



<https://kubernetes.io/community/>

# **도커/쿠버네티스 온라인 부트캠프** **with 카카오엔터프라이즈**

**보안프로젝트 최일선 CTO**  
**isc0304@naver.com**

## 최 일 선

one4all

최 일 선 기술이사

llsun Choi

www.one4all.co.kr

Mobile 010-9891-5357 E-mail gasbugs21c@one4all.co.kr

경기도 부천시 범안로 221, 1층 120호 (옥길헤리움타운) 원포울 주식회사


















- 現 원포울, 보안프로젝트 최일선 CTO
- 現 멀티캠퍼스, 한국데이터진흥원 외부 전문 강사
- 행안부 생애주기별 콘텐츠 자문, 대한병원협회, 한컴MDS, 케이실드 주니어, 국방부 등 기관 및 기업 강의
- 정보보안, 리버싱, 파이썬 프로그래밍, IoT 분석, 데이터분석, AI, 쿠버네티스 외 다수 과목 교육
- 유튜브 재즐보프 운영
- “비박스를 활용한 웹 모의해킹 완벽실습(2016)”, “마인크래프트와 함께 즐겁게 파이썬(2018)”, “시스템 해킹 입문 프로토스타(2019)” 저자

YouTube



재즐보프  
구독자 1.2만명

 <b>AWS Certified Security – Specialty</b> Amazon Web Services Training and...	 <b>AWS Certified Solutions Architect – Professional</b> Amazon Web Services Training and...	 <b>AZ-300 Microsoft Azure Architect Technologies</b> Microsoft
 <b>AZ-301 Microsoft Azure Architect Design</b> Microsoft	 <b>AZ-400 Designing and Implementing Microsoft...</b> Microsoft	 <b>Cisco Certified Network Associate Routing and...</b> Cisco
 <b>CKA: Certified Kubernetes Administrator</b> The Linux Foundation	 <b>CKS: Certified Kubernetes Security Specialist</b> The Linux Foundation	 <b>Microsoft Certified: Azure Administrator Associate</b> Microsoft
 <b>Microsoft Certified: Azure Administrator Associate...</b> Microsoft	 <b>Microsoft Certified: Azure Security Engineer...</b> Microsoft	 <b>Microsoft Certified: Azure Solutions Architect Expert</b> Microsoft
 <b>Microsoft Certified: Azure Solutions Architect Expe...</b> Microsoft	 <b>Microsoft Certified: DevOps Engineer Expert</b> Microsoft	 <b>Microsoft Certified Trainer 2020-2021</b> Microsoft



# Table of Contents

- ▶▶ 클러스터 환경 구축과 리마인드
- ▶▶ 파드 컨테이너 디자인
- ▶▶ 쿠버네티스 저장소
- ▶▶ 컨테이너 롤링 업데이트 전략 이해
- ▶▶ 쿠버네티스 유저 관리
- ▶▶ 엘라스틱서치를 활용한 로그 수집
- ▶▶ 쿠버네티스 보안
- ▶▶ 리소스 로깅과 모니터링
- ▶▶ 쿠버네티스 CI/CD 구성
- ▶▶ 쿠버네티스 마이크로서비스 아키텍처 이해





# 클러스터 환경 구축과 리마인드

▶ 실습을 위한 클러스터 환경 재구성

▶ CKA, CKAD, CKS 시험 제도와 합격 공부 가이드



# 실습을 위한 클러스터 환경 재구성

# 실습을 위한 클러스터 환경 재구성

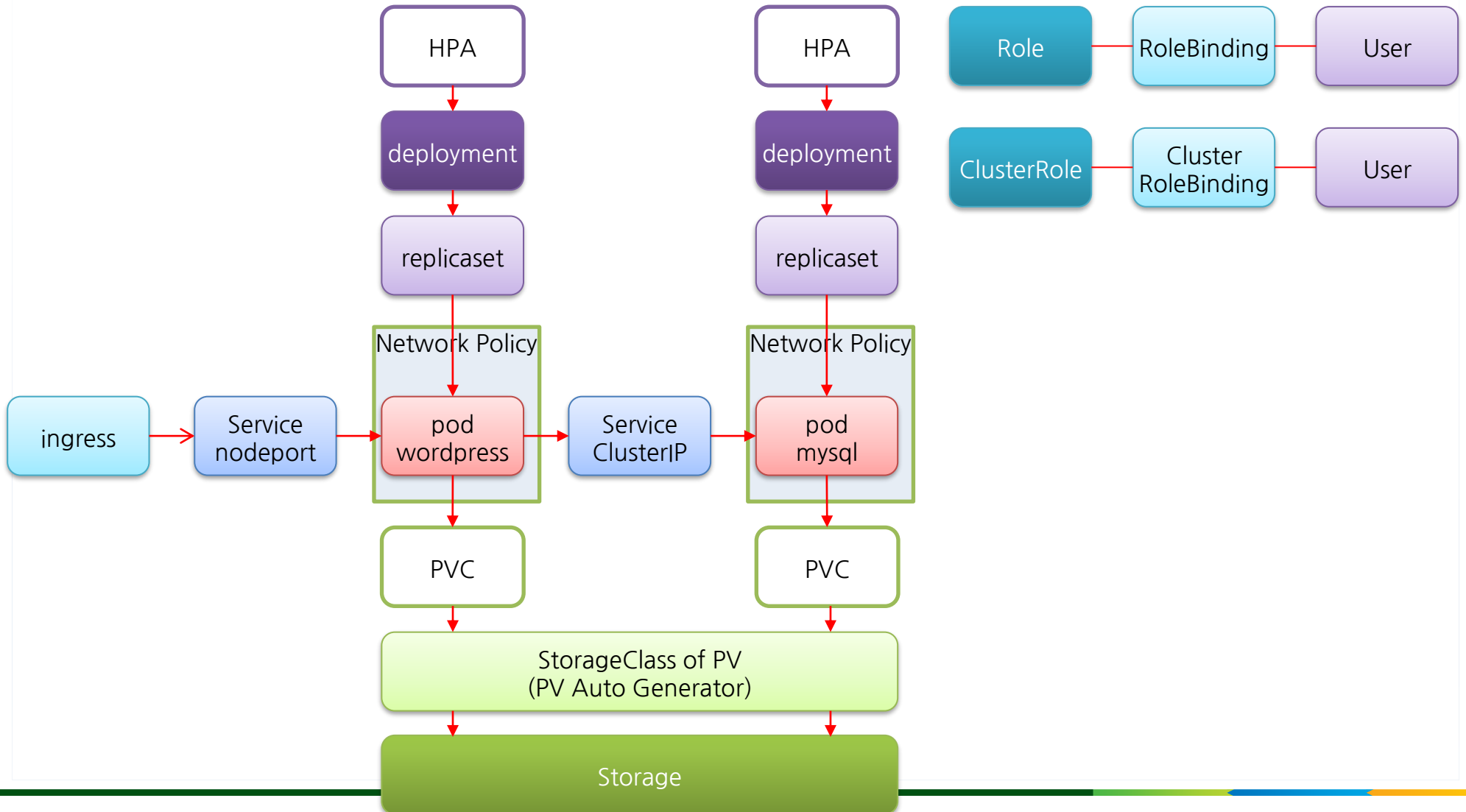
## ▶ (연습문제) 쿠버네티스 설치 필요 사항

- 인스턴스 4대를 사용해 구성
- 마스터 노드 1대 : 쿠버네티스의 마스터 노드가 설정될 호스트
- 워커 노드 3대: 클러스터에 컨테이너를 띄울 Work 노드 추가
- 가상머신은 2CPU, 4GB 메모리 사용 필요

The screenshot shows the Google Cloud Platform console for creating a new VM instance. The 'Compute Engine' section is active, and the 'VM 인스턴스' (VM Instance) tab is selected. A red box highlights the '+ 인스턴스 만들기' (Create Instance) button. Below this, the '부팅 디스크' (Boot Disk) section is expanded, showing the '공개 이미지' (Public Image) tab. A red box highlights the 'Ubuntu' operating system and 'Ubuntu 20.04 LTS' version selection. Another red box highlights the '부팅 디스크 유형' (Boot Disk Type) dropdown, which is set to '균형 있는 영구 디스크' (Balanced Persistent Disk), and the '크기(GB)' (Size in GB) field, which is set to '100'. A red arrow points from the '변경' (Change) button in the '부팅 디스크' section to the '공개 이미지' tab. The '부팅 디스크' section also shows a '새로운 10GB의 균형 있는 영구 디스크 이미지' (New 10GB balanced persistent disk image) option with a '변경' (Change) button.

# 실습을 위한 클러스터 환경 재구성

## ▶ (연습문제) 쿠버네티스 워드프레스 환경 구성하기





# 쿠버네티스 인증 시험 제도와 합격 공부 가이드 - CKA, CKAD, CKS

# 쿠버네티스 인증 시험 제도

## ■ Certified Kubernetes Administrator (CKA) 프로그램

- <https://www.cncf.io/certification/cka/>
- 리눅스 재단과 협력하여 쿠버네티스 생태계 개발을 돕기 위해 CNC(Cloud Native Computing Foundation) 재단에 의해 시작
- CKA/CKAD/CKS 등의 직무 별 다양한 시험 제도 시행
- CNC 재단은 쿠버네티스 관리자 커뮤니티를 성장시키기 위해 노력
- 쿠버네티스를 사용하는 광범위한 회사 및 조직에서 지속적으로 성장을 도모
- 100% 핸즈온(실습) 방식
- \$375의 비용으로 온라인 시험을 치룸 (쿠폰을 검색하면 나오니 쿠폰을 사용합시다!)
- WebCam을 통해 수험자를 모니터링하고 크롬으로 시험을 진행
- 재시험 1회 무료로 응시 가능
- 관련된 사이트를 접속할 수 있도록 하는 오픈 테스트



# 쿠버네티스 인증 시험 제도

## CKA, CKAD, CKS 난이도와 준비 기간

- (지극히 주관적인 강사 개인의 생각)

구분	CKA (Administrator)	CKAD (Application Developer)	CKS (Security Specialist)
난이도	중간	쉬움	어려움
준비 기간 (인프런에 있는 필자의 강의를 잘 수강한 후에 추가로 필요한 "주관적인 추천" 준비 기간이다.)	2주	2일	4주
시험 범위	넓음, 클러스터 쿠버네티스 활용과 클러스터 유지 보수, 트러블 슈팅 등	중간, 클러스터 쿠버네티스 기본 활용 기능	넓음, 클러스터 쿠버네티스의 보안 관련 기본 기능과 추가로 설치하면 좋은 다양한 애드온 기능들

# 쿠버네티스 인증 시험 제도

## ▶ Certified Kubernetes Administrator (CKA) 출제 범위

- Cluster Architecture, Installation & Configuration 25%
- Workloads & Scheduling 15%
- Services & Networking 20%
- Storage 10%
- Troubleshooting 30%



# 쿠버네티스 인증 시험 제도

## ▶ Certified Kubernetes Application Developer (CKAD) 출제 범위

- Core Concepts - 13%
- Configuration - 18%
- Multi-Container Pods - 10%
- Observability - 18%
- Pod Design - 20%
- Services & Networking - 13%
- State Persistence - 8%



# 쿠버네티스 인증 시험 제도

## ▶ Certified Kubernetes Security Specialist (CKS) 출제 범위

- Cluster Setup - 10%
- Cluster Hardening - 15%
- System Hardening - 15%
- Minimize Microservice Vulnerabilities - 20%
- Supply Chain Security - 20%
- Monitoring, Logging and Runtime Security - 20%





# 쿠버네티스 인증 시험 제도

## 기본적으로 알아둬야 하는 편의 옵션 두 가지

- `--dry-run=client`

- 이 옵션을 추가하고 명령을 실행하면 리소스를 생성하지 않음
- 단순히 명령을 테스트하려는 경우에 사용
- 사용자의 명령어가 올바른지 알려줌

- `-o yaml`

- 화면에 YAML 형식의 리소스 정의를 출력

# 쿠버네티스 인증 시험 제도

## 쿠버네티스 치트시트

- 치트 시트 문서에서 다양한 정보 확인 가능
- 필수 환경 적용: kubectl 자동 완성 명령어
- <https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

 **kubernetes**

문서 쿠버네티스 블로그 교육 파트너 커뮤니티 사례 연구 버전 한국어 Korean

검색하기

홈  
시작하기  
개념  
태스크  
튜토리얼  
레퍼런스  
    kubectl  
        kubectl 개요  
        JSONPath 지원  
        kubectl  
        kubectl Commands  
        kubectl 사용 규칙

쿠버네티스 문서 / 레퍼런스 / kubectl / kubectl 치트 시트

## kubectl 치트 시트

이 페이지는 일반적으로 사용하는 `kubectl` 커맨드와 플러그인에 대한 목록을 포함한다.

### Kubectl 자동 완성

#### BASH

```
source <(kubectl completion bash) # bash-completion 패키지를 먼저 설치한 후, bash의 자동 완성을  
echo "source <(kubectl completion bash)" >> ~/.bashrc # 자동 완성을 bash 셸에 영구적으로 추가함
```

[페이지 편집](#)  
[하부 페이지 생성](#)  
[이슈 생성](#)  
[전체 섹션 프린트](#)  
  
Kubectl 자동 완성  
    BASH  
    ZSH  
Kubectl 컨텍스트와 설정  
Kubectl apply  
오브젝트 생성  
리소스 조회 및 찾기  
리소스 업데이트

# 쿠버네티스 인증 시험 제도

## 합격 공부 가이드 - CKA, CKAD, CKS

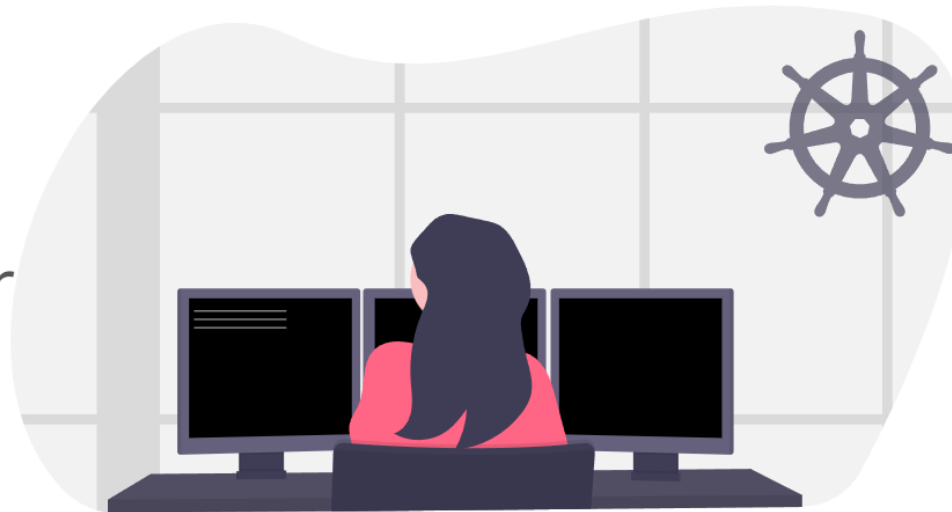
- killer shell 적극 이용하기
- 두 번의 무료 세션 제공 (36시간 \* 2번)
- 시험과 유사한 난이도
- 시험 합격 점수는 65~70점 정도이기 때문에 다 맞을 필요는 없다!



CKS CKA  
CKAD  
Simulator

Get Started

<https://killer.sh/>



# 파드 컨테이너 디자인

- ▶▶ 멀티 컨테이너 파드
- ▶▶ 라이브네스와 레디네스, 스타트업프로브
- ▶▶ 사이드카 컨테이너
- ▶▶ 어댑터 컨테이너
- ▶▶ 앰베서더 컨테이너
- ▶▶ 초기화 컨테이너
- ▶▶ job과 cronjob
- ▶▶ 시스템 리소스 요구사항과 제한 설정



# 멀티 컨테이너

# 멀티 컨테이너

## ▶ 하나의 파드에 다수의 컨테이너를 사용

- 하나의 파드를 사용하는 경우 같은 네트워크 인터페이스와 IPC, 볼륨 등을 공유
- 이 파드는 효율적으로 통신하여 데이터의 지역성을 보장하고 여러 개의 응용프로그램이 결합된 형태로 하나의 파드를 구성할 수 있음

```
$ kubectl exec -it two-containers -- cat  
/usr/share/nginx/html/index.html  
Defaulting container name to nginx-  
container.
```

Use 'kubectl describe pod/two-containers -n default' to see all of the containers in this pod.

Hello from the debian container

pod-mutil-continaer.yaml

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: two-containers  
spec:  
  
  restartPolicy: Never  
  
  volumes:  
  - name: shared-data  
    emptyDir: {}  
  
  containers:  
  
  - name: nginx-container  
    image: nginx  
    volumeMounts:  
    - name: shared-data  
      mountPath: /usr/share/nginx/html  
  
  - name: debian-container  
    image: debian  
    volumeMounts:  
    - name: shared-data  
      mountPath: /pod-data  
    command: ["/bin/sh"]  
    args: ["-c", "echo Hello from the debian container > /pod-data/index.html"]
```



# 멀티 컨테이너

## ▶▶ 연습문제

- 하나의 파드에서 nginx와 redis 이미지를 모두 실행하는 yaml을 만들고 실행하라.

# 라이브네스와 레디네스, 스타트업프로브

# 라이브네스와 레디네스, 스타트업프로브

## ▶ 라이브니스, 레디네스 스타트업 프로브 구성

### ● Liveness Probe

- 컨테이너 살았는지 판단하고 다시 시작하는 기능
- 컨테이너의 상태를 스스로 판단하여 교착 상태에 빠진 컨테이너를 재시작
- 버그가 생겨도 높은 가용성을 보임

### ● Readiness Probe

- 파드가 준비된 상태에 있는지 확인하고 정상 서비스를 시작하는 기능
- 파드가 적절하게 준비되지 않은 경우 로드밸런싱을 하지 않음

### ● Startup Probe

- 애플리케이션의 시작 시기 확인하여 가용성을 높이는 기능
- Liveness와 Readiness의 기능을 비활성화

# 라이브네스와 레디네스, 스타트업프로브

## ▶ Liveness 커맨드 설정 - 파일 존재 여부 확인

- 리눅스 환경 에서커맨드 실행 성공 시 0 (컨테이너 유지)
- 실패하면 그 외 값 출력 (컨테이너 재시작)

exec-liveness.yaml

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
    name: liveness-exec
spec:
  containers:
    - name: liveness
      image: k8s.gcr.io/busybox
      args:
        - /bin/sh
        - -c
        - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
      livenessProbe:
        exec:
          command:
            - cat
            - /tmp/healthy
        initialDelaySeconds: 5
        periodSeconds: 5
```

소스코드 출처:

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>

# 라이브네스와 레디네스, 스타트업프로브

## ▶ Liveness 웹 설정 - http 요청 확인

- 서버 응답 코드가 200이상 400미만 (컨테이너 유지)
- 서버 응답 코드가 그 외 (컨테이너 재시작)

소스코드 출처:

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

http-liveness.yaml

# 라이브네스와 레디네스, 스타트업프로브

## ▶▶ Readiness TCP 설정

- 준비 프로브는 8080포트를 검사
- 5초 후부터 검사 시작
- 검사 주기는 10초
- → 서비스를 시작해도 된다!

## ▶▶ Liveness TCP 설정

- 활성화 프로브는 8080포트를 검사
- 15초 후부터 검사 시작
- 검사 주기는 20초
- → 컨테이너를 재시작하지 않아도 된다!

소스코드 출처:

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>

```
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20
```

tcp-liveness-readiness.yaml



# 라이브네스와 레디네스, 스타트업프로브

## ▶▶ Statup Probe

- 시작할 때까지 검사를 수행
- http 요청을 통해 검사
- 30번을 검사하며 10초 간격으로 수행
- 300(30\*10)초 후에도 파드가 정상 동작하지 않는 경우
- → 300초 동안 파드가 정상 실행되는 시간을 벌어줌

Startup Probe 예제

```
ports:
- name: liveness-port
  containerPort: 8080
  hostPort: 8080

livenessProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 1
  periodSeconds: 10

startupProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 30
  periodSeconds: 10
```

소스코드 출처:

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>

# 사이드카 컨테이너

# 사이드카 컨테이너

## 사이드카 컨테이너란?

- 오토바이의 사이드카에서 유래
- 이륜차에 기존 기능을 향상/확장하는데 사용 (여기서는 파드의 기능을 향상)
- 파드의 파일시스템을 공유하는 형태

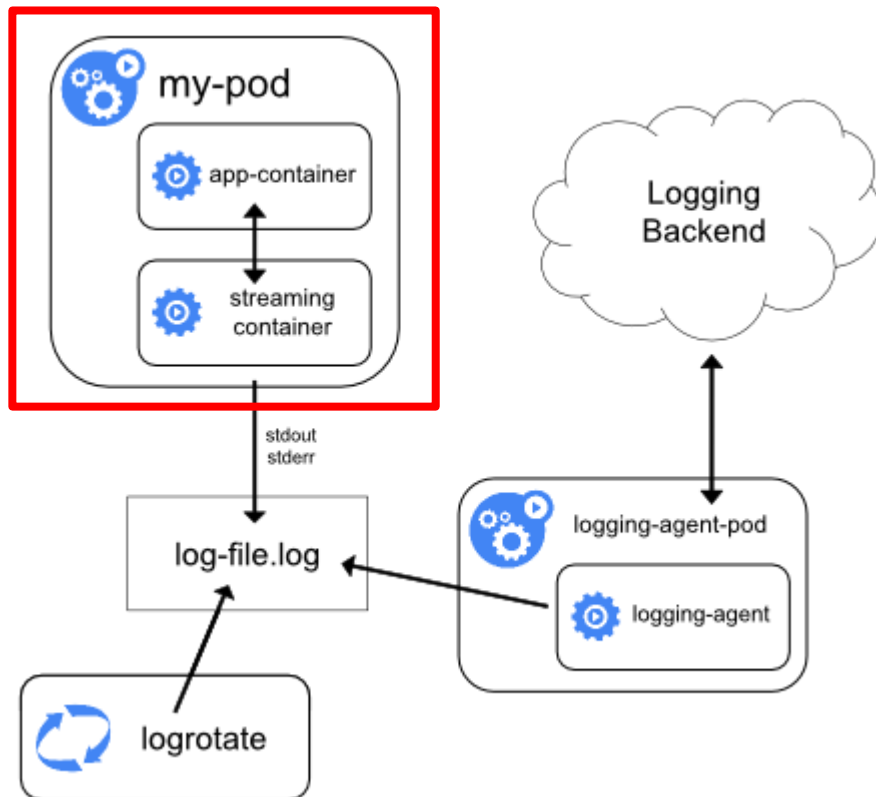
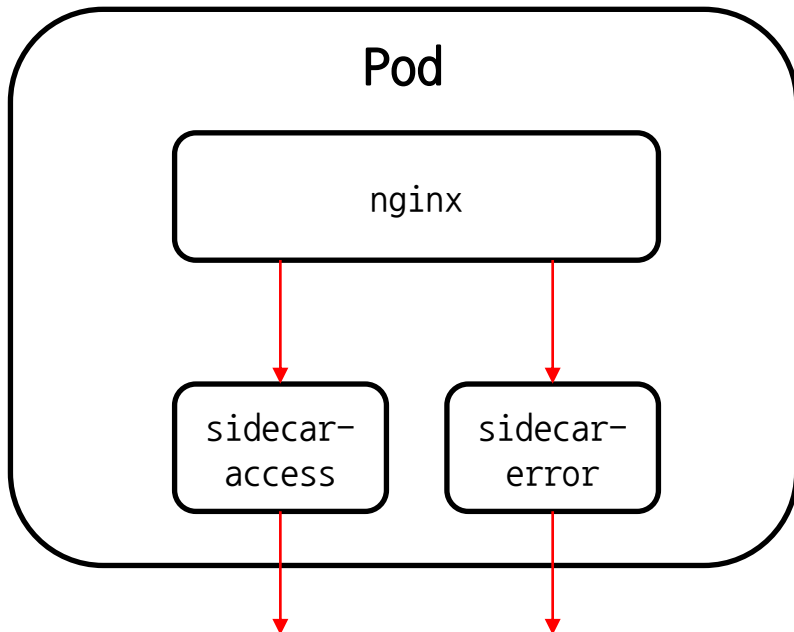


그림 출처: <https://kubernetes.io/ko/docs/concepts/administration/logging/>

# 사이드카 컨테이너

## 사이드카 컨테이너 생성 실습

- 접속 후 다음 명령을 통해서 로그 확인 가능
  - `kubectl logs nginx-sidecar sidecar-access`



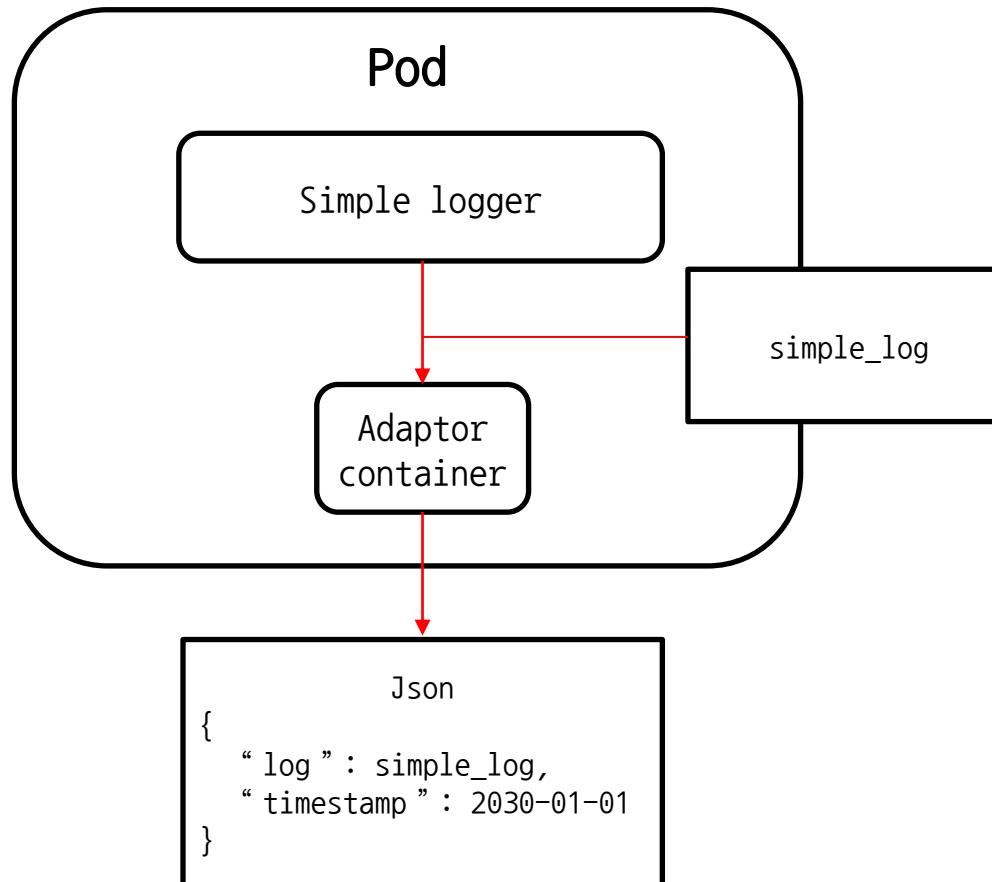
```
# nginx-sidecar.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-sidecar
spec:
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - name: varlognginx
          mountPath: /var/log/nginx
    - name: sidecar-access
      image: busybox
      args: [/bin/sh, -c, 'tail -n+1 -f /var/log/nginx/access.log']
      volumeMounts:
        - name: varlognginx
          mountPath: /var/log/nginx
    - name: sidecar-error
      image: busybox
      args: [/bin/sh, -c, 'tail -n+1 -f /var/log/nginx/error.log']
      volumeMounts:
        - name: varlognginx
          mountPath: /var/log/nginx
  volumes:
    - name: varlognginx
      emptyDir: {}
```

# 어댑터 컨테이너

# 어댑터 컨테이너

## ▶▶ 어댑터 컨테이너란?

- 본질적으로 이질적인 응용프로그램을 적용 가능하도록 개선하는 컨테이너
- 원본 컨테이너에 대한 변경사항 없이 현재 컨테이너 기능을 시스템에 적용시키는 기능





# 어댑터 컨테이너

## 어댑터 컨테이너 소스 예제

- 예제 코드 분석

➤ <https://github.com/bbachi/k8s-adaptor-container-pattern.git>

```
containers:
- image: busybox
  command: ["/bin/sh"]
  args: ["-c", "while true; do echo $(date -u) '#This is log' >> /var/log/file.log; sleep 5;done"]
  name: main-container
  resources: {}
  volumeMounts:
  - name: var-logs
    mountPath: /var/log
```

Main Container

```
app.get('/logs', (req,res) => {
  eachLine('/var/log/file.log', function(line) {
    console.log(line);
    logs.push({
      time: line.split("#")[0],
      message: line.split("#")[1]
    });
  }).then(function() {
    console.log("I'm done!!");
    res.send(logs);
  }).then(function(err){
    console.log(err);
  });
});
```

Adaptor Container

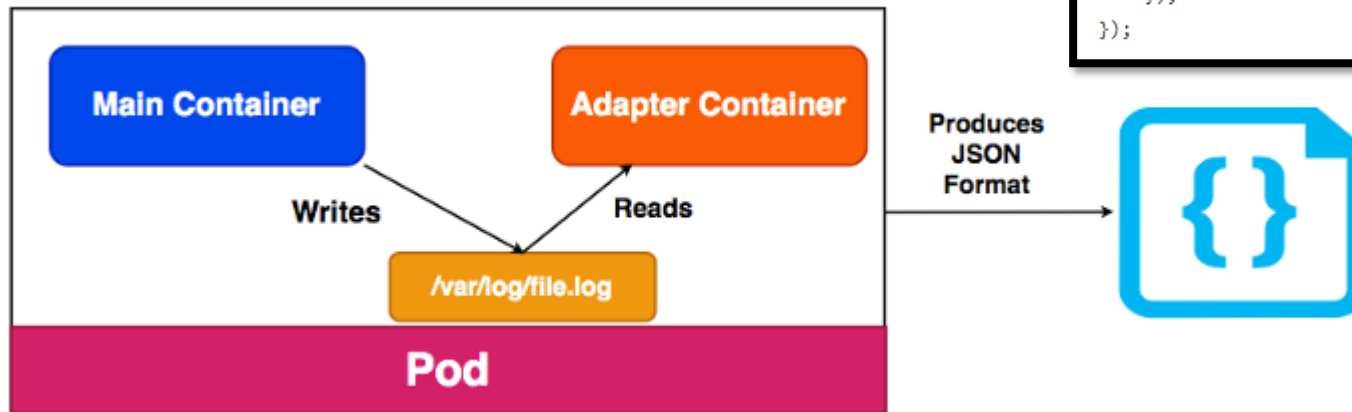


그림 출처: <https://medium.com/bb-tutorials-and-thoughts/kubernetes-learn-adaptor-container-pattern-97674285983c>

# 어댑터 컨테이너

## 어댑터 컨테이너 생성 실습

- 다음 명령을 실행해 클러스터에 배포

```
kubectl apply -f https://raw.githubusercontent.com/bbachi/k8s-adaptor-container-pattern/master/pod.yml
```

- 데이터 요청

```
$ kubectl get pod -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
adapter-container-demo	2/2	Running	0	2m31s	10.244.1.3	node01	<none>	<none>

```
$ curl 10.244.1.3:3080/logs -s > test.txt
```

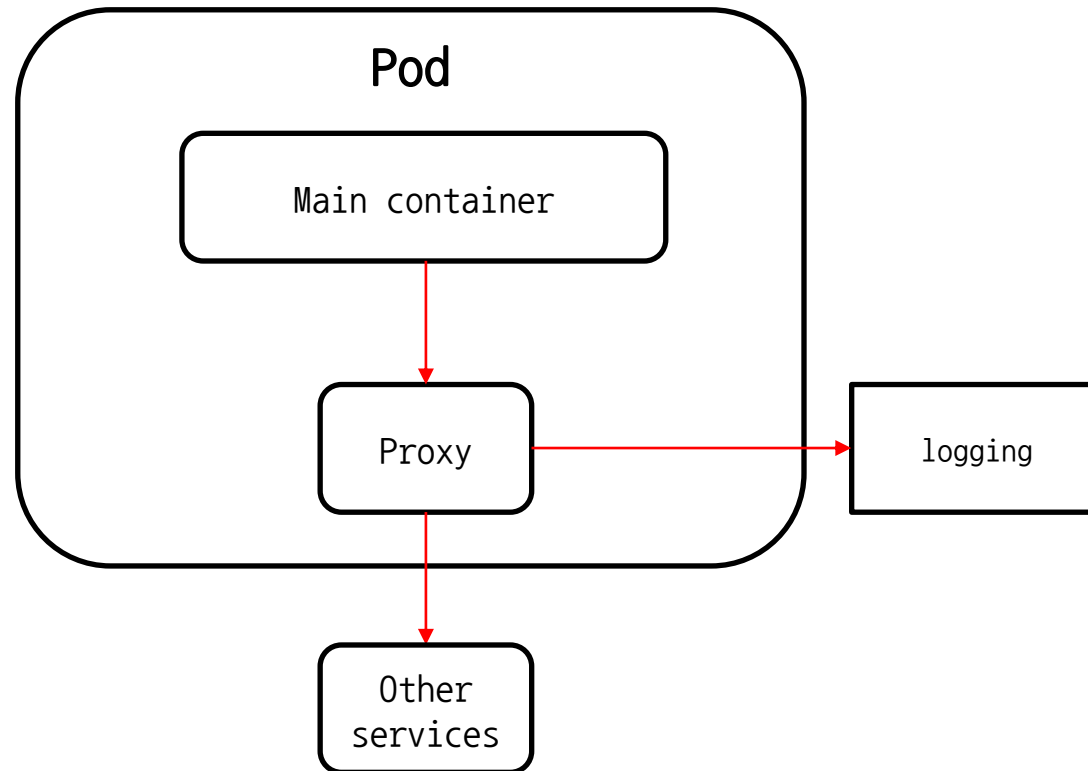
```
{
  - {
    time: "Sun Sep 13 03:46:33 UTC 2020",
    message: "This is log"
  },
  - {
    time: "Sun Sep 13 03:46:38 UTC 2020",
    message: "This is log"
  },
  - {
    time: "Sun Sep 13 03:46:43 UTC 2020",
    message: "This is log"
  },
  - {
    time: "Sun Sep 13 03:46:48 UTC 2020",
    message: "This is log"
  },
}
```

# 앰배서더 컨테이너

# 앰배서더 컨테이너

## ▶ 앰배서더 컨테이너란?

- 앰버서더(ambassador)의 뜻은 국가공무원으로 외교를 대표하는 대사를 의미
- 앰배서더 컨테이너는 파드 외부의 서비스에 대한 액세스를 간소화하는 특수 유형
- 파드에 앰배서더 컨테이너를 배치하여 통신을 대신해주는 역할을 소화
- 서비스의 인증, 데이터의 변조, 감시 등의 다양한 작업이 가능



# 앰배서더 컨테이너

## 앰배서더 컨테이너 소스 예제

### 예제 코드 분석

➤ <https://github.com/bbachi/k8s-ambassador-container-pattern.git>

```
const express = require("express");
const app = express();
const port = 9000;
var rp = require('request-promise');

var options = {
  method: 'GET',
  uri: 'http://localhost:3000'
}

app.get("/", (req, res) => {
  rp(options).then(function (body) {
    res.json(JSON.parse(body))
  }).catch(function (err) {
    console.log(err);
  });
});
```

**Main Container**

```
http {
  server {
    listen 3000;
    location / {
      proxy_pass http://api.mocki.io/v1/b043df5a;
    }
  }
}
```

**Ambassador**

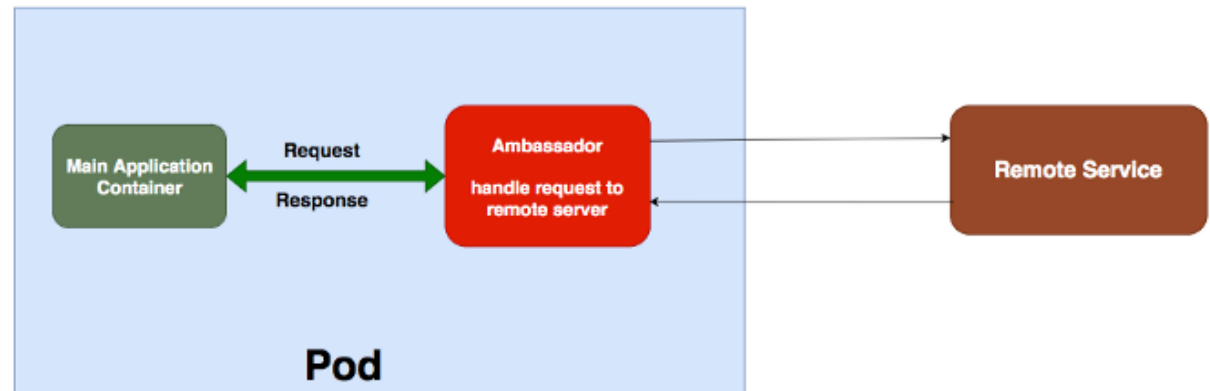


그림 출처: <https://medium.com/bb-tutorials-and-thoughts/kubernetes-learn-ambassador-container-pattern-bc2e1331bd3a>

# 앰배서더 컨테이너

## 앰배서더 컨테이너 생성 실습

- 다음 명령을 실행해 클러스터에 배포

```
kubectl apply -f https://raw.githubusercontent.com/bbachi/k8s-ambassador-container-pattern/master/pod.yml
```

- 데이터 요청

```
$ kubectl exec -it ambassador-container-demo -c ambassador-container -- /bin/sh
```

```
# curl localhost:9000
```

<현재는 403으로 정상적으로 통신 불가>

- 로그에서 통신 정보 확인

- (오류가 뜨지만 통신 되는 것 확인 가능)

```
kubectl logs ambassador-container-demo main-container
```

```
Terminal Host 1 +
httpModule: [Object],
agentClass: [Function: Agent],
href: 'https://api.mocki.io/v1/b043df5a',
ntick: true,
response: [Circular *1],
originalHost: 'api.mocki.io',
originalHostHeaderName: 'host',
agent: [Agent],
_started: true,
req: [ClientRequest],
responseContent: [Circular *1],
_destdata: true,
_ended: true,
_callbackCalled: true,
[Symbol(kCapture)]: false
},
toJSON: [Function: responseToJSON],
caseless: Caseless { dict: [Object] },
body: '{"message":"Forbidden"}',
[Symbol(kCapture)]: false
}
}
```

# 초기화 컨테이너

# 초기화 컨테이너

## init 컨테이너의 특징

- 파드 컨테이너 실행 전에 초기화 역할을 하는 컨테이너
- 완전히 초기화가 진행된 다음에야 주 컨테이너를 실행
- Init 컨테이너가 실패하면, 성공할때까지 파드를 반복해서 재시작
- restartPolicy에 Never를 하면 재시작하지 않음





# 초기화 컨테이너

## ▶▶ init 컨테이너의 특징

- 이 yaml은 mydb와 myservice가 탐지될 때까지 init 컨테이너가 멈추지 않고 돌아감

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox:1.28
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: busybox:1.28
    command: ['sh', '-c', 'until nslookup myservice; do echo waiting for myservice; sleep 2; done;']
  - name: init-mydb
    image: busybox:1.28
    command: ['sh', '-c', 'until nslookup mydb; do echo waiting for mydb; sleep 2; done;']
```

pod-init-container.yaml

# 초기화 컨테이너

## init 컨테이너의 특징

- init 프로세스를 끝낼 수 있는 종결자 등장!

```
svc-pod-mysql.yaml

apiVersion: v1
kind: Service
metadata:
  name: myservice
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
---
apiVersion: v1
kind: Service
metadata:
  name: mydb
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9377
```

# 초기화 컨테이너

## ▶▶ 함께하기

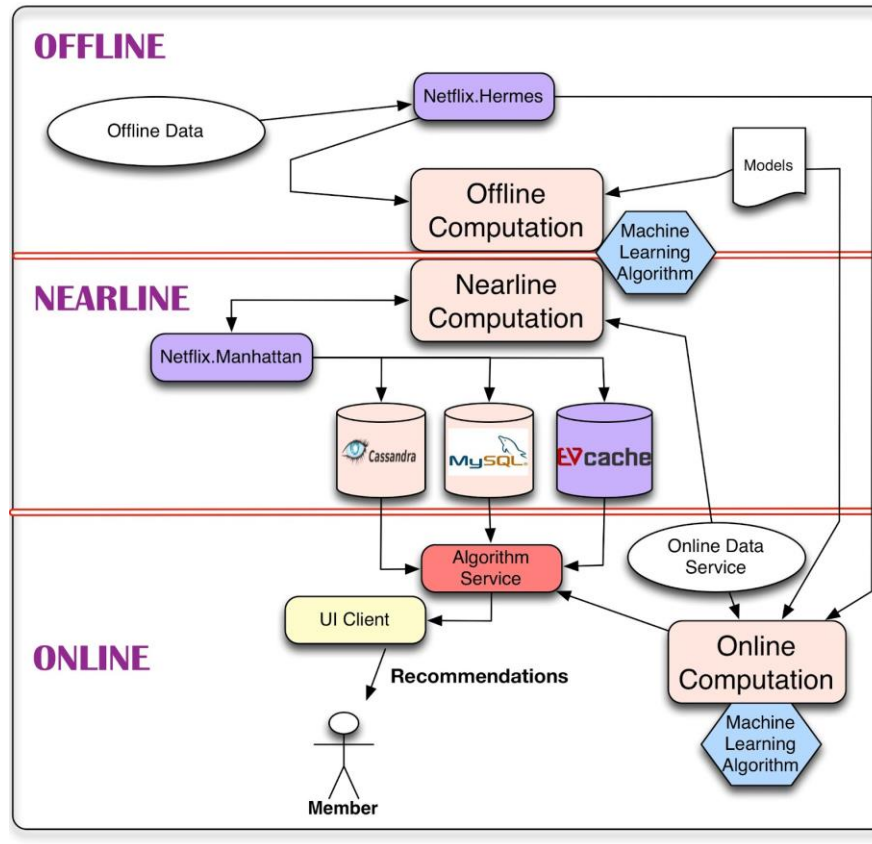
- pod-init-container.yaml를 작성하여 리소스를 생성하라.
- my-app-pod 파드에서 주 컨테이너가 실행되지 않는 현상을 관찰하라.
- 주 컨테이너가 실행되지 않는 이유는 무엇인가?
- svc-pod-mydb.yaml을 작성 및 실행하고 주 컨테이너의 반응을 관찰하라.
- 주 컨테이너가 정상적으로 실행되었는가? 그렇다면 그 이유는 무엇인가?

# job과 cronjob

# job과 cronjob

## Job과 CronJob의 필요성

- 쿠버네티스 클러스터를 운영할 때 일정 주기마다 돌아가야 하는 작업들 존재
- 넷플릭스가 2013년에 공개한 아키텍처



개인에게 맞춤 추천 시스템을 사용하기에는 많은 리소스가 필요하기 때문에 넷플릭스의 추천 시스템은 실시간(OnLine) 분석, 준 실시간(Nearline) 분석, 오프라인(Offline) 분석 나눠서 사용

System Architectures for Personalization and Recommendation  
그림 출처: 넷플릭스

# job과 cronjob

## Job이란?

- 하나 이상의 파드를 만들고 지정된 수의 파드가 성공적으로 종료될 때까지 Pods실행을 계속 재시도
- 작업을 삭제하면 생성한 파드가 정리
- 작업을 일시 중단하면 작업이 다시 다시 재개될 때까지 활성 파드가 삭제
- 작업을 사용하여 여러 파드를 병렬로 실행 가능

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
        restartPolicy: Never
      backoffLimit: 4
```

```
kubectl apply -f https://kubernetes.io/examples/controllers/job.yaml
```

# job과 cronjob

## ▶▶ 잡에 대한 병렬 실행 방법

- 비 병렬 잡: 기본값, 일반적으로 파드가 하나만 실행되고 파드가 종료되면 Job이 완료됨
- 정해진 횟수를 반복하는 잡: .spec.completions에 0이 아닌 양수를 지정하면 설정하면 정해진 횟수까지 파드가 반복적으로 실행
- 병렬 실행 가능 수를 지정: .spec.parallelism에 0이 아닌 양수를 지정하면 정해진 개수만큼 파드가 동시에 실행이 가능

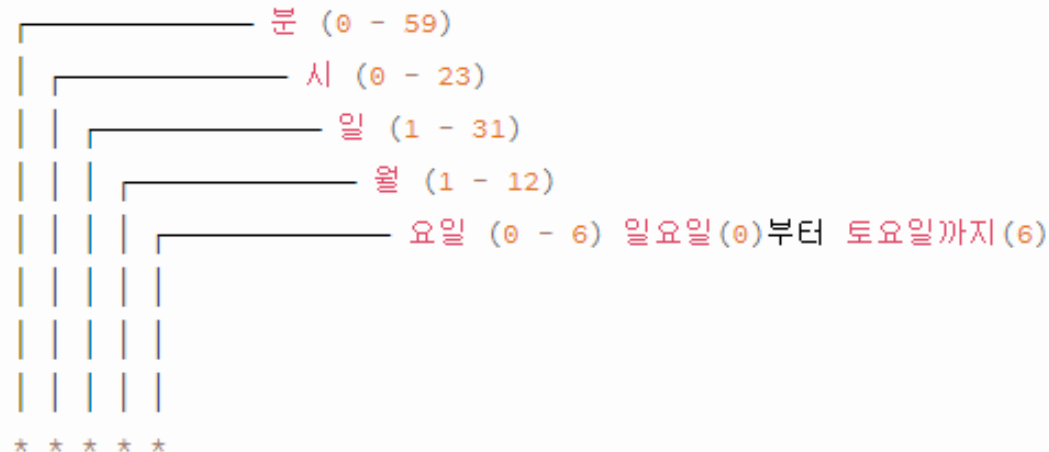
```
cat <<EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: pi-parallelism
spec:
  completions: 5 # 목표 완료 파드 개수
  parallelism: 2 # 동시 실행 가능 파드 개수
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
        restartPolicy: Never
      backoffLimit: 4
EOF
```

## ▶▶ Cronjob 예약 시간 정하기

### ● 예약 시간 작성 요령

- 기존 리눅스 시스템의 크론에서 표기하는 방법과 동일
- CronJob yaml 파일에는 예약 실행할 시간과 실행할 컨테이너를 작성
- 일반적으로 CronJob 하나에 하나의 작업 실행 권장
- 각 다음 내용을 의미하며 숫자로 표기

출처: <https://en.wikipedia.org/wiki/Cron>





# job과 cronjob

## 동시성 정책 설정하기

- spec.concurrencyPolicy는 동시성 정책 설정
- 이미 하나의 크론잡이 실행 중인 경우 크론잡을 추가로 실행할지 결정

정책	의미
Allow	중복 실행을 허용 (기본값)
Forbid	중복 실행을 금지
Replace	현재 실행중인 크론잡을 내리고 새로운 크론잡으로 대체

# job과 cronjob

## CronJob 예제 실행하기

### ● CronJob 간단한 예제

```
# cronjob-1.yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello-1
spec:
  concurrencyPolicy: Allow
  schedule: "*/* * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: busybox
              args:
                - /bin/sh
                - -c
                - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

	내용	설명
예약 실행 시간	*/* * * * *	매분 컨테이너를 실행합니다.
컨테이너 이미지	image: busybox	busybox 이미지를 사용합니다.
커맨드와 아규먼트	args: - /bin/sh - -c - date; echo Hello from the Kubernetes cluster	매분마다 date 명령과 Hello from the Kubernetes cluster 출력합니 다.
concurrencyPolicy	Allow	동시 실행 가능

# job과 cronjob

## ▶▶ CronJob 예제 실행하기

- kubectl 명령을 사용해 cronjob.yaml 파일을 실행하고 관찰
- 1분마다 한번씩 새로운 Pod가 실행되는 모습 확인

```
$ kubectl create -f cronjob-1.yaml
cronjob.batch/hello-1 created
```

```
$ kubectl get pod -w
```

NAME	READY	STATUS	RESTARTS	AGE
hello-1-1585921440-2ndjx	0/1	Pending	0	0s
hello-1-1585921440-2ndjx	0/1	Pending	0	0s
hello-1-1585921440-2ndjx	0/1	ContainerCreating	0	0s
hello-1-1585921440-2ndjx	0/1	Completed	0	4s
hello-1-1585921500-fs6g7	0/1	Pending	0	0s
hello-1-1585921500-fs6g7	0/1	Pending	0	0s
hello-1-1585921500-fs6g7	0/1	ContainerCreating	0	0s

# job과 cronjob

## ▶ CronJob의 Replace 기능 확인하기

- kubectl 명령을 사용해 cronjob.yaml 파일을 실행하고 관찰
- 1분마다 한번씩 새로운 Pod가 실행되는 모습 확인

```
# cronjob-3.yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello-3
spec:
  concurrencyPolicy: Replace
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: busybox
              args:
                - /bin/sh
                - -c
                - date; echo Hello from the Kubernetes cluster; sleep 100
          restartPolicy: OnFailure
```

러닝상태에 들어가기 직전에 앞서 실행하던 파드를 터미네이팅 시키고 자기가 실행되는 모습을 관찰

# 시스템 리소스 요구사항과 제한 설정

# 시스템 리소스 요구사항과 제한 설정

## 컨테이너에서 리소스 요구사항

- CPU와 메모리는 집합적으로 컴퓨팅 리소스 또는 리소스로 부름
- CPU 및 메모리 는 각각 자원 유형을 지니며 자원 유형에는 기본 단위를 사용
- 리소스 요청 설정 방법
  - `spec.containers[].resources.requests.cpu`
  - `spec.containers[].resources.requests.memory`
- 리소스 제한 설정 방법
  - `spec.containers[].resources.limits.cpu`
  - `spec.containers[].resources.limits.memory`

# 시스템 리소스 요구사항과 제한 설정

## 컨테이너에서 리소스 요구사항

- CPU는 코어 단위로 지정되며 메모리는 바이트 단위로 지정

자원 유형	단위
CPU	m(millicpu),
Memory	…… Ti, Gi, Mi, Ki, T, G, M, K

- ※ CPU 0.5가 있는 컨테이너는 CPU 1 개를 요구하는 절반의 CPU
- ※ CPU 0.1은 100m과 동일한 기능
- ※ K, M, G의 단위는 1000씩 증가
- ※ Ki, Mi, Gi의 단위는 1024씩 증가

### ● 환경에 따른 CPU의 의미

- 1 AWS vCPU
- 1 GCP 코어
- 1 Azure vCore
- 1 IBM vCPU
- 1 하이퍼 스레딩 기능이 있는 베어 메탈 인텔 프로세서의 하이퍼 스레드

# 시스템 리소스 요구사항과 제한 설정

## ▶ 컨테이너에서 리소스 요구사항 yaml 작성 요령

각각의 컨테이너마다  
리소스 작성

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

pod-resource-frontend.yaml



# 시스템 리소스 요구사항과 제한 설정

## 연습문제

- 다음 요구사항에 맞는 deploy를 구현하라.

- Deploy name: nginx
- Image: nginx
- 최소 요구사항
  - ✓ CPU: 1m
  - ✓ 메모리: 200Mi
- 리소스 제한
  - ✓ CPU: 2m
  - ✓ 메모리: 400Mi
- Port: 80

# 시스템 리소스 요구사항과 제한 설정

## limitRanges

- <https://kubernetes.io/docs/concepts/policy/limit-range/>
- 네임 스페이스에서 파드 또는 컨테이너별로 리소스를 제한하는 정책
- 리미트 레인지의 기능
  - 네임 스페이스에서 파드나 컨테이너당 최소 및 최대 컴퓨팅 리소스 사용량 제한
  - 네임 스페이스에서 PersistentVolumeClaim 당 최소 및 최대 스토리지 사용량 제한
  - 네임 스페이스에서 리소스에 대한 요청과 제한 사이의 비율 적용
  - 네임 스페이스에서 컴퓨팅 리소스에 대한 디폴트 requests/limit를 설정하고 런타임 중인 컨테이너에 자동으로 입력
- LimitRange 적용 방법
  - Apiserver 옵션에 --enable-admission-plugins=LimitRange를 설정

# 시스템 리소스 요구사항과 제한 설정

## limitRanges

### ● 컨테이너 수준의 리소스 제한

- 제한하기 원하는 네임스페이스에 limitrange 리소스 생성

### ● 예제 해석

- 각 컨테이너에 설정

기준	CPU	메모리
최대	800m	1Gi
최소	100m	99Mi
기본 제한	700m	900Mi
기본 요구사항	110m	111Mi

### ● 리소스 조회

- `kubectl describe limitrange -n 네임스페이스`

limit-mem-cpu-per-container.yaml

```
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-mem-cpu-per-container
spec:
  limits:
    - max:
        cpu: "800m"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "99Mi"
      default:
        cpu: "700m"
        memory: "900Mi"
      defaultRequest:
        cpu: "110m"
        memory: "111Mi"
    type: Container
```

# 시스템 리소스 요구사항과 제한 설정

## limitRanges

### ● 파드 수준의 리소스 제한

- 제한하기 원하는 네임스페이스에 limitrange 리소스 생성

### ● 예제 해석

- 각 파드에 설정

기준	CPU	메모리
최대	2	2Gi
최소	-	-
기본	-	-
최소 요구사항	-	-

### ● 리소스 조회

- `kubectl describe limitrange -n 네임스페이스`

limit-mem-cpu-per-pod.yaml

```
apiVersion: v1
kind: LimitRange
metadata:
  name: limit-mem-cpu-per-pod
spec:
  limits:
    - max:
        cpu: "2"
        memory: "2Gi"
      type: Pod
```

# 시스템 리소스 요구사항과 제한 설정

## limitRanges

### ● 스토리지 리소스 제한

- 제한하기 원하는 네임스페이스에 limitrange 리소스 생성

### ● 예제 해석

- 각 PVC에 설정

기준	용량
최대	2Gi
최소	1Gi

### ● 리소스 조회

- `kubectl describe limitrange -n 네임스페이스`

storagelimits.yaml

```
apiVersion: v1
kind: LimitRange
metadata:
  name: storagelimits
spec:
  limits:
  - type: PersistentVolumeClaim
    max:
      storage: 2Gi
    min:
      storage: 1Gi
```

# 시스템 리소스 요구사항과 제한 설정

## ResourceQuota

- <https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/quota-memory-cpu-namespace/>

### ● 네임스페이스별 리소스 제한

- 제한하기 원하는 네임스페이스에 ResourceQuota 리소스 생성
- 모든 컨테이너에는 CPU, 메모리에 대한 최소요구사항 및 제한 설정이 필요

mem-cpu-demo.yaml

### ● 예제 해석

- 네임스페이스 내의 모든 컨테이너의 합이 다음을 넘어서는 안됨

기준	CPU	메모리
최대	2	2Gi
최소 요구사항	1	1Gi

### ● 리소스 조회

- `kubectl describe resourcequota -n 네임스페이스`

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```