

3장 컨테이너를 다루는 표준 아키텍처, 쿠버네티스 (3, 4)

3.3 쿠버네티스 연결을 담당하는 서비스

3.3.1 가장 간단하게 연결하는 노드포트

[실습] 노드포트 서비스로 외부에서 접속하기

[실습] 부하 분산 테스트 하기 (로드밸런서 기능)

[실습] expose로 노드포트 서비스 생성하기

3.3.2 사용 목적별로 연결하는 인그레스

[실습] 인그레스 사용

3.3.3 클라우드에서 쉽게 구성 가능한 로드밸런서

[코드] 쿠버네티스 클러스터에 로드밸런서 서비스 생성

3.3.4 온프레미스에서 로드밸런서를 제공하는 MetalLB

[실습] MetalLB로 온프레미스 쿠버네티스 환경에서 로드밸런서 서비스 사용 및 구성

3.3.5 부하에 따라 자동으로 파드 수를 조절하는 HPA

[실습] HPA 실습하기

3.4 알아두면 쓸모있는 쿠버네티스 오브젝트

3.4.1 데몬셋

[실습] 데몬셋 작동원리 확인

3.4.2 컨피그맵

[실습] 컨피그맵으로 작성된 MetalLB의 IP 설정 변경하기

3.4.3 PV와 PVC

[실습] NFS 볼륨에 PV/PVC를 만들고 파드에 연결하기

[실습] NFS 볼륨을 파드에 직접 마운트하기

3.4.4 스테이트풀셋

[실습] 스테이트풀셋 생성 과정

3.3 쿠버네티스 연결을 담당하는 서비스

🚩 이번 장의 목표

쿠버네티스 클러스터 내부에서만 파드를 이용하는데 그치지 않고(3.2절) **외부 사용자가 파드를 이용하는 방법을 알아보자!**

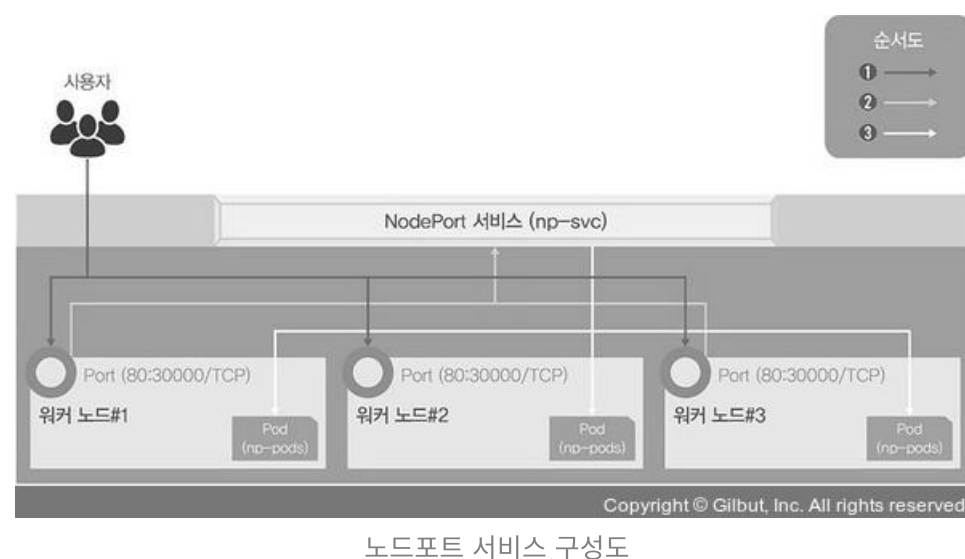
🔍 용어 “서비스”

- **일반적인 서비스** : 웹 서비스나 네트워크 서비스처럼 OS에 속한 데몬 or 개발 중인 서비스
- **쿠버네티스의 서비스** : 외부에서 쿠버네티스 클러스터에 접속하는 방법

3.3.1 가장 간단하게 연결하는 노드포트

노드포트 서비스

- 외부에서 쿠버네티스 클러스터의 내부에 접속하는 가장 쉬운 방법 - **노드포트 서비스 이용**
- 노드포트 서비스를 설정하면 모든 워커 노드의 특정 포트를 열고 여기로 오는 모든 요청을 노드포트 서비스로 전달한다.
- 노드포트 서비스는 해당 업무를 처리할 수 있는 파드로 요청을 전달한다.
- 노드포트 서비스 구성도



[실습] 노드포트 서비스로 외부에서 접속하기

1. 디플로이먼트 파드 생성

```
kubectl create deployment np-pods --image=sysnet4admin/echo-hname
```

2. 배포된 파드 확인

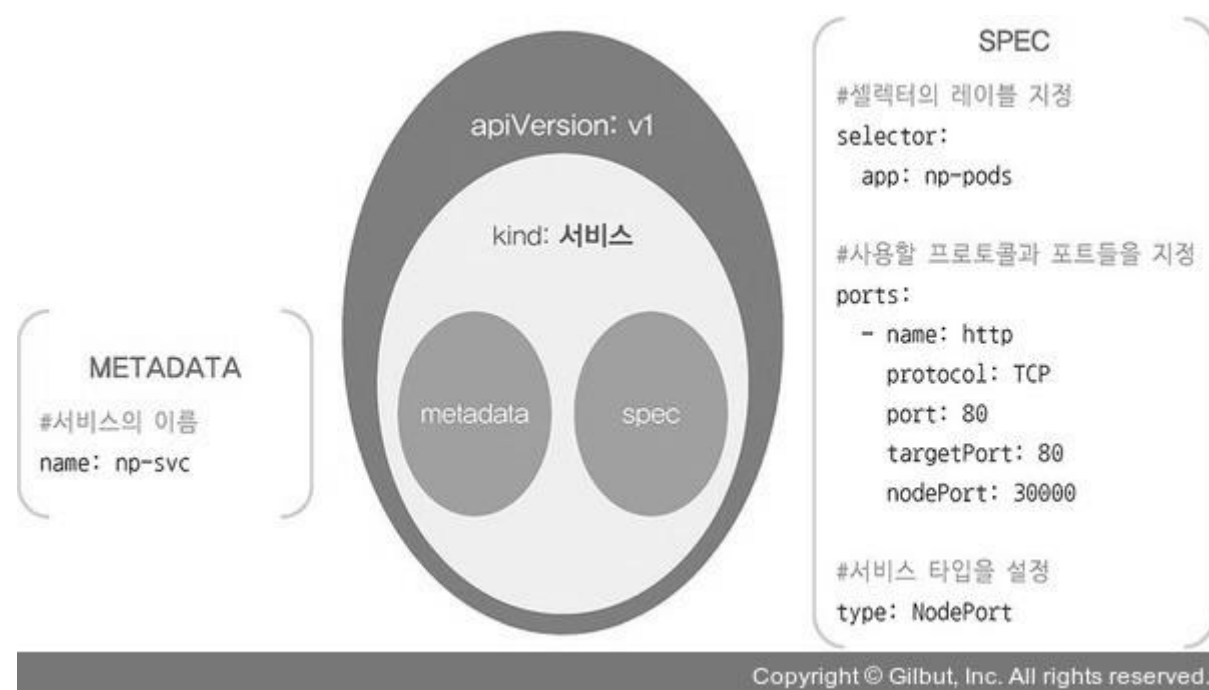
```
kubectl get pods
```

3. 노드포트 서비스 생성 (편의를 위해 이미 정의한 오브젝트 스펙 이용)

```
kubectl create -f ~/Book_k8sInfra/ch3/3.3.1/nodeport.yaml
```

▼ 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
```



nodeport.yaml의 파일 구조

4. 노드포트 서비스로 생성한 np-svc 서비스를 확인

```
kubectl get services
```

5. 쿠버네티스 클러스터의 워커 노드 IP 확인

```
kubectl get nodes -o wide
```

6. 호스트에서 웹 브라우저를 띄우고 **확인한 워커 노드 IP:30000** 으로 접속 확인

[실습] 부하 분산 테스트 하기 (로드밸런서 기능)

1. 반복적으로 **확인한 워커 노드 IP:30000** 으로 접속해 파드 이름을 화면에 표시 (ps에서 실행)

2. 마스터 노드에서 파드를 3개로 증가시키기

```
kubectl scale deployment np-pods --replicas=3
```

3. 배포된 파드 확인

```
kubectl get pods
```

4. 배포된 파드 3개가 돌아가면서 표시되는지 확인 (ps창 확인)

[실습] expose로 노드포트 서비스 생성하기

1. expose 명령어를 사용해 노드포트 서비스 생성

```
kubectl expose deployment np-pods --type=NodePort --name=np-svc-v2 --port=80
```

2. 생성된 서비스 확인 (expose 명령어는 포트번호 지정 불가)

```
kubectl get services
```

3. 호스트에서 웹 브라우저를 띄우고 **IP:PORT** 에 접속

4. 다음 실습 진행을 위한 삭제

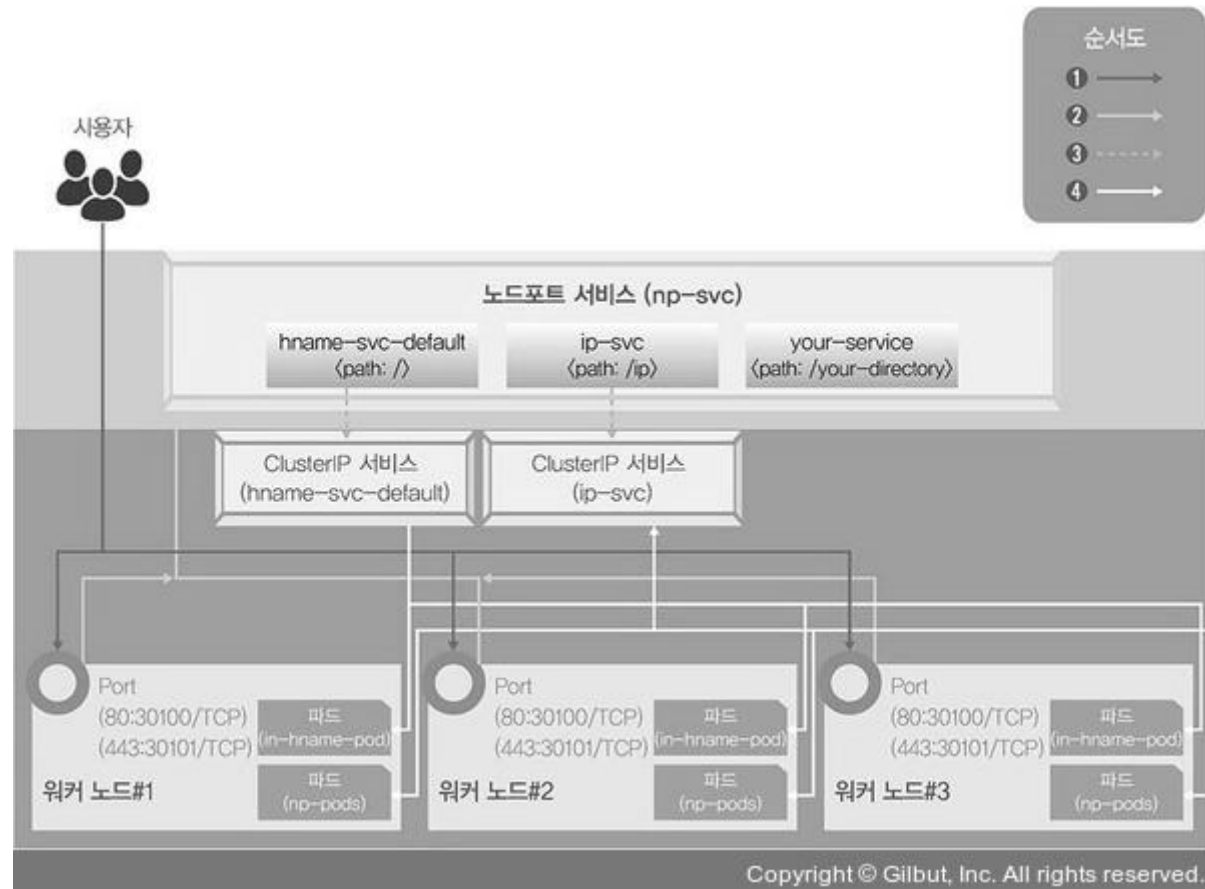
```
kubectl delete deployment np-pods
```

```
kubectl delete service np-svc
```

```
kubectl delete services np-svc-v2
```

3.3.2 사용 목적별로 연결하는 인그레스

- 문제
 - 노드포트 서비스는 포트를 중복사용할 수 없음
 - 1개의 노드포트에 1개의 디플로이먼트만 적용됨
 - 여러 개의 디플로이먼트가 있을때 그 수만큼 노드포트 서비스를 구동해야하는가?
⇒ **인그레스 사용**
- **인그레스(Ingress)**
 - 고유한 주소를 제공해 사용목적에 따라 다른 응답을 제공할 수 있음
 - 트래픽에 대한 L4/L7 로드밸런서 기능 제공
 - 보안 인증서를 처리하는 기능 제공
- **NGINX 인그레스 컨트롤러**
 - 인그레스를 사용하려면 인그레스 컨트롤러가 필요함
 - NGINX 인그레스 컨트롤러는 쿠버네티스에서 프로젝트로 지원함
 - **작동 단계**
 1. 사용자가 노드마다 설정된 노드포트를 통해 노드포트 서비스로 접속
노드포트 서비스를 NGINX 인그레스 컨트롤러로 구성
 2. NGINX 인그레스 컨트롤러는 사용자의 접속 경로에 따라 적합한 클러스터 IP 서비스로 경로 제공
 3. 클러스터 IP 서비스는 사용자를 해당 파드로 연결
 - **구성하려는 NGINX 인그레스 컨트롤러**



[실습] 인그레스 사용

1. 디플로이먼트 2개 배포

```
kubectl create deployment in-hname-pod --image=sysnet4admin/echo-hname
```

```
kubectl create deployment in-ip-pod --image=sysnet4admin/echo-ip
```

2. 배포된 파드 상태 확인

```
kubectl get pods
```

3. NGINX 인그레스 컨트롤러 설치

```
kubectl apply -f ~/Book_k8sInfra/ch3/3.3.2/ingress-nginx.yaml
```

4. NGINX 인그레스 컨트롤러의 파드 배포 확인

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

5. 인그레스를 사용자 요구 사항에 맞게 설정하기 위한 경로와 작동 정의

```
kubectl apply -f ~/Book_k8sInfra/ch3/3.3.2/ingress-config.yaml
```

▼ ingress-config.yaml

```
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 등록되었는지 확인

```
kubectl get ingress
```

7. 인그레스에 요청한 내용이 확실하게 적용되었는지 확인

```
kubectl get ingress -o yaml
```

=====  NGINX 인그레스 컨트롤러 생성 & 인그레스 설정 완료 =====

8. 외부에서 NGINX 인그레스 컨트롤러에 접속할 수 있도록 노트포트 서비스로 NGINX 인그레스 컨트롤러를 외부에 노출

```
kubectl apply -f ~/_Book_k8sInfra/ch3/3.3.2/ingress.yaml
```

▼ ingress.yaml

- http 처리를 위해 30100포트 요청받으면 ⇒ 80포트
- https 처리를 위해 30101포트 요청받으면 ⇒ 443포트
- NGINX 인그레스 컨트롤러가 위치하는 네임스페이스 ⇒ ingress-nginx로 지정
- NGINX 인그레스 컨트롤러의 요구사항에 따라 ⇒ 선택터를 ingress-nginx로 지정

```
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. 노트포트 서비스로 생성된 NGINX 인그레스 컨트롤러를 확인

```
kubectl get services -n ingress-nginx
```

10. expose 명령으로 디플로이먼트도 서비스로 노출


외부와 통신하기 위해 클러스터 내부에서만 사용하는 파드를 클러스터 외부에 노출할 수 있는 구역으로 옮기는 것

```
kubectl expose deployment in-hname-pod --name=hname-svc-default --port=80,443
```

```
kubectl expose deployment in-ip-pod --name=ip-svc --port=80,443
```

11. 생성된 서비스를 점검해 디플로이먼트들이 서비스에 정상적으로 노출되는지 확인

```
kubectl get services
```

=====  모든 준비 완료. 이제 웹으로 작동 확인! =====

12. 워커노드IP:30100 접속해서 확인하기

13. 워커노드IP:30100/ip 접속해서 확인하기

14. https://워커노드IP:30100 접속해서 확인하기

15. https://워커노드IP:30100/ip 접속해서 확인하기

=====  실습 끝. 다음 실습을 위한 삭제 진행! =====

16. 배포한 디플로이먼트와 모든 서비스 삭제

```
kubectl delete deployment in-hname-pod
```

```
kubectl delete deployment in-ip-pod
```

```
kubectl delete services hname-svc-default
```

```
kubectl delete services ip-svc
```

17. NGINX 인그레스 컨트롤러 관련 내용 삭제

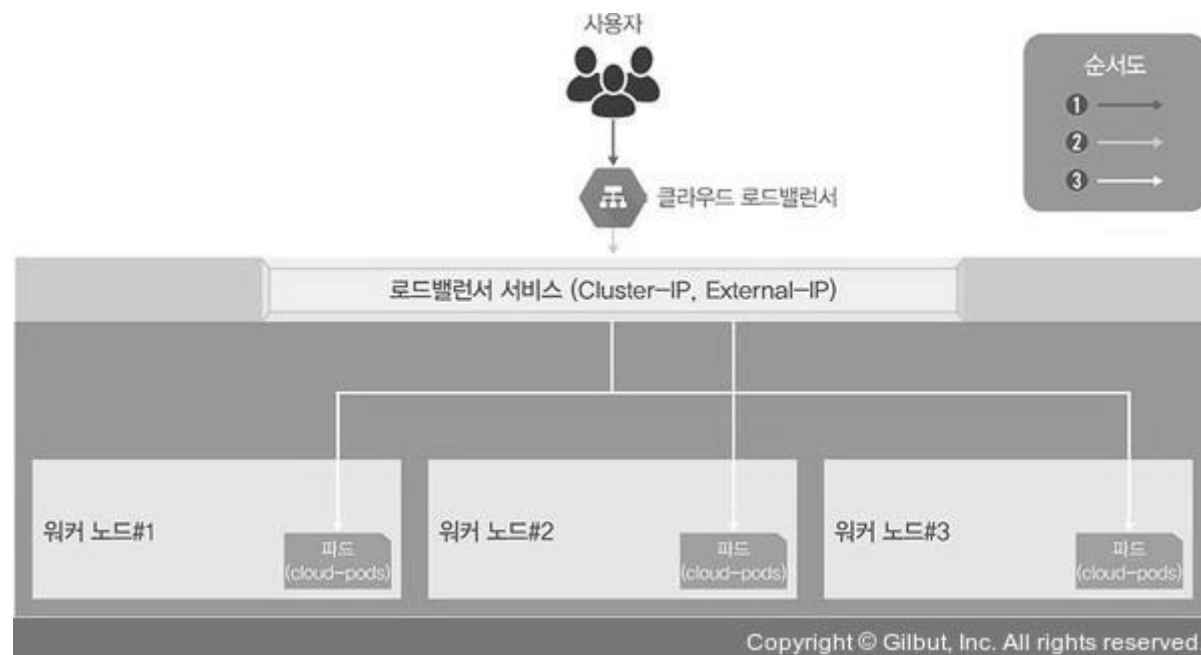
```
kubectl delete -f ~/_Book_k8sInfra/ch3/3.3.2/ingress-nginx.yaml
```

```
kubectl delete -f ~/_Book_k8sInfra/ch3/3.3.2/ingress-config.yaml
```

3.3.3 클라우드에서 쉽게 구성 가능한 로드밸런서

문제 - 앞에서 배운 연결 방식은 들어오는 요청을 모두 워커 노드의 노드포트를 통해 노드포트 서비스로 이동하고 이를 다시 쿠버네티스의 파드로 보내는 구조 ⇒ 비효율적

∴ 쿠버네티스에서 로드밸런서라는 서비스 타입을 제공해 다음 그림과 같은 간단한 구조로 파드를 외부에 노출하고 부하를 분산한다.



[코드] 쿠버네티스 클러스터에 로드밸런서 서비스 생성

다음과 같이 선언하면 클라우드 클러스터에 로드밸런서 서비스가 생성되어 외부와 통신할 수 있는 IP가 부여되고 외부와 통신할 수 있으며 부하도 분산된다.

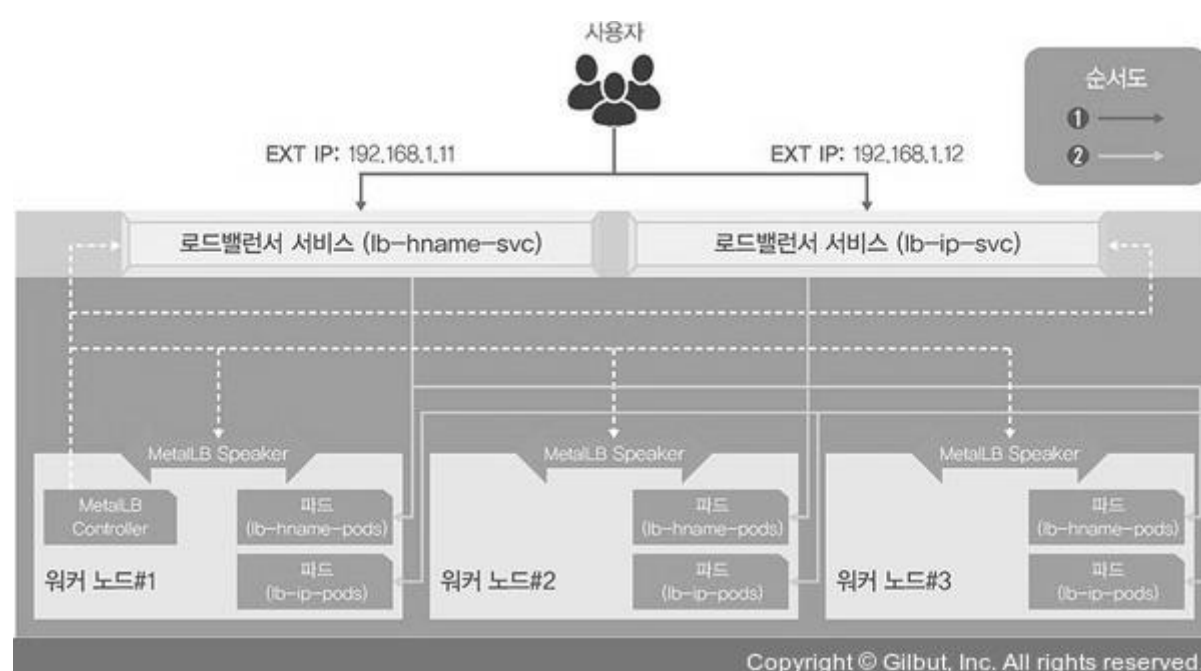
```
kubectl expose deployment ex-lb --type=LoadBalancer --name=ex-svc
kubectl get services ex-svc
```

3.3.4 온프레미스에서 로드밸런서를 제공하는 MetalLB

MetalLB

- 온프레미스에서 로드밸런서를 사용하려면 내부에 로드밸런서 서비스를 받아주는 구성이 필요한데, 이를 지원하는 것이 MetalLB.
- 베어메탈(OS가 설치되지 않은 HW)로 구성된 쿠버네티스에서도 로드밸런서를 사용할 수 있게 고안된 프로젝트
- 특별한 네트워크 설정이나 구성이 있는 것이 아니라 기존의 L2 네트워크와 L3 네트워크로 로드밸런서를 구현
- 네트워크를 새로 배워야할 부담이 없으며 연동하기도 매우 쉬움

이 책에서 어떻게 구현할까?



MetalLB의 서비스 구성도

- 기존의 로드밸런서와 거의 동일한 경로로 통신하며, 테스트 목적으로 두 개의 MetalLB 로드밸런서 서비스를 구현
- MetalLB 컨트롤러는 작동 방식을 정의하고 EXTERNAL-IP를 부여해 관리
- MetalLB 스피커는 정해진 작동 방식에 따라 경로를 만들 수 있도록 네트워크 정보를 광고하고 수집해 각 파드의 경로를 제공
- L2는 스피커 중에서 리더를 선출해 경로 제공을 총괄하게 함

[실습] MetalLB로 온프레미스 쿠버네티스 환경에서 로드밸런서 서비스 사용 및 구성

1. 디플로이먼트를 이용해 2종류 파드 생성. 파드를 3개로 늘려 노드당 3개씩 파드가 배포되도록

```
kubectl create deployment lb-hname-pods --image=sysnet4admin/echo-hname
kubectl scale deployment lb-hname-pods --replicas=3
kubectl create deployment lb-ip-pods --image=sysnet4admin/echo-ip
kubectl scale deployment lb-ip-pods --replicas=3
```

2. 2종류의 파드가 3개씩 총 6개가 배포됐는지 확인

```
kubectl get pods
```

3. 사전 정의된 오브젝트 스펙으로 MetalLB를 구성

```
kubectl apply -f ~/Book_k8sInfra/ch3/3.3.4/metallb.yaml
```

4. 배포된 MetalLB의 파드가 5개(speaker 4, controller 1)인지 확인하고 IP & 상태 확인

```
kubectl get pods -n metallb-system -o wide
```

5. MetalLB 설정 적용을 위해 오브젝트 ConfigMap 활용

```
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. ConfigMap 생성 확인

```
kubectl get configmap -n metallb-system
```

7. MetalLB 설정이 올바르게 적용됐는지 확인

```
kubectl get configmap -n metallb-system -o yaml
```

=====  모든 설정 완료. =====

8. 각 디플로이먼트를 로드밸런서 서비스로 노출

```
kubectl expose deployment lb-hname-pods --type=LoadBalancer --name=lb-hname-svc --port=80
kubectl expose deployment lb-ip-pods --type=LoadBalancer --name=lb-ip-svc --port=80
```

9. 생성된 로드밸런서 서비스별로 CLUSTER-IP와 EXTERNAL-IP가 잘 적용됐는지 확인

```
kubectl get services
```

10. EXTERNAL-IP가 잘 작동하는지 확인

호스트로 192.168.1.11 접속

11. 192.168.1.12 접속해 파드에 요청 방법과 IP가 잘 표시되는지 확인

12. ps에 다음 명령 실행

EXTERNAL-IP(192.168.1.11)에서 반복적인 결과값 가져온다. (정상 작동이라면)

```
$i=0; while($true)
{
  % { $i++; write-host -NoNewline "$i $_" }
```



```
(Invoke-RestMethod "http://192.168.1.11")-replace '\n', " "
}
```

13. 파드를 6개로 늘리기

```
kubectl scale deployment lb-hname-pods --replicas=6
```

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

=====  실습 끝. 다음 실습을 위한 삭제 진행! =====

15. 배포한 deploymnet와 서비스 삭제 (MetalLB 설정은 삭제 X)

```
kubectl delete deploymnet lb-hname-pods
```

```
kubectl delete deployment lb-ip-pods
```

```
kubectl delete service lb-hname-svc
```

```
kubectl delete service lb-ip-svc
```

3.3.5 부하에 따라 자동으로 파드 수를 조절하는 HPA

목적

- 지금까지 - 사용자 1명이 파드에 접근하는 방법을 알아봤음
- 문제 - 사용자가 늘어나면 파드가 더 이상 감당할 수 없어서 서비스 불가 초래할 수 있음
- 해결 - 쿠버네티스는 이런 경우를 대비해 부하량에 따라 디플로이먼트의 파드 수를 유동적으로 관리하는 기능을 제공한다. 이를 **HPA(Horizontal Pod Autoscaler)** 라고 한다.

[실습] HPA 실습하기

1. 디플로이먼트 1개 생성

```
kubectl create deployment hpa-hname-pods --image=sysnet4admin/echo-hname
```

2. expose를 실행해 디플로이먼트를 로드밸런서 서비스로 바로 설정 (앞에서 MetalLB 이미 구성됨)

```
kubectl expose deployment hpa-hname-pods --type=LoadBalancer --name=hpa-hna
```

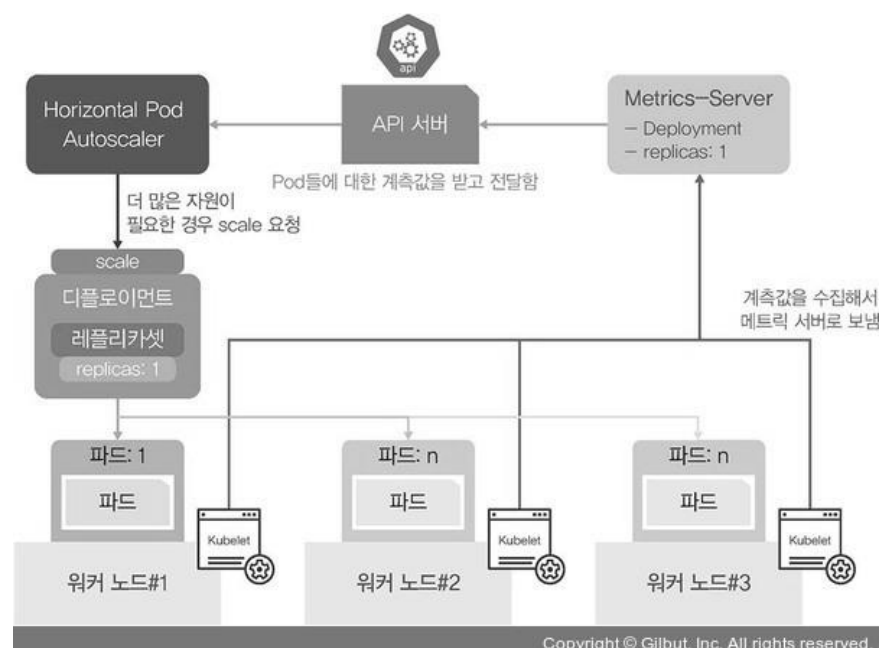
3. 설정된 로드밸런서 서비스와 부여된 IP 확인

```
kubectl get services
```

4. HPA가 작동하려면 파드의 자원이 어느정도 사용되는지 확인해야함 (부하 확인)

```
kubectl top pods
```

- ▼  자원을 요청하는 설정이 없다면 에러가 생기며 진행되지 않음



HPA 작동 구조

HPA가 자원 요청 시 메트릭 서버를 통해 계측값을 전달받아야하는데 지금은 메트릭 서버가 없어서 에러남

5. 쿠버네티스 메트릭 서버의 원본 소스를 sysnet4admin 계정으로 옮겨 메트릭 서버 생성


```
kubectl create -f ~/Book_k8sInfra/ch3/3.3.5/metrics-server.yaml
```

6. 재시도! HPA가 작동하려면 파드의 자원이 어느정도 사용되는지 확인해야함 (부하 확인)

```
kubectl top pods > 오래 걸림
```

7. 배포된 디플로이먼트 내용 확인 후 수정

```
kubectl edit deployment hpa-hname-pods
```

▼ 수정 내용

```
118 imagePullPolicy: Always
119 name: echo-hname
120 resources:
  requests:
    cpu: "10m"
  limits:
    cpu: "50m"
121 terminationMessagePath: /dev/termination-log
```

8. 일정 시간이 지난 후 새로운 파드 생성 확인

```
kubectl top pods
```

9. 디플로이먼트에 autoscale을 설정해 특정 조건이 만족될 때 자동으로 scale 명령하도록 설정

min - 최소 파드의 수 **max** - 최대 파드의 수

cpu-percent - CPU 사용량이 50%를 넘게 되면 autoscale하겠다는 뜻

```
kubectl autoscale deployment hpa-hname-pods --min=1 --max=30 --cpu-percent=50
```

=====  설정 완료. 부하 테스트 진행 =====

10. 화면을 다음처럼 구성



```
마스터 > watch kubectl top pods
```



```
마스터 > watch kubectl get pods
```



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

12. 부하량이 늘어남에 따라 왼쪽 하단 창에서 파드가 새로 생성되는지 확인

13. 부하 분산으로 생성된 파드의 부하량이 증가하는지 확인

14. 더 이상 파드가 새로 생성되지 않는 안정적인 상태가 되는 것을 확인 후 오른쪽 ps 종료

15. 일정 시간 지난 후 더이상 부하가 없으면 autoscale의 최소 조건인 파드 1개의 상태로 돌아가기위해 파드가 종료하는 것을 확인 > 꽤걸림

16. 사용하지 않는 파드 모두 종료되고 1개만 남음

=====  실습 끝. 다음 실습을 위한 삭제 진행! =====

17. 디플로이먼트, 서비스, 메트릭 서버 삭제

```
kubectl delete deployment hpa-hanme-pods
```

```
kubectl delete hpa hpa-hname-pods
```

```
kubectl delete service hpa-hame-svc
```

```
kubectl delete -f ~/Book_k8sInfra/ch3/3.3.5/metrics-server.yaml
```



만약 해당 내용을 직접 확인하고 싶다면, 다음 주소에서 해당 내용을 영상으로 확인할 수 있습니다.

<https://youtu.be/mR0ithbw9Qk>

<https://bit.ly/3ycDVDN>

3.4 알아두면 쓸모있는 쿠버네티스 오브젝트

3.4.1 데몬셋

데몬셋(DaemonSet)

- 디플로이먼트의 replicas가 노드 수만큼 정해져 있는 형태. 노드 하나당 파드 한 개만을 생성함
- 사용 시기
 - 노드 단일 접속 지점으로 노드 외부와 통신할 때
 - e.g. 1) Calico 네트워크 플러그인과 kube-proxy를 생성할 때 사용
 - e.g. 2) MetalLB의 스피커에서 사용
- 노드를 관리하는 파드라면 데몬셋으로 만드는게 가장 효율적임

[실습] 데몬셋 작동원리 확인

1. 현재 MetalLB의 스피커가 각 노드에 분포되어 있는 상태를 확인

```
kubectl get pods -n metallb-system -o wide
```

2. 워커노드 1개 늘리기.

C:\HashiCorp_Book_k8sInfra-main\ch3\3.1.3\Vagrantfile의 5번째 줄 `N = 3` ⇒ `N = 4` 로 수정

3. 새로운 워커노드 w4-k8s 추가

```
cd C:\HashiCorp\_Book_k8sInfra-main\ch3\3.1.3
```

```
vagrant up w4-k8s
```

4. 오브젝트 상태에 변화가 감지되면 변화 출력하도록 확인 후 Ctrl+C

```
kubectl get pods -n metallb-system -o wide -w
```

5. 자동으로 추가된 노드에 설치된 스피커가 데몬셋이 맞는지 확인 (스피커 이름 주의)

```
kubectl get pods [스피커이름] -o yaml -n metallb-system
```

3.4.2 컨피그맵

컨피그맵(ConfigMap)

- 설정을 목적으로 사용하는 오브젝트
- 이그레스에서는 설정을 위해 오브젝트를 인그레스로 선언했는데 MetalLB에서는 컨피그맵을 사용했음. 인그레스는 오브젝트가 인그레스로 지정되어있지만, MetalLB는 프로젝트 타입으로 정해진 오브젝트가 없어서 범용 설정으로 사용되는 컨피그맵을 지정함.

[실습] 컨피그맵으로 작성된 MetalLB의 IP 설정 변경하기

1. 테스트용 디플로이먼트 생성

```
kubectl create deployment cfgmap --image=sysnet4admin/echo-hname
```

2. 디플로이먼트를 로드밸런서(MetalLB)를 통해 노출하고 이름 지정

```
kubectl expose deployment cfgmap --type=LoadBalancer --name=cfgmap
```

3. 생성된 서비스의 IP 확인

```
kubectl get services
```

4. 사전의 구성되어 있는 컨피그맵의 기존 IP를 변경

```
cat ~/Book_k8sInfra/ch3/3.4.2/metallb-l2config.yaml | grep 192.
```

```
sed -i 's/11/21;/s/13/23/' ~/Book_k8sInfra/ch3/3.4.2/metallb-l2config.yaml
```

```
cat ~/Book_k8sInfra/ch3/3.4.2/metallb-l2config.yaml | grep 192.
```

5. 변경된 설정을 적용

```
kubectl apply -f ~/Book_k8sInfra/ch3/3.4.2/metallb-l2config.yaml
```

6. MetalLB와 관련된 모든 파드 삭제

삭제하면 kubelet에서 해당 파드를 자동으로 모두 다시 생성한다.

```
kubectl delete pods --all -n metallb-system
```

7. 새로 생성된 MetalLB의 파드들을 확인

```
kubectl get pods -n metallb-system
```

8. 기존에 노출한 MetalLB 서비스를 삭제하고 다시 생성

새로운 컨피그맵을 적용한 서비스가 올라오도록!

```
kubectl delete service cfgmap-svc
```

```
kubectl expose deployment cfgmap --type=LoadBalancer --name=cfgmap-svc --port=80
```

9. 변경된 설정이 적용되어 MetalLB 서비스의 IP가 변경되었는지 확인

```
kubectl get services
```


10. 호스트의 웹 브라우저에서 192.168.1.21로 접속해 확인

11. 다음 테스트를 위해 생성한 디플로이먼트와 서비스 삭제

```
kubectl delete deployment cfgmap
```

```
kubectl delete service cfgmap-svc
```

3.4.3 PV와 PVC

 파드에서 생성한 내용을 기록하고 보관하거나 모든 파드가 동일한 설정 값을 유지하고 관리하기 위해 공유된 볼륨으로부터 공통된 설정을 가지고 올 수 있도록 설계해야 할 때도 있으므로 **다양한 볼륨을 제공함**

- 임시 : emptyDir
- 로컬 : host Path, local
- 원격 : persistentVolumeClaim, cephfs, cinder, csi, ...
- 특수 목적 : downwardAPI, configMap, ...
- 클라우드: awsElasticBlockStore, azureDisk, gcePersistentDisk, ...

볼륨 스토리지 PV, PVC

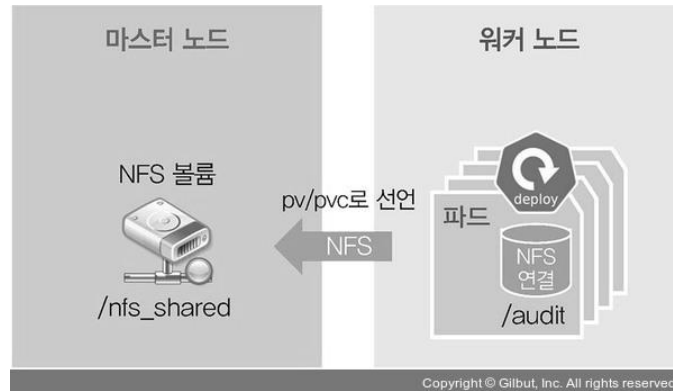
PVC (PersistentVolumeClaim, 지속적으로 사용가능한 볼륨 요청)

PV (PersistentVolume, 지속적으로 사용가능한 볼륨)

- 쿠버네티스는 필요할 때 PVC를 요청해 사용한다.
- PVC를 사용하려면 PV로 볼륨을 선언해야한다.
- 즉, PV는 볼륨을 사용하게 할 수 있게 준비하는 단계이고, PVC는 준비된 볼륨에서 일정 공간을 할당받는 것

[실습] NFS 볼륨에 PV/PVC를 만들고 파드에 연결하기

가장 구현하기 쉬운 NFS 볼륨 타입으로 PV와 PVC를 생성하고 파드에 마운트 해보면서 실제 작동을 확인해보도록 하자.



NFS 볼륨을 이용한 PV/PVC 연결 구성도

1. PV로 선언할 볼륨을 만들기 위해 NFS 서버를 마스터 노드에 구성

```
mkdir /nfs_shared
echo '/nfs_shared 192.168.1.0/24(rw,sync,no_root_squash)' >> /etc/exports
```

2. 해당 내용을 시스템에 적용해 NFS 서버를 활성화하고 다음 시작할 때도 자동 적용

```
systemctl enable --now nfs
```

3. 오브젝트 스펙을 실행해 PV 생성

```
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:
    storage: 100Mi      # 쓸 수 있는 양
  accessModes:         # PV를 어떤 방식으로 사용할지 정의
    - ReadWriteMany    # 여러개의 노드가 읽고 쓸 수 있도록 마운트
  persistentVolumeReclaimPolicy: Retain # PVC가 제거됐을 때 작동하는 방법 (유지)
  nfs:                 # NFS 서버의 연결 위치에 대한 설정
    server: 192.168.1.10
    path: /nfs_shared
```

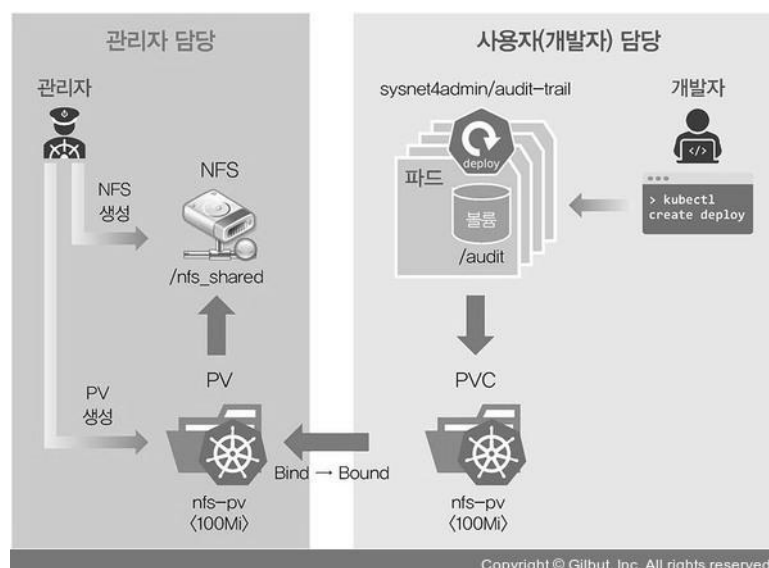
4. 생성된 PV의 상태가 Available임을 확인

```
kubectl get pv
```

5. 오브젝트 스펙을 실행해 PVC를 생성

```
kubectl apply -f ~/Book_k8sInfra/ch3/3.4.3/nfs-pvc.yaml
```

PVC와 PV는 구성이 거의 동일한데 PV는 사용자가 요청할 볼륨 공간을 관리자가 만들고, PVC는 개발자간 볼륨을 요청하는 데 사용한다는 점에서 차이가 있음



PV와 PVC의 관계도

6. 생성된 PVC 확인

```
kubectl get pvc
```

- 1) 상태가 Bound로 변경되었음 (PV와 PVC가 연결됐음을 의미)
- 2) 용량이 설정한 10Mi가 아닌 100Mi (용량은 동적으로 PVC를 따로 요청해 생성하는 경우 아니면 큰상관 X)

7. PV의 상태도 Bound로 변경되었음을 확인

```
kubectl get pv
```

8. 생성된 PVC를 볼륨으로 사용하는 디플로이먼트 오브젝트 스펙을 배포

```
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 이미지를 가져옴
        - name: audit-trail    # 요청을 처리할 때마다 접속 정보를 로그로 기록하는 이미지
          image: sysnet4admin/audit-trail
          volumeMounts:
            # 볼륨이 마운트될 위치 지정
            - name: nfs-vol
              mountPath: /audit    # 볼륨이 마운트될 위치
      volumes:
        - name: nfs-vol
          persistentVolumeClaim:
            claimName: nfs-pvc    # PVC로 생성된 볼륨을 마운트하기 위해 이름 지정
```

9. 생성된 파드 확인

```
kubectl get pods
```

10. 생성된 파드 중 하나에 접속

```
kubectl exec -it [파드NAME] -- /bin/bash
```

11. PVC의 마운트 상태 확인

용량이 100Mi가 아닌 NFS 서버의 용량이 37G임을 확인

```
df -h
```

12. m-k8s 명령창을 더 열고 audit-trail 컨테이너의 기능을 테스트

외부에서 파드에 접속할 수 있도록 로드밸런서 서비스를 생성

```
kubectl expose deployment nfs-pvc-deploy --type=LoadBalancer --name=nfs-pvc-deploy-svc --port=80
```

13. 생성한 로드밸런서 서비스 IP 확인

```
kubectl get services
```

14. 호스트 브라우저에서 192.168.1.21에 접속해 확인

15. 접속한 파드에서 접속 기록 파일이 남았는지 확인

```
ls /audit
```

```
cat /audit/audit_nfs-pvc-deploy-1.log
```

16. 마스터 노드에서 파드를 4개에서 8개로 증가시키기

```
kubectl scale deployment nfs-pvc-deploy --replicas=8
```

17. 생성된 파드 확인

```
kubectl get pods
```

18. 최근 증가한 4개 파드 중 1개를 선택해 접속하고 기록된 audit 로그가 동일한지 확인

```
kubectl exec -it nfs-pvc-deploy-3 -- /bin/bash
```

19. 다른 브라우저를 열고 192.168.1.21에 접속해 다른 이름과 IP가 표시되는지 확인

20. 접속한 파드에서 새로 추가된 audit 로그를 확인

```
ls /audit
```

```
cat /audit/audit_nfs-pvc-deploy-2.log
```

21. 기존에 접속한 파드에서도 동일한 로그가 audit에 기록되어 있는지 확인

```
ls /audit
```

[실습] NFS 볼륨을 파드에 직접 마운트하기

1. 사용자가 관리자와 동일한 단일 시스템이라면 PV와 PVC를 사용할 필요가 없으므로
단순히 볼륨을 마운트하는지 확인

```
kubectl apply -f ~/Book_k8sInfra/ch3/3.4.3/nfs-ip.yaml
```

▼ nfs-ip.yaml

```
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 접속

```
kubectl get pods
```

```
kubectl exec -it [파드NAME] -- /bin/bash
```

3. 접속한 파드에서 동일한 NFS 볼륨을 바라보고 있음을 확인

```
ls /audit
```

4. 다음 진행을 위해 설치한 PV와 PVC를 제외한 나머지 파드, 서비스 삭제

```
kubectl delete deployment nfs-pvc-deploy
```

```
kubectl delete deployment nfs-ip
```

```
kubectl delete service nfs-pvc-deploy-svc
```

3.4.4 스테이트풀셋

스테이트풀셋(StatefulSet)

- 지금까지는 파드가 replicas에 선언된 만큼 무작위로 생성되었는데 **파드가 만들어지는 이름과 순서를 예측해야할 때가 있다.** 주로 레디스, 주키퍼, 카산드라, 몽고DB등의 마스터-슬레이브 구조 시스템에서 필요.
- volumeClaimTemplates 기능을 사용해 PVC를 자동으로 생성할 수 있고, 각 파드가 순서대로 생성되므로 고정된 이름, 볼륨, 설정 등을 가질 수 있음
- 효율성 면에서 좋은 구조가 아니므로 요구사항에 맞게 적용하는 것이 좋음

[실습] 스테이트풀셋 생성 과정

1. 스테이트풀셋 생성 (PV, PVC는 이미 생성되었음)

```
kubectl apply -f ~/Book_k8sInfra/ch3/3.4.4/nfs-pvc-sts.yaml
```

▼ nfs-pvc-sts.yaml

```
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 # same name of volumes's name
              mountPath: /audit
      volumes:
        - name: nfs-vol
          persistentVolumeClaim:
            claimName: nfs-pvc
```

2. 파드 생성 확인 (순서대로 하나씩 생성됨)

```
kubectl get pods -w
```

3. 생성한 스테이트풀셋에 expose 실행 ⇒ ❌ 에러

스테이트풀셋은 expose를 지원하지 않음. 파일로 로드밸런서 서비스를 작성, 실행해야함

```
kubectl expose statefulset nfs-pvc-sts --type=LoadBalancer --name=nfs-pvc-sts-svc --port=80
```

4. 스테이트풀셋을 노출하기 위한 서비스를 생성하고 생성한 로드밸런서 서비스를 확인

```
kubectl apply -f ~/_Book_k8sInfra/ch3/3.4.4/nfs-pvc-sts-svc.yaml
```

▼ 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. 호스트에서 브라우저를 열고 192.168.1.21에 접속해 확인

6. 파드에 접속한 후 새로 접속한 파드의 정보가 추가되었는지 확인 후 exit

```
kubectl exec -it nfs-pvc-sts-0 -- /bin/bash
```

```
ls -l /audit
```

```
exit
```

7. 스테이트풀셋의 파드 삭제

```
kubectl delete statefulset nfs-pvc-sts
```