(예진) 인프라 환경 공부 - 3장 (3)~(4)

☑ Done	☐
Do Date	@2023년 7월 25일

3.3 쿠버네티스 연결을 담당하는 서비스

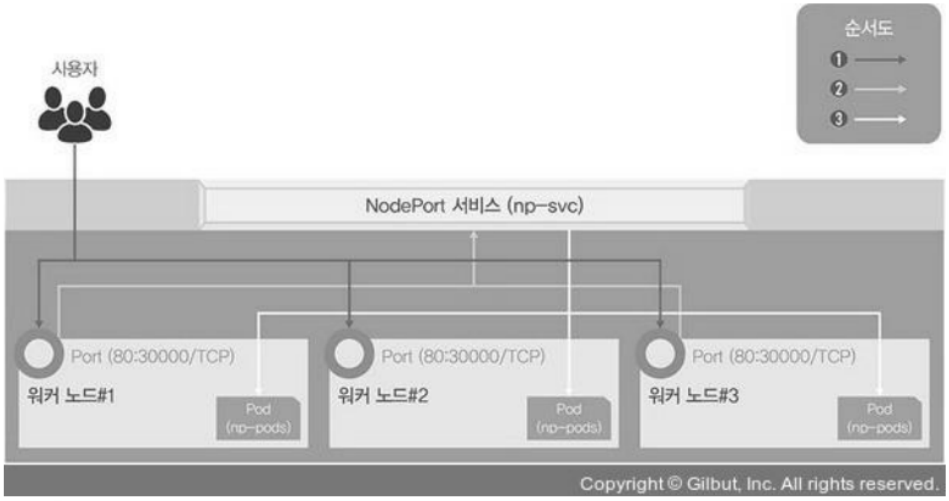
✨이제는 쿠버네티스 외부에서 파드를 이용하는 방법을 배울 차례!✨

✅ **쿠버네티스의 "서비스(service)"**
→ 외부에서 쿠버네티스 클러스터에 접속하는 방법

▼ (1) 가장 간단하게 연결하는 노드포트

노드포트(NodePort)

- 외부에서 쿠버네티스 클러스터의 내부에 접속하는 가장 쉬운 방법!
- 노드포트 서비스를 설정하면, 모든 워커 노드의 특정 포트(노드포트)를 연다
  → 여기로 오는 모든 요청을 노드포트 서비스로 전달
  → 노드포트 서비스가 해당 업무를 처리할 수 있는 파드로 요청 전달



▲ 그림 3-35 노드포트 서비스 구성도

노트포드 서비스로 외부에서 접속하기

1. 디플로이먼트로 파드 생성
   - sysnet4admin 계정에 있는 echo-hname 이미지 사용하기

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

2. 배포된 파드 확인

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

3. 노드포트 서비스 생성
   - 편의를 위해 이미 정의한 오브젝트 스펙을 사용하겠음!

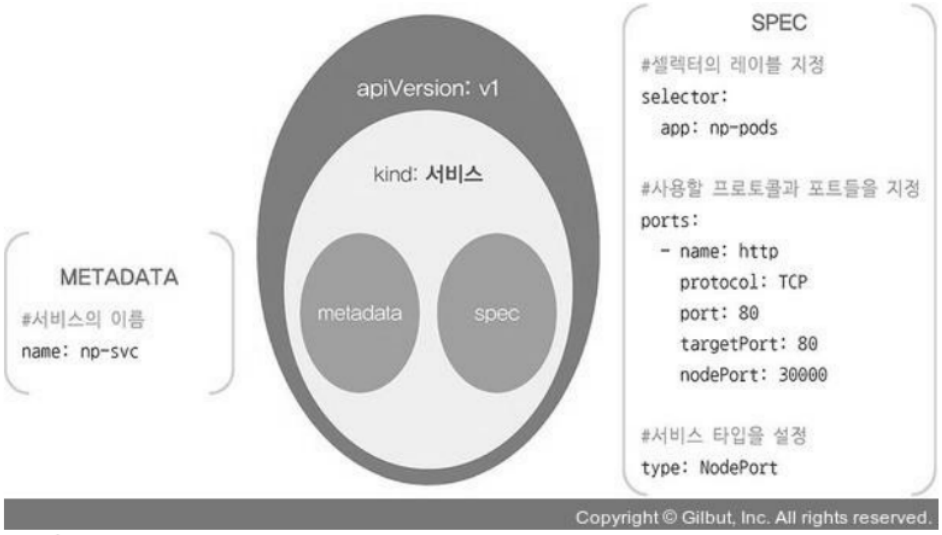
```
[root@m-k8s ~] kubectl create -f ~/_Book_k8sInfra/ch3/3.3.1/nodeport.yaml
service/np-svc created
```

   - 사용하는 오브젝트 스펙은 다음과 같음

     (nodeport.yaml)

```
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
```



▲ 그림 3-36 nodeport.yaml 파일의 구조

- 기존 파드 구조에서 kind가 Service로 바뀌고, spec에 컨테이너에 대한 정보가 없음
- 접속에 필요한 네트워크 관련 정보(protocol, port, targetPort, nodePort)와 서비스의 type을 NodePort로 지정함

4. 노드포트 서비스로 생성한 np-svc 서비스 확인

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

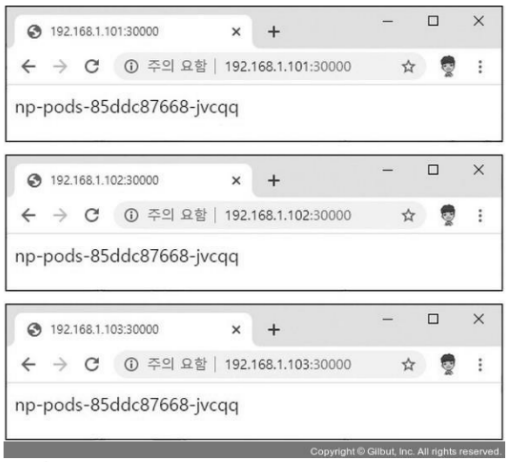
- name에 np-svc, type에 NodePort, port에 30000 확인~
- CLUSTER-IP(10.103.31.217)은 쿠버네티스 클러스터의 내부에서 사용하는 IP로, 자동으로 지정됨

5. 쿠버네티스 클러스터의 워커 노드 IP 확인

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

6. 호스트 노트북(또는 PC)에서 웹 브라우저를 띄우고 192.168.1.101~103(확인한 워커 노드의 IP)와 30000번(노드포트의 포트 번호)으로 접속해 외부에서 접속되는지 확인함. 화면에 파드 이름이 표시되는지도 확인함.

   → 이때, 파드가 하나이므로 화면에 보이는 이름은 모두 동일하다!



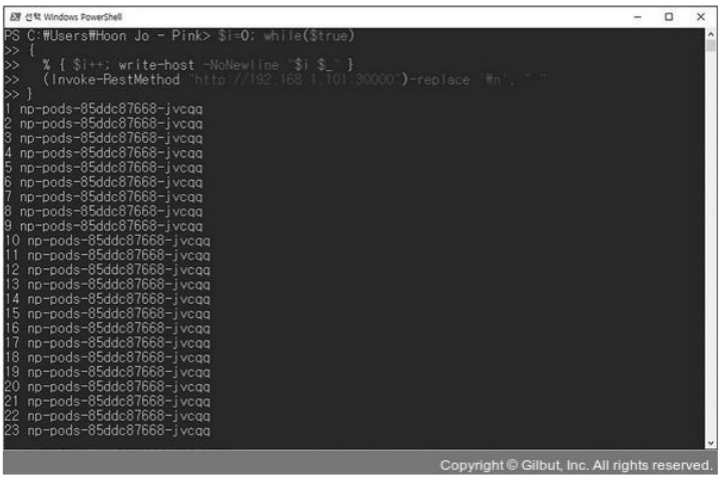
▲ **그림 3-37** 외부에서 워커 노드 IP로 접속하기

부하 분산 테스트하기

- 부하가 분산되는지(로드밸런서 기능) 확인해보기
  - 디플로이먼트로 생성된 파드 1개에 접속하고 있는 중에 파드가 3개로 증가하면 접속이 어떻게 바뀔까?

1. 호스트 노트북(또는 PC)에서 powershell 명령 창을 띄우고, 다음 명령을 실행함.
   - 반복적으로 192.168.1.101:30000에 접속해 접속한 파드 이름을 화면에 표시(Invoke-RestMethod)하는 명령어
   - 파드가 1개에서 3개로 늘어나는 시점 관찰이 가능함

```
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', " "
}
```



▲ **그림 3-38** 파워셸로 노드포트 접속 테스트하기

→ 명령을 실행하면, 현재 접속한 호스트 이름을 순서대로 출력함

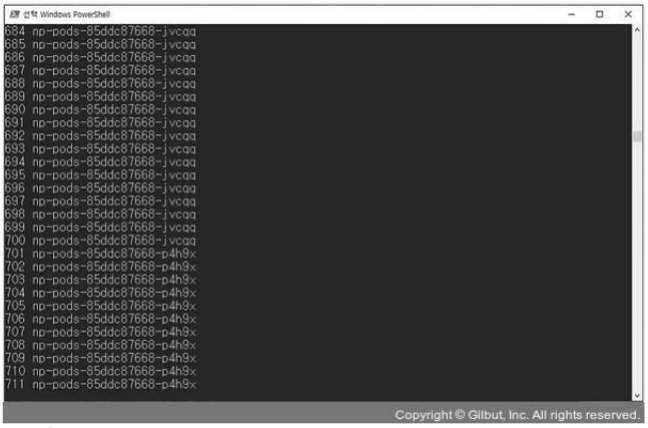
2. powershell로 코드를 실행한 후, 쿠버네티스 마스터 노드에서 scale을 실행해 파드를 3개로 증가시킴

```
[root@m-k8s ~] kubectl scale deployment np-pods --replicas=3
deployment.apps/np-pods scaled
```

3. 배포된 파드를 확인함

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

4. 부하 분산이 제대로 되는지 확인함
   - powershell 명령 창을 확인해 표시하는 파드 이름에 배포된 파드 3개가 돌아가면서 표시되는지 확인함



▲ **그림 3-39** 노드포트로 접속하는 파드 이름 확인하기

❓ 어떻게 추가된 파드를 외부에서 추적해 접속하는 걸까?
→ 노드포트의 오브젝트 스펙에 적힌 np-pods와 디플로이먼트의 이름을 확인해, 동일하면 같은 파드라고 간주하기 때문!

- 노드포트의 오브젝트 스펙에 적힌 np-pods와 디플로이먼트의 이름을 확인해, 동일하면 같은 파드라고 간주하기 때문!
- 추적 방법은 다양하지만, 여기서는 가장 간단하게 이름으로 진행함!

  (노드포트의 오브젝트 스펙인 nodeport.yaml 파일 일부)

```
...
spec:
  selector:
    app: np-pods
...
```

expose로 노드포트 서비스 생성하기

- 노드포트 서비스는 스펙 파일뿐만 아니라, expose 명령어로도 생성이 가능함!

1. expose 명령어를 사용해 서비스로 내보낼 디플로이먼트를 np-pods로 지정함
   - 다음과 같이 지정하기
     - 해당 서비스 이름: np-svc-v2
     - 타입: NodePort (이때, 반드시 대소문자 구분하기!)
     - 서비스가 파드로 보내줄 연결 포트: 80번

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

2. 생성된 서비스 확인

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

   - 오브젝트 스펙으로 생성할 때는 노드포트 포트 번호를 30000번으로 지정했지만, expose를 사용하면 노드포트의 포트 번호를 지정할 수 없음!

     → 30000~32767에서 임의로 지정됨

3. 호스트 노트북(또는 PC)에서 웹 브라우저를 띄우고 192.168.1.101:32122(무작위로 생성된 포트 번호)에 접속함. 배포된 파드 중 하나의 이름이 웹 브라우저에 표시되는지 확인함

▲ 그림 3-40 외부에서 노드포트의 포트 번호로 접속하기

4. 배포한 디플로이먼트, 서비스 2개 모두 삭제 (다음 실습을 위함)

```
[root@m-k8s ~] kubectl delete deployment np-pods
deployment.apps "np-pods" deleted
[root@m-k8s ~] kubectl delete services np-svc
service "np-svc" deleted
[root@m-k8s ~] kubectl delete services np-svc-v2
service "np-svc-v2" deleted
```

▼ (2) 사용 목적별로 연결하는 인그레스

💡 노드포트 서비스는 포트를 중복 사용할 수 없어서 1개의 노드포트에 1개의 디플로이먼트만 적용됨
→ 여러 개의 디플로이먼트가 있을 때, 그 수만큼 노드포트 서비스를 구동해야 할까?
⇒ 이럴 때, 쿠버네티스에서 인그레스를 사용함

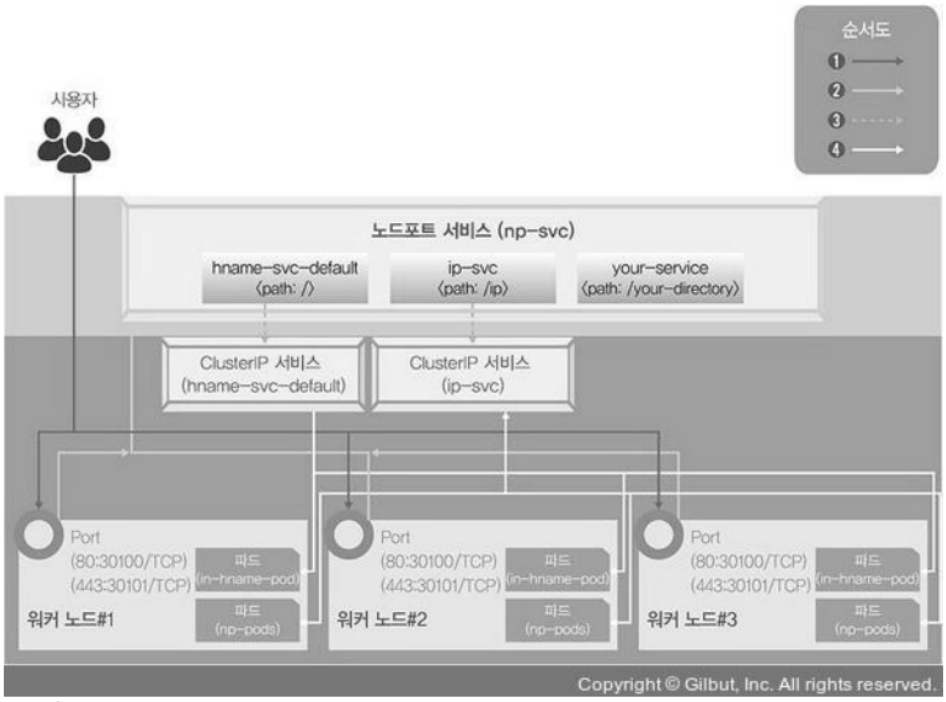
인그레스(Ingress)

- 고유한 주소를 제공해 사용 목적에 다라 다른 응답 제공 가능
- 트래픽에 대한 L4/L7 로드밸런서와 보안 인증서를 처리하는 기능 제공
- 인그레스를 사용하려면 인그레스 컨트롤러가 필요함
  - 다양한 인그레스 컨트롤러 존재

NGINX 인그레스 컨트롤러

- 실습에서는 쿠버네티스에서 프로젝트로 지원하는 NGINX 인그레스 컨트롤러(NGINX Ingress controller)로 구성함
- 작동 단계
  1. 사용자: 노드마다 설정된 노드포트를 통해 노드포트 서비스로 접속함.
     - 이때, 노드포트 서비스를 NGINX 인그레스 컨트롤러로 구성함
  2. NGINX 인그레스 컨트롤러: 사용자의 접속 경로에 따라 적합한 클러스터 IP 서비스로 경로 제공
  3. 클러스터 IP 서비스는 사용자를 해당 파드로 연결해 줌

💡 **인그레스 컨트롤러는 파드와 직접 통신할 수 없음**
→ 노드포트/로드밸런서 서비스와 연동돼야 함
⇒ 실습에서는 노드포트로 연동함!



▲ **그림 3-41** NGINX 인그레스 컨트롤러 서비스 구성도

1. 테스트용으로 디플로이먼트 2개(in-hname-pod, in-ip-pod) 배포

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

2. 배포된 파드의 상태 확인

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

3. NGINX 인그레스 컨트롤러 설치
   - 많은 종류의 오브젝트 스펙이 포함됨
   - 설치되는 요소들은 NGINX 인그레스 컨트롤러 서비스를 제공하기 위해 미리 지정돼 있음

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

4. NGINX 인그레스 컨트롤러의 파드가 배포됐는지 확인
   - NGINX 인그레스 컨트롤러는 default 네임스페이스가 아닌 ingress-nginx 네임스페이스에 속하므로, -n ingress-nginx 옵션을 추가해야 함
     - -n: namespace의 약어. default 외의 네임스페이스를 확인할 때 사용하는 옵션. 파드뿐만 아니라, 서비스를 확인할 때도 동일한 옵션ㅇ르 줌

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

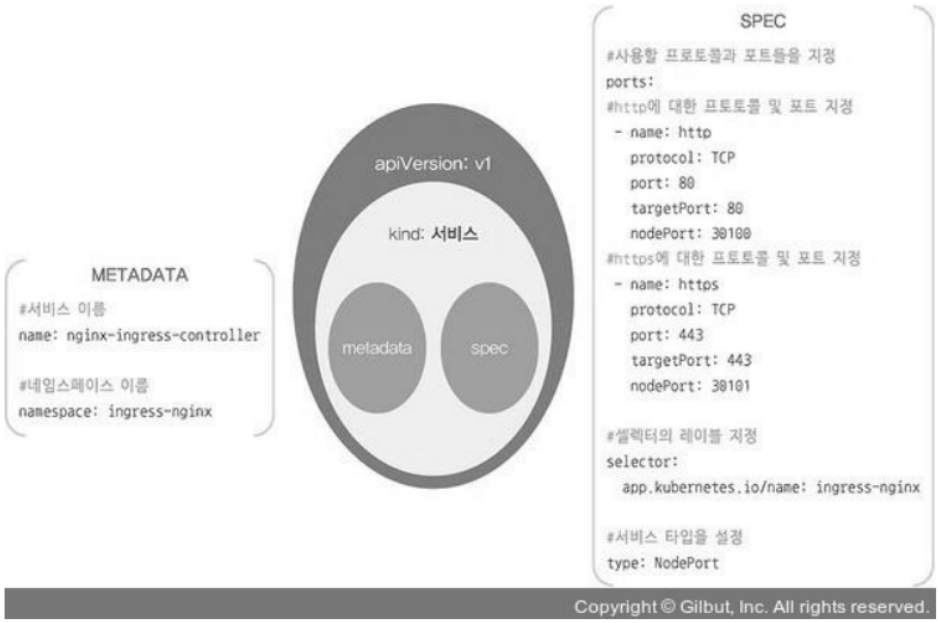
5. 인그레스를 사용자 요구 사항에 맞게 설정하려면 경로 및 작동을 정의해야 함
   - 파일로도 설정할 수 있으며, 실습에서는 다음 경로로 실행해 미리 정의해 둔 설정을 적용함

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

   - 인그레스를 위한 설정 파일
     - 들어오는 주소 값과 포트에 따라 노출된 서비스를 연결하는 역할 설정
     - 외부에서 주소 값과 노드포트를 갖고 들어오는 것은 hname-svc-default 서비스와 연결된 파드로 넘기고, 외부에서 들어오는 주소 값, 노드포트와 함께 뒤에 /ip를 추가한 주소 값은 ip-svc 서비스와 연결된 파드로 접속하게 설정함

       (ingress-config.yaml)

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-nginx
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
ruls:
  -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
```



▲ 그림 3-42 ingress-config.yaml 파일의 구조

6. 인그레스 설정 파일이 제대로 등록됐는지 확인

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

7. 인그레스에 요청한 내용이 확실하게 적용됐는지 확인
   - 이 명령은 인그레스에 적용된 내용을 야믈 형식으로 출력해 적용된 내용을 확인할 수 있음
   - 사용자가 적용한 내용 외에 시스템에서 자동으로 생성하는 것까지 모두 확인할 수 있으므로 이 명령을 응용하면 오브젝트 스펙 파일을 만드는 데 도움이 됨

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

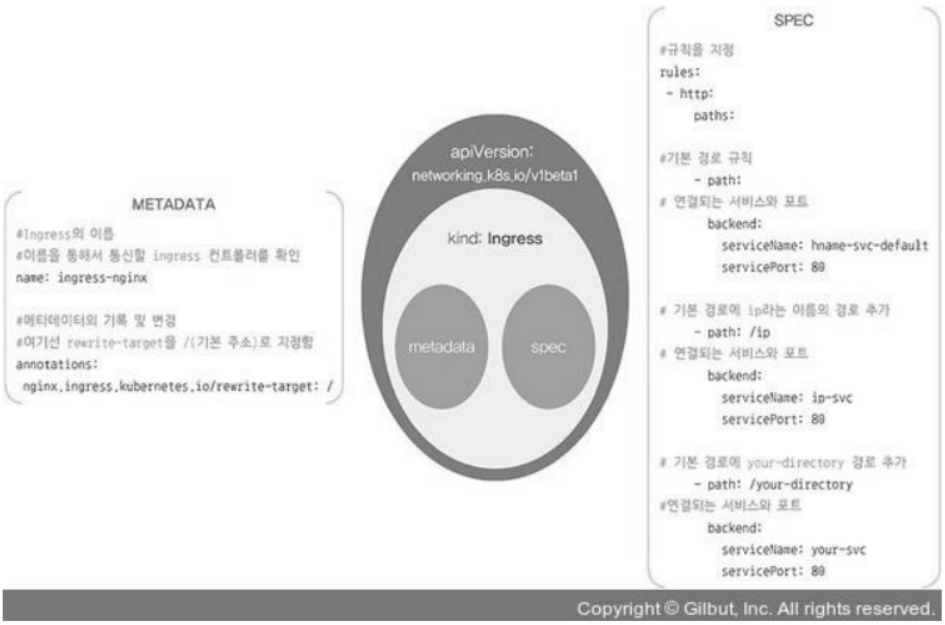
8. NGINX 인그레스 컨트롤러 생성 및 설정 완료함. 외부에서 NGINX 인그레스 컨트롤러에 접속할 수 있게 노드포트 서비스로 NGINX 인그레스 컨트롤러를 외부에 노출함

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

   (ingress.yaml)

   ```
   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
   ```

   - 기존 노드포트와 달리 http를 처리하기 위해 30100번 포트로 들어온 요청을 80번 포트로 넘기고, https를 처리하기 위해 30101번 포트로 들어온 것을 443번 포트로 넘김
   - NGINX 인그레스 컨트롤러가 위치하는 네임스페이스를 ingress-nginx로 지정하고, NGINX 인그레스 컨트롤러의 요구 사항에 따라 셀렉터를 ingress-nginx로 지정함



▲ 그림 3-43 ingress.yaml 파일의 구조

9. 노드포트 서비스로 생성된 NGINX 인그레스 컨트롤러 확인
    - -n ingress-nginx로 네임스페이스를 지정해야만 내용 확인 가능

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

10. 디플로이먼트(in-hname-pod, in-ip-pod)를 서비스로 노출함
    - 외부와 통신하기 위해 클러스터 내부에서만 사용하는 파드를 클러스터 외부에 노출할 수 있는 구역으로 옮기는 것
    - 내부와 외부 네트워크를 분리해 관리하는 DMZ(DeMilitarized Zone, 비무장지대)와 유사한 기능

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

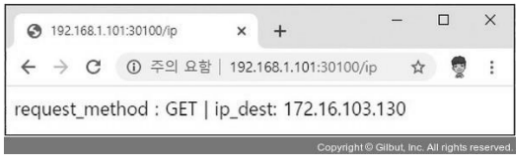
11. 생성된 서비스를 점검해 디플로이먼트들이 서비스에 정상적으로 노출되는지 확인함
    - 새로 생성된 서비스는 default 네임스페이스에 있으므로 -n 옵션으로 네임스페이스를 지정하지 않아도 됨

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

12. 호스트 노트북(또는 PC)에서 웹 브라우저를 띄우고 192.168.1.101:30100에 접속해 외부에서 접속되는 경로에 따라 다르게 작동하는지 확인함
    - 이때, 워커 노드 IP는 192.168.1.101이 아닌 102/103을 사용해도 무방함.
    - 파드 이름이 웹 브라우저에 표시되는지도 확인함

▲ 그림 3-44 192.168.1.101:30100에 접속해 파드 이름이 표시되는지 확인하기

13. 경로를 바꿔서 192.168.1.101:30100 뒤에 /ip를 추가함.
    - 요청 방법과 파드의 ip(CIDR로 임의생성되므로 다를 수 있음)가 반환되는지 확인함



▲ 그림 3-45 192.168.1.101:30100/ip에 접속해 요청 방법과 IP가 표시되는지 확인하기

14. `https://192.168.1.101:30101`로 접속해 HTTP 연결이 아닌 HTTPS 연결도 정상적으로 작동하는지 확인함
    - 30101은 HTTPS의 포트인 443번으로 변환해 접속됨. (브라우저에 따라 경고 메시지가 뜰 수도 있음)
    - 파드 이름이 브라우저에 표시되는지 확인함

▲ 그림 3-46 HTTPS 연결 확인하기

15. `https://192.168.1.101:30101/ip` 를 입력해 요청 방법과 파드의 IP 주소가 웹 브라우저에 표시되는지 확인함

▲ 그림 3-47 https://192.168.1.101:30101/ip에 접속해 요청 방법과 IP가 표시되는지 확인하기

16. 배포한 디플로이먼트와 모든 서비스 삭제 (다음 실습을 위함)

```
[root@m-k8s ~] kubectl delete deployment in-hneme-pod
deployment.apps "in-hname-pod" deleted
[root@m-k8s ~] kubectl delete deployment in-ip-pod
deployment.apps "in-ip-pod" deleted
[root@m-k8s ~] kubectl delete services hname-svc-default
service "hname-svc-default" deleted
[root@m-k8s ~] kubectl delete services ip-svc
service "ip-svc" deleted
```

17. NGINX 인그레스 컨트롤러와 관련된 내용도 모두 삭제. (여러 가지 내용이 혼합되어 설치 파일을 이용해 삭제하는 것이 권장됨)

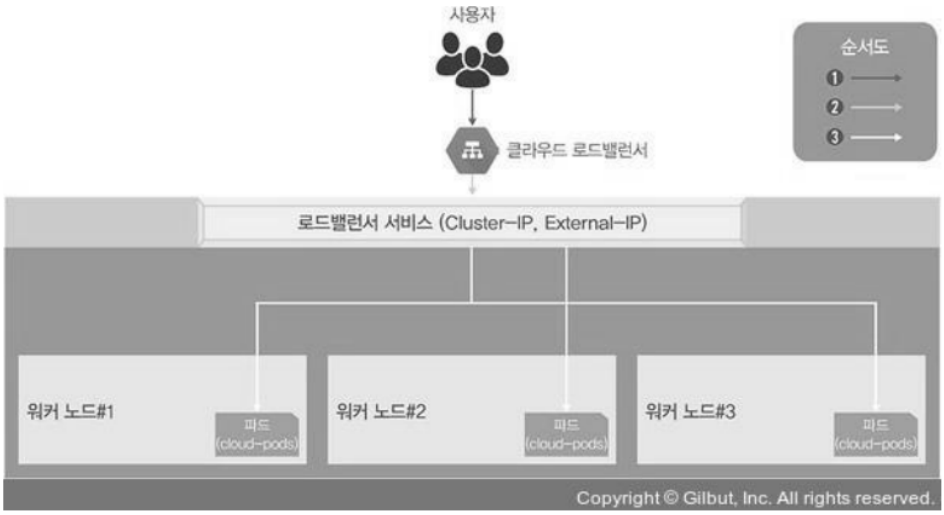
```
[root@m-k8s ~] kubectl delete -f ~/_Book_k8sInfra/ch3/3.3.2/ingress-nginx.yaml
...
[root@m-k8s ~] kubectl delete -f ~/_Book_k8sInfra/ch3/3.3.2/ingress-config.yaml
ingress.networking.k8s.io "ingress-nginx" deleted
```

▼ (3) 클라우드에서 쉽게 구성 가능한 로드밸런서

- 앞선 연결 방식은 들어오는 요청을 모두 워커 노드의 노드포트를 통해 노드포트 서비스로 이동, 이를 다시 쿠버네티스 파드로 보내는 구조

  → 매우 비효율적

  ⇒ 쿠버네티스에서 로드밸런서(LoadBalancer)라는 서비스 타입을 제공해 간단한 구조로 파드를 외부로 노출하고, 부하를 분산함



▲ 그림 3-48 로드밸런서 서비스 구성도

로드밸런서

- 로드밸런서를 사용하려면 로드밸런서를 이미 구현해 둔 서비스업체의 도움을 받아 쿠버네티스 클러스터 외부에 구현해야 함

  → 클라우드에서 제공하는 쿠버네티스를 사용하고 있다면 아래와 같이 선언만 하면 됨.

```
[admin@Cloud_CMD ~] kubectl expose deployment ex-llb --type=LoadBalancer --name=ex-svc
service/ex-svc exposed
[admin@Cloud_CMD ~] kubectl get services ex-svc
```

  - 쿠버네티스 클러스터에 로드밸런서 서비스가 생성돼 외부와 통신할 수 있는 IP(EXTERNAL-IP)가 부여되고, 외부와 통신할 수 있으며, 부하도 분산됨

▼ (4) 온프레미스에서 로드밸런서를 제공하는 MetalLB

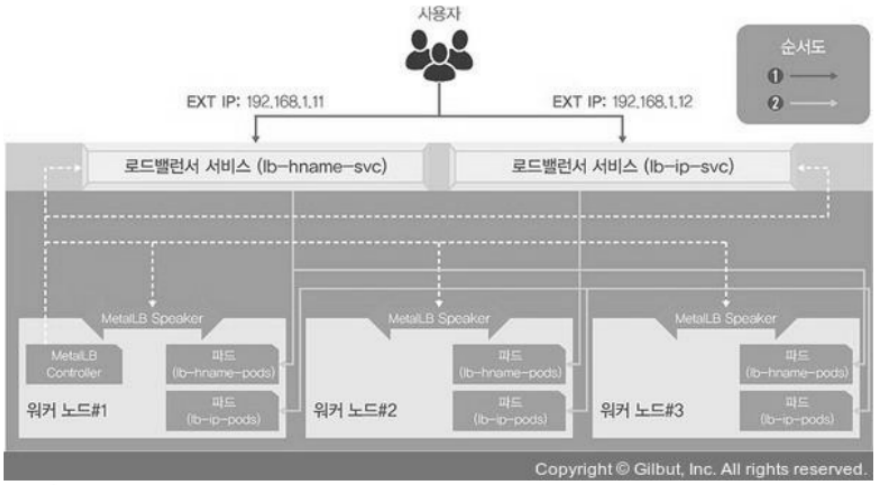
- 우리가 만든 테스트 가상 환경(온프레미스)에서 로드밸런서 사용하기!
- 온프레미스에서 로드밸런서를 사용하려면 내부에 로드밸런서 서비스를 받아주는 구성이 필요함

  → **MetalLB가 이를 지원**함.

MetalLB

- 베어메탈(bare metal, 운영 체제가 설치되지 않은 하드웨어)로 구성된 쿠버네티스에서도 롤드밸런서를 사용할 수 있게 고안된 프로젝트
- 특별한 네트워크 설정/구성이 있는 것이 아니라, 기존의 L2 네트워크(ARP/NDP)와 L3 네트워크(BGP)로 로드밸런서를 구현함

  → 네트워크를 새로 배워야 할 부담 X, 연동하기도 쉬움



▲ 그림 3-49 MetalLB 서비스 구성도

  - 기존의 로드밸런서와 거의 동일한 경로로 통신하며, 테스트 목적으로 두 개의 MetalLB 로드밸런서 서비스를 구현함
  - **MetalLB 컨트롤러**
    - 작동방식(Protocol, 프로토콜)을 정의하고, EXTERNAL-IP를 부여해 관리함
  - **MetalLB 스피커(speaker)**
    - 정해진 작동 방식(L2/ARP, L3/BGP)에 따라 경로를 만들 수 있도록 네트워크 정보를 광고 및 수집해 각 파드의 경로를 제공함
    - 이때, L2는 스피커 중에서 리더를 선출해 경로 제공을 총괄하게 함

MetalLB로 온프레미스 쿠버네티스 환경에서 로드밸런서 서비스 사용하기

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

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

```
deployment.apps/lb-hname-pods scaled
[root@m-k8s ~] kubectl create deployment lb-ip-pods --image=sysnet4admin/echo-ip
deployment.apps/lb-ip-pods created
[root@m-k8s ~] kubectl scale deployment lb-ip-pods --replicas=3
deployment.apps/lb-ip-pods scaled
```

2. 2종류의 파드가 3개씩 총 6개가 배포됐는지 확인함

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

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

```
[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, IP의 상태도 확인함

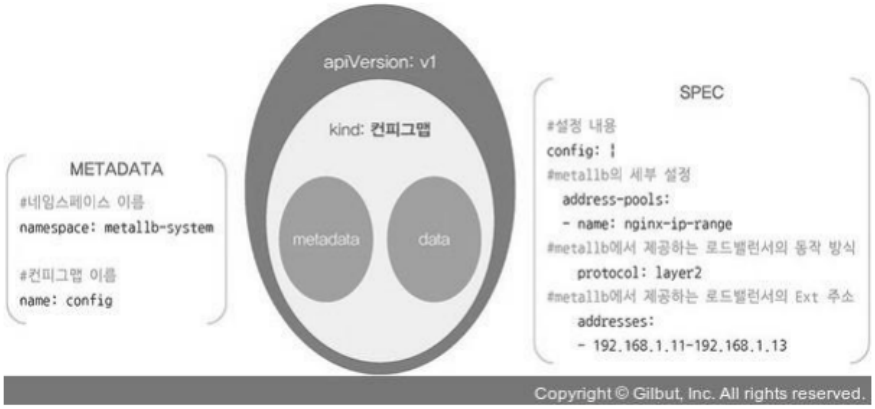
```
[root@m-k8s ~] kubectl apply -f ~/_Book_k8sInfra/ch3/3.3.4/metallb.yaml
...
```

5. 인그레스와 마찬가지로 MetalLB도 설정을 적용해야 하는데, 다음 방법으로 적용함.
   - 이때, 오브젝트는 ConfigMap을 사용함
     - ConfigMap: 설정이 정의된 포맷

```
[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
```



▲ 그림 3-50 metallb-l2config.yaml의 구조

6. ConfigMap이 생성됐는지 확인

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

7. -o yaml 옵션을 주고 다시 실행 → MetalLB의 설정이 올바르게 적용됐는지 확인함

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

8. 모든 설정 완료 → 각 디플로이먼트(lb-hname-pods, lb-ip-pods)를 로드밸런서 서비스로 노출함

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

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

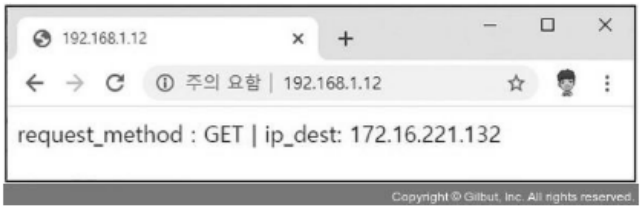
```
[root@m-k8s ~] kubectl get services
```

10. EXTERNAL-IP가 잘 작동하는지 확인
    - 호스트 노트북(또는 PC)에서 브라우저를 띄우고, 192.168.1.11로 접속

      → 배포된 파드 중 하나의 이름이 브라우저에 표시되는지 확인

▲ 그림 3-51 192.168.1.11에 접속해 파드 이름이 표시되는지 확인하기

11. 192.168.1.12를 접속해 파드에 요청 방법과 IP가 표시되는지 확인



▲ 그림 3-52 192.168.1.12에 접속해 요청 방법과 IP가 표시되는지 확인하기

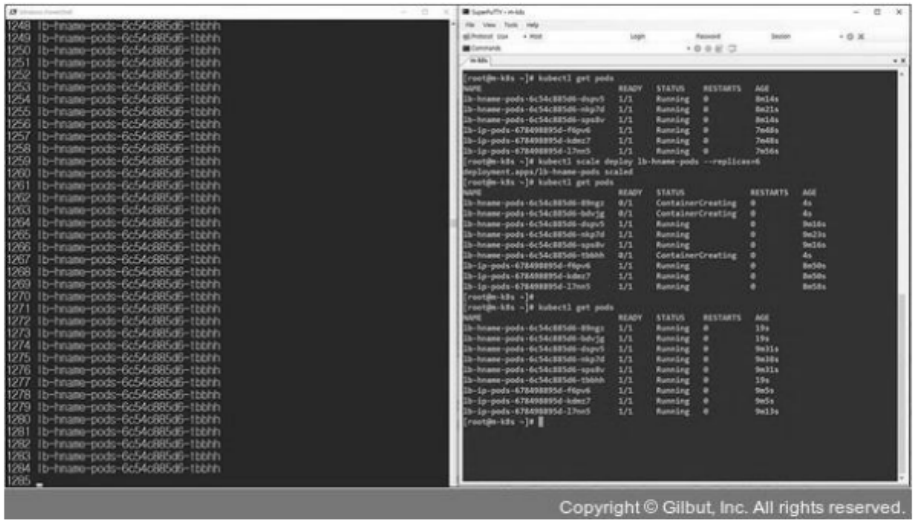
12. powershell 명령 창을 띄우고 shell 스크립트 실행
    - 로드밸러서 기능이 정상적으로 작동하면 192.168.1.11(EXTERNAL-IP)에서 반복적으로 결과값을 갖고 옴

```
[PS C:\Users\Hoon Jo - Pink] > $i=0; while($true)
{
  % { $i++; write-host -NoNewline "$i $_" }
  (Invoke-RestMethod "http://192.168.1.11")-replace '\n', " "
}
```

13. scale 명령으로 파드를 6개로 늘림

```
[root@m-k8s ~] kubectl scale deployment lb-hname-pods --replicas=6
deployment.apps/lb-ip-pods sclaed
```

14. 늘어난 파드 6개도 EXTERNAL-IP를 통해 접근되는지 확인함



▲ 그림 3-53 EXTERNAL-IP 접속 확인하기

15. 배포한 디플로이먼트와 서비스 삭제. MetalLB 설정은 계속 사용하므로 삭제 X

```
[root@m-k8s ~] kubectl delete deployment lb-hname-pods
deployment.apps "lb-hname-pods" deleted
[root@m-k8s ~] kubectl delete deployment lb-ip-pods
deployment.apps "lb-ip-pods" deleted
[root@m-k8s ~] kubectl delete service lb-hname-svc
service "lb-hname-svc" deleted
[root@m-k8s ~] kubectl delete service lb-ip-svc
service "lb-ip-svc" deleted
```

(5) 부하에 따라 자동으로 파드 수를 조절하는 HPA

- 지금까지는 사용자 1명이 파드에 접근하는 방법

  → 사용자가 늘어나면, 파드가 더 이상 감당할 수 없어서 서비스 불가(여기의 서비스는 쿠버네티스의 서비스가 X)

  → 쿠버네티스는 이런 경우를 대비해 부하량에 따라 디플로이먼트의 파드 수를 유동적으로 관리하는 기능 제공

  ⇒ **HAP(Horizontal Pod Autoscaler)**

HPA 설정 및 사용 방법

1. 디플로이먼트 1개를 hpa-hname-pods라는 이름으로 생성

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

2. 앞에서 MetalLB를 구성했으므로, expose를 실행해 hpa-hname-pods를 로드밸런서 서비스로 바로 설정

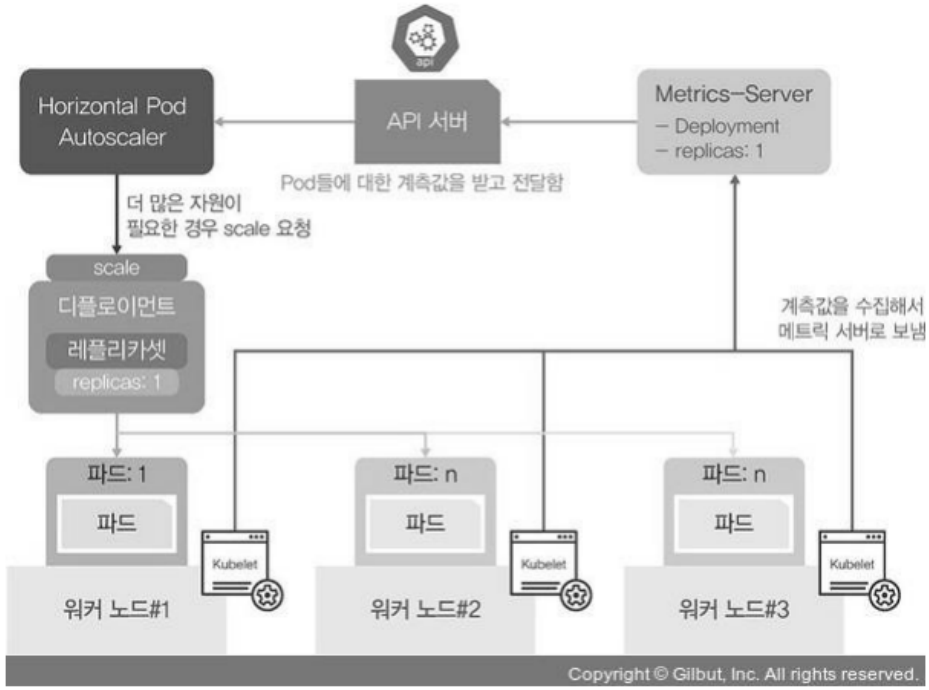
   ```
   [root@m-k8s ~] kubectl create deployment hpa-hname-pods --image=sysnet4admin/echo-hname
   deployment.apps/hpa-hname-pods created
   ```

3. 설정된 로드밸런서 서비스와 부여된 IP 확인
4. HPA가 작동하려면 파드의 자원이 어느 정도 사용되는지 파악해야 함
   - 부하를 확인하는 명령

     ```
     [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 생김

에러가 발생하는 이유?



▲ 그림 3-54 HPA 작동 구조

- HPA가 자원을 요청할 때, 메트릭 서버(Metrics-Server)를 통해 계측값을 전달받음

  → but, 현재 메트릭 서버가 없음

  → 에러 발생!

  ⇒ 계측값을 수집하고 전달해 주는 매트릭 서버를 설정

5. 쿠버네티스 매트릭 서버의 원본 소스(**https://github.com/kubernetes-sigs/metrics-server**)를 sysnet4admin 계정으로 옮겨 메트릭 서버 생성
   - 서비스에서와 마찬가지로 메트릭 서버도 오브젝트 스펙 파일로 설치할 수 있음
     - but, 오브젝트 스펙 파일이 여러 개이므로 git clone 이후에 디렉터리에 있는 파일들을 다시 실행해야 하는 번거로움이 있음
     - 실습에서 사용하려면 몇 가지 추가 설정이 필요함

       → 따라서 원본소스를 옮겨 메트릭 서버 생성!
6. 메트릭 서버를 설정하고 나면, kubectl top pods 명령의 결과를 제대로 확인할 수 있음
   - 확인하는 데까지 시간이 오래 걸림
   - 현재는 아무런 부하가 없으므로 CPU와 MEMORY 값이 매우 낮게 나옴

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

     - 현재는 scale 기준 값이 설정돼 있지 않으므로 파드 증설 시점을 알 수 없음

       → 파드에 부하가 걸리기 전에 scale이 실행되게 디플로이먼트에 기준 값을 기록함.

       (이때, Deployment를 새로 배포하기보다는 기존에 디플로이먼트 내용을 edit 명령으로 직접 수정함)

7. edit 명령을 실행해 배포된 디플로이먼트 내용 확인
    - 40번째 줄에 resources: {} 부분에서 {}를 생략, 그 아래에 다음과 같이 requests, limits 항목과 그 값 추가
        - 이때, 추가한 값은 파드마다 주어진 부하량을 결정하는 기준이 됨
        - 사용한 단위 m은 milliunits의 약어로, 1000m은 1개의 CPU

          ⇒ 10m은 파드의 CPU 0.01 사용을 기준으로 파드를 증설하게 설정한 것
        - 순간적으로 한쪽 파드로 부하가 몰릴 경우를 대비해 CPU 사용 제한을 0.05로 줌
    - 추가가 끝나면 Vim과 동일하게 `:wq`를 입력해 저장하고 나옴
8. 일정 시간이 지난 후 아래 코드를 실행하면, 스펙이 변경돼 새로운 파드가 생성된 것 확인 가능
9. hpa-hname-pods에 autoscale을 설정해서 특정 조건이 만족되는 경우에 자동으로 scale 명령이 수행되도록 하기
    - min: 최소 파드의 수, max: 최대 파드의 수, cpu-percent: CPU 사용량이 50%를 넘게 되면 autoscale하겠다는 뜻
10. 테스트를 위해 화면 왼쪽에 마스터 노드 창 두개, 오른쪽에 powershell 창을 띄움.
    - 오른쪽 창에 호스트 컴퓨터에서 제공하는 부하가 출력됨
    - 왼쪽 상단 창에서는 `watch kubectal top pods`를, 왼쪽 하단 창에서는 `watch kubectl get pods`를 실행함
        - watch를 사용한 이유: 2초에 한 번씩 자동으로 상태를 확인하기 위함
11. HPA를 테스트하기 위해 오른쪽에 있는 powershell 창에서 반복문 실행
    - 부하를 주는 명령은 로드밸런서를 테스트했던 코드와 동일함
    - 왼쪽 상단 창에서 부하량을 감지하는지 확인함.
12. 부하량이 늘어남에 다라 왼쪽 하단 창에서 파드가 새로 생성되는지 확인함
13. 부하 분산으로 생성된 파드의 부하량이 증가하는지 확인
14. 더 이상 파드가 새로 생성되지 않는 안정적인 상태가 되는 것을 확인하고, 부하를 생성하는 오른쪽 powershell 창 종료
15. 일정 시간이 지난 후 더 이상 부하가 없으면, autoscale의 최소 조건인 파드 1개의 상태로 돌아가기 위해 파드가 종료되는 것 확인함
16. 사용하지 않는 파드는 모두 종료되고 1개만 남음
17. 생성한 디플로이먼트, 서비스, 메트릭 서버 삭제. MetalLB는 계속 사용하므로 삭제 X

⇒ 파드 부하량에 따라 HPA가 자동으로 파드 수를 조절하는 것 확인

⇒ HPA를 잘 활용하면, 자원의 사용을 극대화하면서 서비스 가동률을 높일 수 있음

3.4 알아두면 쓸모 있는 쿠버네티스 오브젝트

- 디플로이먼트 외의 용도에 따라 사용할 수 있는 다양한 오브젝트

  → 데몬셋, 컨피그맵, PC, PVC, 스테이트풀셋 등

▼ (1) 데몬셋 (DaemonSet)

- 디플로이먼트의 replicas가 노드 수만큼 정해져 있는 형태
- 노드 하나당 파드 한 개만을 생성함
- 언제 사용?
  - 노드의 단일 접속 지점으로 노드 외부와 통신할 때 → 파드가 1개 이상 필요하지 않음

    ⇒ 노드를 관리하는 파드라면, 데몬셋으로 만드는 게 가장 효율적임
    - Calico 네트워크 플러그인과 kube-proxy 생성 시
    - MetalLB의 스피커

데몬셋 만들기

1. 현재 MetalLB의 스피커가 각 노드에 분포돼 있는 상태 확인

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

2. 워커 노드를 1개 늘림
   - 호스트 컴퓨터의 C:\HashiCorp_Book_k8sInfra-main\ch3\3.1.3 경로로 이동해 Vagrantfile의 5번째 줄에 있는 N 인자의 값을 3에서 4로 수정

     (Vagrantfile 수정)

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  N = 4  # max number of worker nodes
  Ver = '1.18.4'  # Kubernetes Version to install

#================#
# Master Node
#================#
...
```

3. 새로운 워커 노드(w4-k8s) 추가
   - 호스트 컴퓨터의 명령 창에서 C:\HashiCorp_Book_k8sInfra-main\ch3\3.1.3 경로로 이동한 다음 vagrant up w4-k8s를 실행

```
C:\Users\Hoon Jo - Pink > cd C:\HashiCorp\_Book_k8sInfra-main\ch3\3.1.3
C:\HashiCorp\_Book_k8sInfra-main\ch3\3.1.3 > vagrant up w4-k8s
```

4. w4-k8s가 추가되면, m-k8s에서 kubectl get pods -n metallb-system -o wide -w를 수행
   - w: watch의 약어. 오브젝트 상태에 변화가 감지되면 해당 변화를 출력. 리눅스에서 tail -f와 비슷한 역할
   - 변화를 모두 확인했다면 Ctrl+C를 눌러 명령을 중지

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

5. 자동으로 추가된 노드에 설치된 스피커가 데몬셋이 맞는지 확인
   - 스피커 이름(speaker-vnc2k)은 각자 생성된 이름으로 넣어줘야 함

```
[root@m-k8s ~] kubectl get pods speaker-vnc2k -o yaml -n metallb-system
```

▼ (2) 컨피그맵 (ConfigMap)

- 설정(config)을 목적으로 사용하는 오브젝트
- MetalLB를 구성할 때 사용해봄
  - 인그레스에서는 설정을 위해 오브젝트를 인그레스로 선언했는데, 왜 MetalLB에서는 컨피그맵을 사용할까?
    - 인그레스는 오브젝트가 인그레스로 지저오대 있지만, MetalLB는 프로젝트 타입으로 정해진 오브젝트가 없어서 범용 설정으로 사용되는 컨피그맵을 지정함

컨피그맵으로 작성된 MetalLB의 IP 설정 변경

1. 테스트용 디플로이먼트를 cfgmap이라는 이름으로 생성

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

2. cfgmap을 로드밸런서(MetalLB)를 통해 노출하고, 이름은 cfgmap-svc로 지정

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

3. 생성된 서비스의 IP(192.168.1.11)를 확인

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

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 변경함

```
[root@m-k8s ~] cat ~/_Book_k8sInfra/ch3/3.4.2/metallb-l2config.yaml | grep 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/metallb-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
```

5. 변경된 설정 적용

- 컨피그맵 설정 파일(metallb-l2config.yaml)에 apply를 실행

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

6. MetalLB와 관련된 모든 파드 삭제
   - 삭제 후, kubelet에서 해당 파드를 자동으로 모두 다시 생성
   - --all: 파드를 모두 삭제하는 옵션

```
[root@m-k8s ~] kubectl delete pods --all -n metallb-system
pod "controller-65895b47d4-b845q" deleted
...
```

7. 새로 생성된 MetalLB의 파드들을 확인

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

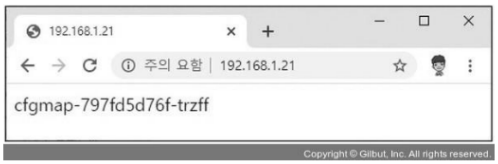
8. 기존에 노출한 MetalLB 서비스(cfgmap-svc)를 삭제하고, 동일한 이름으로 다시 생성해 새로운 컨피그맵을 적용한 서비스가 올라오게 함

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

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

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

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



▲ 그림 3-64 192.168.1.21에 접속해 파드의 이름이 표시되는지 확인하기

11. 이름이 표시되면 설정 변경 성공. 디플로이먼트와 서비스 삭제 (다음 실습을 위해)

```
[root@m-k8s ~] kubectl delete deployment cfgmap
deployment.apps "cfgmap" deleted
[root@m-k8s ~] kubectl delete deployment cfgmap-svc
service "cfgmap-svc" deleted
```

(3) PV와 PVC

- 파드에서 생성한 내용을 기록하고 보관하거나, 모든 파드가 동일한 설정 값을 유지하고 관리하기 위해 공유된 볼륨으로부터 공통된 설정을 가지고 올 수 있도록 설계해야할 때가 있음
- 쿠버네티스는 이런 경우를 위해 아래와 같은 목적으로 다양한 형태의 볼륨을 제공함
  - **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

PVC (PersistentVolumeClaim, 지속적으로 사용 가능한 볼륨 요청)

- 준비된 볼륨에서 일정 공간을 할당받는 것
- 쿠버네티스는 필요할 때 PVC를 요청해 사용함
- PV로 볼륨을 선언해야 PVC 사용 가능

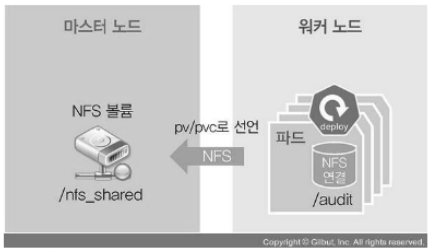
PV (PersistentVolume, 지속적으로 사용 가능한 볼륨)

- 볼륨을 사용할 수 있게 준비하는 단계
- PV로 볼륨을 선언할 수 있는 타입

▲ 그림 3-65 PersistentVolume으로 생성 가능한 볼륨 타입

NFS 볼륨에 PV/PVC를 만들고 파드에 연결하기

- NFC 볼륨 타입이 가장 구현하기 쉬움



▲ 그림 3-66 NFS 볼륨을 이용한 PV/PVC 연결 구성도

NFS 볼륨을 파드에 직접 마운트하기

▼ (4) 스테이트풀셋

- 지금까지는 파드가 replicas에 선언된 만큼 무작위로 생성될 뿐
- 파드가 만들어지는 이름과 순서를 예측해야할 때가 있음
  - 주로 레디스(Redis), 주키퍼(Zookeeper), 카산드라(Cassandra), 몽고DB(MongoDB) 등의 마스터-슬레이브 구조 시스템에서 필요

스테이트풀셋(StatefulSet)

- volumeClaimTemplates 기능을 사용해 PVC를 자동으로 생성할 수 있음
- 각 파드가 순서대로 생성되기 때문에 고정된 이름, 볼륨, 설정 등을 가질 수 있음

  → StatefulSet(이전 상태를 기억하는 세트)
- 효율성 면에서 좋은 구조는 아님

  ⇒ 요구 사항에 맞게 적절히 사용 필요
- 디플로이먼트와 형제나 다름없는 구조이므로 디플로이먼트에서 오브젝트 종류를 변경하면 바로 실습 가능함

스테이트풀셋 만들기

1. 스테이트풀셋 생성
   - PV와 PVC는 앞에서 이미 생성했으므로 바로 스테이트 풀셋 생성함

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

(nfs-pvc-sts.yaml)

```
    ...
7   serviceName: sts-svc-domain  #statefulset need it
    ...
```

- 7번째 줄에 serviceName이 추가된 것 외에는 앞의 nfs-pvc-deploy.yaml 코드와 동일함

2. 파드가 생성됐는지 확인
   - 순서대로 하나씩 생성하는 것을 볼 수 있음

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

3. 생성한 스테이트풀셋에 expose 실행

```
[root@m-k8s ~] kubectl expose statefulset nfs-pvc-sts --type=LoadBalancer --name=nfs-pvc-sts-svc --port=80
error: cannot expose a StatefulSet.apps
```

🚫 에러 발생
→ expose 명령이 스테이트풀셋을 지원하지 않기 때문
⇒ 해결: 파일로 로드밸런서 서비스를 작성, 실행해야 함

💡 expose 명령으로 서비스를 생성할 수 있는 오브젝트는 디플로이먼트, 파드, 레플리카셋, 레플리케이션 컨트롤러임

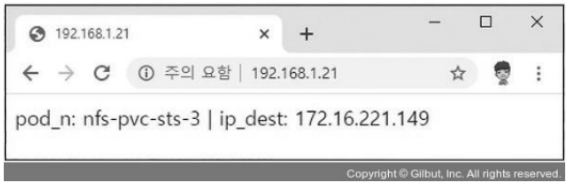
4. 다음 경로를 적용해 스테이트풀셋을 노출하기 위한 서비스를 생성하고, 생성한 로드밸런서 서비스 확인

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

(nfs-pvc-sts-svc.yaml)

```
apiVersion: v1
kind: Service
metadata:
  name: nfs-pvc-sts-svc
spec:
  selector:
    app: nfs-pvc-sts
ports:
  - port: 80
type: LoadBalancer
```

5. 호스트 컴퓨터에서 브라우저를 열고, 192.168.1.21에 접속해 파드 이름과 IP가 표시되는지 확인



▲ 그림 3-70 192.168.1.21에 접속해 파드 이름과 IP가 표시되는지 확인하기

6. exec로 파드에 접속한 후, 새로 접속한 파드의 정보가 추가됐는지 확인. 정보 확인 후, exit로 파드 빠져나옴

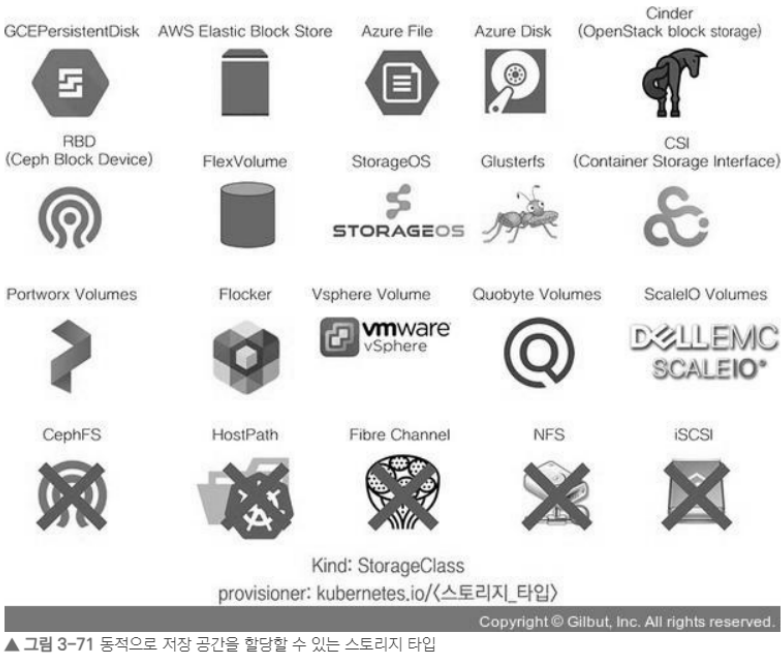
```
[root@m-k8s ~] kubectl exec -it nfs-pvc-sts-0 -- /bin/bash
root@nfs-pvc-sts-0:/ kubectl ls -l /audit
...
-rw-r--r--. 1 root root 96 audit_nfs-pvc-sts-3.log
root@nfs-pvc-sts-0:/ exit
exit
command terminated with exit code 130
[root@m-k8s ~]
```

7. 스테이트풀셋의 파드 삭제
   - 파드는 생성된 순서의 역순으로 삭제됨
   - kubectl get pods -w → 삭제되는 과정 볼 수 있음

```
[root@m-k8s ~] kubectl delete services sts-svc-domain
service "sts-svc-domain" deleted
[root@m-k8s ~] kubectl get pods -w
```

- 일반적으로 스테이트풀셋은 volumeClaimTemplates를 이용해 자동으로 각 파드에 독립적인 스토리지를 할당해 구성 가능
- but, 실습한 NFS 환경에서는 동적으로 할당받을 수 없음

  → 동적으로 할당 가능한 스토리지 타입



▲ 그림 3-71 동적으로 저장 공간을 할당할 수 있는 스토리지 타입