7 15 1일차

컨테이너

추상화

- 복잡함 감추기
- 인터페이스 표준화
- 필요한 기능만 넣자

네트워크 SDN
스토리지 SDS
서버 VM

물리적인 것들을 SW로 연결시켜 능률 상승

OS를 어떻게 추상화할까? 리눅스만 쓰자. 커널 공유가 가능하니까

배포판은 다양하나 내부 커널은 하나다 같다. 코드도 같다

컨테이너는 이러한 커널을 공유하고 리소스를 격리 해 주는 것
리소스: CPU MEMORY DISK PROCESS NETWORK 등
이러한 자원 침범 X

컨테이너는 하드웨어와 소프트웨어를 추상화해서 연결

소비자를 Global scale bc2 최초는 아마존이다.

실제 운영서비스로 전환하면 aws의 비용은 매우 비쌈
많아진 컨테이너를 관리할 것이 생김 ⇒ 쿠버네티스 ⇒ 서비스 메쉬(트래픽까지 제어)

마이크로 서비스 아키텍처

한 개의 서비스 ⇒ 여러 개로 쪼갠다
독립적인 배포 때문에 한다.

도메인 구역을 나눈 뒤 따로

콘웨이의 법칙 :
어플리케이션 아키텍처는 그것을 개발하는 조직 구조를 그대로 반영한다.

역콘웨이의 전략
조직의 구조가 마이크로 서비스에 적용되도록 조직을 설계

느슨한 결합

- 하나의 서비스가 변경될 때 다른 서비스가 변경 x
- 시스템의 그 어떤 부분도 추가 변경 x 변경 후 배포 0
- 강한 결합은 변경을 더 어렵게 한다.

응집력
단일 책임원칙 하에 하나의 마이크로 서비스 안에 함께 변경되는 것들은 같은 곳에 모은다.

결정 경계

차단하자.

모놀리식 ⇒ 모듈러 ⇒ 미니 서비스⇒ 마이크로 서비스⇒ 펑셔널

코드 베이스의 크기이다.

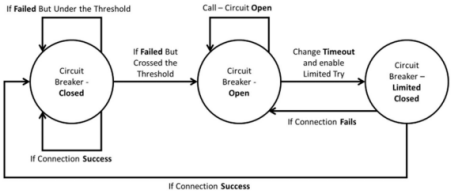
요새는 미니 서비스가 유행이다.

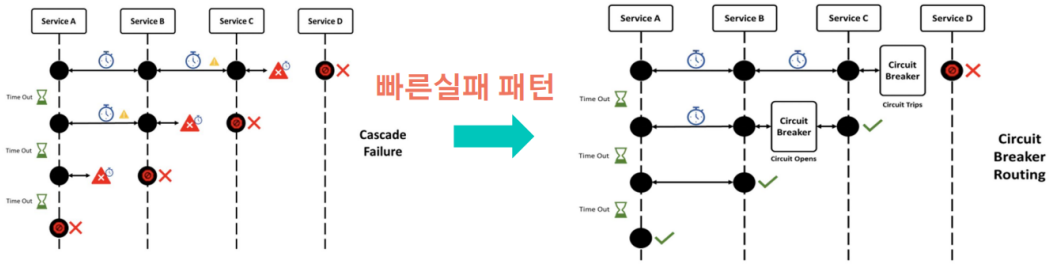
마이크로와 미니의 차이점은 데이터베이스까지 분리하냐 안하냐

이 db를 분리하는게 굉장히 어렵다.

2. 회복성 (Resilience)

- 하나의 서비스 장애가 다른 서비스로 전이 되지 않음
- 전체 장애를 차단 하고 기능은 저하시킴
- 분산 기술에 대한 깊은 이해 필요





hang이 쌓여서 queue가 쌓이면 문제가 된다.

빠른 실패 패턴

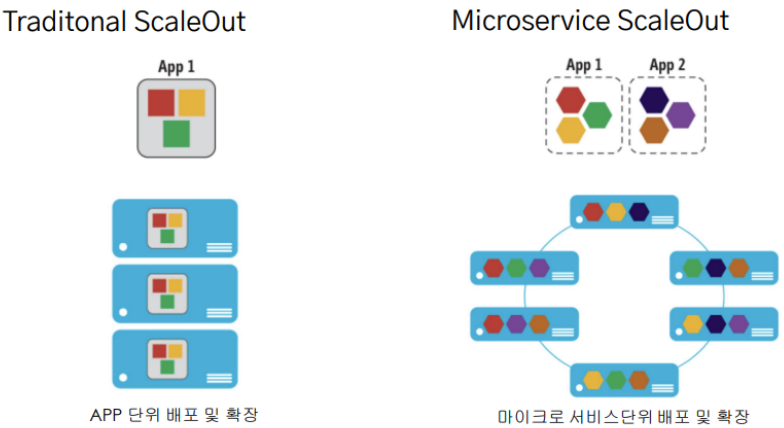
서비스가 도미노 이펙트 하나 죽은거에 영향 받는다. 영향 덜 받게 죽었음 바로 죽었음을 알게끔

마이크로 서비스로 할 때는 네트워크로 하기에 신뢰할 수 없는 이런 네트워크를 커버할 수 있는 패턴이 필요

이런게 쿠버네티스 안에 구현되어 있다. 예전에는 이런걸 다 만들어야 했다.

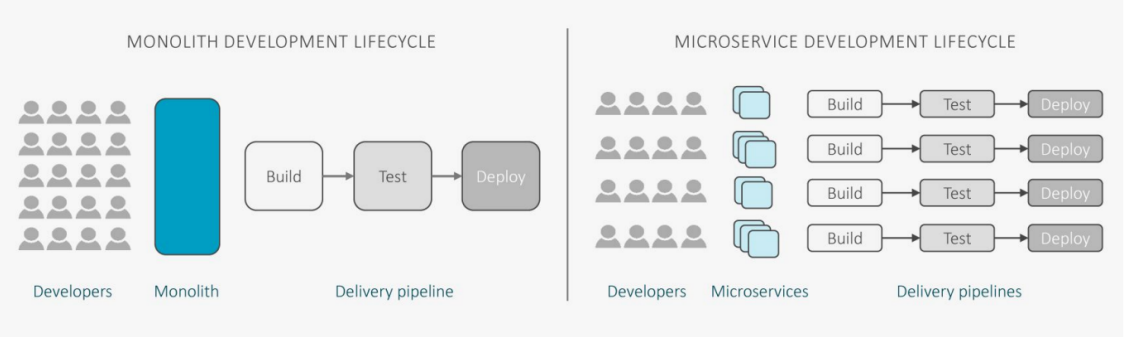
3. 확장성 (Scale Out)

- 하나의 서비스 장애가 다른 서비스로 전이 되지 않음
- 전체 장애를 차단 하고 기능은 저하시킴
- 분산 기술에 대한 깊은 이해 필요
- 전통적인 ScaleOut 과 다름

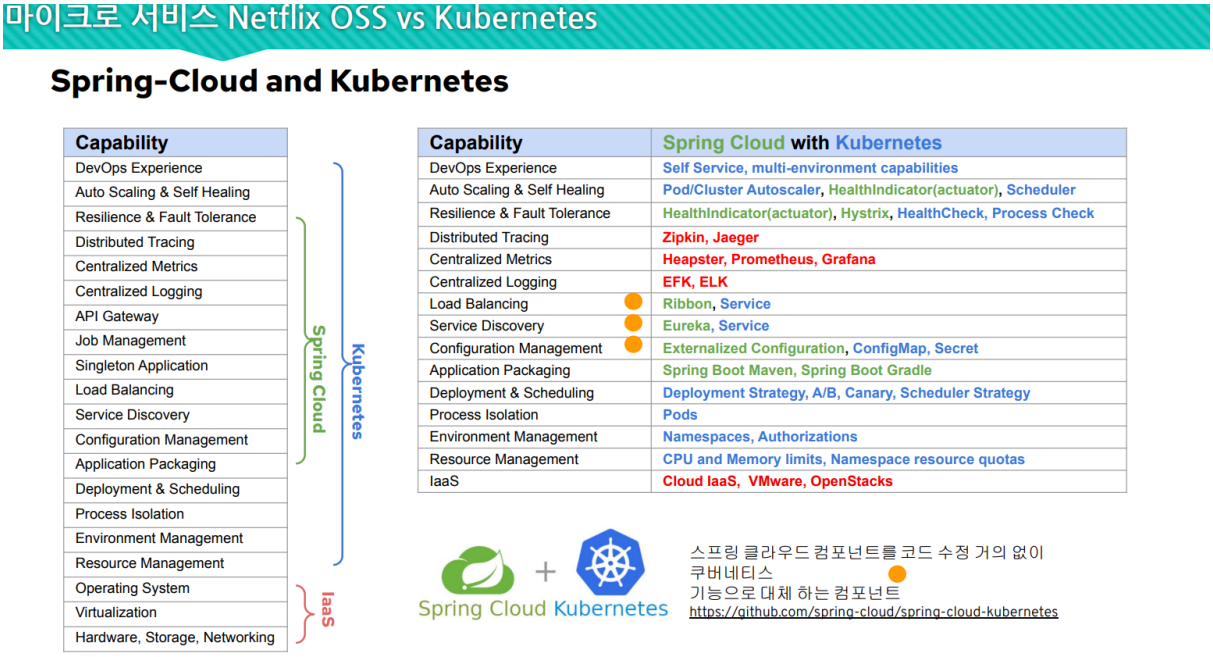


4. 배포 용이성 (Easy Deployment)

- 전체 빌드(Build) 가 아닌 서비스 단위의 빌드 및 배포
- 테스트 코드 또는 TDD 필요
- 컨테이너 환경에 매우 적합
- 다같이 야근 안해도 됨



Microservices Concern	Spring Cloud & Netflix OSS	Kubernetes
Configuration Management	Config Server, Consul, Netflix Archaius	Kubernetes ConfigMap & Secrets
Service Discovery	Netflix Eureka, Hashicorp Consul	Kubernetes Service & Ingress Resources
Load Balancing	Netflix Ribbon	Kubernetes Service
API Gateway	Netflix Zuul	Kubernetes Service & Ingress Resources
Service Security	Spring Cloud Security	-
Centralized Logging	ELK Stack (LogStash)	EFK Stack (Fluentd)
Centralized Metrics	Netflix Spectator & Atlas	Heapster, Prometheus, Grafana
Distributed Tracing	Spring Cloud Sleuth, Zipkin	OpenTracing, Zipkin
Resilience & Fault Tolerance	Netflix Hystrix, Turbine & Ribbon	Kubernetes Health Check & resource isolation
Auto Scaling & Self Healing	-	Kubernetes Health Check, Self Healing, Autoscaling
Packaging, Deployment & Scheduling	Spring Boot	Docker/Rkt, Kubernetes Scheduler & Deployment
Job Management	Spring Batch	Kubernetes Jobs & Scheduled Jobs
Singleton Application	Spring Cloud Cluster	Kubernetes Pods



Spring cloud는 java만..

SRE 엔지니어 = 데브옵스 + CODE 능력

현실적으론 마이크로는 힘들다 미니가 더 현실적인 옵션

CNCF

Sandbox → incubating → zoro

리눅스 재단이 컨테이너화를 가속화하기 위해 만든 재단

https://all.devstats.cncf.io/d/53/projects-health-table?orgId=1

---

kind

kubeadm이라는 오픈 소스가 있다. 쿠버네티스를 로컬이나 소스에 조금 도구화해서 사용할 수 있게끔 한다. 근데 이마저도 어려워서 이걸 더 쉽게 한게 kind이다.

쿠버네티스

끊임없이 서비스를 유지하려고 최선을 다하는 것 manifast대로
쿠버네티스 controller가 계속 감시를 한다.

어플리케이션 개발에 집중할 수 있도록 추상화

기본 구성 단위

- Control plane(master)
  - etcd
    분산된 데이터베이스
    etcd는 최소 3개 이상
    eventually consistency 일관성 당장은 맞지 않지만 언젠가는 맞춰진다.
  - controller manager
  - scheduler

모든게 다 api 통신이다.

kubectl 도커같은 클라이언트

큐브 씨티엘 (x) 큐브 커틀(o) 큐브 컨트롤(세모)

3대를 사용하는 이유
하나 업데이트 하려고 내리는 동안 dr 1 main 1 사용하도록

- nodes(worker)
  - kubelet
    일꾼 명령대로 일하고 node에서 일어나는 모든 일을 컨트롤러에 보고
  - pod
    여러개의 컨테이너를 관리하기 위한 기능을 넣어놓음
    이렇게 서로 관심사를 다 나눠서 분리해서 구현 컨테이너에 기능이 몰빵되지 않게끔
  - 컨테이너 런타임
    컨테이너를 실행하는 엔진 docker
  - OCI
    컨테이너 표준

manifast 파일로 완성해서 컨트롤플레인 (마스터)에 넘기며 워커노드가 배포해서 유지하려고 노력

서버 디스크 네트워크 추상화 ⇒ 쿠버네티스

개발자는 개발만해라 하드웨어 신경쓰지말고 쿠버네티스의 목적

설정을 먼저 만들면 설정을 깃에다 저장 가능

예전에는 생성하고 설정 요즘에는 설정하고 생성

큐브 컨트롤

홈 디렉토리 밑 .kube 디렉토리에 config 파일이 존재 여기에 인증 정보가 있다.

```
PS C:\Users\kdt\.kube> kubectl get no
NAME                          STATUS   ROLES           AGE   VERSION
cwave-cluster-control-plane   Ready    control-plane   14m   v1.29.4
cwave-cluster-worker          Ready    <none>          13m   v1.29.4
cwave-cluster-worker2         Ready    <none>          13m   v1.29.4
```

실습 환경 구축

```yaml
version: "3.8"
name: "cloudwave_practice"

services:
  server:
    image: dangtong/cloudwave_env:1.0
    container_name: "ide"
    networks:
      - kind_network
    environment:
      FILE__PASSWORD: /run/secrets/code-server-password
    env_file:
      - .env
    working_dir: /code
    ports:
      - "8444:8443"
    secrets:
      - code-server-password
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - local_code:/code/local
```

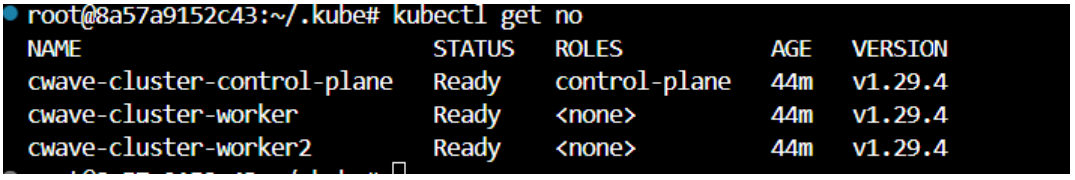
```
      - remote_code:/code/remote
      - config:/config
    # depends_on:
    #   server:
    #     condition: service_completed_successfully
  gitsync:
    profiles:
      - init
    image: registry.k8s.io/git-sync/git-sync:v4.1.0
    container_name: "git"
    environment:
      GITSYNC_REPO: https://github.com/matenduel/cloudwave
      GITSYNC_ROOT: /tmp/git
      GITSYNC_REF: main
      GITSYNC_DEPTH: 1
      GITSYNC_ONE_TIME: 1
    volumes:
      - remote_code:/tmp
networks:
  kind_network:
    name: kind
    external: true

volumes:
  local_code:
    external: true
    name: local_code
  remote_code:
    external: true
    name: remote_code
  config:
    external: true
    name: config

secrets:
  code-server-password:
    file: password.txt
```

docker compose 로 컨테이너 띄운 뒤

네트워크가 kind 네트워크로 묶여 있으니 kubectl 인증 정보를 master node의 ip로 지정해줘야 한다

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCVENDQW
    server: https://192.168.128.4:6443
  name: kind-cwave-cluster
```

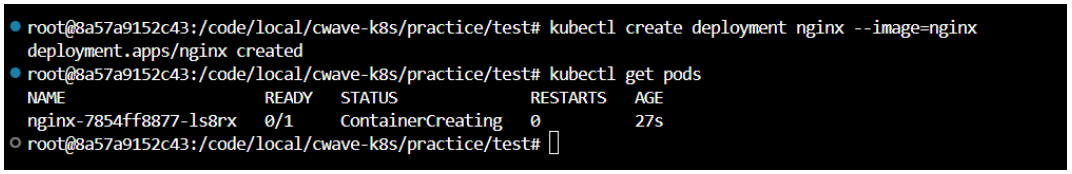
```
contexts:
- context:
    cluster: kind-cwave-cluster
    user: kind-cwave-cluster
  name: kind-cwave-cluster
current-context: kind-cwave-cluster
kind: Config
preferences: {}
users:
- name: kind-cwave-cluster
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURLVENDQWhHZ
    client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFb3dJQkFBS0NBU
```

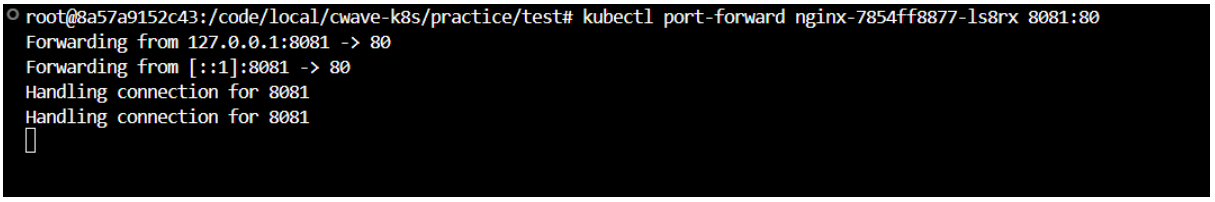


되는 걸 확인 가능하다.

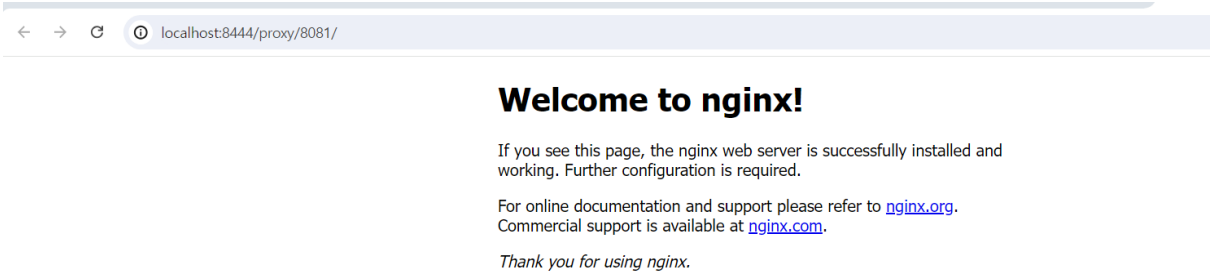
실습1

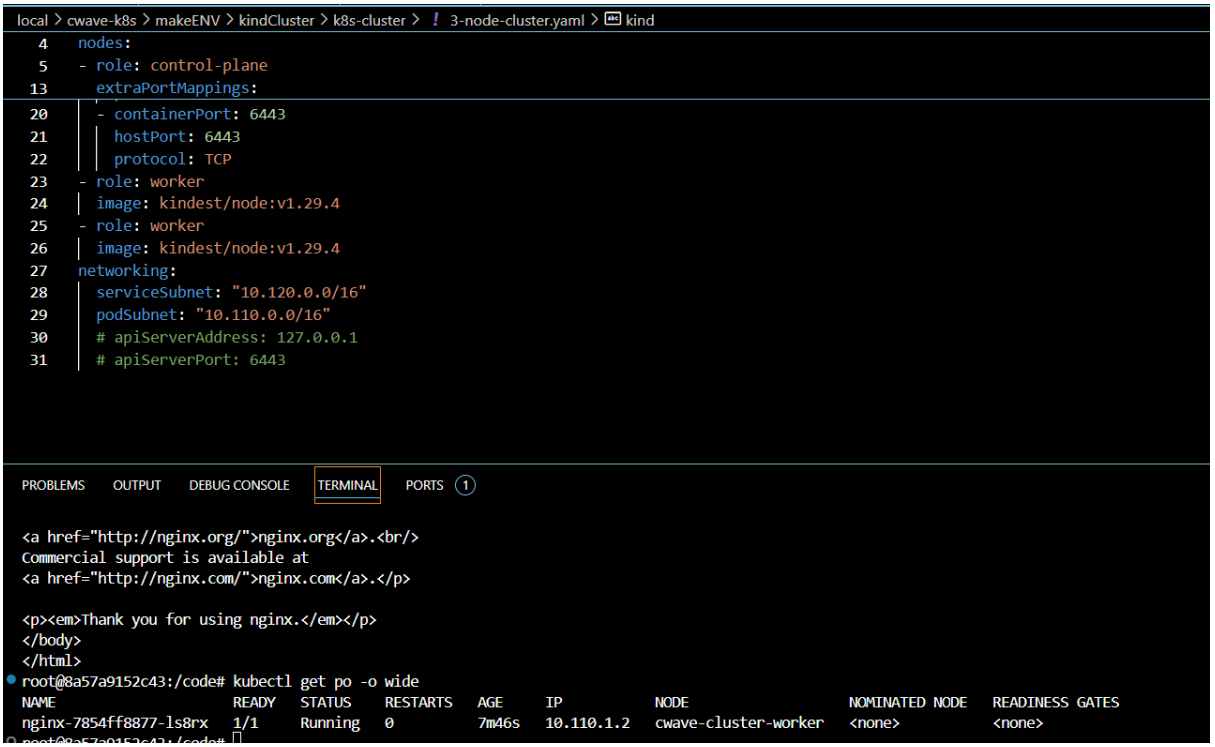
```
kubectl create deployment nginx --image=nginx
```





```
root@8a57a9152c43:/code# curl http://localhost:8081
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
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
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```





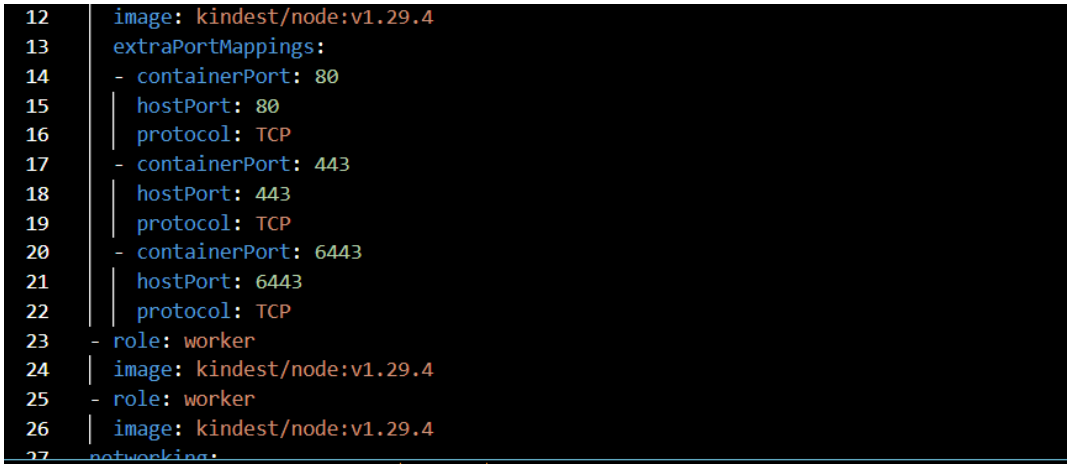
쿠버네티스 입장에선 host가 노트북이 아닌 노트북에 띄워져있는 컨테이너이다.

하지만 그 컨테이너 ip를 쿠버네티스는 사용하지 않고 쿠버네티스만의 네트워크를 만든다.

왜 이렇게 하냐?

- ip 고갈될까봐 쿠버네티스만의 것을 만듦
- 보안을 위해서 내부와 외부 단절

그렇기 때문에 nginx 연결을 하기 위해 kubectl port-forwarding을 한 것



6443을 port forwarding을 해놓았다. 따라서 api 서버는 바로 연결이 된 것

POD

쿠버네티스의 기본 실행 단위이다.

POD안에는 컨테이너가 여러 개 들어갈 수 있다.

하지만 관례가 있다. POD에는 컨테이너가 한 개만 있는 것이 좋다.
POD안에 컨트롤할 걸 여러 개 두지 말자는 것임.

사이드카 패턴:
모니터링 하는 컨테이너는 같이 하나 더 있어도 된다..

POD는 NODE를 걸쳐서 존재하지는 않는다.

지금 실습 환경에서 HOST이자 NODE는 WORKER 2 MASTER PLANE 1 총 세 개인 것임

POD끼리는 PLAT NETWORK로 NAT, GATEWAY 없이 LAN처럼 통신이 가능하다.

보통 YAML로 실행하지 CLI로 POD를 생성하지는 않음 LOG가 안남기 때문에

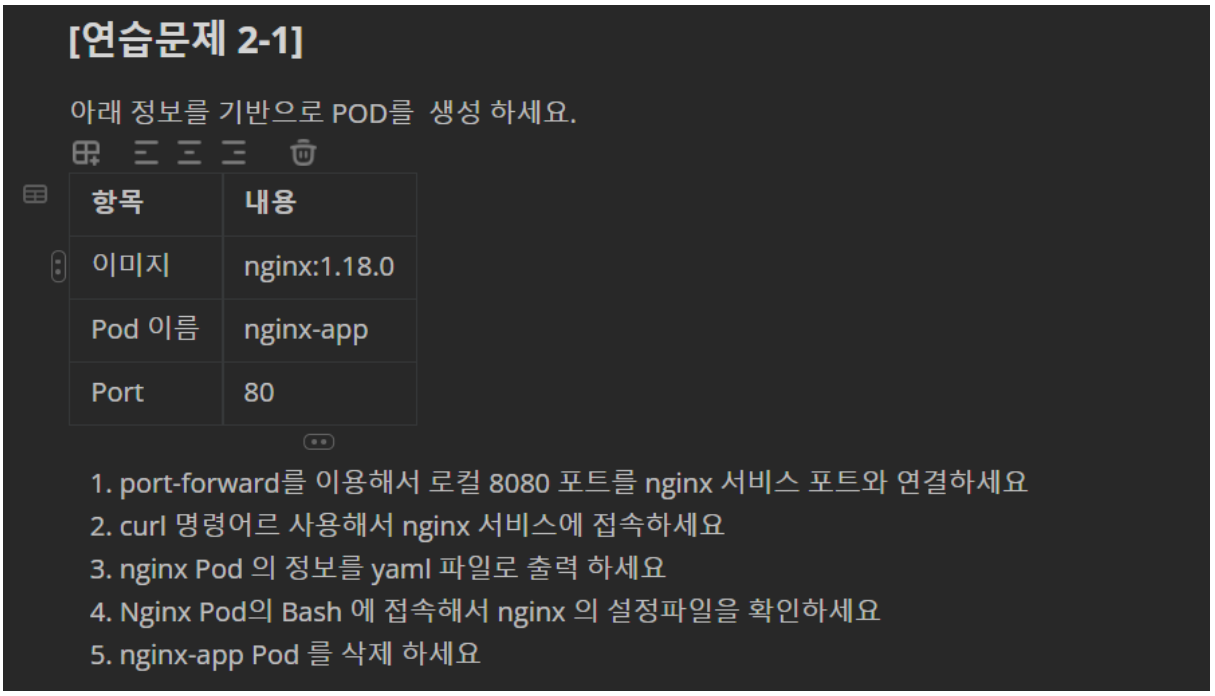
Pod는 컨테이너를 가지고 있는 객체

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx // pod의 이름
spec:
  containers:
  - name: nginx // 컨테이너의 이름
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

limit은 무조건 설정해야 한다. 이거에 따라 master plane의 성능이 크게 달라진다.

kubectl도 get으로 접근 가능

연습 문제 2-1



현재 존재하는 모든 pod 없앤다.

> 💡 kubectl delete all --all

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx-container
    image: nginx:1.18.0
    ports:
    - containerPort: 80
```

1. kubectl apply -f nginx.yaml

   현재 경로에 있는 yaml 파일로 pod를 만든다.

2. kubectl get po

   잘 만들어졌는지 확인한다 .

```
root@8a57a9152c43:/code/local/cwave-k8s/practice/exercise/2-1# kubectl get po
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod   1/1     Running   0          5m23s
```

3. kubectl port-forward nginx-pod 8080:80

   외부와 연결하기 위해 포트 포워딩을 해준다.

   IDE의 8080 PORT를 Worker 둘 중 하나의 port 80으로 연결시키는 것이다.

   연결되는 worker의 ip는 처음 ide를 만들 때 사용한 3-node-cluster.yaml 파일에서 지정한 podsubnet에 포함되는 ip 그 ip는 kubectl get po -o wide를 통해 확인 가능

4. kubectl get po nginx-pod -o yaml

   yaml 파일 상세정보 확인 가능하다..

```
root@8a57a9152c43:/code/local/cwave-k8s/practice/exercise/2-1# kubectl get po ngin
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"name":"nginx-p
  creationTimestamp: "2024-07-15T07:58:49Z"
  name: nginx-pod
  namespace: default
  resourceVersion: "10514"
  uid: 897af0f3-d7e5-4521-97a5-80b33b62fdad
spec:
  containers:
  - image: nginx:1.18.0
    imagePullPolicy: IfNotPresent
    name: nginx-container
    ports:
    - containerPort: 80
      protocol: TCP
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
```

```
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: kube-api-access-tbdcn
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  nodeName: cwave-cluster-worker2
  preemptionPolicy: PreemptLowerPriority
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: kube-api-access-tbdcn
    projected:
      defaultMode: 420
      sources:
      - serviceAccountToken:
          expirationSeconds: 3607
          path: token
      - configMap:
          items:
          - key: ca.crt
            path: ca.crt
          name: kube-root-ca.crt
      - downwardAPI:
          items:
          - fieldRef:
              apiVersion: v1
              fieldPath: metadata.namespace
            path: namespace
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2024-07-15T07:59:01Z"
    status: "True"
    type: PodReadyToStartContainers
  - lastProbeTime: null
    lastTransitionTime: "2024-07-15T07:58:49Z"
    status: "True"
```

```
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: "2024-07-15T07:59:01Z"
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: "2024-07-15T07:59:01Z"
    status: "True"
    type: ContainersReady
  - lastProbeTime: null
    lastTransitionTime: "2024-07-15T07:58:49Z"
    status: "True"
    type: PodScheduled
  containerStatuses:
  - containerID: containerd://b9b528594b49728c3c2466abdec44de566f6a85fb76a46355185
    image: docker.io/library/nginx:1.18.0
    imageID: docker.io/library/nginx@sha256:e90ac5331fe095cea01b121a3627174b2e33e0
    lastState: {}
    name: nginx-container
    ready: true
    restartCount: 0
    started: true
    state:
      running:
        startedAt: "2024-07-15T07:59:00Z"
  hostIP: 192.168.128.2
  hostIPs:
  - ip: 192.168.128.2
  phase: Running
  podIP: 10.110.2.3
  podIPs:
  - ip: 10.110.2.3
  qosClass: BestEffort
  startTime: "2024-07-15T07:58:49Z"
```


5. kubectl exec -it nginx-pod -- bash

   pod에 bash로 접속 가능하다.

```
conf.d  fastcgi_params  koi-utf  koi-win  mime.types  modules  nginx.conf  scgi_params  uwsgi_params  win-utf
root@nginx-pod:/# ls /etc/nginx/nginx.conf
/etc/nginx/nginx.conf
root@nginx-pod:/# cat /etc/nginx/nginx.conf

user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
```

6. curl http://localhost:8080

```
PS C:\kkk\local_code\cwave-k8s\practice\exercise\2-1> curl http://localhost:8080


StatusCode        : 200
StatusDescription : OK
Content           : <!DOCTYPE html>
                    <html>
                    <head>
                    <title>Welcome to nginx!</title>
                    <style>
                        body {
                            width: 35em;
                            margin: 0 auto;
                            font-family: Tahoma, Verdana, Arial, sans-serif;
                        }
                    </style>
                    <...
RawContent        : HTTP/1.1 200 OK
                    Connection: keep-alive
                    Accept-Ranges: bytes
                    Content-Length: 612
                    Content-Type: text/html
                    Date: Mon, 15 Jul 2024 08:10:44 GMT
                    ETag: "5e9efe7d-264"
                    Last-Modified: Tue, 21 Apr 2020 ...
Forms             : {}
Headers           : {[Connection, keep-alive], [Accept-Ranges, bytes], [Content-Length, 612], [Content-Type, text/html]...}
Images            : {}
```

7. kubectl delete pods nginx-pod

```
root@8a57a9152c43:/code/local/cwave-k8s/practice/exercise/2-1# kubectl get po
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod   1/1     Running   0          13m
root@8a57a9152c43:/code/local/cwave-k8s/practice/exercise/2-1# kubectl delete pods nginx-pod
pod "nginx-pod" deleted
root@8a57a9152c43:/code/local/cwave-k8s/practice/exercise/2-1# kubectl get po
No resources found in default namespace.
```

```
root@8a57a9152c43:/code/local/cwave-k8s/practice/exercise/2-1# kubectl logs nginx-pod
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
```

kubectl describe po nginx-pod

쿠버네티스 입장에서 pod를 설명해준다.

> 💡 logs 를 보거나 describe를 보면 웬만한 문제는 잡을 수 있다.

api 그룹

core에 해당하는 그룹은 그룹명을 안써도 된다.

alpha는 쓰지말라 삭제될 수 있다.
beta는 바뀔 수 있다.
stable은 영원히 괜찮다

API 그룹

API 그룹은 쿠버네티스 API를 더 쉽게 확장하게 해준다.

Group	Version
admissionregistration.k8s.io	v1, v1beta1
apiextensions.k8s.io	v1, v1beta1
apiregistration.k8s.io	v1, v1beta1
apps	v1
authentication.k8s.io	v1, v1beta1
authorization.k8s.io	v1, v1beta1
autoscaling	v1, v2beta2, v2beta1
batch	v1, v1beta1
certificates.k8s.io	v1, v1beta1
coordination.k8s.io	v1, v1beta1
core	v1
discovery.k8s.io	v1, v1beta1
events.k8s.io	v1, v1beta1
extensions	v1beta1
flowcontrol.apiserver.k8s.io	v1beta1

단, core API 의 경우는
그룹을 명시 하지 않아도 됨

Group	Version
internal.apiserver.k8s.io	v1alpha1
networking.k8s.io	v1, v1beta1
node.k8s.io	v1, v1beta1, v1alpha1
policy	v1, v1beta1
rbac.authorization.k8s.io	v1, v1beta1, v1alpha1
scheduling.k8s.io	v1, v1beta1, v1alpha1
storage.k8s.io	v1, v1beta1, v1alpha1

알파
(Alpha)
- 기본적으로 비활성화
- 활성화 하면 버그에 노출 될 수 있음
- 다음 릴리즈 에서 언제든지 삭제 되거나 변경 될 수 있음

(예: v1alpha1
)

베타
(Beta)
- 기본적으로 활성화 되어 있음. 코드테스트를 많이 했으며, 안전함
- 전반적인 기능에 대한 기술 지원이 중단되지 않음 (지속 가능함)
- 문법이나 사용 방식이 다음 릴리지 에서 바뀔 수 있음

(예: v2beta3
)

안정화
(Stable)
- 버전이름이 vX 와 같이 표기 한다. 여기서 X 는 정수 이다.

(예: v1
)

라벨 및 주석 기본

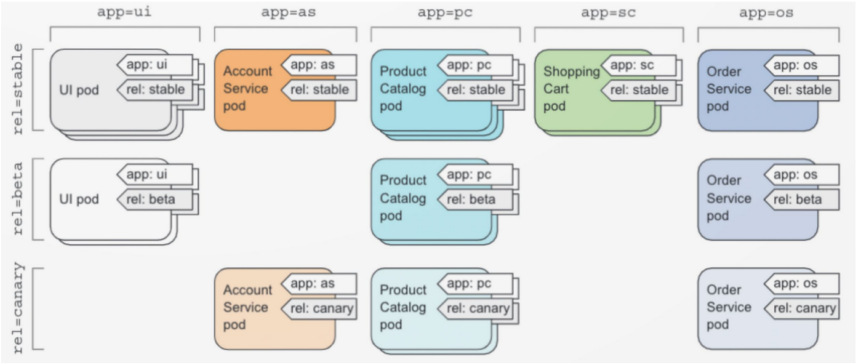
네모박스 하나가 pod

라벨을 붙이는 이유 대규모 서비스에는 pod의 수도 많고 pod 안에 컨테이너도 많기에 필터링이 필요함 라벨은 이런 필터링을 위함

통신 또한 라벨명으로 가능하다. ip 혹은 digest같은 넘버로 안해도 된다. 쿠버네티스가 알아서 한다.

라벨의 실제 활용

- app는 Pod가 속한 애플리케이션, 구성요소 또는 마이크로 서비스 지정
- rel은 Pod에서 실행 중인 애플리케이션의 안정적 버전, 베타 혹은 카나리 버전인지 여부
- app과 rel 을 추가 하면서 Pod 들을 2가지 차원으로 구성



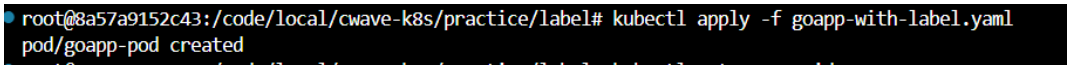
주석

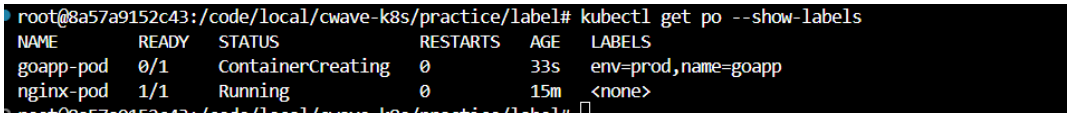
클라우드로 가면 주석을 라벨처럼 사용한다.

라벨하고 똑같지만 글자 크기가 다르다.

라벨 실습

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: goapp-pod

  labels:
    name: goapp
    env: prod

spec:
  containers:
  - name: goapp-container

    image: dangtong/goapp
    resources:
      limits:
        memory: "128Mi"
        cpu: "500m"
    ports:
      - containerPort: 8080
```





💡 kubectl get po --show-labels

pod의 라벨들이 출력이 된다.

```
root@8a57a9152c43:/code/local/cwave-k8s/practice/label# kubectl get po -L env
NAME        READY   STATUS              RESTARTS   AGE    ENV
goapp-pod   0/1     ContainerCreating   0          104s   prod
nginx-pod   1/1     Running             0          16m
root@8a57a9152c43:/code/local/cwave-k8s/practice/label# kubectl get po -L env -L name
NAME        READY   STATUS              RESTARTS   AGE    ENV    NAME
goapp-pod   0/1     ContainerCreating   0          112s   prod   goapp
nginx-pod   1/1     Running             0          16m
```

-L로 필터를 사용할 수 있다.

describe로도 확인 가능

```
root@8a57a9152c43:/code/local/cwave-k8s/practice/label# kubectl describe po goapp-pod
Name:             goapp-pod
Namespace:        default
Priority:         0
Service Account:  default
Node:             cwave-cluster-worker2/192.168.128.2
Start Time:       Mon, 15 Jul 2024 17:42:31 +0900
Labels:           app=application
                  env=prod
                  name=goapp
                  tier=backend
```

라벨로 검색

```
root@8a57a9152c43:/code/local/cwave-k8s/practice/label# kubectl get po -l name=goapp
NAME        READY   STATUS    RESTARTS   AGE
goapp-pod   1/1     Running   0          7m24s
```

라벨추가

```
root@8a57a9152c43:/code/local/cwave-k8s/practice/label# kubectl label po goapp-pod app="application" tier="backend"
pod/goapp-pod labeled
```

라벨삭제

```
root@8a57a9152c43:/code/local/cwave-k8s/practice/label# kubectl label po goapp-pod env-
pod/goapp-pod unlabeled
```

-만 붙이면 라벨이 삭제가 된다.