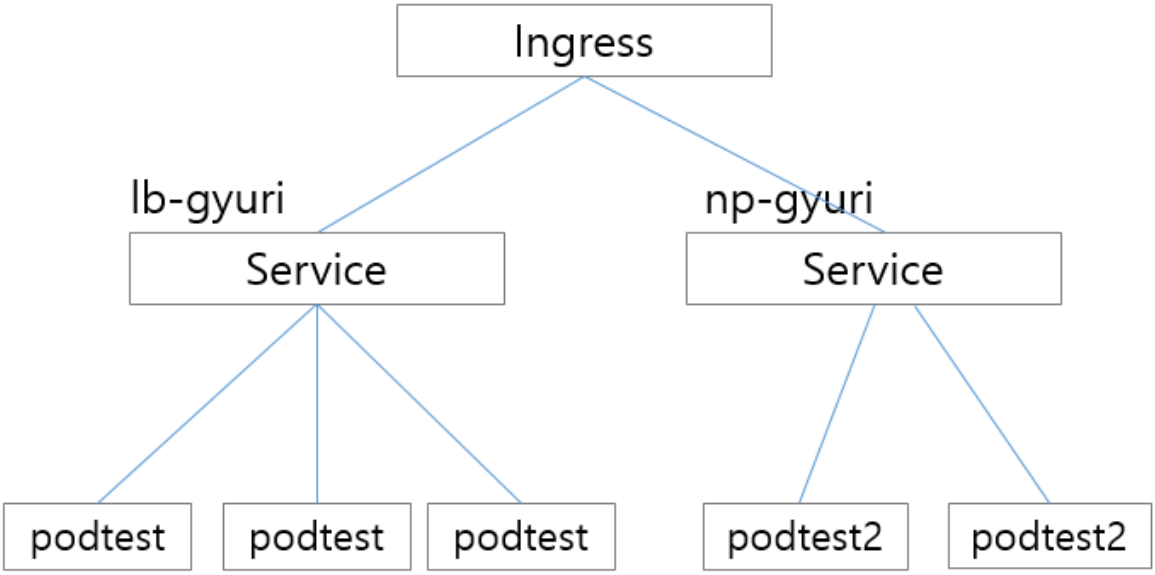
위 사진을 보면 ingress-nginx 네임스페이스가 생성된 것을 확인할 수 있다. 파드들을 확인해보면 인그레스 컨트롤러 파드가 ingress-nginx 네임스페이스에 할당된 것을 확인할 수 있다.

```
# kubectl get pods -n ingress-nginx
```

```
[root@master ~]# kubectl get pods -n ingress-nginx
NAME                                        READY   STATUS      RESTARTS   AGE
ingress-nginx-admission-create-z4hlx        0/1     Completed   0          29m
ingress-nginx-admission-patch-ck4bj         0/1     Completed   2          29m
ingress-nginx-controller-75557995f8-6mfkx   1/1     Running     0          29m
```

> **Minikube와의 차이**
>
> Minikube는 기본적으로 addon을 제공한다. addon이란 쿠버네티스 클러스터에서 필요한 기능을 확장하고 구현하는 역할이다. Minikube에서 addon이 사용하는 Namespace는 일반적으로 kube-system이다.
>
> Minikube에서는 Ingress가 addon으로 주어지기 때문에 kube-system 네임스페이스에 ingress-controller 파드가 속하게 된다.

2. 인그레스 yaml 파일 작성



현재 pod, 서비스 상태

```
# deployments.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: deptest
spec:
  replicas: 3
  selector:
    matchLabels:
      app: podtest
  template:
    metadata:
      labels:
        app: podtest
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment2
  labels:
    app: deptest2
spec:
  replicas: 2
  selector:
    matchLabels:
      app: podtest2
  template:
    metadata:
```

```
      labels:
        app: podtest2
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
```

```
[root@master ~]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-6b7b564c58-f2bxv   1/1     Running   0          29m
nginx-deployment-6b7b564c58-jcgcr   1/1     Running   0          29m
nginx-deployment-6b7b564c58-xw694   1/1     Running   0          29m
nginx-deployment2-8b44867c4-nd7dd   1/1     Running   0          29m
nginx-deployment2-8b44867c4-rtlct   1/1     Running   0          29m
```

```
#service.yaml
apiVersion: v1
kind: Service
metadata:
  name: lb-gyuri
spec:
  selector:
    app: podtest
  ports:
  - protocol: TCP
    port: 80
  type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
   name: np-gyuri
spec:
  selector:
    app: podtest2
  ports:
  - port: 80
    targetPort: 80
  type: NodePort
```

```
[root@master ~]# kubectl get svc
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)        AGE
kubernetes   ClusterIP      10.96.0.1       <none>           443/TCP        4d22
h
lb-gyuri     LoadBalancer   10.102.156.80   192.168.72.102   80:30848/TCP   29m
np-gyuri     NodePort       10.98.149.184   <none>           80:31115/TCP   29m
```

Ingress 설정

```
#ingress.yaml

apiVersion: networking.k8s.io/v1beta1
kind: Ingress
```

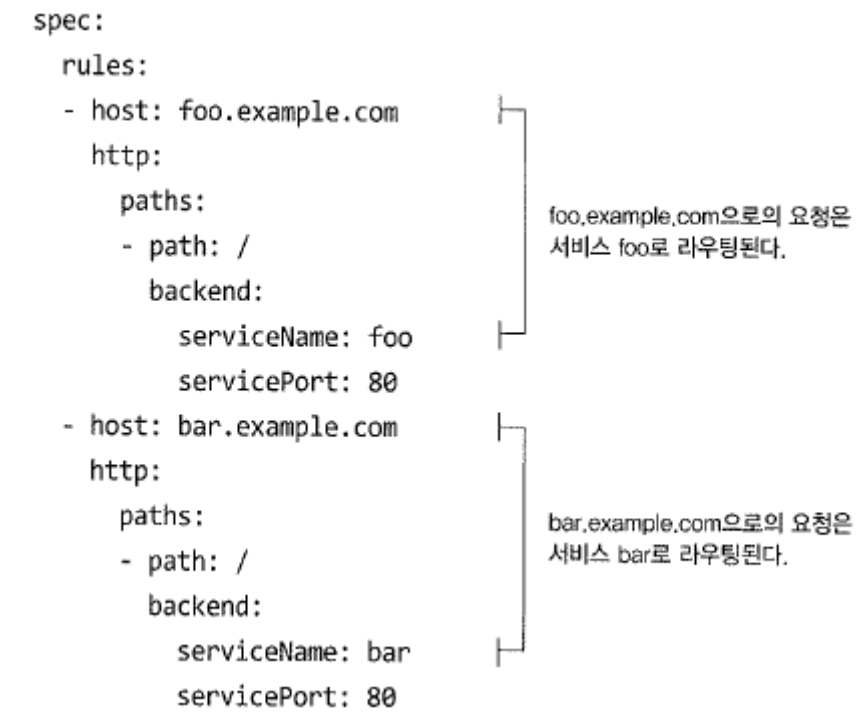
```
metadata:
    name: gyuri-ingr
    annotations:
        kubernetes.io/ingress.class: "nginx"
        ingress.kubernetes.io/rewrite-target: /
        ingress.kubernetes.io/ssl-redirect: "false"
spec:
   tls:
   - hosts:
     - gyuriexample.com
     secretName: tls-secret
   rules:
   - host: gyuriexample.com
     http:
        paths:
        - path: /lb-gyuri
          backend:
             serviceName: lb-gyuri
             servicePort: 80
        - path: /np-gyuri
          backend:
             serviceName: np-gyuri
             servicePort: 80
```

```
[root@master ~]# kubectl get ingress
NAME         CLASS    HOSTS              ADDRESS          PORTS   AGE
gyuri-ingr   <none>   gyuriexample.com   192.168.236.25   80      20m
```

```
[root@master ~]# kubectl describe ingress
Name:             gyuri-ingr
Namespace:        default
Address:          192.168.236.25
Default backend:  default-http-backend:80 (<error: endpoints "default-http-backend" not found>)
TLS:
  tls-secret terminates gyuriexample.com
Rules:
  Host              Path  Backends
  ----              ----  --------
  gyuriexample.com
                    /lb-gyuri   lb-gyuri:80 (10.32.0.3:80,10.32.0.4:80,10.32.0.7:80)
                    /np-gyuri   np-gyuri:80 (10.32.0.5:80,10.32.0.6:80)
Annotations:        ingress.kubernetes.io/rewrite-target: /
                    ingress.kubernetes.io/ssl-redirect: false
                    kubernetes.io/ingress.class: nginx
Events:             <none>
```

SVC

Pod



*에러 대처

kubectl apply -f ingress.yaml 을 했을 경우 다음과 같은 에러가 발생될 경우가 있다.

```
Error from server (InternalError): error when creating "ingress.yaml": Internal
error occurred: failed calling webhook "validate.nginx.ingress.kubernetes.io"
```

정확한 원인 해결은 아니지만 ingress가 update된 후에 발생하는 오류로, 인증과 관련된 부분에서 발생한 에러라고 함. 관련된 부분을 삭제하여 임의로 해결.

```
kubectl delete validatingwebhookconfiguration ingress-nginx-admission
```

도메인 IP 추가

```
#/etc/hosts

192.168.236.25 gyuriexmple.com
```

* 해결하지 못한 에러 : HTTP, HTTPS Path 접근

의심 상황 : ingress service controller 타입이 NodePort로 지정되어있음. 컨트롤러를 설치하면서 해당 설정을 바꾼후 시도해보아야 함.

```
#curl 명령어
curl gyuriexample.com
-> 에러 발생 : Failed connect to gyuriexample.com:80

curl gyuriexample.com:30078
-> nginx 확인 가능

curl gyuriexample.com:30078/lb-gyuri
-> 404 not found error
```

TLS 트래픽 처리 구성

TLS란?

안전한 보안 통신을 위한 보안용 프로토콜 -> 이러한 보안용 프로토콜을 이용하여 암호화하여 안전한 HTTPS 통신을 할 수 있음

```
#개인키 생성
openssl genrsa -out tls.key 2048
#인증서 생성
openssl req -new -x509 -key tls.key -out tls.cert -days 360 -subj
/CN=gyuriexample.com
#시크릿 생성
kubectl create secret tls tls-secret --cert=tls.cert --key=tls.key
```

이 후 ingress에 tls관련 설정을 저장한다.

```
spec:
   tls:
   - hosts:
     - gyuriexample.com
     secretName: tls-secret
```

이 후 Ingress를 확인하면 port에 443이 뜬 것을 볼 수 있다.

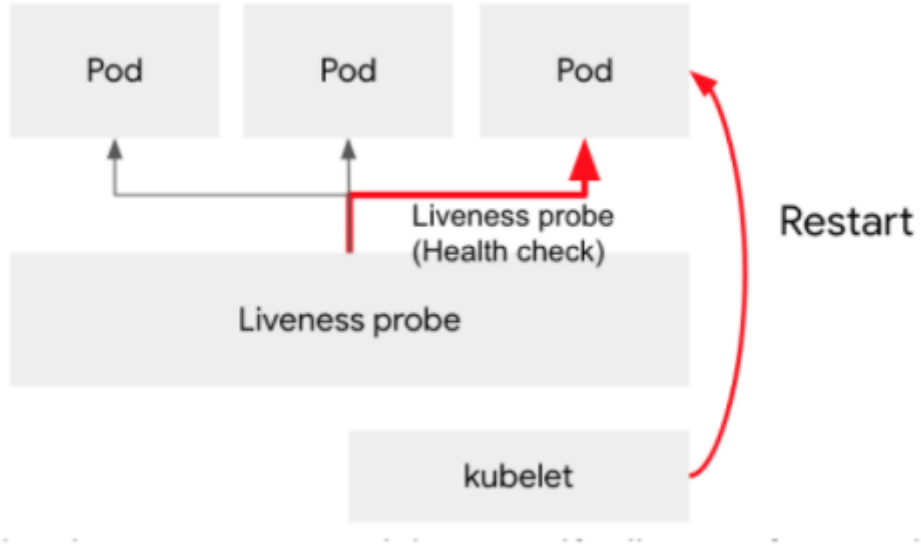
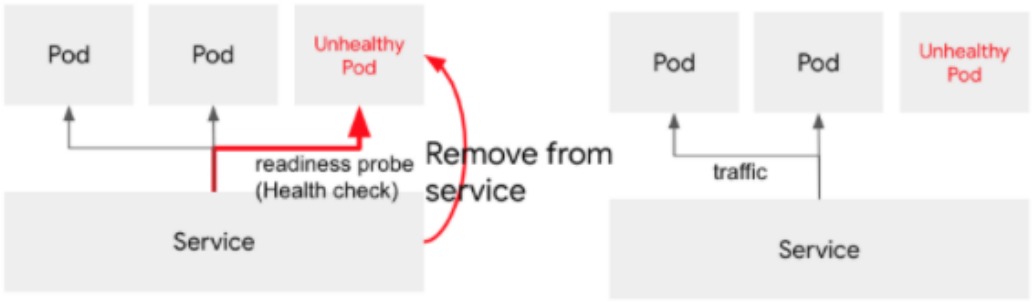
```
[root@master ~]# kubectl get ingress
NAME         CLASS    HOSTS              ADDRESS          PORTS     AGE
gyuri-ingr   <none>   gyuriexample.com   192.168.236.25   80, 443   34h
```

```
curl -k -v https://gyuriexample.com/lb-gyuri
```

Readiness probe

컨테이너가 요청을 처리할 준비가 되었는지(서비스가 가능한 상태인지) 여부를 나타내는 probe이다.

> Liveness probe VS Readiness probe

> Liveness probe : 컨테이너 상태를 주기적으로 체크하다 컨테이너에 응답이 없으면 컨테이너를 자동으로 **재 시작**한다. 컨테이너가 살아있는지를 체크하는 기능이다.
>
> 
>
>
> Readiness probe : 컨테이너의 서비스 가용상태를 체크하다 서비스가 불가능하면 새롭게 pod를 재 생성하지 않고 서비스의 엔드포인트에서 **해당 파드 IP주소를 삭제한다.** 컨테이너를 재기동한다해도 서비스가 불가능할 경우가 있기 때문에 사용하는 기능이다. : 서비스 레벨
>
> 
>

컨테이너의 상태가 정상이라는 것을 Readiness probe를 통해 확인받은 후 트래픽을 컨테이너에 보내기 때문에 대량의 데이터 혹은 migration을 처리해야할 때 Readiness probe를 지정하게 된다.

Readiness probe 유형

1) Exec

프로세스를 실행한 후 반환된 프로세스의 종료코드로 Readiness probe가 작동된다.

ls 명령어 : 파일 존재(성공) 시 0 반환 - Readiness probe 점검 성공 / 파일 존재하지 않을(실패) 시 0이외의 값 반환 - Readiness probe 점검 실패

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: deptest
spec:
  replicas: 3
  selector:
    matchLabels:
      app: podtest
  template:
    metadata:
      labels:
        app: podtest
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
        readinessProbe:
          exec:
            command:
            - ls
            - /var/ready
```

```
[root@master ~]# kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-6b7b564c58-jpsg6   1/1     Running   0          3h31m
nginx-deployment-6b7b564c58-sk9fk   1/1     Running   0          3h31m
nginx-deployment-6b7b564c58-tf4sf   1/1     Running   0          3h31m
nginx-deployment2-8b44867c4-mh8tq   1/1     Running   0          3h31m
nginx-deployment2-8b44867c4-zbdt6   1/1     Running   0          3h31m
[root@master ~]# kubectl exec nginx-deployment-6b7b564c58-
nginx-deployment-6b7b564c58-jpsg6  nginx-deployment-6b7b564c58-tf4sf
nginx-deployment-6b7b564c58-sk9fk
[root@master ~]# kubectl exec nginx-deployment-6b7b564c58-jpsg6 -- touch /var/ready
[root@master ~]# kubectl exec nginx-deployment-6b7b564c58-sk9fk -- touch /var/ready
```

Pod 템플릿을 변경해도 기존 파드에는 영향을 미치지 않기 때문에 파드를 삭제한 후 새로운 파드를 생성하게 한다. 새 파드에는 /var/ready가 존재하지 않기 때문에 Readiness probe에 실패하고 해당 파드는 서비스의 엔드포인트에 포함되지 않는다.

```
[root@master ~]# kubectl get po
NAME                                READY   STATUS              RESTARTS   AGE
nginx-deployment-5d4cb75f9d-k8rwz   0/1     ContainerCreating   0          26s
nginx-deployment-6b7b564c58-jpsg6   1/1     Running             0          3h38m
nginx-deployment-6b7b564c58-sk9fk   1/1     Running             0          3h38m
nginx-deployment-6b7b564c58-tf4sf   1/1     Running             0          3h38m
nginx-deployment2-8b44867c4-mh8tq   1/1     Running             0          3h38m
nginx-deployment2-8b44867c4-zbdt6   1/1     Running             0          3h38m
```

```
[root@master ~]# kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-5d4cb75f9d-dnd64   0/1     Running   0          69s
nginx-deployment-6b7b564c58-jpsg6   1/1     Running   0          3h43m
nginx-deployment-6b7b564c58-sk9fk   1/1     Running   0          3h43m
nginx-deployment-6b7b564c58-tf4sf   1/1     Running   0          3h43m
nginx-deployment2-8b44867c4-mh8tq   1/1     Running   0          3h43m
nginx-deployment2-8b44867c4-zbdt6   1/1     Running   0          3h43m
```

```
[root@master ~]# kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-5d4cb75f9d-psmfc   0/1     Running   0          3m52s
nginx-deployment-5d4cb75f9d-r57nk   0/1     Running   0          3m52s
nginx-deployment-5d4cb75f9d-vvfrj   0/1     Running   0          3m52s
nginx-deployment2-8b44867c4-rpvss   1/1     Running   0          3m52s
nginx-deployment2-8b44867c4-x76sz   1/1     Running   0          3m52s
```

```
  Warning  Unhealthy      1s (x7 over 61s)   kubelet            Readiness probe failed: ls
: cannot access '/var/ready': No such file or directory
```

```
[root@master ~]# kubectl exec nginx-deployment-5d4cb75f9d-psmfc touch /var/ready
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kub
ectl exec [POD] -- [COMMAND] instead.
[root@master ~]# kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-5d4cb75f9d-psmfc   1/1     Running   0          8m23s
nginx-deployment-5d4cb75f9d-r57nk   0/1     Running   0          8m23s
nginx-deployment-5d4cb75f9d-vvfrj   0/1     Running   0          8m23s
nginx-deployment2-8b44867c4-rpvss   1/1     Running   0          8m23s
nginx-deployment2-8b44867c4-x76sz   1/1     Running   0          8m23s
```

```
[root@master ~]# kubectl describe lb-gyuri
error: the server doesn't have a resource type "lb-gyuri"
[root@master ~]# kubectl describe svc lb-gyuri
Name:                     lb-gyuri
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=podtest
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.106.232.160
IPs:                      10.106.232.160
LoadBalancer Ingress:     192.168.72.102
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30078/TCP
Endpoints:                10.32.0.3:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

2) HTTP GET : IP 주소에 대해 HTTP Get 요청을 수행해서 응답의 상태코드가 200보다 크고 400보다 작으면 성공, 그렇지 않다면 실패

3) TCP : IP주소에 대해 TCP 검사를 수행해서 포트가 활성화되어있다면 성공, 그렇지 않다면 실패

Headless Service

클라이언트 DNS 조회 : 서비스의 클러스터 IP 반환

if 서비스 == Headless Service :

클라이언트 DNS 조회 : 서비스에 포함된 모든 파드의 IP 반환

서비스 생성 시점에 endpoint 지정을 하지 않음.

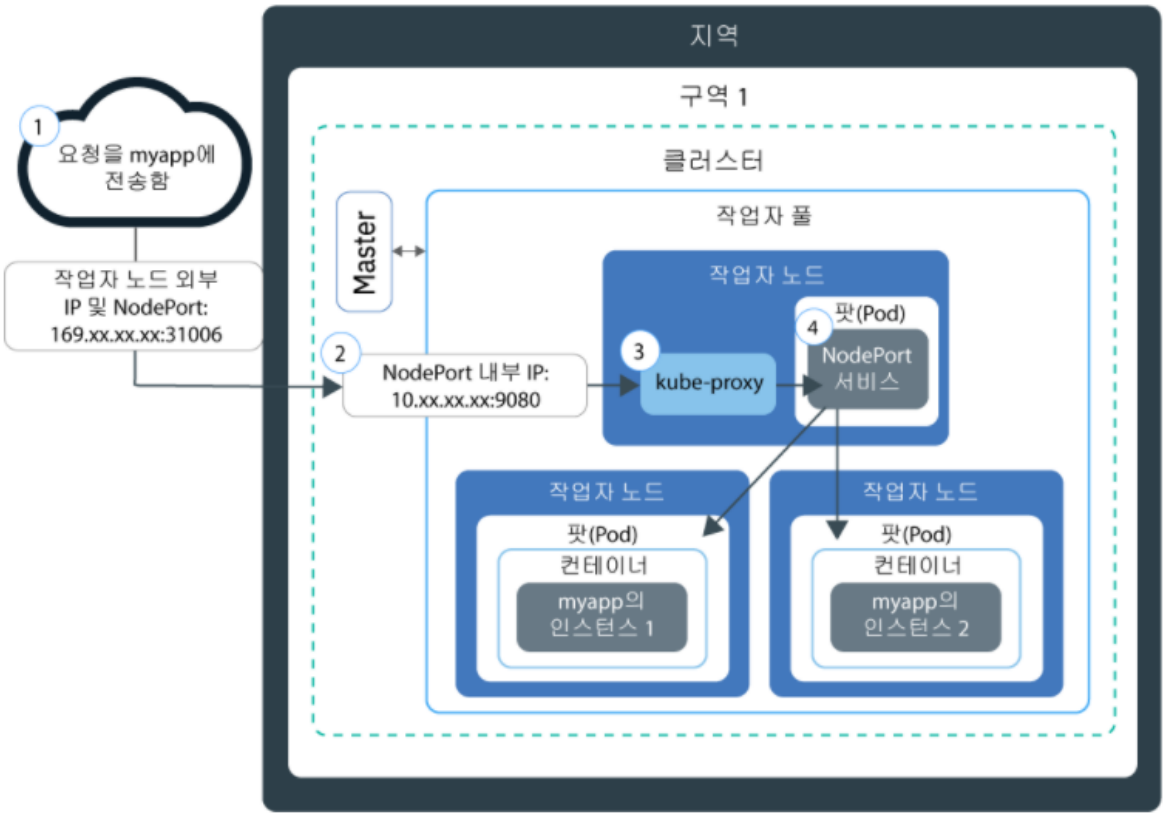
```
apiVersion: v1
kind: Service
metadata:
  name: kubia-headless
spec:
  clusterIP: None
  ports:
  - port: 80
```

이 부분이 서비스를 헤드리스 서비스로 만든다.

모든 파드 검색 - 준비되지 않은 파드 포함

```
kind: Service
metadata:
  annotations:
    service.alpha.kubernetes.io/tolerate-unready-endpoints: "true"
```

Nodeport



출처

- 쿠버네티스 공식 사이트 - 인그레스
- 조대협 블로그 : Readiness Probe