

7 17 3일차

네트워크

vpc 안에 서브넷 2개

하나는 private 하나는 public

private안에는 tomcat ← db와 통신하기 위한 용도

public 안에는 nginx같은 외부 통신

- 네트워크 계층 구조

- 덜 민감한 웹 서버

- 민감한 웹서버

- 톰캣

- db

만들 때부터 IAC로 만들어야 한다.

수정이 편하게 손으로 직접 만들면 나중에 옮기기 굉장히 힘들다.

Public에서 Private으로 변경할 때

서버가 여러 대라면 문제는 더 심화된다.

Code로 즉 IAC로 할 경우 인프라의 구조를 이해하기에도 쉽고 공부도 더 잘된다.

Provider

어떤 서비스를 만들었을 때 Interface spec과 Interface API를 만든다.

EKS ⇒ 쿠버네티스가 제공하는 API와 SPEC을 이용하여 디스크 네트워크를 다 구성

따라서 EKS 환경이 쿠버네티스에서 사용하는 방법과 거의 80% 유사하다.

다시 배울 필요가 없다는 점은 굉장히 중요

실습

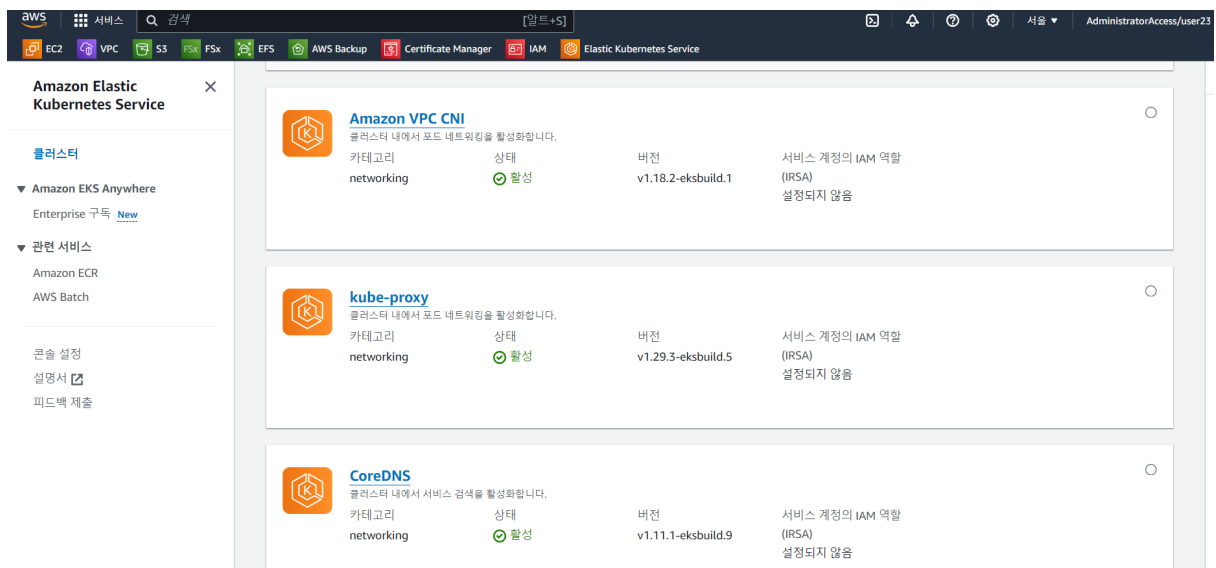
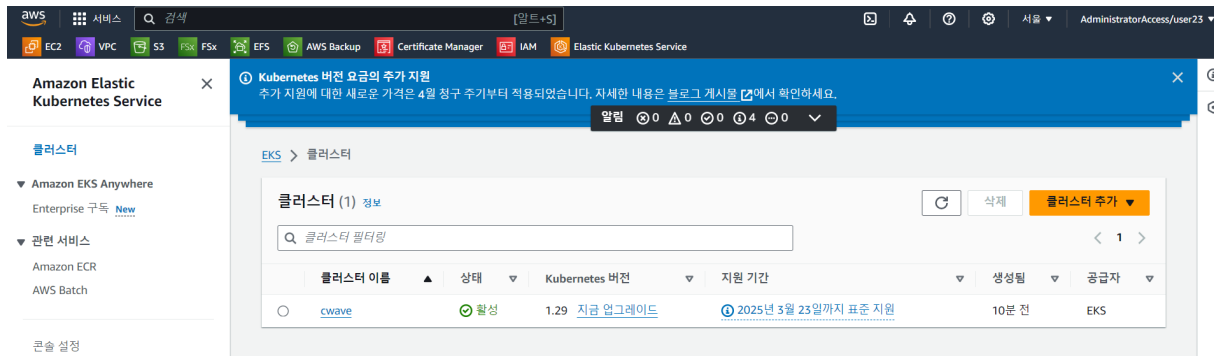
- terraform apply

```
Apply complete! Resources: 74 added, 0 changed, 0 destroyed.

Outputs:

bastion-public-ip = "3.39.222.152"
```

- eks 클러스터 활성화 확인 가능하다.



- kubectl

```
root@i348ca531c4f:/code/local/cwave-k8s/makeENV/eks# kubectl
arn:aws:eks:ap-northeast-2:211125410568:cluster/cwave
kind-cwave-cluster
```

- rename

```
root@8348ca531c4f:/code/local/cwave-k8s/makeENV/eks# aws eks update-kubeconfig --region ap-northeast-2 --name cwave
Added new context arn:aws:eks:ap-northeast-2:211125410568:cluster/cwave to /config/.kube/config
```

- rename

```
root@8348ca531c4f:/code/local/cwave-k8s/makeENV/eks# kubectl config rename-context arn:aws:eks:ap-northeast-2:211125410568:cluster/cwave eks
Context "arn:aws:eks:ap-northeast-2:211125410568:cluster/cwave" renamed to "eks".
root@8348ca531c4f:/code/local/cwave-k8s/makeENV/eks# kubectl config rename-context kind-cwave-cluster local
Context "kind-cwave-cluster" renamed to "local".
```

- 이름 바뀐 것을 확인 가능하다. 또 현재는 eks에 위치한다.

```
root@8348ca531c4f:/code/local/cwave-k8s/makeENV/eks# kubectlx
eks
local
```

- kubectl get no로 worker node를 확인 가능하다.
이 node들이 관리해야 할 것의 30%를 대체해준다.
이 node들은 eks에 위치한 worker node들이다.

```
root@8348ca531c4f:/code/local/cwave-k8s/makeENV/eks# kubectl get no
NAME                                                    STATUS    ROLES    AGE   VERSION
ip-10-1-3-208.ap-northeast-2.compute.internal         Ready    <none>    4m19s v1.29.3-eks-ae9a62a
ip-10-1-4-244.ap-northeast-2.compute.internal         Ready    <none>    4m20s v1.29.3-eks-ae9a62a
```

- local로 전환한 뒤 worker node들을 확인한다.

```
root@8348ca531c4f:/code/local/cwave-k8s/makeENV/eks# kubectlx local
Switched to context "local".
root@8348ca531c4f:/code/local/cwave-k8s/makeENV/eks# kubectl get no
NAME                                STATUS    ROLES    AGE   VERSION
cwave-cluster-control-plane         Ready    control-plane   42h   v1.29.4
cwave-cluster-worker                Ready    <none>         42h   v1.29.4
cwave-cluster-worker2              Ready    <none>         42h   v1.29.4
```

- 현재 local에 있음을 확인 가능하다.

```
root@8348ca531c4f:/code/local/cwave-k8s/makeENV/eks# kubectlx
eks
local
```

- 수정후 재 apply

```
local > cwave-k8s > makeENV > eks > 004_eks.tf
4   module "eks" {
30     eks_managed_node_group_defaults = {
31       instance_types = [ "m7i.large" ]
32     }
33
34     eks_managed_node_groups = {
35       green = {
36         min_size      = 2
37         max_size      = 4
38         desired_size  = 3
39       }
40       instance_types = [ "m7i.large" ]
41     }
42   }
43 }
44
```

```
Terraform will perform the following actions:

# module.eks.module.eks_managed_node_group["green"].aws_eks_node_group.this[0] will be updated in-place
~ resource "aws_eks_node_group" "this" {
  id          = "cwave:green-20240717004406770400000019"
  tags       = {
    "Name" = "green"
  }
  # (16 unchanged attributes hidden)

  ~ scaling_config {
    ~ max_size = 2 -> 4
    # (2 unchanged attributes hidden)
  }

  # (3 unchanged blocks hidden)
}

Plan: 0 to add, 1 to change, 0 to destroy.
```

서비스

K8s 에서 제공하는 서비스의 종류

구분항목	내부접속		외부접속	
	Cluster IP	NodePort	LoadBalancer	Ingress
외부 접속	X	O	O	O
L7 지원	X	X	X	O
30000 이하 포트 사용	O	X	O	O
가상 호스팅 (virtual Hosting)	X	X	X	O
성능	좋음	안좋음	좋음	좋음
Production 권장	O	X	O	O

넷플릭스는 마이크로 서비스이기에 어떤 서비스의 tomcat에 접근하기 위해선
그 tomcat의 상황을 알 필요가 있는데 알 방법이 없었음 따라서 유레카 서비스를 개발
유레카 서비스는 일종의 모든 log를 기록해놓은 db같은 녀석이다.
따라서 유레카 서비스를 이용해 접근하고자 하는 서비스의 상태를 물어보고 접속

항상 모든 서비스는 core dns를 조회한 뒤에 찾아감
ip로 찾아가지 말라고 core dns를 사용

pod가 뜨면 그 pod를 찾아가는 dns 규칙이 이미 다 정해져 있다.

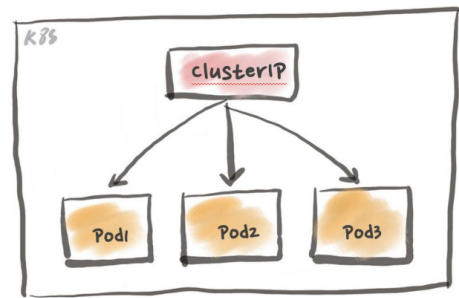
쿠버네티스 특성상 노드 또는 pod에 장애가 발생할 경우 다른 노드 및 pod에 옮겨서 실행
⇒ 정적 라우팅 불가능함

쿠버네티스에서 4가지의 서비스 제공

- cluster ip

Clusterip 란?

- kube-proxy 기반으로 동작 (IPTables 또는 IPVS 모드 선택 가능)
- IPTables 모드에서 라우팅은 랜덤 선택
- IPVS 모드에서는 아래와 같은 라우팅 알고리즘 선택 가능
- 다른 서비스의 컴포넌트로 포함



알고리즘	
RR	라운드 로빈(번갈아 가며 한번씩)
LC	최소연결(커백션이 가장 적은 노드에 라우팅)
DH	목적지 해싱
SH	소스 해싱
SED	최단 예상 지연(Shortest expected delay)
NQ	큐 미사용(Never Queue)

IPTable 모드에서는 netfilter 를 사용해 유저스페이스와 커널 사이에 변환이 필요 없어 성능이 더 좋음

위에 로드 밸런서가 하나 있어서 아래에 있는 node중 하나에 제공
cluster ip는 내부에 있는 것들과의 통신에 사용된다.

kube-proxy-5cl2j	1/1	Running	1 (25h ago)	43h
kube-proxy-ms9mj	1/1	Running	1 (25h ago)	43h
kube-proxy-n4z7n	1/1	Running	1 (25h ago)	43h

kube proxy가 구현체이다.
소스 해싱: 특정 위치에 있는 pod만 지정해서 하겠다.

오늘 내가 최초로 접속했을 때 라운드 로빈이 적용되는 것이 아닌 세션이 만료되고 재접속 했을 때 적용된다.

로그인 정보만 레디스에 보관한다. pod가 죽어도 서비스 이용자들이 로그인을 유지할 수 있게

- NodePort
노드 포트 서비스를 통해서 로컬 쿠버네티스로 외부에서 접근이 가능하다.

노드의 특정 ip로 port forwarding하면 clusterip로 연결이 된다.

특정 노드로 접속하더라도 clusterip로 연동되기 때문에 로드밸런싱이 된다.

nodeport는 운영 용도가 아닌 개발 하는데 사용된다.
host, container, service port로 총 3개의 port를 열어서 사용한다.

- LoadBalancer

외부와 통신을 위한 서비스 1

트래픽을 로드밸런서가 받고 로드밸런서가 클러스터 ip랑만 연결되어서 클러스터 ip에 전달해주고 클러스터 ip가 아래 pod들에게 전달

클러스터 ip가 자동으로 생성된다.

layer 4 이다 ip하고 port만 본다

- Ingress

외부와 통신을 위한 서비스 2

그림을 잘못 그렸다 ingress도 밖에 있고 virtual hosting을 지원한다.

url기반으로 라우팅이 가능하다. 즉 도메인 안에서 쪼개서 다른 호스팅을 제공하는 것임

layer 7이다 모든 것을 볼 수 있다. 그래서 가상 호스팅이 가능한 것이다. packet을 다 까서 볼 수 있기 때문이다. 헬스 체크까지 할 수도 있다. http

helm.tf에

```
38 resource "helm_release" "eks_common_alb" {
39   provider = helm.cwave-eks-helm
40   name     = "aws-load-balancer-controller"
41   chart    = "aws-load-balancer-controller"
42   version  = "1.6.2"
43   repository = "https://aws.github.io/eks-charts"
44   namespace = "kube-system"
45
46   dynamic "set" {
47     for_each = {
48       "clusterName"           = var.cluster-name
49       "serviceAccount.create" = "true"
50       "serviceAccount.name"   = local.lb_controller_service_account_name
51       "region"                = "ap-northeast-2"
52       "vpcId"                 = aws_vpc.vpc.id
53       "image.repository"      = "602401143452.dkr.ecr.ap-northeast-2.amazonaws.com/amazon/aws-load-bala
54       "serviceAccount.annotations.eks\\.amazonaws\\.com/role-arn" = module.cwave_eks_lb_controller_role.iam_role_arn
55     }
56
57     content {
58       name = set.key
59       value = set.value
60     }
61   }
62 }
```

이렇게 코드가 존재하고 이 코드가 I4 ,I7 로드 밸런서를 만드는 모듈이다.

Headless service

헤드리스 서비스는 본인의 ip를 주는 것이 아닌 본인 뒤에 있는 것들의 ip를 제공한다.

로드 밸런싱 뒤에 있는 것을 클러스터로 운영할 때 헤드리스 서비스 사용

대표적인 서비스 : redis, 카산드라, 몽고db

뒤에 있는 것들 중 어디에 저장할지 그리고 뒤에 있는 놈들 중 어떤 게 살아있는지 등등의 이유로 사용된다.

클러스터화

클러스터는 항상 reader가 있다. 5개 4개의 node가 있다면 한 node가 죽으면 모든 node가 알고 있다.

클러스터의 조건

1. 노드가 2개 이상이어야 한다.
2. 한 노드가 죽으면 모든 노드가 알게끔 컨센서스를 구성

실습

app.js

```
const http = require('http');
const os = require('os');

console.log("Kubia server starting...");

var handler = function(request, response) {
  console.log("Received request from " + request.connection.remoteAddress);
  response.writeHead(200);
  response.end("You've hit " + os.hostname() + "\n");
};

var www = http.createServer(handler);
www.listen(8080);
```

Dockerfile

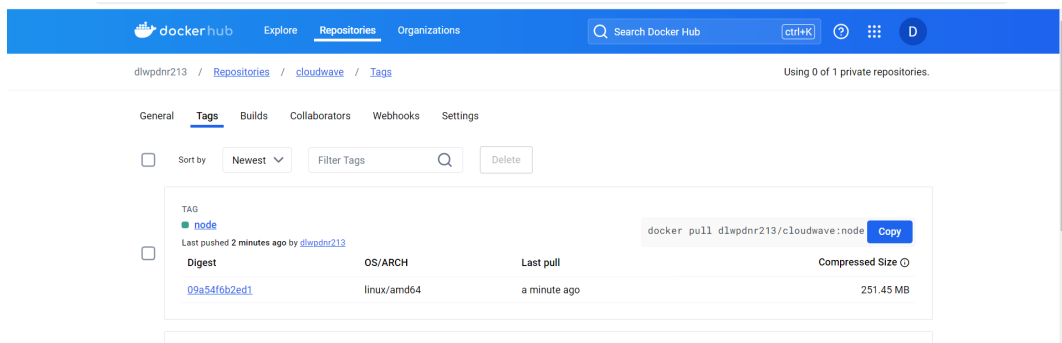
```
# FROM 으로 BASE 이미지 로드
FROM node:7

# ADD 명령어로 이미지에 app.js 파일 추가
ADD app.js /app.js

# ENTRYPOINT 명령어로 node 를 실행하고 매개변수로 app.js 를 전달
ENTRYPOINT ["node", "app.js"]
```

```
docker build -t dlwpdnr213/cloudwave:node # 이미지 해당 태그로 빌드
docker push dlwpdnr213/cloudwave:node # 이미지 내 계정 도커 허브로 푸시
```


- 푸쉬된 것 확인 가능하다.



- nodeapp-deploy.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodeapp-deploy
spec:
  selector:
    matchLabels:
      app: nodeapp-pod
  replicas: 3
  template:
    metadata:
      labels:
        app: nodeapp-pod
    spec:
      containers:
        - name: nodeapp-container
          image: dlwpdnr213/cloudwave:node
          resources:
            limits:
              memory: "128Mi"
              cpu: "500m"
          ports:
            - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: nodeapp-svc
spec:
  selector:
    app: nodeapp-pod
  ports:

```

```
- port: 8081
  targetPort: 8080
```

- 세 개를 붙여서 다른 파일을 이어서 작성 가능하다.

```
root@8348ca531c4f:/code/local/cwave-k8s/practice/service/clusterip/manifest# kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/nodeapp-deploy-7d6c4bf4f8-76vtz	1/1	Running	0	2m24s
pod/nodeapp-deploy-7d6c4bf4f8-hjwhc	1/1	Running	0	2m17s
pod/nodeapp-deploy-7d6c4bf4f8-lnmng	1/1	Running	0	2m32s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.120.0.1	<none>	443/TCP	4m51s
service/nodeapp-svc	ClusterIP	10.120.51.22	<none>	8081/TCP	4m40s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/nodeapp-deploy	3/3	3	3	4m40s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/nodeapp-deploy-7d6c4bf4f8	3	3	3	2m32s
replicaset.apps/nodeapp-deploy-7dfbd5dd5b	0	0	0	4m40s

- 그냥 curl 했을 때 안되는거 확인 가능하다.

```
root@8348ca531c4f:/code/local/cwave-k8s/practice/service/clusterip/manifest# curl http://10.120.51.22:8081
curl: (7) Failed to connect to 10.120.51.22 port 8081 after 2478 ms: Connection refused
root@8348ca531c4f:/code/local/cwave-k8s/practice/service/clusterip/manifest# kubectl port-forward service/nodeapp-svc 8081:8081
```

- port-forwarding 이후에 하면?

```
root@8348ca531c4f:/code# curl http://localhost:8081
You've hit nodeapp-deploy-7d6c4bf4f8-lnmng
```

연습문제 12-1

[연습문제 12-1] ClusterIP

다음 조건을 만족하는 서비스와 ClusterIP 를 생성 하세요

- Deployment

항목	내용
kind	Deployment
image	nginx:1.8.9
replicas	3
containerPort	80

- Service

항목	내용
kind	Service
type	ClusterIP
port	80
targetPort	80

- nginx:1.8.9 이미지를 이용하여 Replica=3 인 Deployment 를 생성하세요
- nginx 서비스를 로드밸링싱 하는 서비스를 ClusterIP 로 생성하세요
- kubernetes port-forward를 이용해서 네트워크를 연결하고 curl 명령어로 웹사이트를 조회 하세요

- nginx.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
spec:
  selector:
    matchLabels:
      app: nginx-pod
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx-pod
    spec:
      containers:
        - name: nginx-container
          image: nginx:1.9.1
          resources:
```

```

      limits:
        memory: "128Mi"
        cpu: "500m"
      ports:
      - containerPort: 80
    ---
  apiVersion: v1
  kind: Service
  metadata:
    name: nginx-svc
  spec:
    selector:
      app: nginx-pod
    ports:
    - port: 80
      targetPort: 80

```

- kubectl get all로 확인

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-1# kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/nginx-deploy-ffddcb795-94zhg    1/1     Running   0           32s
pod/nginx-deploy-ffddcb795-ggxzb    1/1     Running   0           30s
pod/nginx-deploy-ffddcb795-vp728    1/1     Running   0           31s
pod/nodeapp-deploy-7d6c4bf4f8-76vtz 1/1     Running   0           48m
pod/nodeapp-deploy-7d6c4bf4f8-hjwhc 1/1     Running   0           48m
pod/nodeapp-deploy-7d6c4bf4f8-lnmng 1/1     Running   0           48m

NAME                                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/kubernetes                  ClusterIP    10.120.0.1    <none>        443/TCP    50m
service/nginx-svc                  ClusterIP    10.120.25.40 <none>        80/TCP     68s
service/nodeapp-svc                ClusterIP    10.120.51.22 <none>        8081/TCP   50m

```

- port-forwarding으로 열기

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-1# kubectl port-forward service/nginx-svc 80:80
Forwarding from 127.0.0.1:80 -> 80
Forwarding from [::1]:80 -> 80
Handling connection for 80

```

- localhost로 접근하기

```
● root@8348ca531c4f:/code# curl http://localhost:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
```

sticky session은 웹 서버와 백엔드 서비스로 갈 때 붙는다

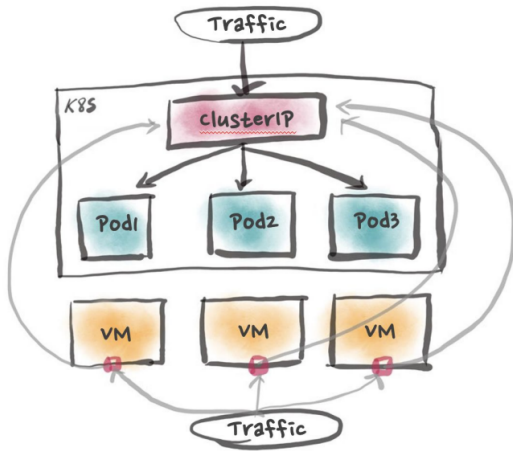
사용자가 웹 서버 갈 때는 아무 곳이나 붙는다. 랜덤으로 붙음

따라서 내가 이렇게 local로 접근했을 때 replica로 만들어진 3개의 node에 랜덤으로 붙는다는 것이다.
nginx는 웹서버니까 사용자가 직접 접근해서 랜덤이다.

NodePort

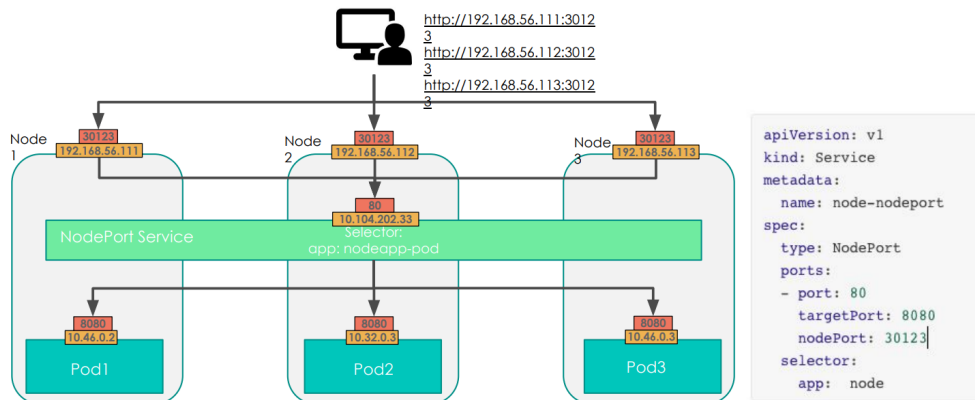
NodePort 란?

- 노드의 특정 포트를 할당 및 개방 하여 서비스(ClusterIP) 와 연동함
- 노드의 포트 할당 범위가 (30000 ~ 32767 : --service-node-port-range) 로 제한됨. : Product에 적합하지 않은 이유
- 특정 노드로 접속하더라도 ClusterIP 로 연동 되기 때문에 로드밸런싱 됨



NodePort 상세 구성 및 단점

- 서비스 포트가 30000 ~ 32767 사이로 제한됨
- 장비나 VM의 IP 혹은 포트 변경 발생시 대응 어려움



실습

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodeapp-deploy
```

```
spec:
  selector:
    matchLabels:
      app: nodeapp-pod
  replicas: 3
  template:
    metadata:
      labels:
        app: nodeapp-pod
    spec:
      containers:
        - name: nodeapp-container
          image: dangtong/nodeapp
          resources:
            limits:
              memory: "128Mi"
              cpu: "500m"
            ports:
              - containerPort: 8080

---
apiVersion: v1
kind: Service
metadata:
  name: nodeapp-svc
spec:
  type: NodePort # 이걸 명시 안해주면 클러스터가된다.
  selector:
    app: nodeapp-pod
  ports:
    - port: 8081
      targetPort: 8080
      nodePort: 30123
```

- 확인

```
root@8348ca531c4f:/code/local/cwave-k8s/practice/service/nodeport# kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/nodeapp-deploy-55dd79c44d-2sgln	0/1	ContainerCreating	0	5s
pod/nodeapp-deploy-55dd79c44d-7vdmw	1/1	Running	0	5s
pod/nodeapp-deploy-55dd79c44d-t2zhp	1/1	Running	0	5s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.120.0.1	<none>	443/TCP	33s
service/nodeapp-svc	NodePort	10.120.8.168	<none>	8081:30123/TCP	5s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/nodeapp-deploy	2/3	3	2	5s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/nodeapp-deploy-55dd79c44d	3	3	2	5s

- curl로 확인하기

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/service/nodeport# curl http://192.168.128.3:30123
You've hit nodeapp-deploy-55dd79c44d-t2zhp
root@8348ca531c4f:/code/local/cwave-k8s/practice/service/nodeport# curl http://192.168.128.4:30123
You've hit nodeapp-deploy-55dd79c44d-t2zhp
root@8348ca531c4f:/code/local/cwave-k8s/practice/service/nodeport# curl http://192.168.128.2:30123
You've hit nodeapp-deploy-55dd79c44d-t2zhp

```

30123 port로 모든 node 안의 pod에 접근이 가능함으로 nodePort가 되는 것을 확인할 수 있다.

eks local 환경 비교

모든 노드마다 서비스는 kube proxy를 사용한다.

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/service/nodeport# kubectl get po -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE                                NOMINATED NODE   READINESS GATES
nodeapp-deploy-55dd79c44d-2sgln     1/1     Running   0           12m   10.110.1.52     cwave-cluster-worker               <none>            <none>
nodeapp-deploy-55dd79c44d-7vdmw     1/1     Running   0           12m   10.110.1.51     cwave-cluster-worker               <none>            <none>
nodeapp-deploy-55dd79c44d-t2zhp     1/1     Running   0           12m   10.110.2.61     cwave-cluster-worker2             <none>            <none>
root@8348ca531c4f:/code/local/cwave-k8s/practice/service/nodeport# kubectl get svc -o wide
NAME      TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE   SELECTOR
kubernetes ClusterIP  10.120.0.1    <none>        443/TCP          14m   <none>
nodeapp-svc NodePort    10.120.8.168 <none>        8081:30123/TCP   14m   app=nodeapp-pod

```

ex) local kind 클러스터에 ide에서 부터 접속한다.

pod에다 하는 건 pod 하나만 딱 골라서 api서버로 한 것임

서비스에다 하는 거는 클러스터 아래에 모든 노드에 존재하는 pod port에다 한 것

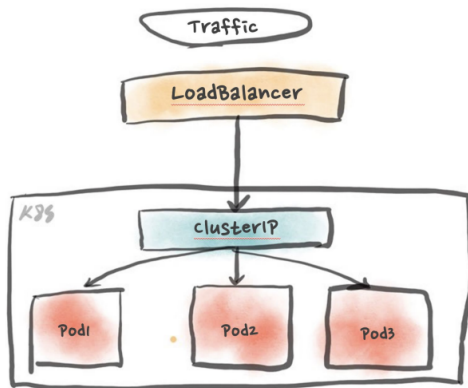
서비스는 모든 컨테이너 걸쳐서 떴기 때문에

클러스터 ip는 무조건 portforwarding을 해야 한다.

Road balance

LoadBalancer 란?

- ClusterIP + 외부 로드밸런서
- ClusterIP 가 외부 통신이 불가능 하기 때문에 앞 단계 LoadBalancer 와 결합하여 외부 트래픽 유입이 가능한 서비스
- 외부 로드밸런서는 Layer4 로드밸런서. (GCP의 경우 TCP 로드밸런스 할당)



실습

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodeapp
spec:
  selector:
    matchLabels:
      app: nodeapp
  replicas: 3
  template:
    metadata:
      labels:
        app: nodeapp
    spec:
      containers:
        - name: nodeapp
          image: dlwvdpnr213/cloudwave:node
          resources:
            limits:
              memory: "128Mi"
              cpu: "500m"
          ports:
            - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
```

```

name: nodeapp-lb
annotations:
  service.beta.kubernetes.io/aws-load-balancer-type: external
  service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
  service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing

spec:
  type: LoadBalancer
  selector:
    app: nodeapp
  ports:
    - port: 80
      targetPort: 8080

```

aws에선 어노테이션을 메타데이터에 추가해줘야한다.

- 어노테이션

```

service.beta.kubernetes.io/aws-load-balancer-type: external
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing

```

- kubectl get po,svc,deploy

```

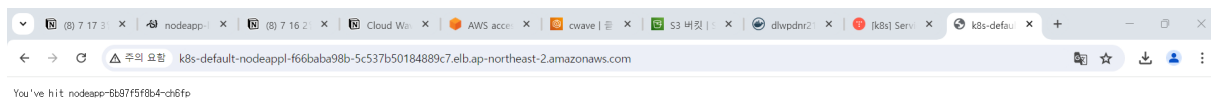
root@8348ca531c4f:/code/local/cwave-k8s/practice/service/loadbalancer# kubectl apply -f nodeapp-lb.yaml
deployment.apps/nodeapp created
service/nodeapp-lb configured
root@8348ca531c4f:/code/local/cwave-k8s/practice/service/loadbalancer# kubectl get po,svc,deploy
NAME                                READY    STATUS    RESTARTS   AGE
pod/nodeapp-6b97f5f8b4-ch6fp        1/1      Running   0           18s
pod/nodeapp-6b97f5f8b4-m4fg8        1/1      Running   0           18s
pod/nodeapp-6b97f5f8b4-qgkqg        1/1      Running   0           18s

NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/kubernetes                  ClusterIP           172.20.0.1      <none>           443/TCP          4h33m
service/nodeapp-lb                 LoadBalancer       172.20.138.44   k8s-default-nodeapp1-f66baba98b-5c537b50184889c7.elb.ap-northeast-2.amazonaws.com  80:31672/TCP    72s

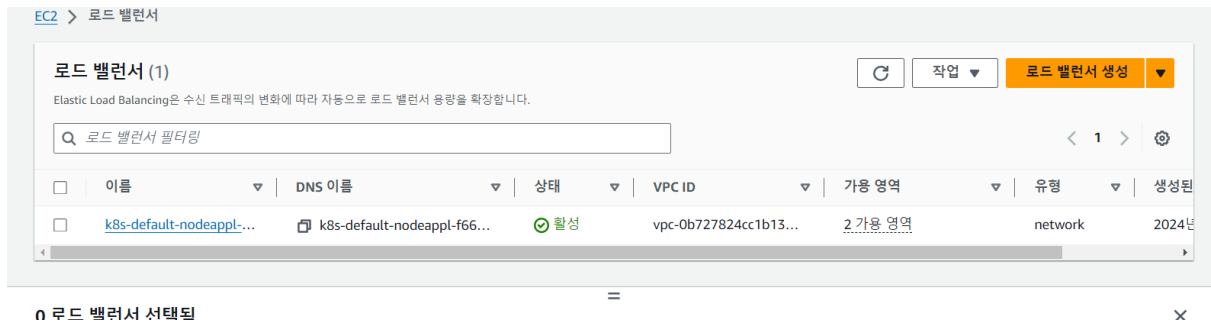
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/nodeapp              3/3      3              3            18s

```

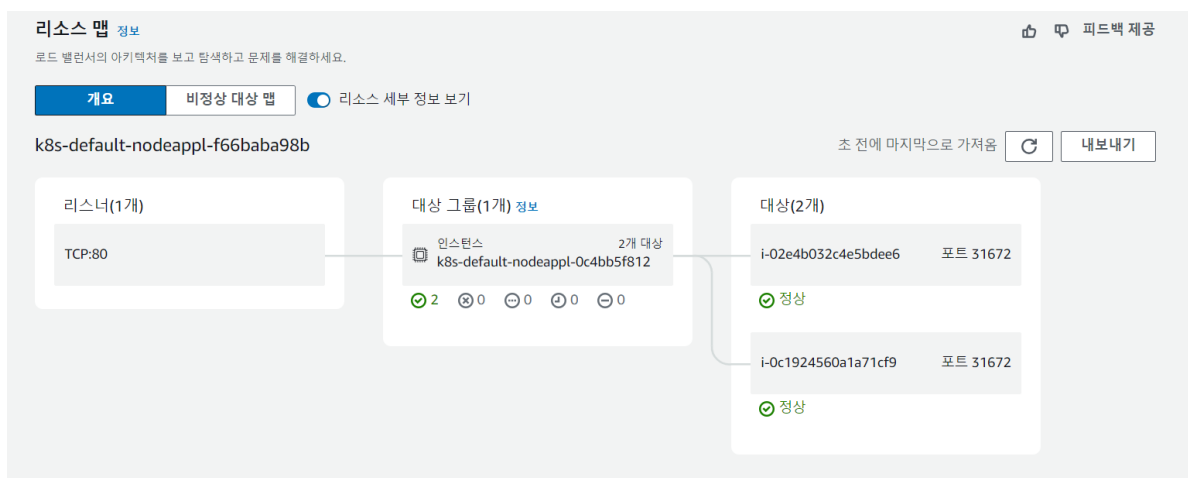
- external-ip로 접속



- aws에서 생성이 된 것을 확인 가능하다.



• 대상그룹 확인



metallb 실습

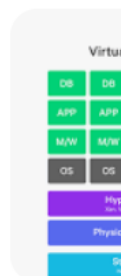
'베어메탈(Bare Metal)'이란 용어는 원래 하드웨어 상에 어떤 소프트웨어도 설치되어 있지 않은 상태를 뜻합니다. 즉, 베어메탈 서버는 가상화를 위한 하이퍼바이저 OS 없이 물리 서버를 그대로 제공하는 것을 말합니다. 따라서 하드웨어에 대한 직접 제어 및 OS 설정까지 가능합니다.



가비아

<https://library.gabia.com/contents/infrahosting>

베어메탈 서버(Bare Metal Server)란? - 가비아 라이브러리



MetalLB란?

Kubernetes 사용 시 AWS, GCP, Azure 와 같은 클라우드 플랫폼에서는 자체적으로 로드 밸런서(Load Balancer)를 제공해 주지만, 온프레미스 클러스터에서는 로드 밸런싱 기능을 제공하는 모듈을 추가적으로 설치해야 한다.

MetalLB는 MetalLB는 **BareMetalLoadBalancer** 약자로 베어메탈 환경에서 사용할 수 있는 로드 밸런서를 제공하는 오픈소스 프로젝트이다. 클라우드 환경의 서비스(로드밸런서 타입)와는 동작이 조금 다르다.

서비스(로드 밸런서)의 External IP 전파를 위해서 표준 프로토콜인 ARP(IPv4)/NDP(IPv6), BGP 를 사용한다. 데몬셋으로 speaker 파드를 생성하여 External IP 전파한다.

- kind 네트워크 서브넷 확인

```
PS C:\kkk\local_code\cwave-k8s\makeENV\IDE> docker inspect kind
[
  {
    "Name": "kind",
    "Id": "5146cd0027405155d81d1008b5656edb889b107584bad3509619c38e4fe918e5",
    "Created": "2024-07-15T06:08:40.112663439Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": true,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "192.168.128.0/20",
          "Gateway": "192.168.128.1"
        }
      ]
    }
  }
]
```

- metallb 다운로드

```
kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.14.5/config/
```

- metallb address address 대역대를 내 서브넷 네트워크 대역대로 수정 후 실행

```

local > cwave-k8s > makeENV > kindCluster > metallb > ! metallb-addressPool.yaml > ...
1  apiVersion: metallb.io/v1beta1
2  kind: IPAddressPool
3  metadata:
4    name: cwave-pool
5    namespace: metallb-system
6  spec:
7    addresses:
8      - 192.168.128.10-192.168.128.20 # set by make target
9    ---
10 apiVersion: metallb.io/v1beta1
11 kind: L2Advertisement
12 metadata:
13   name: cwave-loadbalancer-advertisement
14   namespace: metallb-system
15 spec:
16   ipAddressPools:
17     - cwave-pool

```

- 로드밸런서 external- ip 확인

```

root@8348ca531c4f:/code/local/cwave-k8s/makeENV/kindCluster/metallb# kubectl apply -f metallb-addressPool.yaml
ipaddresspool.metallb.io/cwave-pool created
l2advertisement.metallb.io/cwave-loadbalancer-advertisement created
root@8348ca531c4f:/code/local/cwave-k8s/makeENV/kindCluster/metallb# kubectl get all

```

NAME	READY	STATUS	RESTARTS	AGE
pod/nodeapp-6b97f5f8b4-7sb8b	1/1	Running	0	3m27s
pod/nodeapp-6b97f5f8b4-b58h7	1/1	Running	0	3m27s
pod/nodeapp-6b97f5f8b4-lwkdp	1/1	Running	0	3m27s
pod/nodeapp-deploy-55dd79c44d-2sgln	1/1	Running	0	167m
pod/nodeapp-deploy-55dd79c44d-7vdmw	1/1	Running	0	167m
pod/nodeapp-deploy-55dd79c44d-t2zhp	1/1	Running	0	167m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.120.0.1	<none>	443/TCP	168m
service/nodeapp-lb	LoadBalancer	10.120.216.210	192.168.128.10	80:31956/TCP	3m27s
service/nodeapp-svc	NodePort	10.120.8.168	<none>	8081:30123/TCP	167m

- curl을 보낸다

```

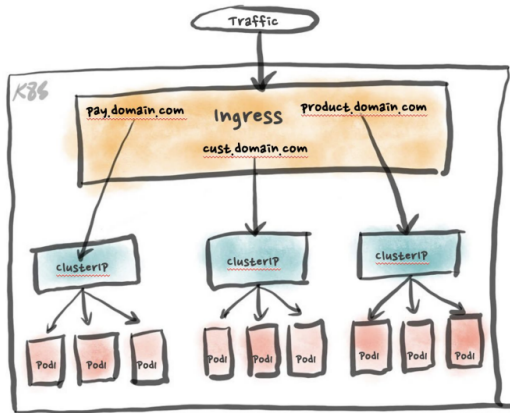
root@8348ca531c4f:/code/local/cwave-k8s/practice/service/loadbalancer# curl http://192.168.128.10:80
You've hit nodeapp-6b97f5f8b4-7sb8b

```

Ingress

Ingress 란?

- NodePort + 외부 로드밸런서
- 반드시 Ingress 컨트롤러가 있어야함 (Nginx 등)
- Ingress Controller 리스트 : <https://kubernetes.io/ko/docs/concepts/services-networking/ingress-controllers/>
- 외부 로드밸런서는 Layer7 로드밸런서.



Ingress가 맨 앞단에 위치 Ingress는 L7 layer이기에 http로 온 패킷을 까서 dns주소를 ip로 전환

Ingress 하단에 존재하는 load balancer에게 전달이 가능하다. 로드밸런스는 L4 layer이기에 클러스터에게 전달 cluster들은 전달 받은 대로 그 아래에 존재하는 pod에게 전달한다

이 pod들은 deployment로 배포된 곳 안에 존재한다.

실습

- deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
```

```

        image: nginx:1.9.1
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 80

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: goapp
spec:
  selector:
    matchLabels:
      app: goapp
  replicas: 3
  template:
    metadata:
      labels:
        app: goapp
    spec:
      containers:
      - name: goapp
        image: dangtong/goapp
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 8080

```

ingress가 load balancer를 지정해주고 load balancer가 pod를 지정해준다

- load balance 생성 yaml

```

apiVersion: v1
kind: Service
metadata:
  name: nginx-lb
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:

```

```

- port: 80 # 내가 정하는거
  targetPort: 80 # 정해져있는거 nginx가 정한거

---
apiVersion: v1
kind: Service
metadata:
  name: goapp-lb
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  type: LoadBalancer
  selector:
    app: goapp
  ports:
    - port: 80
      targetPort: 8080

```

- 두 파일 실행

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/service/ingress# kubectl get all
NAME                                READY    STATUS    RESTARTS   AGE
pod/goapp-6ccf95f594-4lztz         1/1      Running   0           4m32s
pod/goapp-6ccf95f594-55vz7         1/1      Running   0           4m32s
pod/goapp-6ccf95f594-wswms         1/1      Running   0           4m32s
pod/nginx-545b74fc67-jvsmr         1/1      Running   0           4m32s
pod/nginx-545b74fc67-tj7lp         1/1      Running   0           4m32s
pod/nginx-545b74fc67-zvx49         1/1      Running   0           4m32s

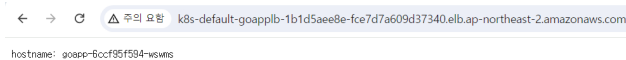
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/goapp-lb                    LoadBalancer        172.20.214.57   k8s-default-goapplb-1b1d5aee8e-fce7d7a609d37340.elb.ap-northeast-2.amazonaws.com 80:32365/TCP    14m
service/kubernetes                  ClusterIP            172.20.0.1      <none>           443/TCP          6h28m
service/nginx-lb                    LoadBalancer        172.20.14.179   k8s-default-nginxb-7d20bdfd27-1e5d75c0cfab6ff3.elb.ap-northeast-2.amazonaws.com 80:31459/TCP    14m

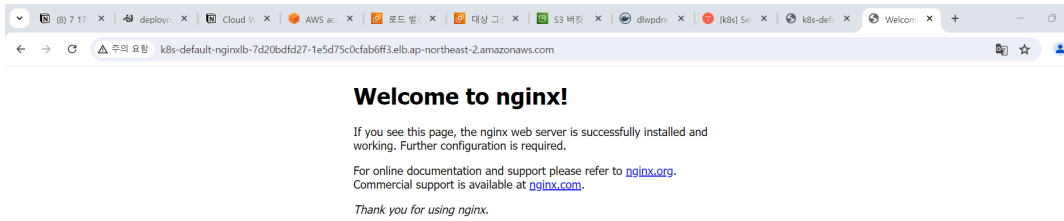
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/goapp                3/3      3              3            4m32s
deployment.apps/nginx                3/3      3              3            4m32s

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/goapp-6ccf95f594     3          3          3        4m33s
replicaset.apps/nginx-545b74fc67     3          3          3        4m33s

```

- 결과 확인


 주위 요함 k8s-default-goapplb-1b1d5aee8e-fce7d7a609d37340.elb.ap-northeast-2.amazonaws.com
 hostname: goapp-6ccf95f594-wswms



- ingress.yaml 작성

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb-ingress-class
spec:
  controller: ingress.k8s.aws/alb
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
  labels:
    name: ingress
spec:
  ingressClassName: "alb-ingress-class"
  rules:
    - host: nginx.duldul100.com
      http:
        paths:
          - pathType: Prefix
            path: "/"
            backend:
              service:
                name: nginx-lb
                port:
                  number: 80
    - host: goapp.duldul100.com
      http:
        paths:
          - pathType: Prefix
            path: "/"
            backend:
              service:
```

```
name: goapp-lb
port:
  number: 80
```

- ingress 확인

```
root@8348ca531c4f:/code/local/cwave-k8s/practice/service/ingress# kubectl get ingress
NAME      CLASS      HOSTS
ingress   alb-ingress-class  nginx.duldul100.com,goapp.duldul100.com  k8s-default-ingress-f8e435b167-394172120.ap-northeast-2.elb.amazonaws.com  80
```

- aws 확인

로드 밸런서 (1/3)

Elastic Load Balancing은 수신 트래픽의 변화에 따라 자동으로 로드 밸런서 용량을 확장합니다.

로드 밸런서 필터링

	이름	DNS 이름	상태	VPC ID	가용 영역	유형	생성
<input type="checkbox"/>	k8s-default-nginxlb-7...	k8s-default-nginxlb-7d20...	✓ 활성화	vpc-0b727824cc1b13...	2 가용 영역	network	202
<input type="checkbox"/>	k8s-default-goapplb-1...	k8s-default-goapplb-1b1d...	✓ 활성화	vpc-0b727824cc1b13...	2 가용 영역	network	202
<input type="checkbox"/>	k8s-default-ingress-f8...	k8s-default-ingress-f8e43...	프로비저닝 중	vpc-0b727824cc1b13...	2 가용 영역	application	202

- 연결 됐는지 확인법

1. describe ingress backend ip 3개 물렸는지 확인

```
root@8348ca531c4f:/code/local/cwave-k8s/practice/service/ingress# kubectl describe ingress
Name: ingress
Labels: name=ingress
Namespace: default
Address: k8s-default-ingress-f8e435b167-394172120.ap-northeast-2.elb.amazonaws.com
Ingress Class: alb-ingress-class
Default backend: <default>
Rules:
  Host      Path  Backends
  ----
  nginx.duldul100.com  /  nginx-lb:80 (10.1.3.161:80,10.1.4.112:80,10.1.4.183:80)
  goapp.duldul100.com  /  goapp-lb:80 (10.1.3.149:8080,10.1.3.228:8080,10.1.4.44:8080)
Annotations: alb.ingress.kubernetes.io/scheme: internet-facing
```

2. endpoint에 물려있으면 ok

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/service/ingress# kubectl describe svc nginx-lb
Name: nginx-lb
Namespace: default
Labels: <none>
Annotations: service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
              service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
              service.beta.kubernetes.io/aws-load-balancer-type: external
Selector: app=nginx
Type: LoadBalancer
IP Family Policy: SingleStack
IP Families: IPv4
IP: 172.20.14.179
IPs: 172.20.14.179
LoadBalancer Ingress: k8s-default-nginxlb-7d20bdfd27-1e5d75c0cfab6ff3.elb.ap-northeast-2.amazonaws.com
Port: <unset> 80/TCP
TargetPort: 80/TCP
NodePort: <unset> 31459/TCP
Endpoints: 10.1.3.161:80,10.1.4.112:80,10.1.4.183:80

```

- ingress 생성 확인

로드 밸런서 (1/3)

Elastic Load Balancing은 수신 트래픽의 변화에 따라 자동으로 로드 밸런서 용량을 확장합니다.

로드 밸런서 필터링

이름	DNS 이름	상태	VPC ID	가용 영역	유형	생성
k8s-default-nginxlb-7...	k8s-default-nginxlb-7d20...	✓ 활성	vpc-0b727824cc1b13...	2 가용 영역	network	202
k8s-default-goapplb-1...	k8s-default-goapplb-1b1d...	✓ 활성	vpc-0b727824cc1b13...	2 가용 영역	network	202
k8s-default-ingress-f8...	k8s-default-ingress-f8e43...	✓ 활성	vpc-0b727824cc1b13...	2 가용 영역	application	202

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/service/ingress# kubectl get ingress
NAME CLASS HOSTS ADDRESS PORTS
ingress alb-ingress-class nginx.duldul100.com,goapp.duldul100.com k8s-default-ingress-f8e435b167-394172120.ap-northeast-2.elb.amazonaws.com 80
4m55s

```

- nslookup 확인

```

leejeuk@DESKTOP-KM62UF8:/mnt/c/Users/kdt$ nslookup
> k8s-default-ingress-f8e435b167-394172120.ap-northeast-2.elb.amazonaws.com
Server: 10.255.255.254
Address: 10.255.255.254#53

Non-authoritative answer:
Name: k8s-default-ingress-f8e435b167-394172120.ap-northeast-2.elb.amazonaws.com
Address: 13.124.205.158
Name: k8s-default-ingress-f8e435b167-394172120.ap-northeast-2.elb.amazonaws.com
Address: 15.165.141.191

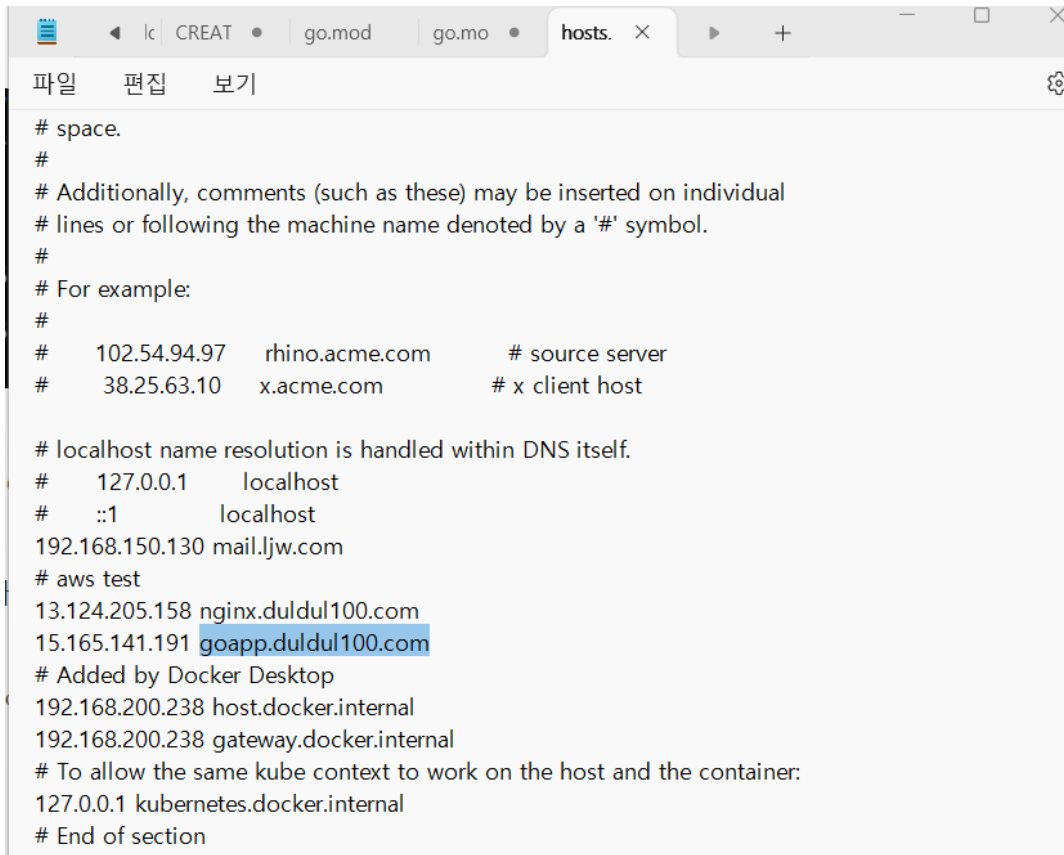
```

dns 서버에 묻기전에 1차적으로 리눅스는 /etc/host 밑에를 뒀진다.

윈도우 browser에서 검색을 시도할 것이기 때문에

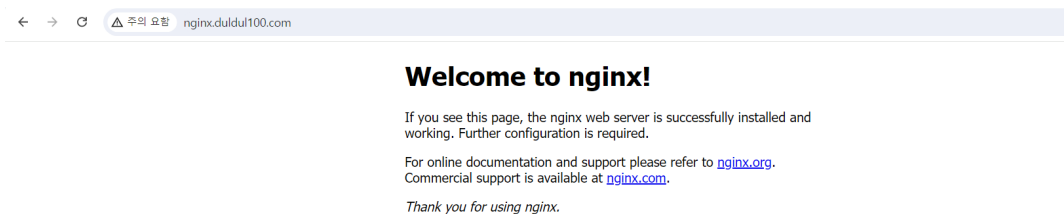
C:\Windows\System32\drivers\etc\hosts 파일을 수정

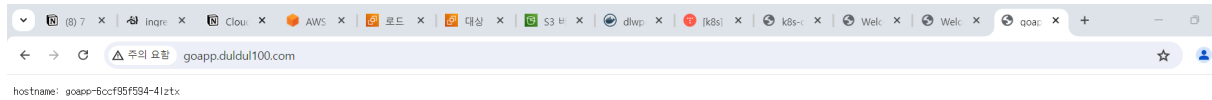
- 완성된 2 ip로 파일 수정



```
# space.  
#  
# Additionally, comments (such as these) may be inserted on individual  
# lines or following the machine name denoted by a '#' symbol.  
#  
# For example:  
#  
# 102.54.94.97 rhino.acme.com      # source server  
# 38.25.63.10  x.acme.com         # x client host  
  
# localhost name resolution is handled within DNS itself.  
# 127.0.0.1 localhost  
# ::1 localhost  
192.168.150.130 mail.ljw.com  
# aws test  
13.124.205.158 nginx.duldul100.com  
15.165.141.191 goapp.duldul100.com  
# Added by Docker Desktop  
192.168.200.238 host.docker.internal  
192.168.200.238 gateway.docker.internal  
# To allow the same kube context to work on the host and the container:  
127.0.0.1 kubernetes.docker.internal  
# End of section
```

- dns 주소 확인





	aws	kind	설치필요
cluster	0	0	
loadbalancer	0	metallb 설치 필요	
nodeport	방화벽 해제해야 가능	0	
ingress	0 설치가 필요한데 이미 배포할때 해놨음	nginx 컨트롤러 설치 → ingress 생성 가능	

연습문제 12-2

kind에서 ingress 사용하는 환경을 구축해보자!

<https://kind.sigs.k8s.io/docs/user/ingress/>

ingress는 7계층이기 때문에 http, https만 가능 tcp 안된다.

```
cat <<EOF | kind create cluster --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    protocol: TCP
  - containerPort: 443
    hostPort: 443
    protocol: TCP
EOF
```

일단 tcp port mapping만 해놔라 뒷단 하게

1. Ingress controller 설치

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-2# kubectl get ns
NAME                STATUS   AGE
default              Active   2d2h
first-namespace      Active   31h
ingress-nginx        Active   19m
kube-node-lease      Active   2d2h
kube-public          Active   2d2h
kube-system          Active   2d2h
local-path-storage   Active   2d2h
metallb-system       Active   149m
my-dev              Active   30h
my-ns                Active   31h

```

```

my-ns                Active   31h
root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-2# kubectl get all -n ingress-nginx
NAME                                                    READY   STATUS    RESTARTS   AGE
pod/ingress-nginx-admission-create-n22fz              0/1     Completed 0           19m
pod/ingress-nginx-admission-patch-lgw8v              0/1     Completed 0           19m
pod/ingress-nginx-controller-7b7b4c4647-jwh2w        1/1     Running   0           19m

NAME                                                    TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
service/ingress-nginx-controller                      NodePort      10.120.241.97 <none>         80:32109/TCP,443:31660/TCP 19m
service/ingress-nginx-controller-admission            ClusterIP     10.120.76.64  <none>         443/TCP          19m

NAME                                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ingress-nginx-controller              1/1     1            1           19m

NAME                                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/ingress-nginx-controller-7b7b4c4647  1         1         1       19m

NAME                                                    COMPLETIONS   DURATION   AGE
job.batch/ingress-nginx-admission-create              1/1           16s       19m
job.batch/ingress-nginx-admission-patch              1/1           16s       19m

```

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-2# kubectl get crd
NAME                                                    CREATED AT
bfdprofiles.metallb.io                                2024-07-17T05:54:43Z
bgpadvertisements.metallb.io                         2024-07-17T05:54:43Z
bgppeers.metallb.io                                  2024-07-17T05:54:43Z
communities.metallb.io                               2024-07-17T05:54:43Z
ipaddresspools.metallb.io                            2024-07-17T05:54:43Z
l2advertisements.metallb.io                          2024-07-17T05:54:43Z
serviceL2statuses.metallb.io                         2024-07-17T05:54:43Z

```

metallb는 crd가 만들어진 것을 확인 가능 ingress는 이미 잘 구현되어 있기에 기본 스펙에 포함되어 있어 서비스만 만들어주면 된다. crd를 만들지는 않는다.

2. 테스트

```

kubectl wait --namespace ingress-nginx \
  --for=condition=ready pod \
  --selector=app.kubernetes.io/component=controller \
  --timeout=90s

```

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-2# kubectl wait --namespace ingress-nginx \
--for=condition=ready pod \
--selector=app.kubernetes.io/component=controller \
--timeout=90s
pod/ingress-nginx-controller-7b7b4c4647-jwh2w condition met

```

컨테이너가 running으로 될 때까지 waiting하는 스크립트이다.

3. kind-ingress.yaml 파일 생성

aws에서 ingress 아래 lb가 있었다면 여기서 즉 local kind에선 ingress 아래 svc 즉 service를 두겠다.

물론 metal lb가 있기에 lb로도 할 수 있다. 하지만 지금 예제에서는 metal lb가 없다 가정하고 서비스로 하는 것이다.

여기서 서비스는 cluster ip이다.

```

kind: Pod
apiVersion: v1
metadata:
  name: foo-app
  labels:
    app: foo
spec:
  containers:
  - command:
    - /agnhost
    - netexec
    - --http-port
    - "8080"
    image: registry.k8s.io/e2e-test-images/agnhost:2.39
    name: foo-app
  ---
kind: Service
apiVersion: v1
metadata:
  name: foo-service
spec:
  selector:
    app: foo
  ports:
    # Default port used by the image
    - port: 8080
  ---
kind: Pod

```

```

apiVersion: v1
metadata:
  name: bar-app
  labels:
    app: bar
spec:
  containers:
  - command:
    - /agnhost
    - netexec
    - --http-port
    - "8080"
    image: registry.k8s.io/e2e-test-images/agnhost:2.39
    name: bar-app
---
kind: Service
apiVersion: v1
metadata:
  name: bar-service
spec:
  selector:
    app: bar
  ports:
    # Default port used by the image
    - port: 8080
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
  - http:
    paths:
    - pathType: Prefix
      path: /foo
      backend:
        service:
          name: foo-service
          port:
            number: 8080
    - pathType: Prefix
      path: /bar
      backend:
        service:
          name: bar-service
          port:
            number: 8080
---

```



```

root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-2# kubectl get ingress
NAME          CLASS    HOSTS    ADDRESS    PORTS    AGE
example-ingress <none>   *        localhost  80       30m
root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-2# kubectl get svc
NAME          TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
bar-service   ClusterIP    10.120.73.85   <none>         8080/TCP   76s
foo-service   ClusterIP    10.120.184.65  <none>         8080/TCP   76s
kubernetes    ClusterIP    10.120.0.1     <none>         443/TCP    16m

```

3. curl

```

},
"23126153467ac3437fe81ba62fce763e924422abd19c416abbea015a6a3224d8": {
  "Name": "cwave-cluster-control-plane",
  "EndpointID": "0eacebc691aab62be158b817a619ac15bd6ca5440d33e089f87fcee53193cb0",
  "MacAddress": "02:42:c0:a8:80:02",
  "IPv4Address": "192.168.128.2/20",
  "IPv6Address": "fc00:f853:ccd:e793::2/64"
},

```

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-2# curl http://192.168.128.2/foo
○ NOW: 2024-07-17 08:40:05.672710026 +0000 UTC m=+246.439608017root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-2#
root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-2#

```

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-2# curl http://192.168.128.2/bar
○ NOW: 2024-07-17 08:40:40.78933296 +0000 UTC m=+281.529764565root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-2#

```

4. 접속

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-2# kubectl get po -n ingress-nginx
NAME                                READY    STATUS    RESTARTS    AGE
ingress-nginx-admission-create-n22fz 0/1      Completed 0            37m
ingress-nginx-admission-patch-lgw8v   0/1      Completed 0            37m
ingress-nginx-controller-7b7b4c4647-jwh2w 1/1      Running   0            37m

```

```

root@8348ca531c4f:/code/local/cwave-k8s/practice/exercise/12-2# kubectl get ingress
NAME          CLASS    HOSTS    ADDRESS    PORTS    AGE
example-ingress <none>   *        localhost  80       45m

```

ingress는 너의 host ip라는 것 즉 control plane이라는 것임 따라서 위에 curl이 된 것

localhost는 ingress가 떠 있는 ip를 의미한다. 거기에다 80 port에다 curl을 던지니까 됐다.

ingress가 main node 즉 control plane에 떠있고 다른 서비스들이 worker node에 떠도 걸쳐있기 때문에 결국은 ingress에서 찾아 들어간다.