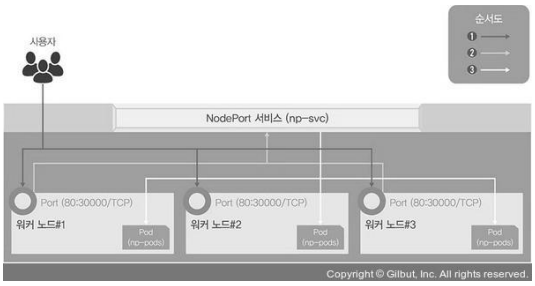
# week6

## ▼ ✨ 3.3 쿠버네티스 연결을 담당하는 서비스

- 외부 사용자가 파드를 이용하는 방법
- **서비스 : 외부에서 쿠버네티스 클러스터에 접속하는 방법**

### ▼ 💛 3.3.1 가장 간단하게 연결하는 노드포트




💡 **노드포트**(NodePort)

- 모든 워커 노드의 **특정 포트(노드포트)**를 열고
- 여기로 오는 모든 **요청**을 **노드포트 서비스로 전달**합니다.
- 그리고 노드포트 서비스는 **해당 업무를 처리할 수 있는 파드로 요청을 전달**합니다.

▼ 🚩 노드포트 서비스로 외부에서 접속하기 (yaml 파일)

**1.** 디플로이먼트로 파드를 생성합니다. 이때 이미지는 sysnet4admin 계정에 있는 echo-hname을 사용합니다.

```
[root@m-k8s ~]# kubectl create deployment np-pods --image=sysnet4admin/echo-hname
```

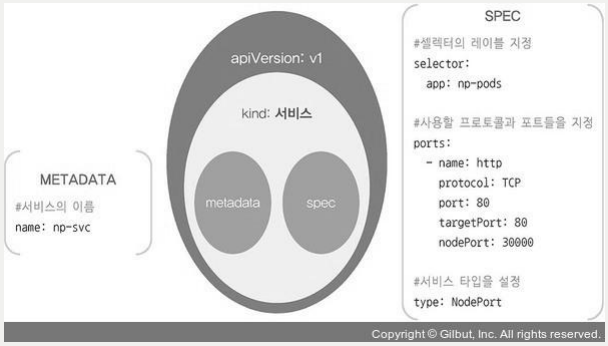
**2.** 배포된 파드를 확인합니다.

```
[root@m-k8s ~]# kubectl get pods
```

**3.** 노드포트 서비스를 생성합니다. 여기서는 편의를 위해 이미 정의한 오브젝트 스펙을 이용합니다.

```
[root@m-k8s ~]# kubectl create -f ~/_Book_k8sInfra/ch3/3.3.1/nodeport.yaml
```

▼ nodeport.yaml파일의 구조




```
apiVersion: v1
kind: Service
metadata:
  name: np-svc
spec:
  selector:
    app: np-pods
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30000
  type: NodePort
```

- kind: Service
- spec에 컨테이너에 대한 정보 없음
- 접속에 필요한 네트워크 관련 정보(protocol, port, targetPort, nodePort)와 서비스의 type을 NodePort로 지정

**4.** 노드포트 서비스로 생성한 np-svc 서비스를 확인합니다.

```
[root@m-k8s ~]# kubectl get services
```

```
[root@m-k8s ~]# kubectl get services
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP          19m
np-svc       NodePort    10.107.152.242   <none>        80:30000/TCP     45s
```

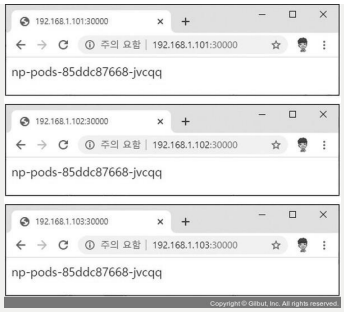
노드포트의 포트 번호가 30000번으로 지정됐습니다. CLUSTER-IP는 쿠버네티스 클러스터의 내부에서 사용하는 IP로, 자동으로 지정됩니다.

**5.** 쿠버네티스 클러스터의 워커 노드 IP를 확인합니다.

```
[root@m-k8s ~]# kubectl get nodes -o wide
```

```
[root@m-k8s ~]# kubectl get nodes -o wide
NAME     STATUS   ROLES    AGE     VERSION   INTERNAL-IP
m-k8s    Ready    master   25m     v1.18.4   192.168.1.10
w1-k8s   Ready    <none>   19m     v1.18.4   192.168.1.101
w2-k8s   Ready    <none>   14m     v1.18.4   192.168.1.102
w3-k8s   Ready    <none>   9m51s   v1.18.4   192.168.1.103
```

**6.** 외부에서 접속되는지 확인합니다. 화면에 파드 이름이 표시되는지도 확인합니다. **이때 파드가 하나이므로 화면에 보이는 이름은 모두 동일**합니다.




▼ 🚩 부하 분산 테스트하기

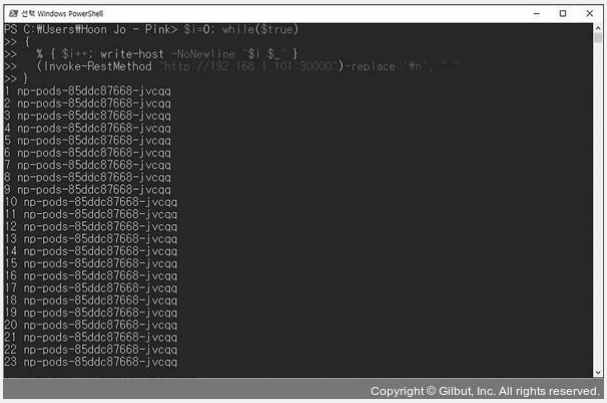
디플로이먼트로 생성된 **파드 1개에 접속하고 있는 중에 파드가 3개로 증가**하면 접속이 어떻게 바뀔까요?

**부하가 분산되는지(로드밸런서 기능)** 확인해 보겠습니다.

💡 **1.** 파워셸(powershell) 명령 창을 띄우고 다음 명령을 실행합니다.

이 명령은 반복적으로 192.168.1.101:30000에 접속해 **접속한 파드 이름을 화면에 표시(Invoke-RestMethod)**합니다.

```
PS C:\Users\Hoon Jo - Pink> $i=0; while($true)
{
 % { $i++; write-host -NoNewline "$i $_" }
 (Invoke-RestMethod "http://192.168.1.101:30000")-replace '\n', " "
}
```

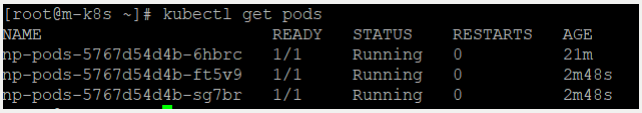



**2.** scale을 실행해 파드를 3개로 증가시킵니다.

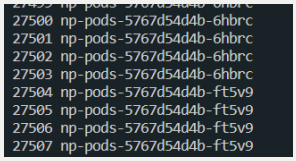
```
[root@m-k8s ~]# kubectl scale deployment np-pods --replicas=3
```

**3.** 배포된 파드를 확인합니다.

```
[root@m-k8s ~]# kubectl get pods
```




파워셸 명령 창을 확인해 표시하는 파드 이름에 배포된 파드 3개가 돌아가면서 표시되는지 확인합니다. 즉, 부하 분산이 제대로 되는지 확인합니다.




어떻게 추가된 파드를 외부에서 추적해 접속하는 것일까요?

이는 **노드포트의 오브젝트 스펙에 적힌 np-pods와 디플로이먼트의 이름을 확인해 동일하면 같은 파드라고 간주하기 때문**입니다.

```
#nodeport.yaml
spec:
```

```
selector:
  app: np-pods
```

▼ 🚩 expose 명령어로 노드포트 서비스 생성하기

💡 **1.** expose 명령어를 사용해 **서비스로 내보낼 디플로이먼트를 np-pods로 지정**합니다.
**서비스의 이름은 np-svc-v2**로,
**타입은 NodePort**로 지정합니다(이때 서비스 타입은 반드시 대소문자를 구분해야 합니다).
마지막으로 서비스가 파드로 보내줄 **연결 포트를 80번**으로 지정합니다.

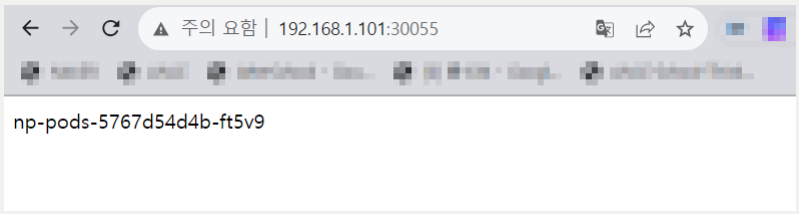
```
[root@m-k8s ~]# kubectl expose deployment np-pods --type=NodePort --name=np-svc-v2 --port=80
```

**2.** 생성된 서비스를 확인합니다.

```
[root@m-k8s ~]# kubectl get services
```

```
[root@m-k8s ~]# kubectl get services
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP        72m
np-svc       NodePort    10.107.152.242   <none>        80:30000/TCP   53m
np-svc-v2    NodePort    10.102.194.129   <none>        80:30055/TCP   11s
```

**3.** 배포된 파드 중 하나의 이름이 웹 브라우저에 표시되는지 확인합니다.

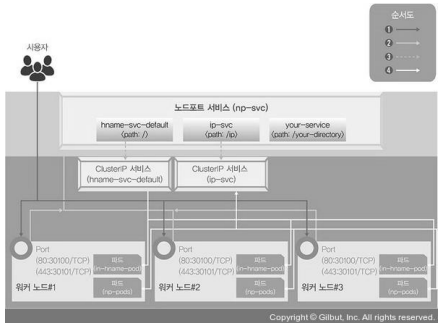



**4.** 다음 실습 진행을 위해 배포한 디플로이먼트와 서비스 2개를 모두 삭제합니다.

```
[root@m-k8s ~]# kubectl delete deployment np-pods
[root@m-k8s ~]# kubectl delete services np-svc
[root@m-k8s ~]# kubectl delete services np-svc-v2
```

## ▼ 💛 3.3.2 사용 목적별로 연결하는 인그레스

- 노드포트 서비스는 **포트를 중복 사용할 수 없어서 1개의 노드포트에 1개의 디플로이먼트만 적용**됩니다. 그렇다면 여러 개의 디플로이먼트가 있을 때 그 수만큼 노드포트 서비스를 구동해야 할까요?
- **인그레스**(Ingress)
  - **고유한 주소를 제공해 사용 목적에 따라 다른 응답**을 제공할 수 있습니다.
  - 트래픽에 대한 **L4/L7 로드밸런서와 보안 인증서를 처리**하는 기능을 제공합니다.
- NGINX 인그레스 컨트롤러




  - 인그레스를 사용하려면 인그레스 컨트롤러가 필요합니다.

💡 인그레스 컨트롤러

**1.** 사용자는 노드마다 설정된 노드포트를 통해 노드포트 서비스로 접속합니다. 이때 **노드포트 서비스를 NGINX 인그레스 컨트롤러로 구성**합니다.

**2.** NGINX 인그레스 컨트롤러는 사용자의 **접속 경로에 따라 적합한 클러스터 IP 서비스로 경로를 제공**합니다.

**3.** 클러스터 IP 서비스는 사용자를 해당 **파드로 연결**해 줍니다.

- **인그레스 컨트롤러는 파드와 직접 통신할 수 없어서 노드포트 또는 로드밸런서 서비스와 연동되어야 합니다**. 따라서 노드포트로 이를 연동했습니다.
- 인그레스 컨트롤러의 궁극적인 목적은 **사용자가 접속하는 경로에 따라 다른 결괏값을 제공**하는 것입니다.

▼ 🚩 코드 실습

**1.** 디플로이먼트 2개(in-hname-pod, in-ip-pod)를 배포합니다.

```
[root@m-k8s ~]# kubectl create deployment in-hname-pod --image=sysnet4admin/echo-hname
[root@m-k8s ~]# kubectl create deployment in-ip-pod --image=sysnet4admin/echo-ip
```

**2.** 배포된 파드의 상태를 확인합니다.

```
[root@m-k8s ~]# kubectl get pods
```

**3.** NGINX 인그레스 컨트롤러를 설치합니다. 여기에는 많은 종류의 오브젝트 스펙이 포함됩니다. 설치되는 요소들은 NGINX 인그레스 컨트롤러 서비스를 제공하기 위해 미리 지정돼 있습니다.

```
[root@m-k8s ~]# kubectl apply -f ~/_Book_k8sInfra/ch3/3.3.2/ ingress-nginx .yaml
```

**4.** NGINX 인그레스 컨트롤러의 파드가 배포됐는지 확인합니다. NGINX 인그레스 컨트롤러는 **default 네임스페이스가 아닌 ingress-nginx 네임스페이스**에 속하므로 **-n ingress-nginx 옵션을 추가**해야 합니다.

- -n : namespace의 약어로, default 외의 네임스페이스를 확인할 때 사용하는 옵션입니다.

```
[root@m-k8s ~]# kubectl get pods -n ingress-nginx
```

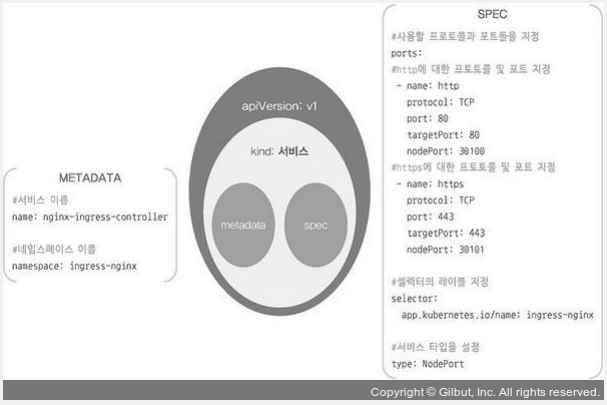
**5.** 인그레스를 사용자 요구 사항에 맞게 설정하려면 **경로와 작동을 정의**해야 합니다. 파일로도 설정할 수 있으므로 다음 경로로 실행해서 미리 정의해 둔 설정을 적용합니다.

```
[root@m-k8s ~]# kubectl apply -f ~/_Book_k8sInfra/ch3/3.3.2/ingress- config .yaml
```

▼ ingress-config.yaml

- 주소 값과 포트에 따라 노출된 서비스를 연결하는 역할을 설정합니다.
- 외부에서 주소 값과 노드포트를 가지고 들어오는 것은 **hname-svc-default 서비스와 연결된 파드**로 넘기고,
- 외부에서 들어오는 주소 값, 노드포트와 함께 뒤에 **/ip를 추가한 주소 값**은 **ip-svc 서비스와 연결된 파드**로 접속하게 설정했습니다.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-nginx
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path:
        backend:
          serviceName: hname-svc-default
          servicePort: 80
      - path: /ip
        backend:
          serviceName: ip-svc
          servicePort: 80
      - path: /your-directory
        backend:
          serviceName: your-svc
          servicePort: 80
```




**6.** 인그레스 설정 파일이 제대로 등록됐는지 kubectl get ingress로 확인합니다.

```
[root@m-k8s ~]# kubectl get ingress
```

```
[root@m-k8s ~]# kubectl get ingress
NAME            CLASS    HOSTS   ADDRESS   PORTS   AGE
ingress-nginx   <none>   *                 80      8m35s
```

**7.** 인그레스에 요청한 내용이 확실하게 적용됐는지 확인합니다. 이 명령은 **인그레스에 적용된 내용을 야믈 형식으로 출력**해 적용된 내용을 확인할 수 있습니다. 우리가 적용한 내용 외에 시스템에서 자동으로 생성하는 것까지 모두 확인할 수 있으므로 이 명령을 응용하면 오브젝트 스펙 파일을 만드는 데 도움이 됩니다.

```
[root@m-k8s ~]# kubectl get ingress -o yaml
```

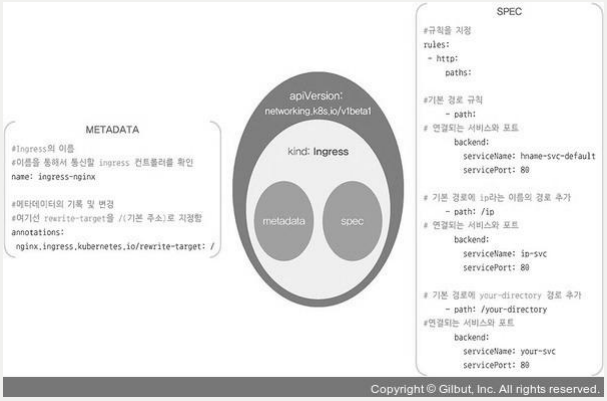
**8.** NGINX 인그레스 컨트롤러 생성과 인그레스 설정을 완료했습니다. 이제 **외부에서 NGINX 인그레스 컨트롤러에 접속할 수 있게 노드포트 서비스로 NGINX 인그레스 컨트롤러를 외부에 노출**합니다. (인그레스 컨트롤러는 파드와 직접 통신 불가)

```
[root@m-k8s ~]# kubectl apply -f ~/_Book_k8sInfra/ch3/3.3.2/ingress.yaml
```

▼ ingress.yaml

- 기존 노드포트와 달리 http를 처리하기 위해 30100번 포트로 들어온 요청을 80번 포트로 넘기고,
- https를 처리하기 위해 30101번 포트로 들어온 것을 443번 포트로 넘깁니다.
- NGINX 인그레스 컨트롤러가 위치하는 네임스페이스를 ingress-nginx로 지정하고 NGINX 인그레스 컨트롤러의 요구 사항에 따라 **셀렉터를 ingress-nginx로 지정**했습니다.
  - "ingress-nginx"라는 이름을 가진 파드들과 이 서비스가 연결되어 클라이언트 요청을 해당 파드들로 전달합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-ingress-controller
  namespace: ingress-nginx
spec:
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 80
    nodePort: 30100
  - name: https
    protocol: TCP
    port: 443
    targetPort: 443
    nodePort: 30101
  selector:
    app.kubernetes.io/name: ingress-nginx
  type: NodePort
```




**9.** 노드포트 서비스로 생성된 NGINX 인그레스 컨트롤러(nginx-ingress-controller)를 확인합니다. 이때도 -n ingress-nginx로 네임스페이스를 지정해야만 내용을 확인할 수 있습니다.

```
[root@m-k8s ~]# kubectl get services -n ingress-nginx
```

```
[root@m-k8s ~]#  kubectl get services -n ingress-nginx
NAME                       TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)
nginx-ingress-controller   NodePort   10.110.145.253   <none>        80:30100/TCP,443:30101/TCP
```

**10. expose 명령으로 디플로이먼트(in-hname-pod, in-ip-pod)도 서비스로 노출**합니다.

- 외부와 통신하기 위해 클러스터 내부에서만 사용하는 파드를 **클러스터 외부에 노출할 수 있는 구역으로 옮기는 것**입니다.
- 내부와 외부 네트워크를 분리해 관리하는 DMZ(DeMilitarized Zone, 비무장지대)와 유사한 기능입니다. 비유적으로 표현하면 각 방에 있는 물건을 외부로 내보내기 전에 공용 공간인 거실로 모두 옮기는 것과 같습니다.

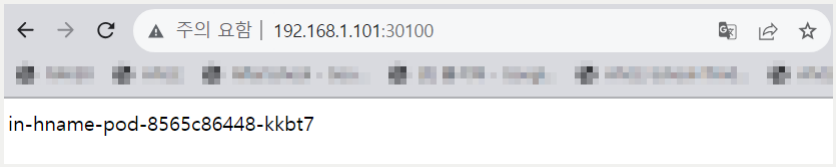
```
[root@m-k8s ~]# kubectl expose deployment in-hname-pod --name=hname-svc-default --port=80,443
[root@m-k8s ~]# kubectl expose deployment in-ip-pod --name=ip-svc --port=80,443
```

**11.** 생성된 서비스를 점검해 디플로이먼트들이 서비스에 정상적으로 노출되는지 확인합니다. *새로 생성된 서비스는 default 네임스페이스에 있으므로 -n 옵션으로 네임스페이스를 지정하지 않아도 됩니다.*

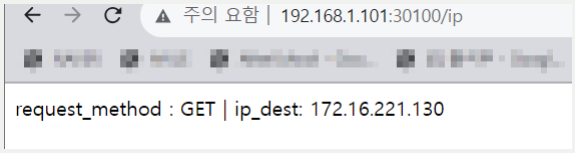
```
[root@m-k8s ~]# kubectl get services
```

```
[root@m-k8s ~]# kubectl get services
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
hname-svc-default   ClusterIP   10.107.14.64     <none>        80/TCP,443/TCP
ip-svc              ClusterIP   10.107.249.90    <none>        80/TCP,443/TCP
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP
```

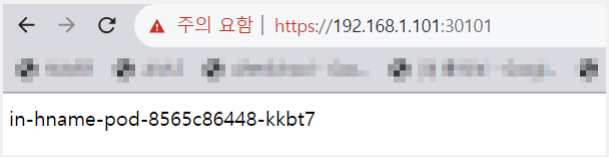
**12.** 192.168.1.101:30100에 접속해 외부에서 접속되는 경로에 따라 다르게 작동하는지 확인합니다. 이때 워커 노드 IP는 192.168.1.101이 아닌 102 또는 103을 사용해도 무방합니다.



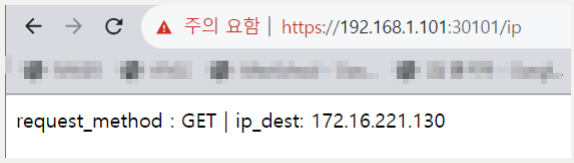

**13.** 192.168.1.101:30100 뒤에 /ip를 추가합니다. 요청 방법과 파드의 ip가 반환되는지 확인합니다.




**14.** https://192.168.1.101:30101으로 접속해 HTTP 연결이 아닌 HTTPS 연결도 정상적으로 작동하는지 확인합니다. 30101은 HTTPS의 포트인 443번으로 변환해 접속됩니다.




**15.** https://192.168.1.101:30101/ip를 입력해 마찬가지로 요청 방법과 파드의 IP 주소가 웹 브라우저에 표시되는지 확인합니다.



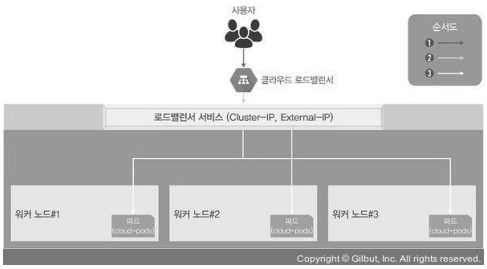

**16.** NGINX 인그레스 컨트롤러 구성과 테스트가 끝났습니다. 역시 다음 실습 진행을 위해 배포한 디플로이먼트와 모든 서비스를 삭제합니다.

```
[root@m-k8s ~]# kubectl delete deployment in-hname-pod
[root@m-k8s ~]# kubectl delete deployment in-ip-pod
[root@m-k8s ~]# kubectl delete services hname-svc-default
[root@m-k8s ~]# kubectl delete services ip-svc
```

**17.** NGINX 인그레스 컨트롤러와 관련된 내용도 모두 삭제합니다.

```
[root@m-k8s ~]# kubectl delete -f ~/_Book_k8sInfra/ch3/3.3.2/ingress-nginx.yaml
[root@m-k8s ~]# kubectl delete -f ~/_Book_k8sInfra/ch3/3.3.2/ingress-config.yaml
```

### ▼ 💛 3.3.3 클라우드에서 쉽게 구성 가능한 로드밸런서




- 앞에서 배운 연결 방식은 들어오는 요청을 **모두 워커 노드의 노드포트를 통해 노드포트 서비스로 이동하고 이를 다시 쿠버네티스의 파드로 보내는 구조**였습니다. 이 방식은 매우 비효율적입니다.
- 그래서 쿠버네티스에서는 **로드밸런서**(LoadBalancer)라는 서비스 타입을 제공해 **파드를 외부에 노출하고 부하를 분산**합니다.

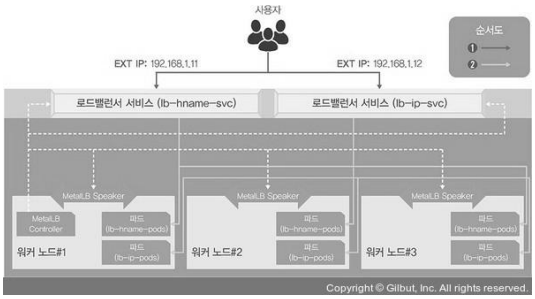
- 로드밸런서를 사용하려면 **로드밸런서를 이미 구현해 둔 서비스업체의 도움을 받아 쿠버네티스 클러스터 외부에 구현해야 합니다**. 클라우드에서 제공하는 쿠버네티스를 사용하고 있다면 다음과 같이 선언만 하면 됩니다. 이 실습은 클라우드 사(EKS, GKE, AKS)에서만 가능합니다.

```
[admin@Cloud_CMD ~]# kubectl expose deployment ex-lb --type=LoadBalancer --name=ex-svc
[admin@Cloud_CMD ~]# kubectl get services ex-svc
```

그러면 쿠버네티스 클러스터에 로드밸런서 서비스가 생성돼 외부와 통신할 수 있는 IP(EXTERNAL-IP)가 부여되고 외부와 통신할 수 있으며 부하도 분산됩니다.

## ▼ 💛 3.3.4 온프레미스에서 로드밸런서를 제공하는 MetalLB

- 그렇다면 **우리가 만든 테스트 가상 환경(온프레미스)에서는 로드밸런서를 사용**하는 것은 불가능할까요?




- **MetalLB**
  - 온프레미스에서 로드밸런서를 사용하려면 **내부에 로드밸런서 서비스를 받아주는 구성**이 필요한데, 이를 지원하는 것이 MetalLB입니다.
    - **베어메탈(bare metal, 운영 체제가 설치되지 않은 하드웨어)**로 구성된 쿠버네티스에서도 로드밸런서를 사용할 수 있게 고안된 프로젝트입니다.
    - MetalLB는 특별한 네트워크 설정이나 구성이 있는 것이 아니라 기존의 L2 네트워크(ARP/NDP)와 L3 네트워크(BGP)로 로드밸런서를 구현합니다. 그러므로 네트워크를 새로 배워야 할 부담이 없으며 연동하기도 매우 쉽습니다.
    - 이 책에서는 MetalLB의 L2 네트워크로 로드밸런서를 구현합니다. 그림에서 알 수 있듯이 기존의 로드밸런서와 거의 동일한 경로로 통신하며, 테스트 목적으로 두 개의 MetalLB 로드밸런서 서비스를 구현합니다.
  - **MetalLB 컨트롤러**는 **작동 방식(Protocol, 프로토콜)**을 정의하고 **EXTERNAL-IP를 부여**해 관리합니다.
    - 외부 로드 밸런서에 대한 구성 및 제어를 담당합니다. 클러스터 내의 리소스 상태를 모니터링하고, 사용자가 생성한 서비스의 유형 및 구성에 따라 적절한 로드 밸런서 구현을 결정합니다.
  - **MetalLB 스피커(speaker)**는 정해진 작동 방식(L2/ARP, L3/BGP)에 따라 경로를 만들 수 있도록 네트워크 정보를 광고하고 수집해 각 **파드의 경로를 제공**합니다.
  - 이때 **L2는 스피커 중에서 리더를 선출해 경로 제공을 총괄**하게 합니다.

▼ 🚩 MetalLB로 온프레미스 쿠버네티스 환경에서 로드밸런서 서비스 사용

**1.** 디플로이먼트를 이용해 2종류(lb-hname-pods, lb-ip-pods)의 파드를 생성합니다. 그리고 scale 명령으로 파드를 3개로 늘려 노드당 1개씩 파드가 배포되게 합니다.

```
[root@m-k8s ~]# kubectl create deployment lb-hname-pods --image=sysnet4admin/echo-hname
[root@m-k8s ~]# kubectl scale deployment lb-hname-pods --replicas=3
```

```
[root@m-k8s ~]# kubectl create deployment lb-ip-pods --image=sysnet4admin/echo-ip
[root@m-k8s ~]# kubectl scale deployment lb-ip-pods --replicas=3
```

**2.** 2종류의 파드가 3개씩 총 6개가 배포됐는지 확인합니다.

```
[root@m-k8s ~]# kubectl get pods
```

**3.** 인그레스와 마찬가지로 사전에 정의된 오브젝트 스펙으로 MetalLB를 구성합니다. 이렇게 하면 MetalLB에 필요한 요소가 모두 설치되고 독립적인 네임스페이스(metallb-system)도 함께 만들어집니다.

```
[root@m-k8s ~]# kubectl apply -f ~/_Book_k8sInfra/ch3/3.3.4/metallb.yaml
```

**4.** 배포된 MetalLB의 파드가 5개(controller 1개, speaker 4개)인지 확인하고, IP와 상태도 확인합니다.

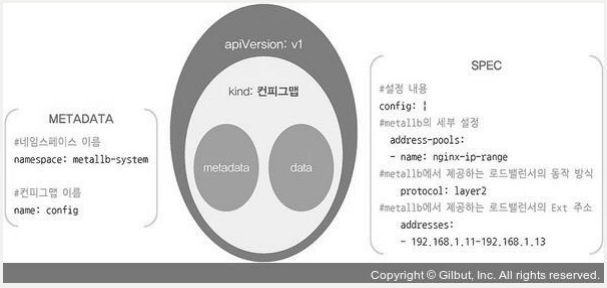
```
[root@m-k8s ~]# kubectl get pods -n metallb-system -o wide
```

```
[root@m-k8s ~]# kubectl get pods -n metallb-system -o wide
NAME                          READY   STATUS              RESTARTS   AGE   IP             NODE
controller-5d48db7f99-vbqfn   0/1     ContainerCreating   0          27s   <none>         w3-k8s
speaker-4rn8l                 0/1     ContainerCreating   0          27s   192.168.1.10   m-k8s
speaker-g8272                 0/1     ContainerCreating   0          27s   192.168.1.103  w3-k8s
speaker-vdzld                 0/1     ContainerCreating   0          27s   192.168.1.101  w1-k8s
speaker-wdst9                 0/1     ContainerCreating   0          27s   192.168.1.102  w2-k8s
```

**5.** 인그레스와 마찬가지로 MetalLB도 설정을 적용해야 하는데, 다음 방법으로 적용합니다. 이때 오브젝트는 ConfigMap을 사용합니다. **ConfigMap은 설정이 정의된 포맷**이라고 생각하면 됩니다.

```
[root@m-k8s ~]# kubectl apply -f ~/_Book_k8sInfra/ch3/3.3.4/metallb-l2config.yaml
```

▼ metallb-l2config.yaml




```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: nginx-ip-range
      protocol: layer2
      addresses:
      - 192.168.1.11-192.168.1.13
```

**6.** ConfigMap이 생성됐는지 확인합니다.

```
[root@m-k8s ~]# kubectl get configmap -n metallb-system
```

**7.** -o yaml 옵션을 주고 다시 실행해 MetalLB의 설정이 올바르게 적용됐는지 확인합니다.

```
[root@m-k8s ~]# kubectl get configmap -n metallb-system -o yaml
```

**8.** 모든 설정이 완료됐으니 이제 각 디플로이먼트(lb-hname-pods, lb-ip-pods)를 로드밸런서 서비스로 노출합니다.

```
[root@m-k8s ~]# kubectl expose deployment lb-hname-pods --type=LoadBalancer --name=lb-hname-svc --port=80
```

```
[root@m-k8s ~]# kubectl expose deployment lb-ip-pods --type=LoadBalancer --name=lb-ip-svc --port=80
```

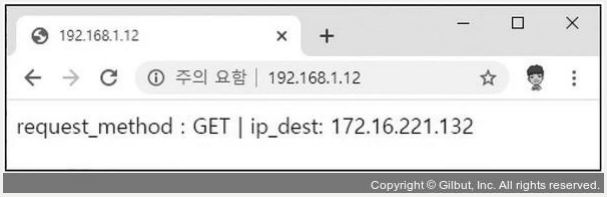
**9.** 생성된 로드밸런서 서비스별로 CLUSTER-IP와 EXTERNAL-IP가 잘 적용됐는지 확인합니다. 특히 EXTERNAL-IP에 ConfigMap을 통해 부여한 IP를 확인합니다.

```
[root@m-k8s ~]# kubectl get services
```

```
[root@m-k8s ~]# kubectl get services
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)        AGE
kubernetes      ClusterIP      10.96.0.1        <none>         443/TCP        4h9m
lb-hname-svc    LoadBalancer   10.106.122.14    192.168.1.11   80:31920/TCP   71s
lb-ip-svc       LoadBalancer   10.102.181.168   192.168.1.12   80:30071/TCP   67s
```

**10.** EXTERNAL-IP가 잘 작동하는지도 확인해 봅시다. 192.168.1.11로 접속합니다. 배포된 파드 중 하나의 이름이 브라우저에 표시되는지 확인합니다.


**11.** 192.168.1.12를 접속해 파드에 요청 방법과 IP가 표시되는지 확인합니다.




**12.** 파워셸 명령 창을 띄우고 셸 스크립트를 실행합니다. 로드밸런서 기능이 정상적으로 작동하면 192.168.1.11(EXTERNAL-IP)에서 반복적으로 결괏값을 가지고 옵니다.

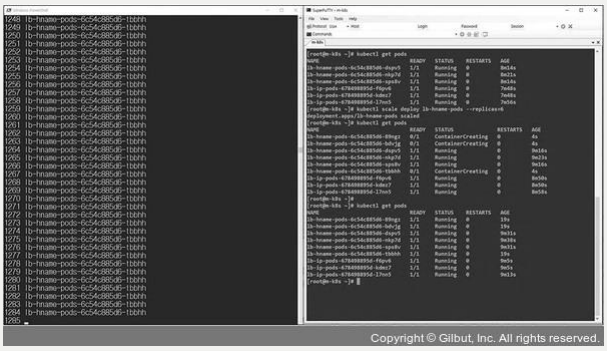
```
$i=0; while($true)
{
  % { $i++; write-host -NoNewline "$i $_" }
  (Invoke-RestMethod "http://192.168.1.11")-replace '\n', " "
}
```

```
1198 lb-hname-pods-79b95c7c7b-bvw6r
1199 lb-hname-pods-79b95c7c7b-bvw6r
1200 lb-hname-pods-79b95c7c7b-bvw6r
1201 lb-hname-pods-79b95c7c7b-n2nzp
1202 lb-hname-pods-79b95c7c7b-n2nzp
1203 lb-hname-pods-79b95c7c7b-n2nzp
1204 lb-hname-pods-79b95c7c7b-n2nzp
```

**13.** scale 명령으로 파드를 6개로 늘립니다.

```
[root@m-k8s ~]# kubectl scale deployment lb-hname-pods --replicas=6
```

**14.** 늘어난 파드 6개도 EXTERNAL-IP를 통해 접근되는지 확인합니다.




**15.** 다음 실습을 진행하기 전에 배포한 Deployment와 서비스는 삭제합니다. 단, MetalLB 설정은 계속 사용하므로 삭제하지 않습니다.

```
[root@m-k8s ~]# kubectl delete deployment lb-hname-pods
[root@m-k8s ~]# kubectl delete deployment lb-ip-pods
[root@m-k8s ~]# kubectl delete service lb-hname-svc
[root@m-k8s ~]# kubectl delete service lb-ip-svc
```

### ▼ 💛 3.3.5 부하에 따라 자동으로 파드 수를 조절하는 HPA

- 지금까지는 사용자 1명이 파드에 접근하는 방법을 알아봤습니다. 그런데 사용자가 갑자기 늘어난다면 어떻게 될까요? 파드가 더 이상 감당할 수 없어서 서비스 불가(여기서 서비스는 쿠버네티스의 서비스가 아닙니다)라는 결과를 초래할 수도 있습니다.
- 쿠버네티스는 이런 경우를 대비해 **부하량에 따라 디플로이먼트의 파드 수를 유동적으로 관리하는 기능을 제공**합니다. 이를 **HPA**(Horizontal Pod Autoscaler)라고 합니다.

▼ 🚩 코드 실습

**1.** 디플로이먼트 1개를 hpa-hname-pods라는 이름으로 생성합니다.

```
[root@m-k8s ~]# kubectl create deployment hpa-hname-pods --image=sysnet4admin/echo-hname
```

**2.** 앞에서 MetalLB를 구성했으므로 expose를 실행해 hpa-hname-pods를 로드밸런서 서비스로 바로 설정할 수 있습니다.

```
[root@m-k8s ~]# kubectl expose deployment hpa-hname-pods --type=LoadBalancer --name=hpa-hname-svc --port=80
```

**3.** 설정된 로드밸런서 서비스와 부여된 IP를 확인합니다.

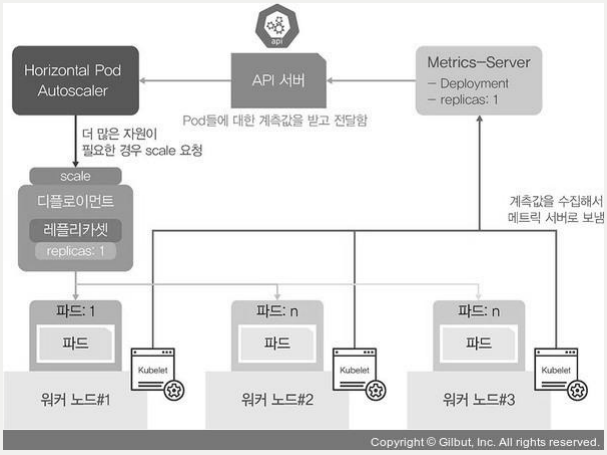
```
[root@m-k8s ~]# kubectl get services
```

```
[root@m-k8s ~]# kubectl get services
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)
hpa-hname-svc   LoadBalancer   10.99.185.174   192.168.1.11   80:30096/TCP
kubernetes      ClusterIP      10.96.0.1       <none>         443/TCP
```

**4.** HPA가 작동하려면 파드의 자원이 어느 정도 사용되는지 파악해야 합니다. **부하를 확인하는 명령**은 리눅스의 top(table of processes)과 비슷한 kubectl top pods입니다.

```
[root@m-k8s ~]# kubectl top pods
Error from server (NotFound): the server could not find the requested resource (get services http:heapster:)
```

자원을 요청하는 설정이 없다며 에러가 생기고 진행되지 않습니다. 왜 에러가 발생하는지 HPA가 작동하는 구조를 간단하게 살펴보겠습니다.




그림을 보면 HPA가 자원을 요청할 때 메트릭 서버(Metrics-Server)를 통해 계측값을 전달받습니다. 그런데 우리에게는 **현재 메트릭 서버가 없기 때문에 에러가 발생**하는 것입니다. 따라서 **계측값을 수집하고 전달해 주는 메트릭 서버를 설정**해야 합니다.

**5.** 쿠버네티스 메트릭 서버의 원본 소스(**https://github.com/kubernetes-sigs/metrics-server**)를 sysnet4admin 계정으로 옮겨 메트릭 서버를 생성하겠습니다.

```
[root@m-k8s ~]# kubectl create -f ~/_Book_k8sInfra/ch3/3.3.5/metrics-server.yaml
```

▼ 참고

- 서비스에서와 마찬가지로 메트릭 서버도 오브젝트 스펙 파일로 설치할 수 있습니다. 그러나 오브젝트 스펙 파일이 여러 개라서 git clone 이후에 디렉터리에 있는 파일들을 다시 실행해야 하는 번거로움이 있습니다. 또한 실습에서 사용하려면 몇 가지 추가 설정이 필요합니다.
- 수정된 코드

```
containers:
      - name: metrics-server
        image: k8s.gcr.io/metrics-server-amd64:v0.3.6
        args:
        # Manually Add for lab env(Sysnet4admin/k8s)
        # skip tls internal usage purpose
          - --kubelet-insecure-tls
        # kubelet could use internalIP communication
          - --kubelet-preferred-address-types=InternalIP
          - --cert-dir=/tmp
          - --secure-port=4443
```

  - **102번째 줄**: TLS(Transport Layer Security) 인증을 무시하게 합니다.
  - **104~106번째 줄**: kubelet이 내부 주소를 우선 사용하게 합니다.

**6.** 메트릭 서버를 설정하고 나면 kubectl top pods 명령의 결과를 제대로 확인할 수 있습니다. 파드의 top 값을 확인합니다. 현재는 아무런 부하가 없으므로 CPU와 MEMORY 값이 매우 낮게 나옵니다.

```
[root@m-k8s ~]# kubectl top pods
```

```
[root@m-k8s ~]# kubectl top pods
NAME                              CPU(cores)   MEMORY(bytes)
hpa-hname-pods-75f874d48c-gvfsm   0m           1Mi
```

현재는 scale 기준 값이 설정돼 있지 않아서 **파드 증설 시점**을 알 수가 없습니다. 따라서 **파드에 부하가 걸리기 전에 scale이 실행되게 디플로이먼트에 기준 값을 기록**합니다. 이때 Deployment를 새로 배포하기보다는 기존에 배포한 디플로이먼트 내용을 **edit 명령으로 직접 수정**합니다.

**7.** edit 명령을 실행해 배포된 디플로이먼트 내용을 확인합니다.

```
[root@m-k8s ~]# kubectl edit deployment hpa-hname-pods
```

```
    spec:
      containers:
      - image: sysnet4admin/echo-hname
        imagePullPolicy: Always
        name: echo-hname
        resources:
          limits:
            cpu: "50m"
          requests:
            cpu: "10m"
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
```

이때 추가한 값은 파드마다 주어진 부하량을 결정하는 기준이 됩니다.

여기서 사용한 단위 m은 milliunits의 약어로 1000m은 1개의 CPU가 됩니다. 따라서 10m은 파드의 CPU 0.01 사용을 기준으로 파드를 증설하게 설정한 것입니다. 또한 순간적으로 한쪽 파드로 부하가 몰릴 경우를 대비해 CPU 사용 제한을 0.05로 주었습니다.

**8.** 일정 시간이 지난 후 kubectl top pods를 실행하면 스펙이 변경돼 새로운 파드가 생성된 것을 확인할 수 있습니다.

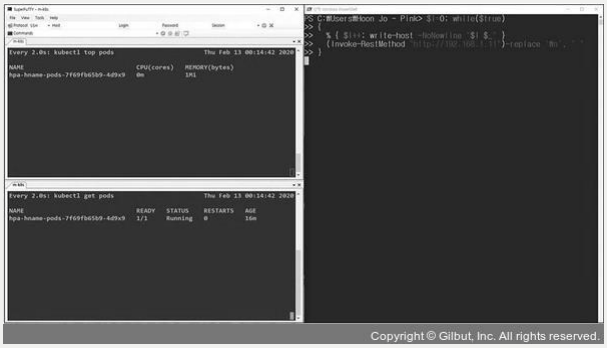
```
[root@m-k8s ~]# kubectl top pods
```

```
[root@m-k8s ~]#  kubectl top pods
NAME                                CPU(cores)   MEMORY(bytes)
hpa-hname-pods-696b8fcc99-6lxq6   0m           1Mi
```

**9.** hpa-hname-pods에 **autoscale을 설정**해서 특정 조건이 만족되는 경우에 자동으로 scale 명령이 수행되도록 하겠습니다. 여기서 min은 최소 파드의 수, max는 최대 파드의 수입니다. **cpu-percent는 CPU 사용량이 50%를 넘게 되면 autoscale하겠다는 뜻**입니다.

```
[root@m-k8s ~]# kubectl autoscale deployment hpa-hname-pods --min=1 --max=30 --cpu-percent=50
```

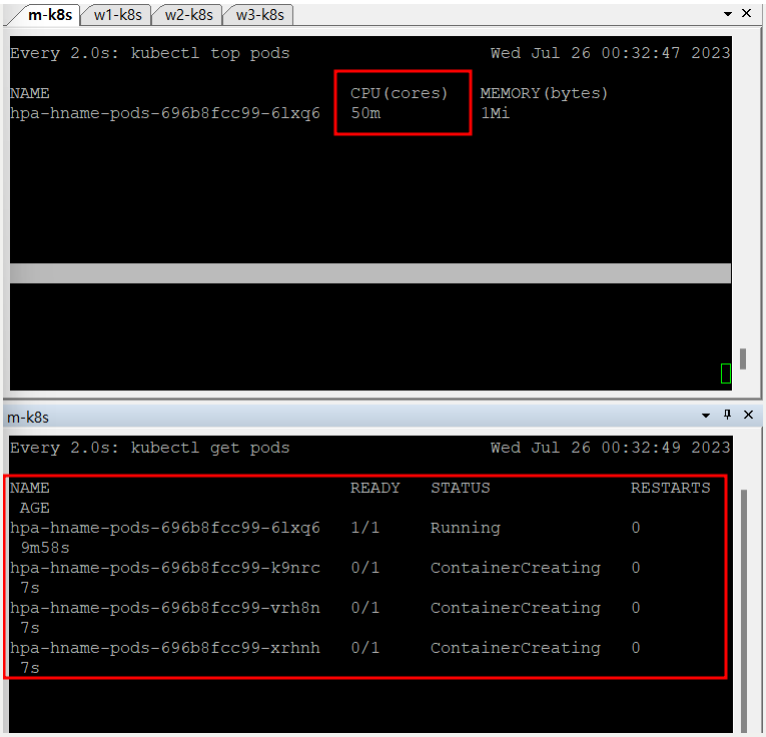
**10.** 테스트를 위해 화면을 다음과 같이 구성합니다. 왼쪽에 마스터 노드 창 두 개를 띄웁니다. 오른쪽에는 파워셸 창을 띄웁니다. 여기에 호스트 컴퓨터에서 제공하는 부하가 출력됩니다. 왼쪽 상단 창에서는 `watch kubectl top pods`를, 왼쪽 하단 창에서는 `watch kubectl get pods`를 실행합니다. (watch : 2초에 한 번씩 자동으로 상태를 확인)



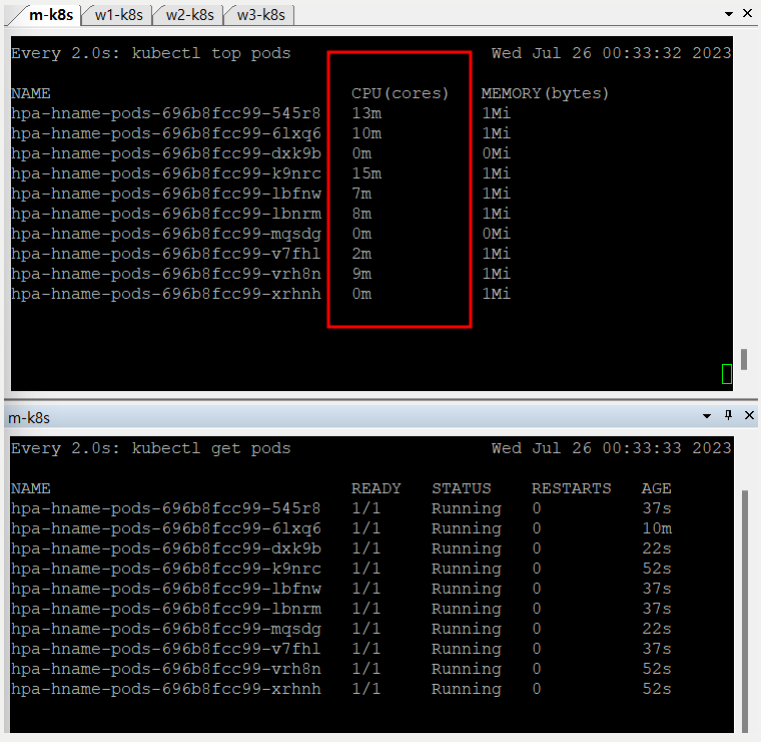

**11.** HPA를 테스트하기 위해 오른쪽에 있는 파워셸 창에서 반복문을 실행합니다. 부하를 주는 명령은 로드밸런서를 테스트했던 코드와 동일합니다. **왼쪽 상단 창에서 부하량을 감지**하는지 확인합니다.

```
$i=0; while($true)
{
  % { $i++; write-host -NoNewline "$i $_" }
  (Invoke-RestMethod "http://192.168.1.11")-replace '\n', " "
}
```

**12.** 부하량이 늘어남에 따라 **왼쪽 하단 창에서 파드가 새로 생성**되는지 확인합니다.

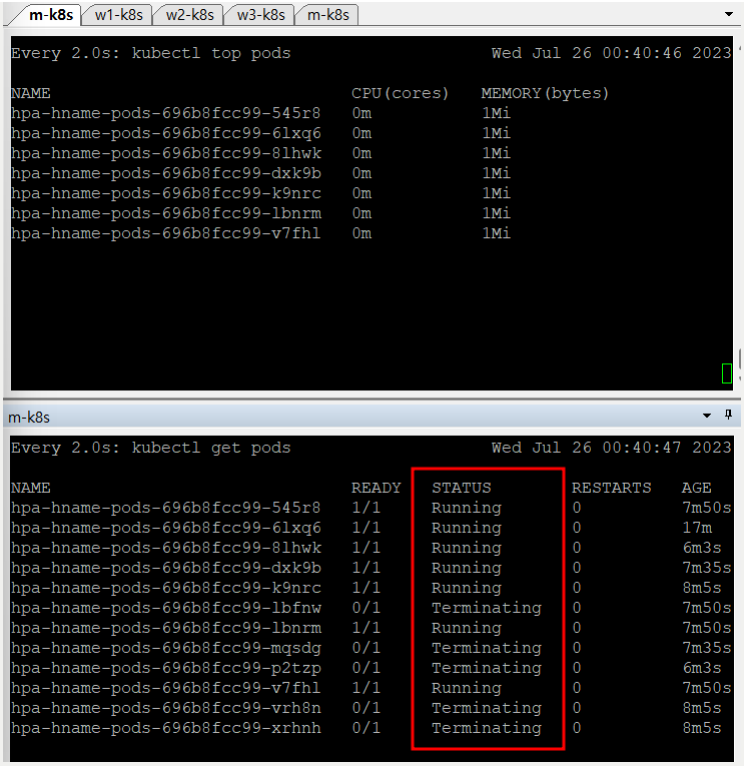



**13.** 부하 분산으로 생성된 **파드의 부하량이 증가**하는지 확인합니다.

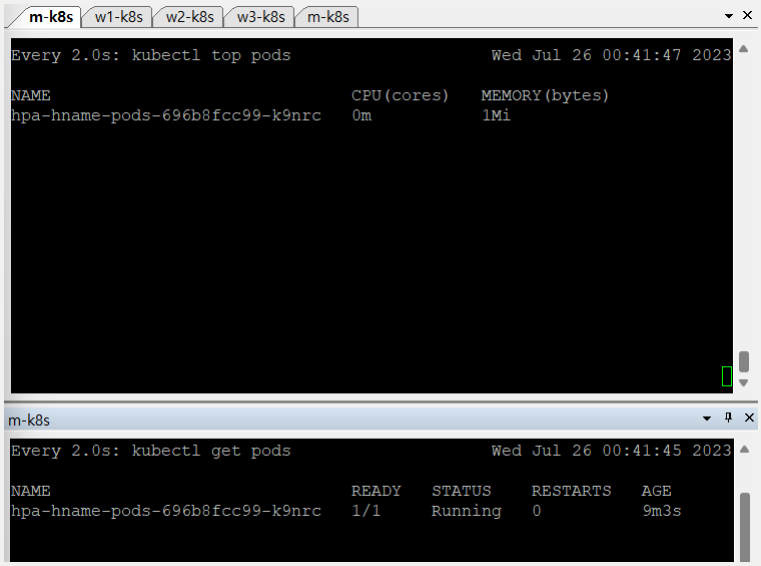



**14.** 더 이상 파드가 새로 생성되지 않는 안정적인 상태가 되는 것을 확인하고, 부하를 생성하는 오른쪽 파워셸 창을 종료합니다.

**15.** 일정 시간이 지난 후 더 이상 부하가 없으면 autoscale의 최소 조건인 파드 1개의 상태로 돌아가기 위해 파드가 종료되는 것을 확인합니다.




**16.** 사용하지 않는 파드는 모두 종료되고 1개만 남습니다.




**17.** 부하 테스트가 끝났습니다. 파드 부하량에 따라 HPA가 자동으로 파드 수를 조절하는 것을 확인했습니다. HPA를 잘 활용하면 자원의 사용을 극대화하면서 서비스 가동률을 높일 수 있습니다. 앞에서와 마찬가지로 생성한 디플로이먼트, 서비스, 메트릭 서버를 삭제합니다. MetalLB는 계속 사용하므로 삭제하지 않습니다.

```
[root@m-k8s ~]# kubectl delete deployment hpa-hname-pods
[root@m-k8s ~]# kubectl delete hpa hpa-hname-pods
[root@m-k8s ~]# kubectl delete service hpa-hname-svc
[root@m-k8s ~]# kubectl delete -f ~/_Book_k8sInfra/ch3/3.3.5/metrics-server.yaml
```

# ▼ ✨ 3.4 알아두면 쓸모 있는 쿠버네티스 오브젝트

## ▼ 💛 3.4.1 데몬셋

- **디플로이먼트의 replicas가 노드 수만큼 정해져 있는 형태**라고 할 수 있는데, 노드 하나당 파드 한 개만을 생성합니다.

- **노드의 단일 접속 지점으로 노드 외부와 통신**합니다.
- 파드가 1개 이상 필요하지 않습니다. 결국 **노드를 관리하는 파드라면 데몬셋으로 만드는 게 가장 효율적**입니다.

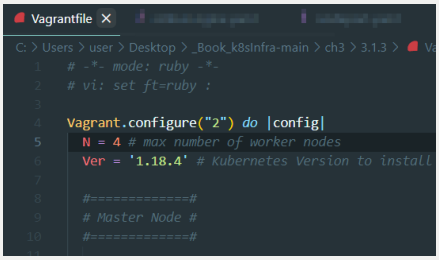
▼ 🚩 코드 실습

💡 **1.** 현재 MetalLB의 스피커가 각 노드에 분포돼 있는 상태를 확인합니다.

`[root@m-k8s ~]# kubectl get pods -n metallb-system -o wide`

```
[root@m-k8s ~]# kubectl get pods -n metallb-system -o wide
NAME                          READY   STATUS    RESTARTS   AGE   IP              NODE
controller-5d48db7f99-vbqfn   1/1     Running   0          69m   172.16.132.5    w3-k8s
speaker-4rn81                 1/1     Running   0          69m   192.168.1.10    m-k8s
speaker-g8272                 1/1     Running   0          69m   192.168.1.103   w3-k8s
speaker-vdzld                 1/1     Running   0          69m   192.168.1.101   w1-k8s
speaker-wdst9                 1/1     Running   0          69m   192.168.1.102   w2-k8s
```

**2.** 워커 노드를 1개 늘립니다.




**3.** 호스트 컴퓨터의 명령 창에서 C:\HashiCorp_Book_k8sInfra-main\ch3\3.1.3 경로로 이동한 다음 `vagrant up w4-k8s`를 실행해 새로운 워커 노드(w4-k8s)를 추가합니다.

**4.** w4-k8s이 추가되면 m-k8s에서 kubectl get pods -n metallb-system -o wide -w를 수행합니다. 여기서 -w는 watch의 약어로 **오브젝트 상태에 변화가 감지되면 해당 변화를 출력**합니다. 변화를 모두 확인했다면 Ctrl+C를 눌러 명령을 중지합니다.

`[root@m-k8s ~]# kubectl get pods -n metallb-system -o wide -w`

```
[root@m-k8s ~]# kubectl get pods -n metallb-system -o wide -w
NAME                          READY   STATUS    RESTARTS   AGE   IP              NODE     NOMINATED NODE   READINESS GATES
controller-5d48db7f99-vbqfn   1/1     Running   0          75m   172.16.132.5    w3-k8s   <none>           <none>
speaker-4rn81                 1/1     Running   0          75m   192.168.1.10    m-k8s    <none>           <none>
speaker-g8272                 1/1     Running   0          75m   192.168.1.103   w3-k8s   <none>           <none>
speaker-nnwzp                 1/1     Running   0          73s   192.168.1.104   w4-k8s   <none>           <none>
speaker-vdzld                 1/1     Running   0          75m   192.168.1.101   w1-k8s   <none>           <none>
speaker-wdst9                 1/1     Running   0          75m   192.168.1.102   w2-k8s   <none>           <none>
```

**5.** 자동으로 추가된 노드에 설치된 스피커가 데몬셋이 맞는지 확인합니다.

`[root@m-k8s ~]# kubectl get pods speaker-nnwzp -o yaml -n metallb-system`

```
  name: speaker-nnwzp
  namespace: metallb-system
  ownerReferences:
  - apiVersion: apps/v1
    blockOwnerDeletion: true
    controller: true
    kind: DaemonSet
    name: speaker
```

### ▼ 💛 3.4.2 컨피그맵

- 컨피그맵(ConfigMap)은 이름 그대로 **설정(config)을 목적으로 사용**하는 오브젝트입니다.

▼ 🚩 컨피그맵으로 작성된 **MetalLB의 IP 설정을 변경**해봅시다.

**1.** 테스트용 디플로이먼트를 cfgmap이라는 이름으로 생성합니다.

```
[root@m-k8s ~]# kubectl create deployment cfgmap --image=sysnet4admin/echo-hname
```

**2.** cfgmap을 로드밸런서(MetalLB)를 통해 노출하고 이름은 cfgmap-svc로 지정합니다.

```
[root@m-k8s ~]# kubectl expose deployment cfgmap --type=LoadBalancer --name=cfgmap-svc --port=80
```

**3.** 생성된 서비스의 IP(**192.168.1.11**)를 확인합니다.

```
[root@m-k8s ~]# kubectl get services
```

```
[root@m-k8s ~]#  kubectl get services
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)
  AGE
cfgmap-svc   LoadBalancer   10.101.81.161   192.168.1.11   80:30828/TCP
  8h
kubernetes   ClusterIP      10.96.0.1       <none>         443/TCP
```

**4.** 사전에 구성돼 있는 컨피그맵의 기존 IP(192.168.1.11~192.168.1.13)를 sed 명령을 사용해 192.168.1.21~192.168.1.23으로 변경합니다.

```
[root@m-k8s ~]# cat ~/_Book_k8sInfra/ch3/3.4.2/ metallb-l2config.yaml  | grep 192.
[root@m-k8s ~]#  sed -i 's/11/21/;s/13/23/' ~/_Book_k8sInfra/ch3/3.4.2/metallb-l2config.yaml
[root@m-k8s ~]# cat ~/_Book_k8sInfra/ch3/3.4.2/metallb-l2config.yaml | grep 192.
```

```
[root@m-k8s ~]# cat ~/_Book_k8sInfra/ch3/3.4.2/metallb-l2config.yaml | g
rep 192.
      - 192.168.1.11-192.168.1.13
[root@m-k8s ~]# sed -i 's/11/21/;s/13/23/' ~/_Book_k8sInfra/ch3/3.4.2/me
tallb-l2config.yaml
[root@m-k8s ~]#  cat ~/_Book_k8sInfra/ch3/3.4.2/metallb-l2config.yaml |
grep 192.
      - 192.168.1.21-192.168.1.23
[root@m-k8s ~]#
```

**5.** 컨피그맵 설정 파일(metallb-l2config.yaml)에 apply를 실행해 변경된 설정을 적용합니다.

```
[root@m-k8s ~]# kubectl apply -f ~/_Book_k8sInfra/ch3/3.4.2/metallb-l2config.yaml
```

**6.** MetalLB와 관련된 모든 파드를 삭제합니다. 삭제하고 나면 kubelet에서 해당 파드를 자동으로 모두 다시 생성합니다. --all은 파드를 모두 삭제하는 옵션입니다.

```
[root@m-k8s ~]# kubectl delete pods --all -n metallb-system
```

```
[root@m-k8s ~]# kubectl delete pods --all -n metallb-system
pod "controller-5d48db7f99-vbqfn" deleted
pod "speaker-4rn8l" deleted
pod "speaker-g8272" deleted
pod "speaker-nnwzp" deleted
pod "speaker-vdzld" deleted
pod "speaker-wdst9" deleted
```

**7.** 새로 생성된 MetalLB의 파드들을 확인합니다.

```
[root@m-k8s ~]# kubectl get pods -n metallb-system
```

```
[root@m-k8s ~]# kubectl get pods -n metallb-system
NAME                          READY   STATUS              RESTARTS
controller-5d48db7f99-cvv8c   0/1     ContainerCreating   0
speaker-f8k2f                 1/1     Running             0
speaker-hqlwb                 1/1     Running             0
speaker-q4g2b                 1/1     Running             0
speaker-q7wfw                 1/1     Running             0
speaker-vsn6x                 1/1     Running             0
```

**8.** 기존에 노출한 MetalLB 서비스(cfgmap-svc)를 삭제(delete)하고 동일한 이름으로 다시 생성해 새로운 컨피그맵을 적용한 서비스가 올라오게 합니다.

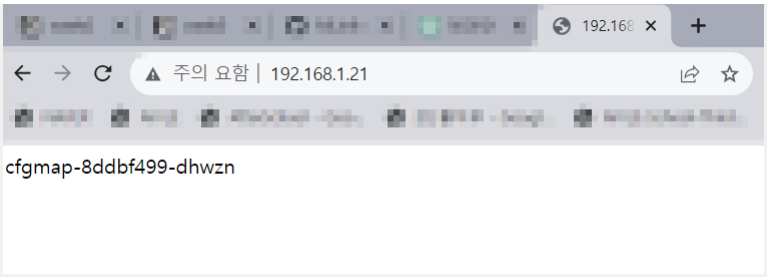
```
[root@m-k8s ~]# kubectl delete service cfgmap-svc
[root@m-k8s ~]# kubectl expose deployment cfgmap --type=LoadBalancer --name=cfgmap-svc --port=80
```

**9.** 변경된 설정이 적용돼 새로운 MetalLB 서비스의 IP가 192.168.1.21로 바뀌었는지 확인합니다.

```
[root@m-k8s ~]# kubectl get services
```

```
[root@m-k8s ~]# kubectl get services
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)
AGE
cfgmap-svc   LoadBalancer   10.97.52.48    192.168.1.21   80:31974/TCP
12s
kubernetes   ClusterIP      10.96.0.1      <none>         443/TCP
13h
```

**10.** 호스트 컴퓨터의 브라우저에서 192.168.1.21로 접속해 파드의 이름이 화면에 표시되는지 확인합니다.



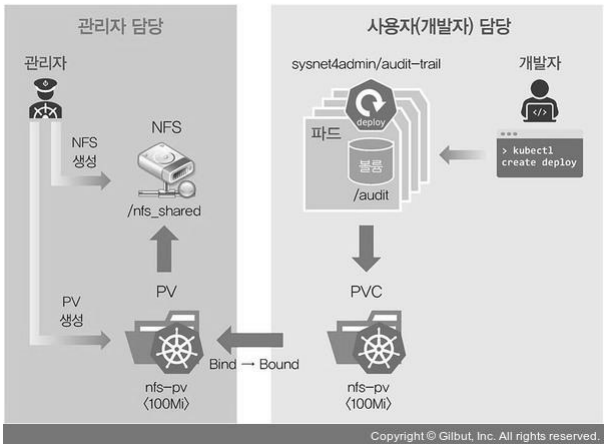

**11.** 이름이 표시되므로 설정 변경이 성공했습니다. 다음 테스트를 위해 생성한 디플로이먼트와 서비스를 삭제합니다.

```
[root@m-k8s ~]# kubectl delete deployment cfgmap
[root@m-k8s ~]# kubectl delete service cfgmap-svc
```

### ▼ 💛 3.4.3 PV와 PVC

- 때때로 **파드에서 생성한 내용을 기록하고 보관**하거나 **모든 파드가 동일한 설정 값을 유지하고 관리하기 위해 공유된 볼륨으로부터 공통된 설정을 가지고 올 수 있도록 설계**해야 할 때도 있습니다.
- 쿠버네티스는 이런 경우를 위해 다음과 같은 목적으로 다양한 형태의 볼륨을 제공합니다.
  - **임시**: emptyDir
  - **로컬**: host Path, local
  - **원격**: **persistentVolumeClaim**, cephfs, cinder, csi, fc(fibre channel), flexVolume, flocker, glusterfs, iscsi, nfs, portworxVolume, quobyte, rbd, scaleIO, storageos, vsphereVolume
  - **특수 목적**: downwardAPI, configMap, secret, azureFile, projected
  - **클라우드**: awsElasticBlockStore, azureDisk, gcePersistentDisk
- **PV와 PVC**
  - 쿠버네티스는 필요할 때 **PVC(PersistentVolumeClaim, 지속적으로 사용 가능한 볼륨 요청)**를 요청해 사용합니다.
  - PVC를 사용하려면 **PV(PersistentVolume, 지속적으로 사용 가능한 볼륨)로 볼륨을 선언**해야 합니다.
  - 간단하게 **PV는 볼륨을 사용할 수 있게 준비하는 단계**이고, **PVC는 준비된 볼륨에서 일정 공간을 할당받는 것**입니다.
  - 비유하면 PV는 요리사(관리자)가 피자를 굽는 것이고, PVC는 손님(사용자)가 원하는 만큼의 피자를 접시에 담아 가져오는 것입니다.
  - 사용자가 PVC를 요청 → PV가 NFS 볼륨에 연결 → 해당 PV를 마운트하여 NFS 서버의 공유된 디렉토리를 사용

#### ▼ 🚩 NFS 볼륨에 PV/PVC를 만들고 파드에 연결하기



**1.** PV로 선언할 볼륨을 만들기 위해 **NFS 서버를 마스터 노드에 구성합**니다.

```
[root@m-k8s ~]# mkdir /nfs_shared
[root@m-k8s ~]# echo '/nfs_shared 192.168.1.0/24(rw,sync,no_root_squash)' >> /etc/exports
```

- 공유되는 디렉터리는 /nfs_shared로 생성
- 해당 디렉터리를 NFS로 받아들일 IP 영역은 192.168.1.0/24
- 옵션을 적용해 /etc/exports에 기록
  - `/etc/exports` 파일은 **NFS 서버가 클라이언트들과 파일 시스템을 공유할 수 있도록 허용하는 규칙을 정의**하는 파일입니다. NFS 서버는 이 파일에 적힌 규칙에 따라 클라이언트가 접근할 수 있는 디렉토리 및 권한을 제어합니다.
  - 여기서 사용된 텍스트 `'/nfs_shared 192.168.1.0/24(rw,sync,no_root_squash)'` 는 NFS를 통해 `/nfs_shared` **디렉토리를 네트워크 세그먼트 `192.168.1.0/24` 의 호스트들에게 읽기 및 쓰기 권한(`rw`)으로 공유한다는 의미**입니다. `sync`는 파일의 변경 내용이 디스크에 동기화되기를 기다린다는 의미이며, `no_root_squash`는 원격 호스트에서 `root` 사용자로 접근할 때 `root`의 권한을 변경하지 않는다는 의미입니다
  - 이때 nfs-utils.x86_64는 현재 CentOS에 이미 설치돼 있으므로 설치하지 않아도 됩니다.

**2.** 해당 내용을 시스템에 적용해 NFS 서버를 활성화하고 다음에 시작할 때도 자동으로 적용되도록 systemctl enable --now nfs 명령을 실행합니다.

```
[root@m-k8s ~]# systemctl enable --now nfs
```

**3.** 다음 경로에 있는 오브젝트 스펙을 실행해 **PV를 생성**합니다.

```
[root@m-k8s ~]# kubectl apply -f ~/_Book_k8sInfra/ch3/3.4.3/nfs-pv.yaml
```

▼ nfs-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: nfs-pv
spec:
 capacity:
# 실제 사용량 제한이 아닌 쓸 수 있는 양을 레이블로 붙임
 storage: 100Mi
 accessModes:
# 여러 개의 노드가 읽고 쓸 수 있도록 마운트
 - ReadWriteMany
# PV가 제거됐을 때 작동하는 방법을 정의
 persistentVolumeReclaimPolicy: Retain # 유지 옵션
# nfs 서버의 연결 위치에 대한 설정
 nfs:
 server: 192.168.1.10
 path: /nfs_shared
```

**4.** kubectl get pv를 실행해 생성된 PV의 상태가 Available(사용 가능)임을 확인합니다.

```
[root@m-k8s ~]# kubectl get pv
```

**5.** 다음 경로에서 오브젝트 스펙을 실행해 **PVC를 생성**합니다.

```
[root@m-k8s ~]# kubectl apply -f ~/_Book_k8sInfra/ch3/3.4.3/nfs-pvc.yaml
```

▼ nfs-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Mi
```

PVC는 PV와 구성이 거의 동일합니다. 하지만 **PV는 사용자가 요청할 볼륨 공간을 관리자가 만들고, PVC는 사용자(개발자)간 볼륨을 요청하는 데 사용한다는** 점에서 차이가 있습니다. 여기서 요청하는 storage: 10Mi는 동적 볼륨이 아닌 경우에는 레이블 정도의 의미를 가집니다.

**6.** 생성된 PVC를 kubectl get pvc로 확인합니다.

- Bound(묶여짐) : PV와 PVC가 연결

```
[root@m-k8s ~]# kubectl get pvc
```

```
[root@m-k8s ~]# kubectl get pvc
NAME      STATUS   VOLUME   CAPACITY   ACCESS MODES
nfs-pvc   Bound    nfs-pv   100Mi      RWX
```

**7.** PV의 상태도 Bound로 바뀌었음을 kubectl get pv로 확인합니다.

```
[root@m-k8s ~]# kubectl get pv
```

```
[root@m-k8s ~]# kubectl get pv
NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
nfs-pv   100Mi      RWX            Retain           Bound    default/nfs-pvc
```

**8.** 생성한 **PVC를 볼륨으로 사용**하는 디플로이먼트 오브젝트 스펙을 배포합니다.

```
[root@m-k8s ~]# kubectl apply -f ~/_Book_k8sInfra/ch3/3.4.3/nfs-pvc-deploy.yaml
```

▼ nfs-pvc-deploy.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nfs-pvc-deploy
spec:
  replicas: 4
  selector:
    matchLabels:
      app: nfs-pvc-deploy
  template:
    metadata:
      labels:
        app: nfs-pvc-deploy
    spec:
      containers:
      # audit-trail 이미지 가져오기
      # 요청을 처리할 때마다 접속 정보를 로그로 기록
      - name: audit-trail
        image: sysnet4admin/audit-trail
        volumeMounts:
        - name: nfs-vol
          #볼륨 마운트 위치 지정
          mountPath: /audit
      volumes:
      - name: nfs-vol
        persistentVolumeClaim:
          # pvc로 생성된 볼륨을 마운트하기 위해 nfs-pvc 이름 사용
          claimName: nfs-pvc
```

**9.** 생성된 파드를 확인합니다.

```
[root@m-k8s ~]# kubectl get pods
```

**10.** 생성한 파드 중 하나에 exec로 접속합니다.

```
[root@m-k8s ~]# kubectl exec -it nfs-pvc-deploy-5fd9876c46-8gv98 -- /bin/bash
```

- 이후 사용자는 Pod 내부에서 `/bin/bash` 쉘 환경을 사용하여 원하는 명령어를 실행하거나 작업을 수행할 수 있게 됩니다.

**11.** df -h를 실행해 PVC의 마운트 상태를 확인합니다. 용량이 100Mi가 아닌(nfs-pv.yaml 참고) NFS 서버의 용량이 37G임을 확인합니다.

```
root@nfs-pvc-deploy-5fd9876c46-8gv98:/# df -h
```

```
root@nfs-pvc-deploy-5fd9876c46-8gv98:/# df -h
Filesystem                  Size  Used Avail Use% Mounted on
overlay                      37G  3.0G   35G   8% /
tmpfs                       1.3G     0  1.3G   0% /dev
tmpfs                       1.3G     0  1.3G   0% /sys/fs/cgroup
192.168.1.10:/nfs_shared     37G  3.4G   34G  10% /audit
/dev/mapper/centos_k8s-root  37G  3.0G   35G   8% /etc/hosts
shm                          64M     0   64M   0% /dev/shm
tmpfs                       1.3G   12K  1.3G   1% /run/secrets/kuberne
/serviceaccount
tmpfs                       1.3G     0  1.3G   0% /proc/acpi
tmpfs                       1.3G     0  1.3G   0% /proc/scsi
tmpfs                       1.3G     0  1.3G   0% /sys/firmware
```

**12.** 오른쪽에 m-k8s 명령 창을 1개 더 열고 **audit-trail (요청을 처리할 때마다 접속 정보를 로그로 기록) 컨테이너의 기능을 테스트**합니다. 외부에서 파드(nfs-pv-deploy)에 접속할 수 있도록 expose로 로드밸런서 서비스를 생성합니다.

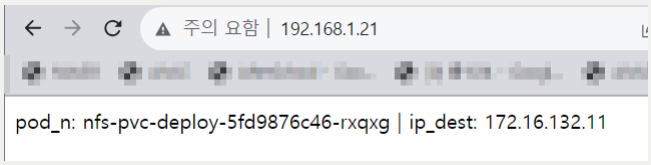
```
[root@m-k8s ~]# kubectl expose deployment nfs-pvc-deploy --type=LoadBalancer --name=nfs-pvc-deploy-svc --port=80
```

**13.** 생성한 로드밸런서 서비스의 IP를 확인합니다.

```
[root@m-k8s ~]# kubectl get services
```

```
[root@m-k8s ~]#  kubectl get services
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)
    AGE
kubernetes          ClusterIP      10.96.0.1       <none>         443/TCP
    14h
nfs-pvc-deploy-svc  LoadBalancer   10.100.50.94    192.168.1.21   80:30939/
```

**14.** 호스트 컴퓨터에서 브라우저를 엽니다. 192.168.1.21에 접속해 파드 이름과 IP가 표시되는지 확인합니다.




**15.** exec를 통해 접속한 파드에서 ls /audit 명령을 실행해 접속 기록 파일이 남았는지 확인합니다. cat으로 해당 파일의 내용도 함께 확인합니다.

```
root@nfs-pvc-deploy-5fd9876c46-8gv98:/# ls /audit
root@nfs-pvc-deploy-5fd9876c46-8gv98:/# cat /audit/audit_nfs-pvc-deploy-5fd9876c46-rxqxg.log
```

```
root@nfs-pvc-deploy-5fd9876c46-8gv98:/# cat /audit/audit_nfs-pvc
76c46-rxqxg.log
26/Jul/2023:10:11:34 +0900  172.16.132.11  GET
26/Jul/2023:10:11:34 +0900  172.16.132.11  GET
```

**16.** 마스터 노드(m-k8s)에서 scale 명령으로 파드를 4개에서 8개로 증가시킵니다.

```
[root@m-k8s ~]# kubectl scale deployment nfs-pvc-deploy --replicas=8
```

```
[root@m-k8s ~]# kubectl get pods
NAME                              READY   STATUS
nfs-pvc-deploy-5fd9876c46-8gv98   1/1     Running
nfs-pvc-deploy-5fd9876c46-ctv8q   0/1     ContainerCreating
nfs-pvc-deploy-5fd9876c46-ggtkk   0/1     ContainerCreating
nfs-pvc-deploy-5fd9876c46-kjlxv   0/1     ContainerCreating
nfs-pvc-deploy-5fd9876c46-rxqxg   1/1     Running
nfs-pvc-deploy-5fd9876c46-s7lmc   1/1     Running
nfs-pvc-deploy-5fd9876c46-tjpsj   0/1     ContainerCreating
nfs-pvc-deploy-5fd9876c46-wfmjz   1/1     Running
```

**17.** 생성된 파드를 확인합니다.

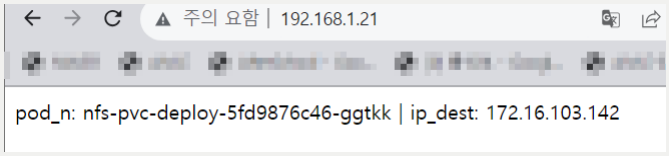
```
[root@m-k8s ~]# kubectl get pods
```

**18.** 최근에 증가한 4개의 파드 중 1개를 선택해 exec로 접속하고 기록된 audit 로그가 동일한지 확인합니다.

```
[root@m-k8s ~]# kubectl exec -it nfs-pvc-deploy-5fd9876c46-ctv8q -- /bin/bash
```

```
root@nfs-pvc-deploy-5fd9876c46-ctv8q:~# cat /audit/audit_nfs-pvc-deploy-5f
76c46-rxqxg.log
26/Jul/2023:10:11:34 +0900  172.16.132.11  GET
26/Jul/2023:10:11:34 +0900  172.16.132.11  GET
```

**19.** 다른 브라우저를 열고 192.168.1.21로 접속해 다른 파드 이름과 IP가 표시되는지를 확인합니다.




**20.** exec로 접속한 파드에서 ls /audit을 실행해 새로 추가된 audit 로그를 확인합니다. 그리고 cat으로 기록된 내용도 함께 확인합니다.

```
root@nfs-pvc-deploy-5fd9876c46-ggtkk:~# ls /audit
root@nfs-pvc-deploy-5fd9876c46-ggtkk:~# cat /audit/audit_nfs-pvc-deploy-5fd9876c46-ggtkk.log
```

```
root@nfs-pvc-deploy-5fd9876c46-ctv8q:~# cat /audit/audit_nfs-pvc
76c46-ggtkk.log
26/Jul/2023:10:23:51 +0900  172.16.103.142  GET
26/Jul/2023:10:23:51 +0900  172.16.103.142  GET
```

**21.** 기존에 접속한 파드에서도 동일한 로그가 audit에 기록돼 있는지 확인합니다.

```
root@nfs-pvc-deploy-5fd9876c46-8gv98:/# ls /audit
```

```
root@nfs-pvc-deploy-5fd9876c46-8gv98:/# ls /audit
audit_nfs-pvc-deploy-5fd9876c46-ggtkk.log
audit_nfs-pvc-deploy-5fd9876c46-rxqxg.log
```

▼ 🚩 NFS 볼륨을 파드에 직접 마운트하기

💡 **1.** 사용자가 관리자와 **동일한 단일 시스템이라면 PV와 PVC를 사용할 필요가 없습니다.** 따라서 단순히 볼륨을 마운트하는지 확인하고 넘어가겠습니다.

`[root@m-k8s ~]# kubectl apply -f ~/_Book_k8sInfra/ch3/3.4.3/nfs-ip.yaml`

▼ nfs-ip.yaml

PV와 PVC를 거치지 않고 바로 NFS 서버로 접속하는 것을 확인할 수 있습니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nfs-ip
spec:
  replicas: 4
  selector:
    matchLabels:
      app: nfs-ip
  template:
    metadata:
      labels:
        app: nfs-ip
    spec:
      containers:
      - name: audit-trail
        image: sysnet4admin/audit-trail
        volumeMounts:
        - name: nfs-vol
          mountPath: /audit
      volumes:
      - name: nfs-vol
        nfs:
          server: 192.168.1.10
          path: /nfs_shared
```

**2.** 새로 배포된 파드를 확인하고 그중 하나에 exec로 접속합니다.

`[root@m-k8s ~]# kubectl get pods`

`[root@m-k8s ~]# kubectl exec -it nfs-ip-7789f445b7-vl7ht -- /bin/bash`

**3.** 접속한 파드에서 ls /audit를 실행해 동일한 NFS 볼륨을 바라보고 있음을 확인합니다.

`root@nfs-ip-84fd4d6f69-475vb:/# ls /audit`

```
[root@m-k8s ~]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nfs-ip-7789f445b7-8ghgs             1/1     Running   0          30s
nfs-ip-7789f445b7-ktw47             1/1     Running   0          30s
nfs-ip-7789f445b7-tqn4x             1/1     Running   0          30s
nfs-ip-7789f445b7-vl7ht             1/1     Running   0          30s
nfs-pvc-deploy-5fd9876c46-8gv98     1/1     Running   0          32m
nfs-pvc-deploy-5fd9876c46-ctv8q     1/1     Running   0          15m
nfs-pvc-deploy-5fd9876c46-ggtkk     1/1     Running   0          15m
nfs-pvc-deploy-5fd9876c46-kjlxv     1/1     Running   0          15m
nfs-pvc-deploy-5fd9876c46-rxqxg     1/1     Running   0          32m
nfs-pvc-deploy-5fd9876c46-s7lmc     1/1     Running   0          32m
nfs-pvc-deploy-5fd9876c46-tjpsj     1/1     Running   0          15m
nfs-pvc-deploy-5fd9876c46-wfmjz     1/1     Running   0          32m
[root@m-k8s ~]# kubectl exec -it nfs-ip-7789f445b7-vl7ht -- /bin/bash
root@nfs-ip-7789f445b7-vl7ht:/# ls /audit
audit_nfs-pvc-deploy-5fd9876c46-ggtkk.log
audit_nfs-pvc-deploy-5fd9876c46-rxqxg.log
```

**4.** 다음 진행을 위해 설치한 PV와 PVC를 제외한 나머지인 파드와 서비스를 삭제합니다.

`[root@m-k8s ~]# kubectl delete deployment nfs-pvc-deploy`

`[root@m-k8s ~]# kubectl delete deployment nfs-ip`

`[root@m-k8s ~]# kubectl delete service nfs-pvc-deploy-svc`

실제로 PV와 PVC를 구성해서 **PV와 PVC를 구성하는 주체가 관리자와 사용자로 나뉜다**는 것을 확인했습니다. **또한 관리자와 사용자가 나뉘어 있지 않다면 굳이 PV와 PVC를 통하지 않고 바로 파드에 공유가 가능한 NFS 볼륨을 마운트**할 수 있음을 알았습니다.

## ▼ 💛 3.4.4 스테이트풀셋

- 지금까지는 파드가 replicas에 선언된 만큼 무작위로 생성될 뿐이었습니다. **그런데 파드가 만들어지는 이름과 순서를 예측**해야 할 때가 있습니다.
- 주로 레디스(Redis), 주키퍼(Zookeeper), 카산드라(Cassandra), 몽고DB(MongoDB) 등의 마스터-슬레이브 구조 시스템에서 필요합니다.
- **스테이트풀셋**(StatefulSet)
  - **volumeClaimTemplates 기능을 사용해 PVC를 자동으로 생성**할 수 있고, **각 파드가 순서대로 생성**되기 때문에 고정된 이름, 볼륨, 설정 등을 가질 수 있습니다. 그래서 StatefulSet(이전 상태를 기억하는 세트)이라는 이름을 사용합니다. 다만, 효율성 면에서 좋은 구조가 아니므로 요구 사항에 맞게 적절히 사용하는 것이 좋습니다.
  - 스테이트풀셋은 디플로이먼트와 형제나 다름없는 구조라 디플로이먼트에서 오브젝트 종류를 변경하면 바로 실습할 수 있습니다.

▼ 🚩 코드 실습

**1.** PV와 PVC는 앞에서 이미 생성했으므로 바로 스테이트풀셋을 다음 명령으로 생성합니다.

```
[root@m-k8s ~]# kubectl apply -f ~/_Book_k8sInfra/ch3/3.4.4/nfs-pvc-sts.yaml
```

▼ nfs-pvc-sts.yaml

- `serviceName: sts-svc-domain #statefulset need it` 이 추가된 것 외에는 앞의 nfs-pvc-deploy.yaml 코드와 동일합니다.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nfs-pvc-sts
spec:
  replicas: 4
  serviceName: sts-svc-domain #statefulset need it
  selector:
    matchLabels:
      app: nfs-pvc-sts
  template:
    metadata:
      labels:
        app: nfs-pvc-sts
    spec:
      containers:
      - name: audit-trail
        image: sysnet4admin/audit-trail
        volumeMounts:
        - name: nfs-vol
          mountPath: /audit
      volumes:
      - name: nfs-vol
        persistentVolumeClaim:
          claimName: nfs-pvc
```

**2.** 앞의 명령을 실행한 후에 파드가 생성되는지는 kubectl get pods -w 명령으로 확인합니다. 다음과 같이 순서대로 하나씩 생성하는 것을 볼 수 있습니다.

```
[root@m-k8s ~]# kubectl get pods -w
```

```
[root@m-k8s ~]# kubectl get pods -w
NAME            READY   STATUS    RESTARTS   AGE
nfs-pvc-sts-0   1/1     Running   0          34s
nfs-pvc-sts-1   1/1     Running   0          24s
nfs-pvc-sts-2   1/1     Running   0          14s
nfs-pvc-sts-3   1/1     Running   0          7s
```

**3.** 생성한 스테이트풀셋에 expose를 실행합니다. 그런데 에러가 발생합니다. 이는 **expose 명령이 스테이트풀셋을 지원하지 않기 때문**입니다. **해결하려면 파일로 로드밸런서 서비스를 작성, 실행**해야 합니다.

```
[root@m-k8s ~]# kubectl expose statefulset nfs-pvc-sts --type=LoadBalancer --name=nfs-pvc-sts-svc --port=80
```

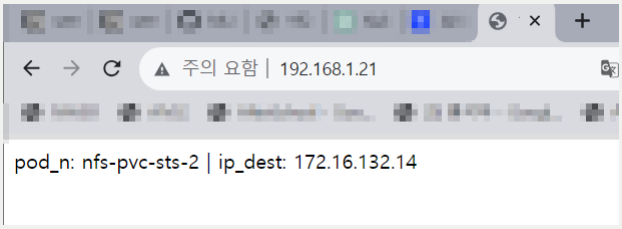
```
error: cannot expose a StatefulSet.apps
```

**4.** 다음 경로를 적용해 **스테이트풀셋을 노출하기 위한 서비스를 생성**하고, kubectl get service 명령으로 생성한 로드밸런서 서비스를 확인합니다.

```
[root@m-k8s ~]# kubectl apply -f ~/_Book_k8sInfra/ch3/3.4.4/nfs-pvc-sts-svc.yaml
[root@m-k8s ~]# kubectl get services
```

```
[root@m-k8s ~]# kubectl get services
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)
  AGE
kubernetes        ClusterIP      10.96.0.1       <none>         443/TCP
  14h
nfs-pvc-sts-svc   LoadBalancer   10.104.186.15   192.168.1.21   80:32648/TCP
```

**5.** 호스트 컴퓨터에서 브라우저를 엽니다. 192.168.1.21에 접속해 파드 이름과 IP가 표시되는지를 확인합니다.




**6.** exec로 파드에 접속한 후에 ls /audit -l로 새로 접속한 파드의 정보가 추가됐는지 확인합니다. 정보를 확인하고 나면 exit로 파드를 빠져나옵니다.

```
[root@m-k8s ~]# kubectl exec -it nfs-pvc-sts-0 -- /bin/bash
root@nfs-pvc-sts-0:/# ls -l /audit
```

#exit

```
root@nfs-pvc-sts-0:/# ls -l /audit
total 12
-rw-r--r--. 1 root root  96 Jul 26 10:23 audit_nfs-pvc-deploy-5fd9876
k.log
-rw-r--r--. 1 root root 188 Jul 26 10:23 audit_nfs-pvc-deploy-5fd9876
g.log
-rw-r--r--. 1 root root  94 Jul 26 10:37 audit_nfs-pvc-sts-2.log
```

**7.** 스테이트풀셋의 파드를 삭제합니다. 파드는 생성된 순서의 **역순으로 삭제**되는데, kubectl get pods -w를 실행하면 삭제되는 과정을 볼 수 있습니다.

```
[root@m-k8s ~]# kubectl delete statefulset nfs-pvc-sts
[root@m-k8s ~]# kubectl get pods -w
```

```
[root@m-k8s ~]# kubectl get pods -w

NAME            READY   STATUS        RESTARTS   AGE
nfs-pvc-sts-0   1/1     Terminating   0          7m
nfs-pvc-sts-1   1/1     Terminating   0          6m50s
nfs-pvc-sts-2   1/1     Terminating   0          6m40s
nfs-pvc-sts-3   1/1     Terminating   0          6m33s
nfs-pvc-sts-3   0/1     Terminating   0          6m35s
nfs-pvc-sts-2   0/1     Terminating   0          6m42s
nfs-pvc-sts-1   0/1     Terminating   0          6m53s
nfs-pvc-sts-0   0/1     Terminating   0          7m3s
```

일반적으로 스테이트풀셋은 volumeClaimTemplates를 이용해 자동으로 각 파드에 독립적인 스토리지를 할당해 구성할 수 있습니다. ~~그러나 이 책의 NFS 환경에서는 동적으로 할당받을 수 없기 때문에 이 부분은 과제로 남겨두겠습니다.~~