



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

도커/쿠버네티스 온라인 부트캠프 with 카카오엔터프라이즈

보안프로젝트 최일선 CTO

isc0304@naver.com

• 최 일 선



최 일 선 기술이사

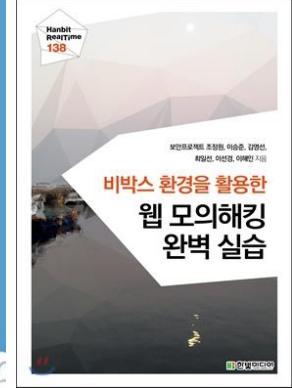
llsun Choi

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

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



www.one4all.co.kr



- 현 원포울, 보안프로젝트 최일선 CTO
- 현 멀티캠퍼스, 한국데이터진흥원 외부 전문 강사
- 행안부 생애주기별 컨텐츠 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 채널 운영
- “비박스를 활용한 웹 모의해킹 완벽실습(2016)”, “마인크래프트와 함께 즐겁게 파이썬(2018)”, “시스템 해킹 입문 프로토스타(2019)” 저자



재즐보프
구독자 1.2만명

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

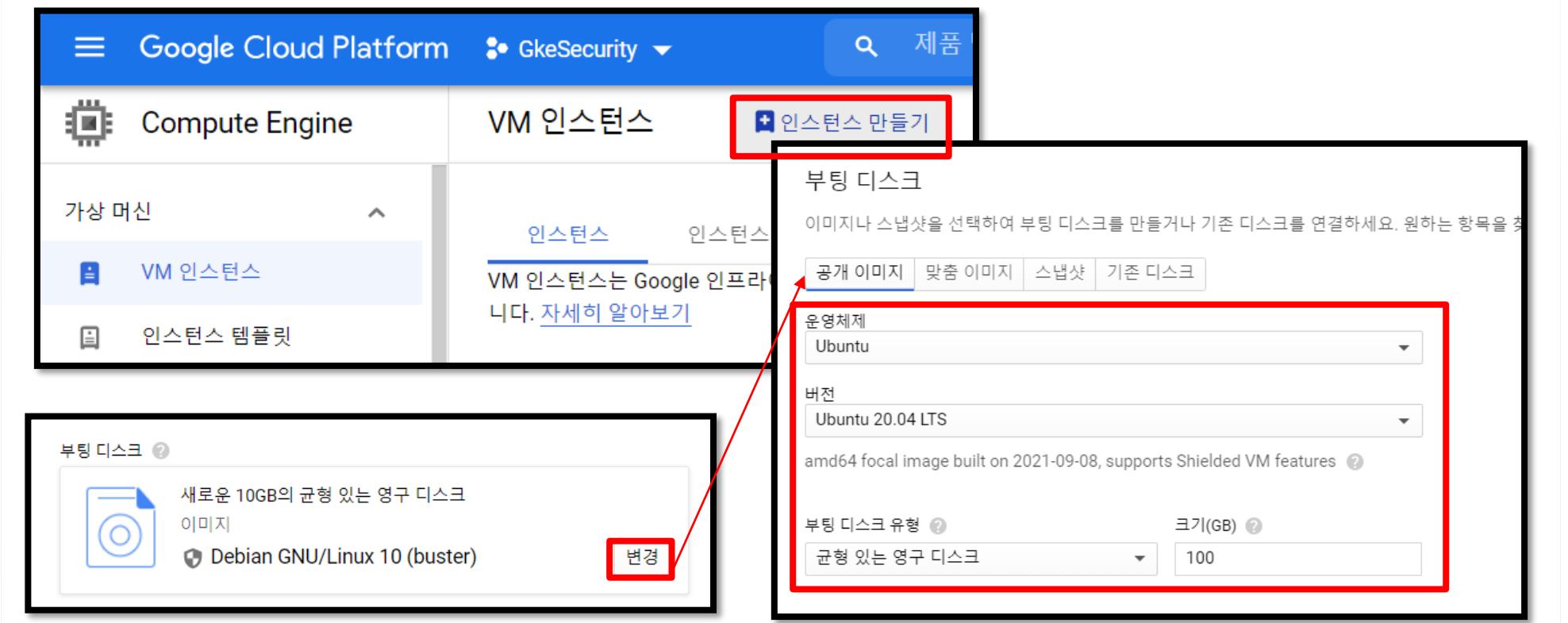


실습을 위한 클러스터 환경 재구성

실습을 위한 클러스터 환경 재구성

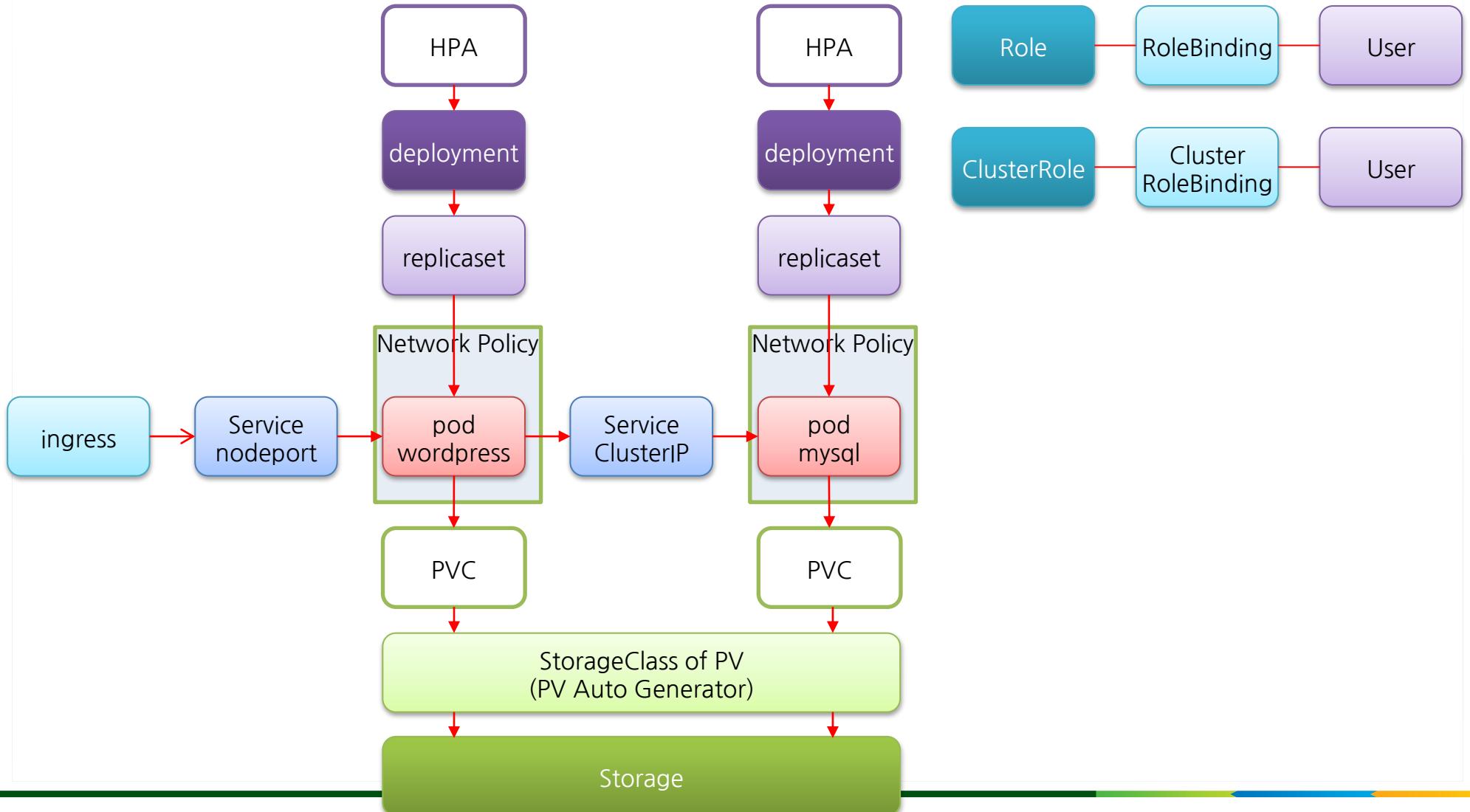
▶ (연습문제) 쿠버네티스 설치 필요 사항

- 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



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 돋기 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 "주관적 인 추천" 준비 기간이다.)	2주	2일	4주
시험 범위	넓음, 클러스터 쿠버네티 스 활용과 클러스터 유지 보수, 트러블 슈팅 등	중간, 클러스터 쿠버네티 스 기본 활용 가능	넓음, 클러스터 쿠버네티 스의 보안 관련 기본 기 능과 추가로 설치하면 좋 은 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 도큐먼트에서 다양한 정보 확인 가능
- 필수 환경 적용: kubectl 자동 완성 명령어
- <https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

The screenshot shows the official Kubernetes documentation website at <https://kubernetes.io>. The page title is "kubectl 치트 시트". The left sidebar contains navigation links for "Home", "Getting Started", "Concepts", "Tasks", "Tutorials", "References", and specific sections for "kubectl", "kubectl completion", "JSONPath", "kubectl", "Commands", and "Usage Guidelines". The main content area displays the "kubectl cheatsheet" with a search bar and a sidebar for printing options. A code block in the center shows the command to enable bash completion:

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

The right sidebar lists other cheat sheets and resources: "kubectl 자동 완성", "BASH", "ZSH", "kubectl 텍스트와 설정", "kubectl apply", "오브젝트 생성", "리소스 조회 및 찾기", and "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!

\$ killer_ Kubernetes Exam Simulator

CKS CKA CKAD Dashboard Store Account Logout

CKS CKA
CKAD
Simulator

Get Started

<https://killer.sh/>

파드 컨테이너 디자인

▶ 멀티 컨테이너 파드

▶ 라이브네스와 레디네스, 스타트업프로브

▶ 사이드카 컨테이너

▶ 어댑터 컨테이너

▶ 앰배서더 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

```
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"]
```

pod-mutil-continaer.yaml

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 환경에서 커맨드 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)

```
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

소스코드 출처:

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

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

Startup 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

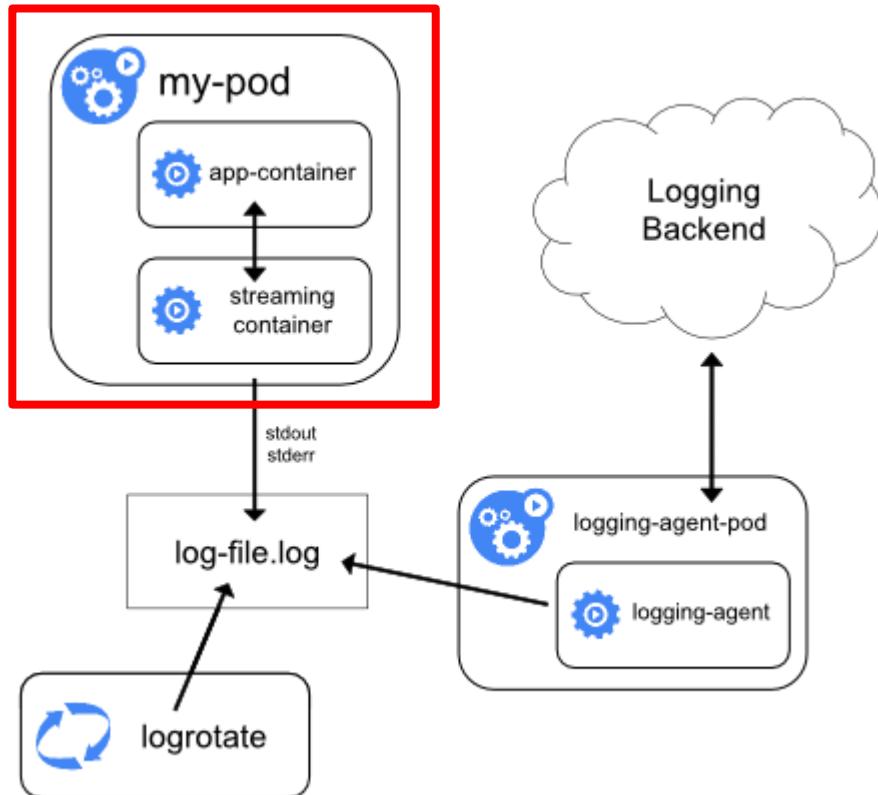
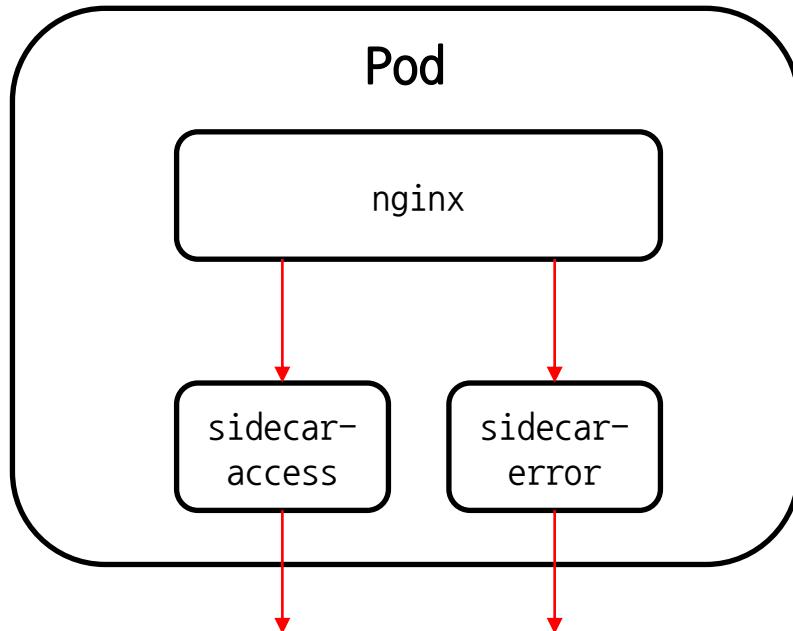


그림 출처: <https://kubernetes.io/ko/docs/concepts/administration/logging/>

사이드카 컨테이너

■ 사이드카 컨테이너 생성 실습

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



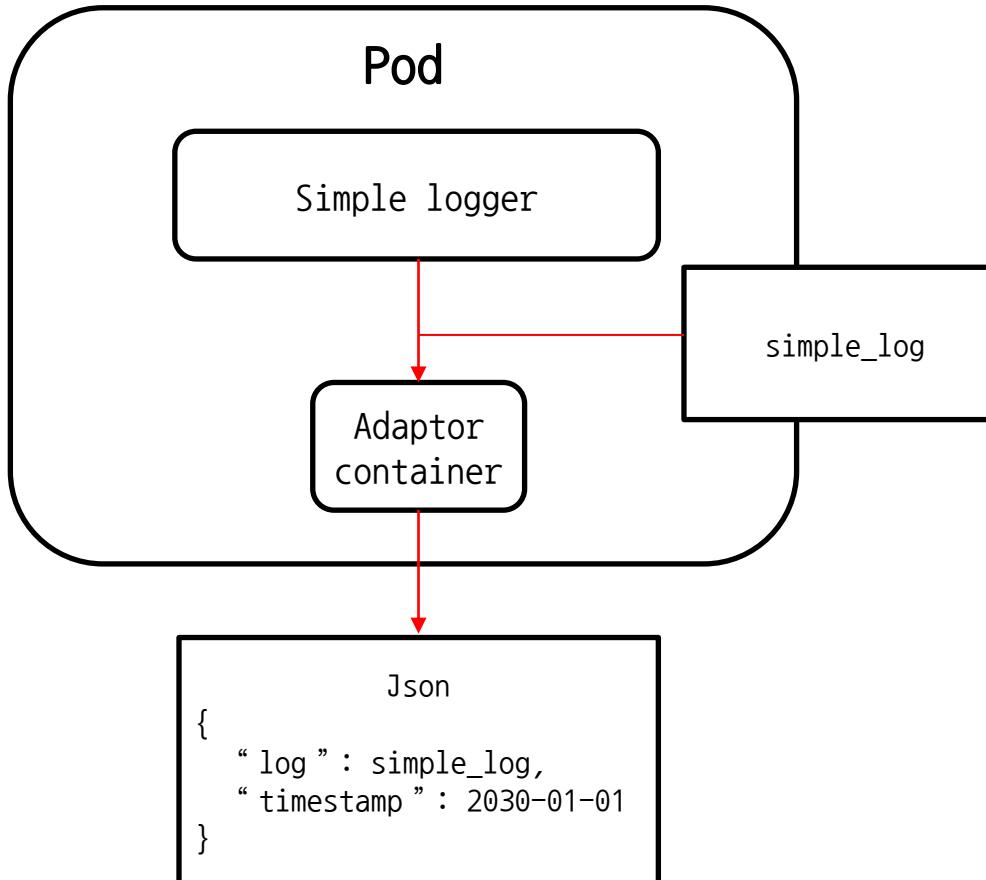
```
# nginx-sidecar.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-sidecar
spec:
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - name: varlognginx
          mountPath: /var/log/nginx
    - name: sidecar-access
      image: 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

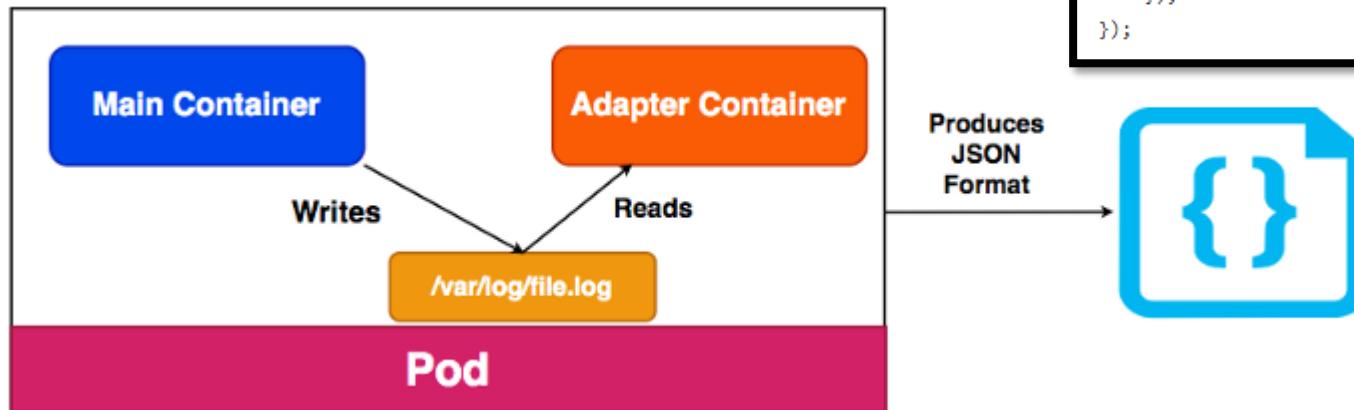


그림 출처: <https://medium.com/bb-tutorials-and-thoughts/kubernetes-learn-adaptor-container-pattern-97674285983c>

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
```

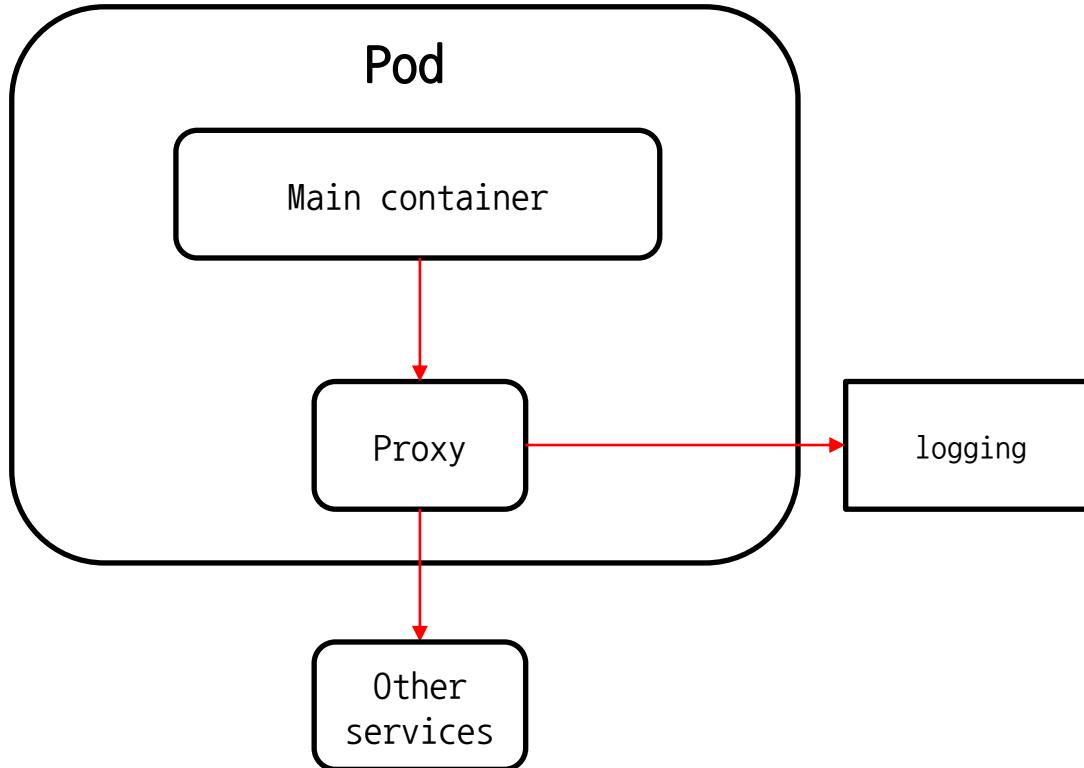
```
[  
  - {  
      time: "Sun Sep 13 03:46:33 UTC 2020",  
      message: "This is log"  
    },  
  - {  
      time: "Sun Sep 13 03:46:38 UTC 2020",  
      message: "This is log"  
    },  
  - {  
      time: "Sun Sep 13 03:46:43 UTC 2020",  
      message: "This is log"  
    },  
  - {  
  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

앰배서더 컨테이너

앰배서더 컨테이너

■ 앰배서더 컨테이너란?

- 엔보서더(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

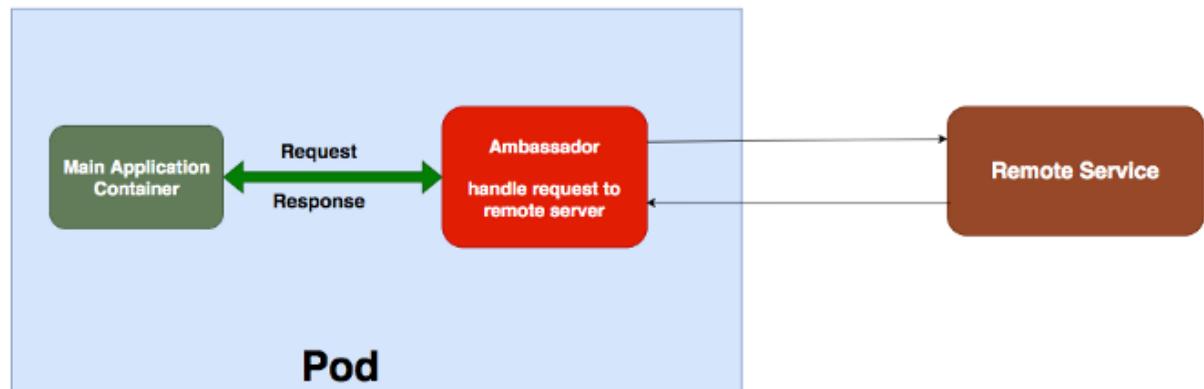


그림 출처: <https://medium.com/bb-tutorials-and-thoughts/kubernetes-learn-ambassador-container-pattern-bc2e1331bd3a>

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
```

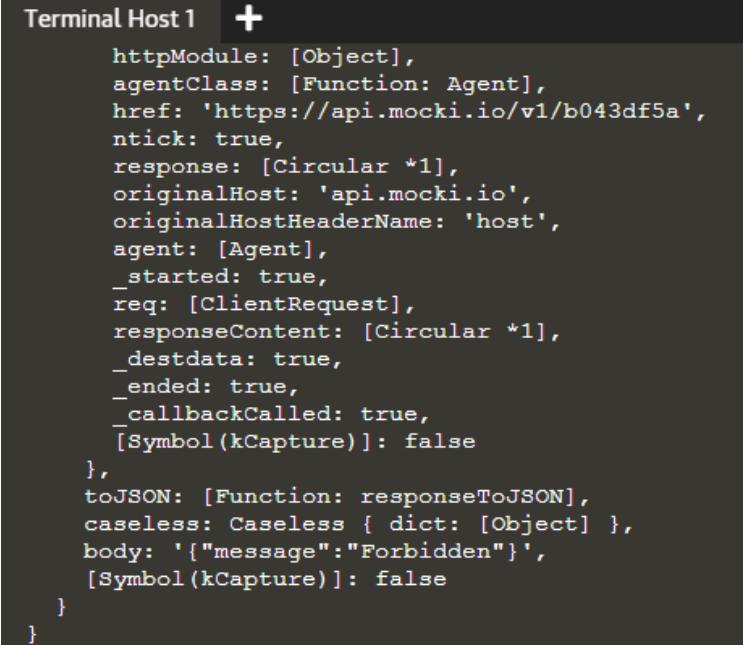
- 데이터 요청

```
$ 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 myse  
rvice; sleep 2; done;']  
    - name: init-mydb  
      image: busybox:1.28  
      command: ['sh', '-c', 'until nslookup mydb; do echo waiting for mydb; sle  
ep 2; done;']
```

초기화 컨테이너

▶ init 컨테이너의 특징

- init 프로세스를 끝낼 수 있는 종결자 등장!

```
apiVersion: v1                                svc-pod-mydb.yaml
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

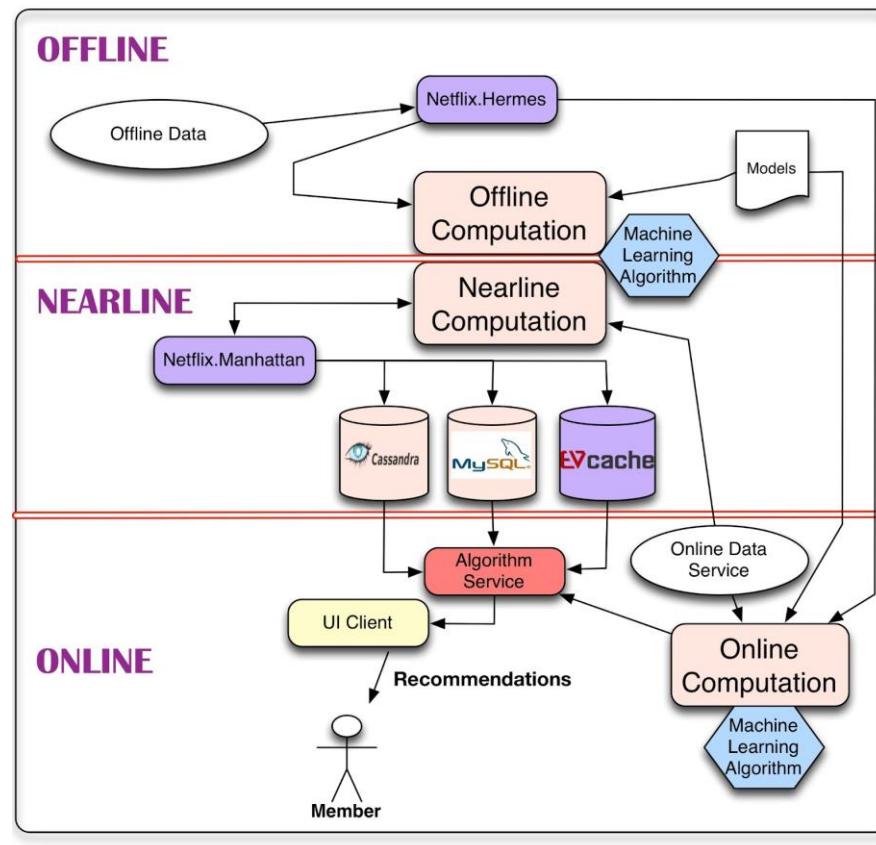
- pod-init-container.yaml을 작성하여 리소스를 생성하라.
- my-app-pod 파드에서 주 컨테이너가 실행되지 않는 현상을 관찰하라.
- 주 컨테이너가 실행되지 않는 이유는 무엇인가?
- svc-pod-mydb.yaml을 작성 및 실행하고 주 컨테이너의 반응을 관찰하라.
- 주 컨테이너가 정상적으로 실행되었는가? 그렇다면 그 이유는 무엇인가?

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 Pods 실행을 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

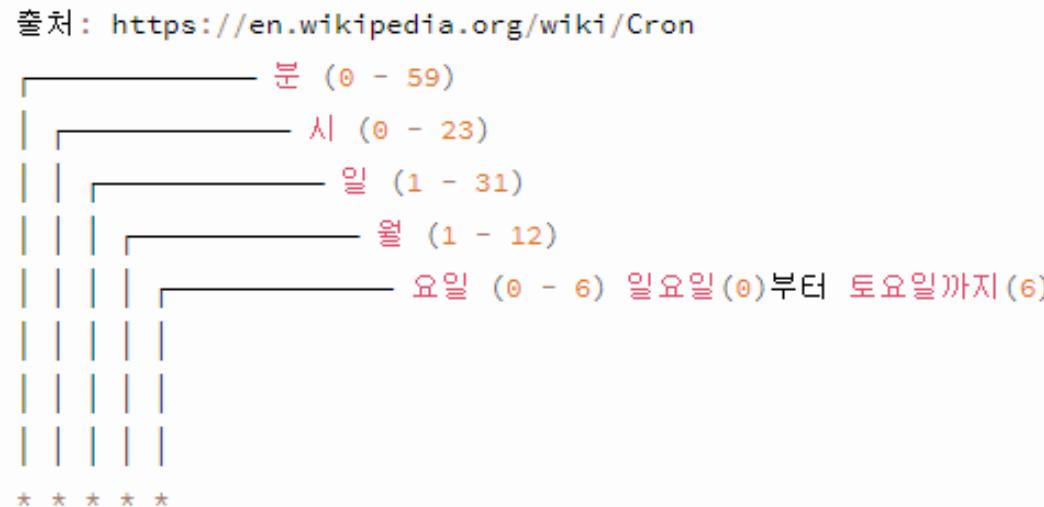
```
cat <<EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: pi-parallelism
spec:
  completions: 5 # 목표 완료 파드 개수
  parallelism: 2 # 동시 실행 가능 파드 개수
  template:
    spec:
      containers:
        - name: pi
          image: perl
          command: ["perl", "-Mbignum=bpi", "-wle",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

job과 cronjob

▶ 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



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: "*/1 * * * *"
  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	*/1 * * * *	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 출력합니다.
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 앞
서 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"
```

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

storagelimits.yaml

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

● 리소스 조회

- kubectl describe limitrange -n 네임스페이스

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

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

● 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

쿠버네티스 저장소

▶ 볼륨 개요

▶ Secrets, configmap 마운트

▶ NFS를 활용한 네트워크 스토리지

▶ PV와 PVC

▶ StorageClass

▶ rook-ceph를 활용한
프라이빗 클라우드 스토리지클래스

▶ 스테이트풀셋



볼륨 개요

볼륨 개요

▶ 볼륨(Volume)

- 컨테이너가 외부 스토리지에 액세스하고 공유하는 방법
- 파드의 각 컨테이너에는 고유의 분리된 파일 시스템 존재
- 볼륨은 파드의 컴포넌트이며 파드의 스펙에 의해 정의
- 독립적인 쿠버네티스 오브젝트가 아니며 스스로 생성, 삭제 불가
- 각 컨테이너의 파일 시스템의 볼륨을 마운트하여 생성
- 볼륨의 종류

임시 볼륨	로컬 볼륨	네트워크 볼륨	네트워크 볼륨 (클라우드 종속적)
emptyDir	hostpath local	iSCSI NFS cephFS glusterFS ...	gcePersistentDisk awsEBS azureFile ...

▶ 주요 사용 가능한 볼륨의 유형

- **emptyDir**: 일시적인 데이터 저장, 비어 있는 디렉터리
- **hostPath**: 파드에 호스트 노드의 파일 시스템에서 파일이나 디렉토리를 마운트
- **nfs**: 기존 NFS (네트워크 파일 시스템) 공유가 파드에 장착
- **gcePersistentDisk**: 구글 컴퓨터 엔진 (GCE) 영구디스크 마운트
(awsElasticBlockStore, azureDisk 또한 클라우드에서 사용하는 형태)
- **persistentVolumeClaim**: 사용자가 특정 클라우드 환경의 세부 사항을 모른 채 GCE PersistentDisk 또는 iSCSI 볼륨과 같은 내구성 스토리지를 요구(Claim)할 수 있는 방법
- **configMap, Secret, downwardAPI**: 특수한 유형의 볼륨
- 볼륨 관련 레퍼런스
 - <https://kubernetes.io/docs/concepts/storage/volumes/#persistentvolumeclaim>

Secrets, configmap 마운트

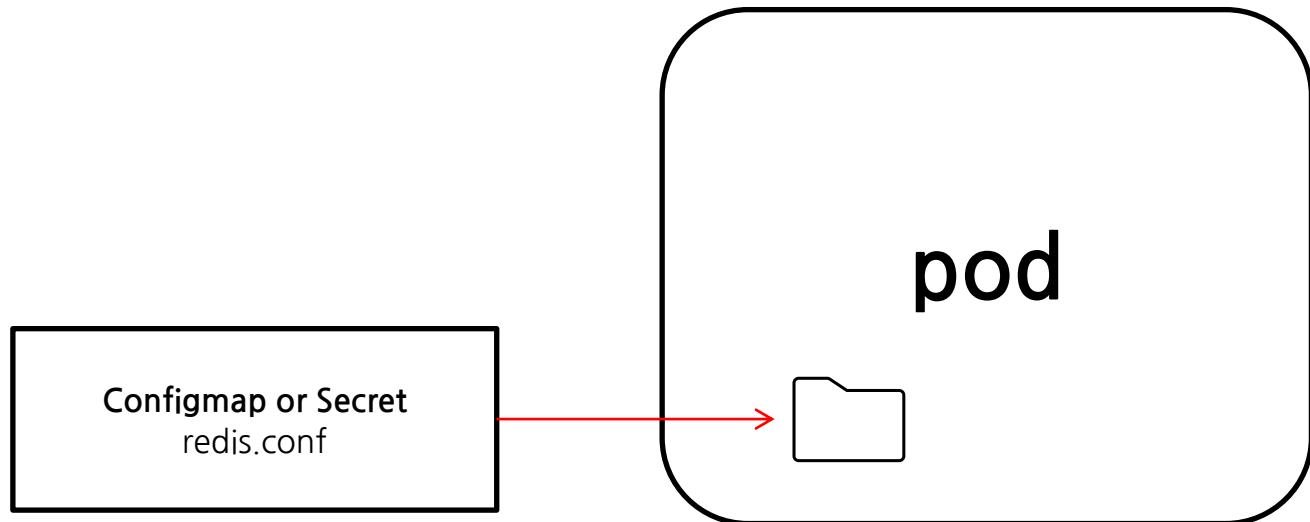
Secrets, configmap 마운트

▣ 컨피그맵(ConfigMap) 마운트

- 컨피그맵은 키-값 쌍으로 기밀이 아닌 데이터를 저장하는 데 사용하는 API 오브젝트
- 파드는 볼륨에서 환경 변수, 커맨드-라인 인수 또는 구성 파일로 컨피그맵을 사용
- 컨피그맵을 사용하면 컨테이너 이미지에서 환경별 구성을 분리하여, 애플리케이션을 쉽게 이식

▣ 시크릿(Secret) 마운트

- 시크릿은 암호, 토큰 또는 키와 같은 소량의 중요한 데이터를 포함하는 오브젝트
- 시크릿을 사용해 사용자의 기밀 데이터를 애플리케이션 코드에 넣을 필요가 없음



Secrets, configmap 마운트

▣ 컨피그맵(ConfigMap) 마운트

- 컨피그맵에 원하는 내용을 넣어서 파일을 구성

```
cat <<EOF >./example-redis-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-redis-config
data:
  redis-config: |
    maxmemory 2mb
    maxmemory-policy allkeys-lru
EOF
```

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/website/main/content/en/examples/pods/config/redis-pod.yaml
```

Secrets, configmap 마운트

▣ 컨피그맵(ConfigMap) 마운트

- 컨피그맵이 적절히 전달 됐는지 확인

```
$ kubectl exec -it redis -- redis-cli
```

```
127.0.0.1:6379> CONFIG GET maxmemory-policy
1) "maxmemory-policy"
2) "allkeys-lru"
```

```
127.0.0.1:6379> CONFIG GET maxmemory
1) "maxmemory"
2) "2097152"
```

Secrets, configmap 마운트

▶ 시크릿(Secret) 마운트

- 시크릿에 전달할 데이터와 시크릿 생성

```
echo -n admin > username  
echo -n 1q2w3e > password
```

```
kubectl create secret generic mysecret --from-file=username --from-file=password
```

Secrets, configmap 마운트

▶ 시크릿(Secret) 마운트

- Secret을 통해서 파드에 마운트

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: mypod
      image: redis
    volumeMounts:
      - name: foo
        mountPath: "/etc/foo"
        readOnly: true
  volumes:
    - name: foo
      secret:
        secretName: mysecret
EOF
```

NFS를 활용한 네트워크 스토리지

NFS를 활용한 네트워크 스토리지

▶ NFS 네트워크 볼륨 사용하기

- NFS 네트워크 볼륨이 있어야 K8S와 테스트 가능

- 서버 설치 방법
 - ✓ apt-get update
 - ✓ apt-get install nfs-common nfs-kernel-server portmap
- 공유할 디렉토리 생성
 - ✓ mkdir /home/nfs
 - ✓ chmod 777 /home/nfs
- /etc(exports 파일에 다음 내용 추가
 - ✓ /home/nfs 10.0.2.15(rw,sync,no_subtree_check) 10.0.2.4(rw,sync,no_subtree_check)
10.0.2.5(rw,sync,no_subtree_check)
 - ✓ service nfs-server restart
 - ✓ showmount -e 127.0.0.1
- NFS 클라이언트에서는 mount 명령어로 마운트해서 사용
 - ✓ mount -t nfs 10.0.2.5:/home/nfs /mnt

NFS를 활용한 네트워크 스토리지

▶ NFS 네트워크 볼륨 사용하기

- 각 노드에 NFS 관련 라이브러리 설치

- apt-get update
- apt-get install nfs-common nfs-kernel-server portmap

- /home/nfs에 index.html을 생성

- nfs-httdp.yaml 파일을 실행 후 접속 테스트

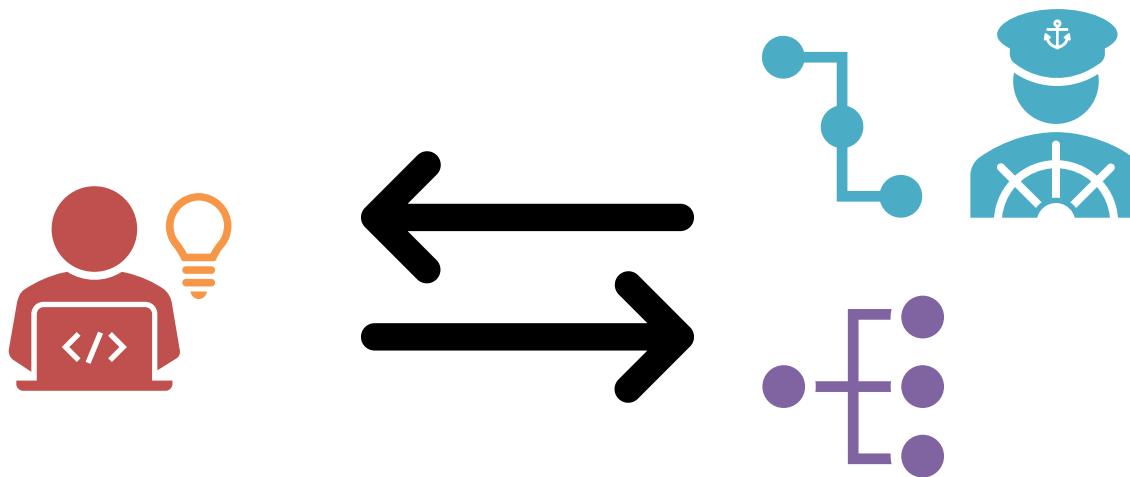
nfs-httdp.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nfs-httdp
spec:
  containers:
    - image: httpd
      name: web
    volumeMounts:
      - mountPath: /usr/local/apache2/htdocs
        name: nfs-volume
        readOnly: true
  volumes:
    - name: nfs-volume
      nfs:
        server: 10.0.2.5
        path: /home/nfs
```

PV와 PVC

▶ 파드 개발자 입장에서의 추상화

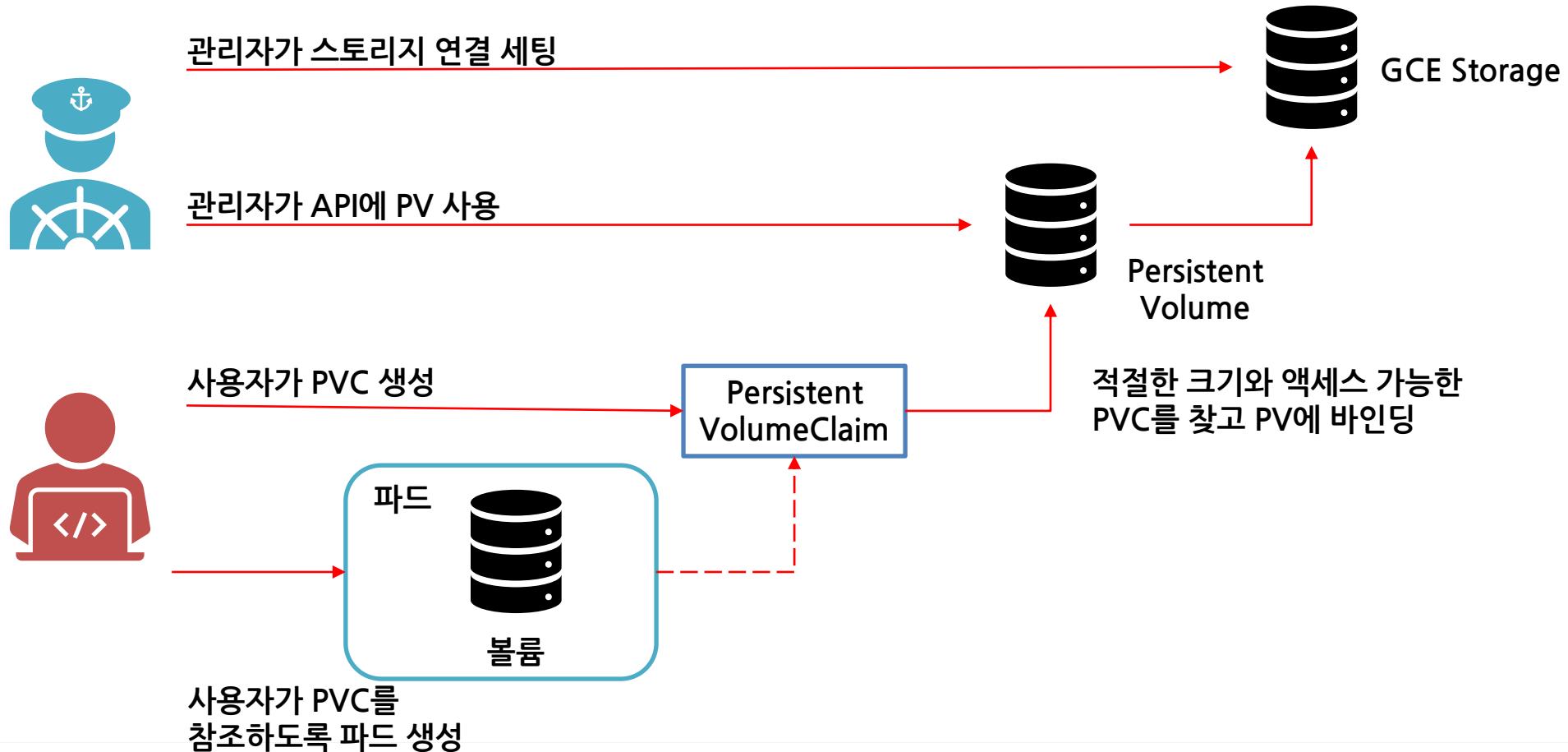
- 파드 개발자가 클러스터에서 스토리지를 사용할 때 인프라를 알아야 할까?
- 실제 네트워크 스토리지를 사용하려면 알아야 함
- 애플리케이션을 배포하는 개발자가 스토리지 기술의 종류를 몰라도 상관없도록 하는 것이 이상적
- 인프라 관련 처리는 클러스터 관리자의 유일한 도메인!
- pv와 pvc를 사용해 관리자와 사용자의 영역을 나눔



PV와 PVC

PersistentVolume(PV)과 PersistentVolumeClaim(PVC)

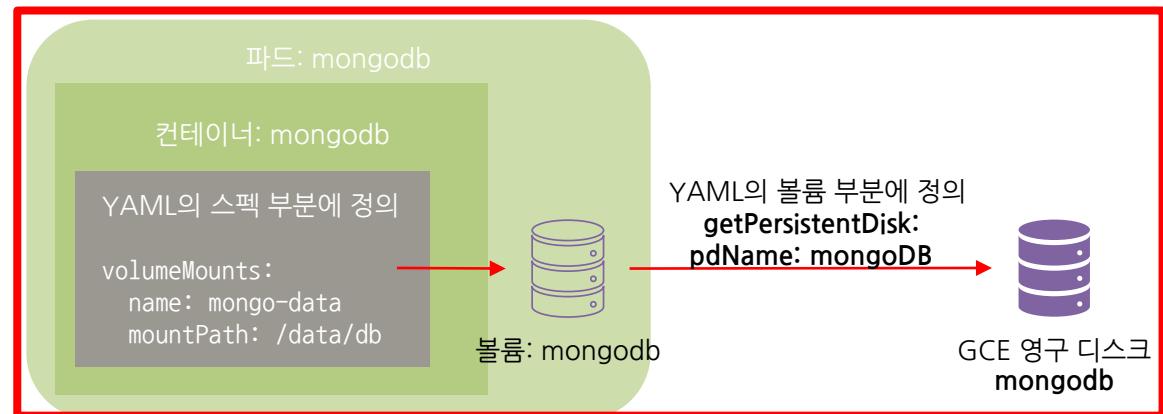
- 인프라 세부 사항을 알지 못해도 클러스터의 스토리지를 사용할 수 있도록 제공해주는 리소스
- 파드 안에 영구 볼륨을 사용하도록 하는 방법은 다소 복잡



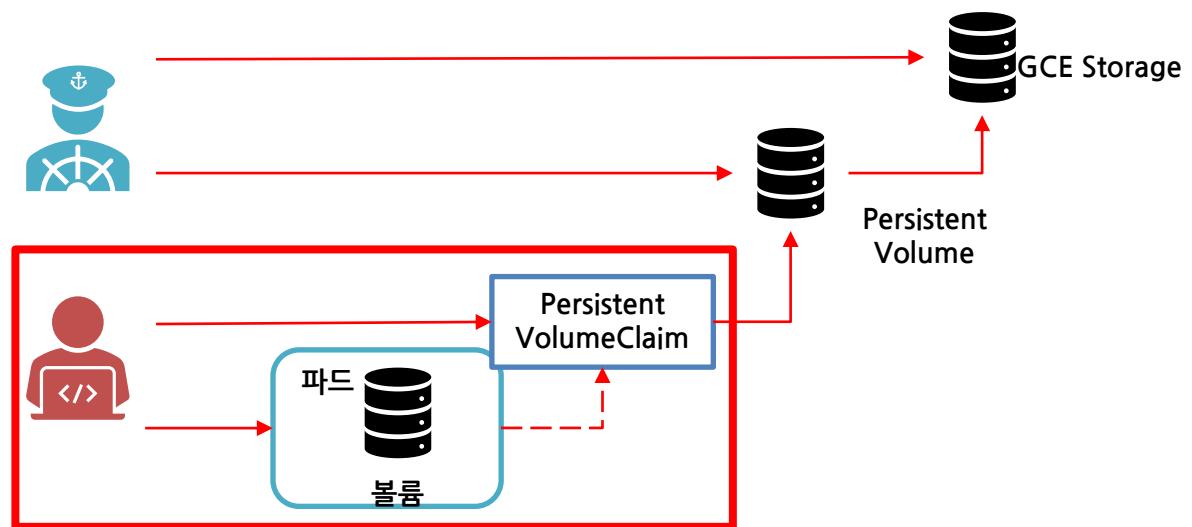
PV와 PVC

▶ PV, PVC의 장점 비교

getPersistentDisk를 사용할 때
사용자가 알아야 하는 부분



pv, pvc를 사용할 때
사용자가 알아야 하는 부분



PV와 PVC

▶ PersistentVolume(PV)과 PersistentVolumeClaim(PVC) 정의

- 두 가지 새로운 요소에 대해 정의: PVC

mongo-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc    클레임 사용 때 필요
spec:
  resources:
    requests:
      storage: 1Gi    요청하는 스토리지 양
  accessModes:
    - ReadWriteOnce   단일 클라이언트에
                      읽기쓰기 지원
  storageClassName: ""    동적 프로비저닝에서 사용
```

PV와 PVC

▶ PersistentVolume(PV)과 PersistentVolumeClaim(PVC) 정의

● 두 가지 새로운 요소에 대해 정의: PV

➤ 참고: PV는 네임스페이스에 속하지 않는다!

mongo-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongodb-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
    - ReadOnlyMany
  persistentVolumeReclaimPolicy: Retain
  gcePersistentDisk:
    pdName: mongodb
    fsType: ext4
```

몽고DB에 대한 정의

단일 클라이언트에 읽기쓰기 가능
여러 번 읽기만 가능

Reclaiming	설명
Retain(유지)	PersistentVolumeClaim삭제하면 PersistentVolume여전히 존재하고 볼륨은 "해제된"것으로 간주 연관된 스토리지 자산의 데이터를 수동으로 정리
Delete(삭제)	외부 인프라의 연관된 스토리지 자산을 모두 제거
Recycle(재사용)	rm -rf /thevolume/*볼륨에 대한 기본 스크립트()을 수행하고 새 클레임에 대해 다시 사용할 수 있도록 함

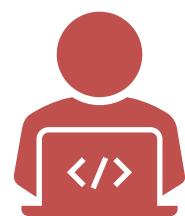
※ 설정은 생성 후에도 재설정 가능

PV와 PVC

PersistentVolume(PV)과 PersistentVolumeClaim(PVC)

- 인프라 세부 사항을 알지 못해도 클러스터의 스토리지를 사용할 수 있도록 제공해주는 리소스
- 파드 안에 영구 볼륨을 사용하도록 하는 방법은 다소 복잡

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: ""
```



Persistent
VolumeClaim



GCE Storage

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongodb-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
    - ReadOnlyMany
  persistentVolumeReclaimPolicy: Retain
  gcePersistentDisk:
    pdName: mongodb
    fsType: ext4
```

PV와 PVC

▶ PV, PVC 생성과 조회

```
$ kubectl delete all --all  
$ kubectl create -f mongo-pv.yaml  
$ kubectl create -f mongo-pvc.yaml
```

```
$ kubectl get pvc  
NAME      STATUS   VOLUME      CAPACITY   ACCESS MODES   STORAGECLASS   AGE  
mongodb-pvc  Bound    mongodb-pv  1Gi        RWO,ROX          default        30s
```

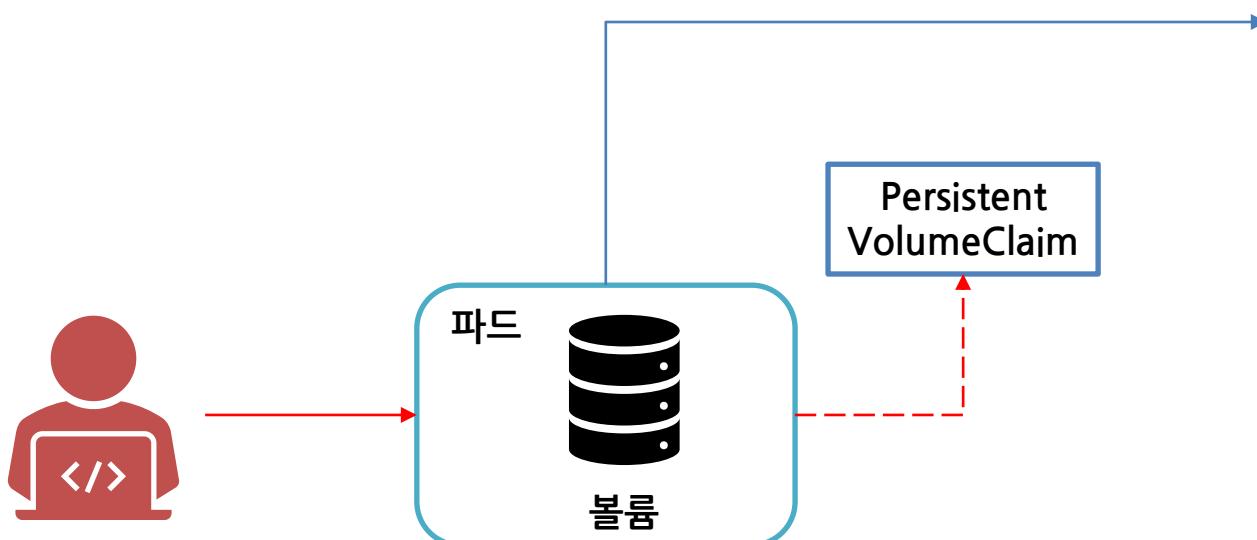
```
$ kubectl get pv  
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM           ...  
mongodb-pv  1Gi        RWO,ROX       Retain           Bound    default/mongodb-pvc  ...
```

PV와 PVC

▶ PVC를 활용한 파드 생성

```
$ kubectl create -f mongo-pvc-pod.yaml  
pod/mongodb created
```

```
$ kubectl exec -it mongodb mongo  
MongoDB shell version v4.0.10  
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb  
[...]  
> use mystore  
switched to db mystore  
> db.foo.find()  
{ "_id" : ObjectId("5d0f0b03a3edc611a05bda93"), "name" : "foo" }
```



mongo-pvc-pod.yaml

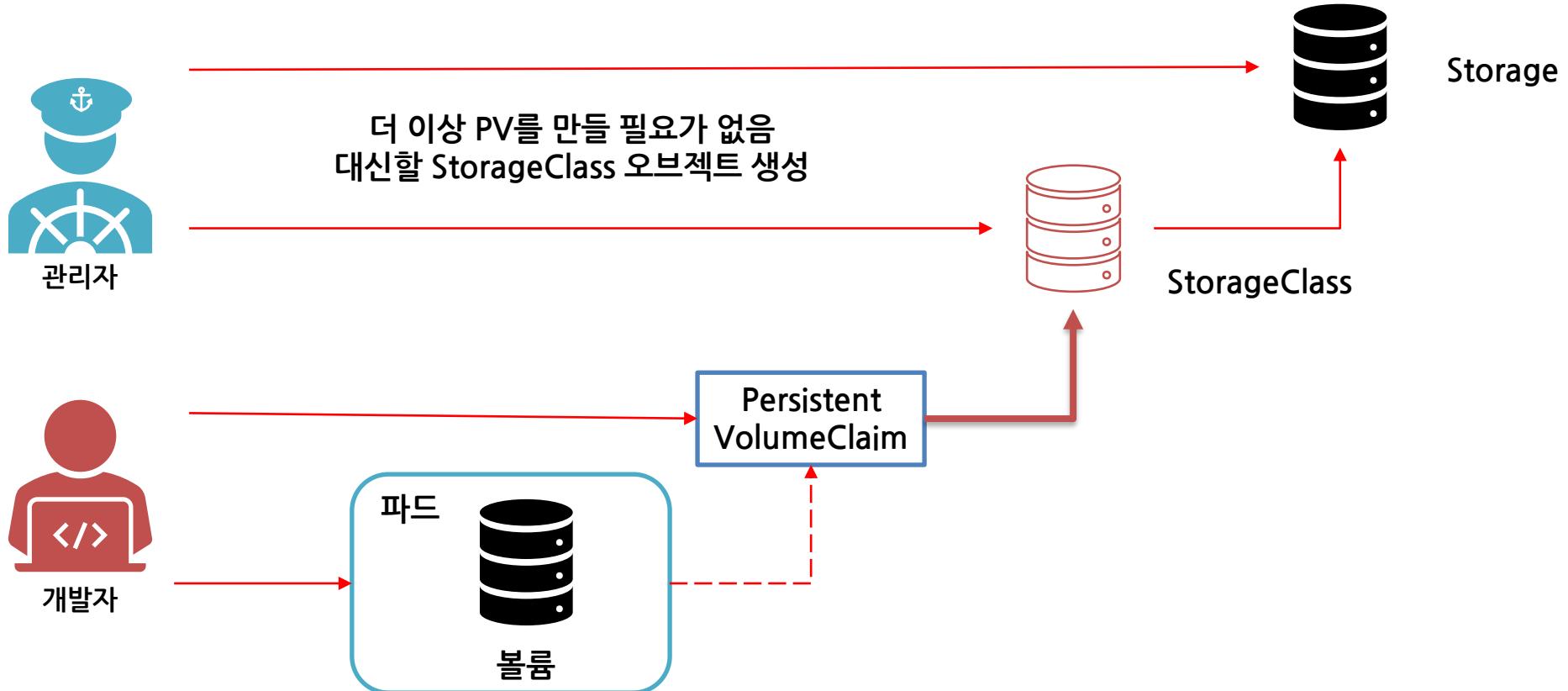
```
apiVersion: v1  
kind: Pod  
metadata:  
  name: mongodb  
spec:  
  containers:  
    - image: mongo  
      name: mongodb  
      volumeMounts:  
        - name: mongodb-data  
          mountPath: /data/db  
  ports:  
    - containerPort: 27017  
      protocol: TCP  
volumes:  
  - name: mongodb-data  
    persistentVolumeClaim:  
      claimName: mongodb-pvc
```

StorageClass

StorageClass

▶ PV 동적 프로비저닝

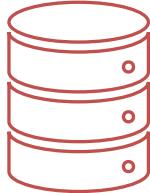
- PV를 직접 만드는 대신 사용자가 원하는 PV 유형을 선택하도록 오브젝트 정의 가능



StorageClass

▶ PV 동적 프로비저닝

- StorageClass yaml 파일 제작



storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: storage-class
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```

프로비저닝에 사용할 플러그인 선택

제공자에게 전달될 매개 변수

```
$ kubectl create -f storageclass.yaml
storageclass.storage.k8s.io/storage-class created
```

```
$ kubectl get sc
```

NAME	PROVISIONER	AGE
standard (default)	kubernetes.io/gce-pd	2d7h
storage-class	kubernetes.io/gce-pd	26s

StorageClass

▶ PV 동적 프로비저닝

● PVC 파일 제작

- 파드와 PVC 모두 삭제 후 재 업로드 (apply 명령어 시 권한 에러 발생)
- mongo-pvc.yaml 내용 변경

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: ""
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: storage-class
```

StorageClass

▶ PV 동적 프로비저닝

- PV 동적 프로비저닝을 사용하면 사용할 디스크와 PV가 자동으로 생성됨

```
$ gcloud compute disks list
```

NAME	...	SIZE_GB	TYPE	STATUS
...	... 20	pd-standard	READY	
...	... 20	pd-standard	READY	
gke-standard-cluster-1-default-pool-e4bbb8da-9th0		...	20	pd-standard READY
...				
gke-standard-cluster-1-pvc-c285a414-95c7-11e9-81f9-42010a9200df	...	1	pd-ssd	READY

```
$ kubectl get pv
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM
STORAGECLASS	REASON AGE				
mongodb-pv	1Gi	RWO,ROX	Retain		Released
default/mongodb-pvc	38m				
pvc-c285a414-95c7-11e9-81f9-42010a9200df	1Gi	RWO	Delete		Bound
default/mongodb-pvc	storage-class	15m			

StorageClass

▶ PV 동적 프로비저닝 동작 순서 정리

mongo-pvc-pod.yaml (변경사항 없음)

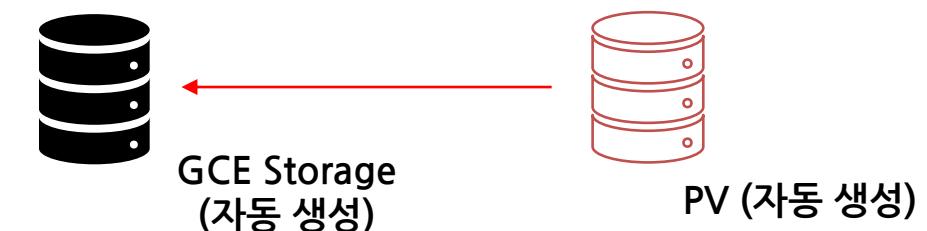
```
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  containers:
    - image: mongo
      name: mongodb
    volumeMounts:
      - name: mongodb-data
        mountPath: /data/db
    ports:
      - containerPort: 27017
        protocol: TCP
    volumes:
      - name: mongodb-data
        persistentVolumeClaim:
          claimName: mongodb-pvc
```

mongo-pvc.yaml (storageclass 이름 변경)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: storage-class
```

storageclass.yaml (새로 생성)

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: storage-class
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```



StorageClass

연습문제

- **httpd를 사용할 수 있도록 pod, pvc, pv 정의하여 수동 프로비저닝 수행하기**
 - httpd를 사용할 수 있도록 수동으로 pod, pvc, pv, disk를 정의하고 생성하라.

- **httpd를 사용할 수 있도록 pod, pvc, storageclass 정의하여 동적 프로비저닝 수행하기**
 - httpd를 사용할 수 있도록 자동으로 pod, pvc, storageclass를 정의하고 생성하라.

rook-ceph를 활용한 프라이빗 클라우드 스토리지 클래스

rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

rook-ceph?

- 온프레미스 환경에서 storage-class를 구성하는 도구
- ceph은 파일 스토리지를 가상화시키는 클러스터를 구성할 수 있는 소프트웨어
- 직접 설치하는 방법도 있지만 rook 패키지를 활용하면 쿠버네티스에서 보다 편리하게 ceph을 설치하고 관리 가능

<http://rook.io/>



Documentation

Community

Blog

Get Started

Open-Source,
Cloud-Native Storage
for Kubernetes

Production ready management for File, Block and Object Storage

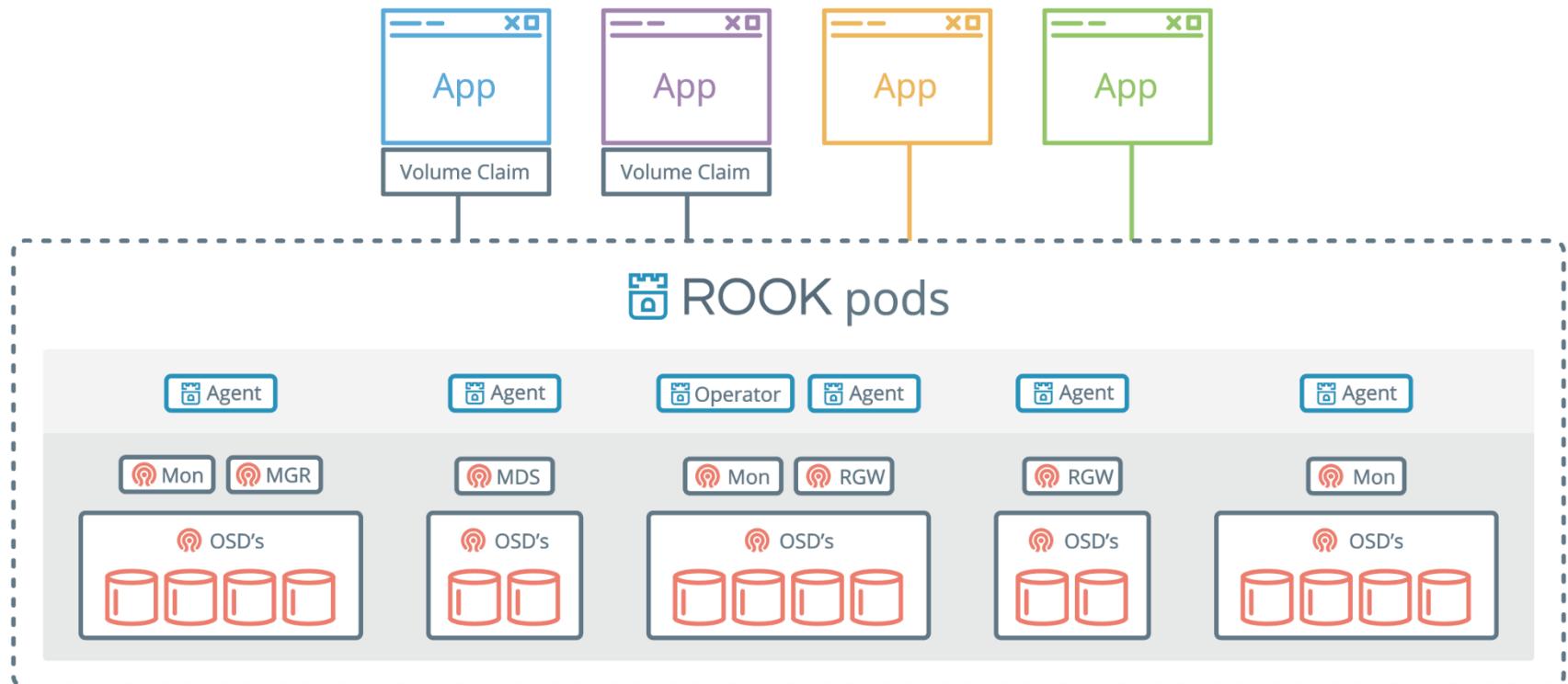
Try it out now



rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

rook-ceph 아키텍처

Rook Architecture

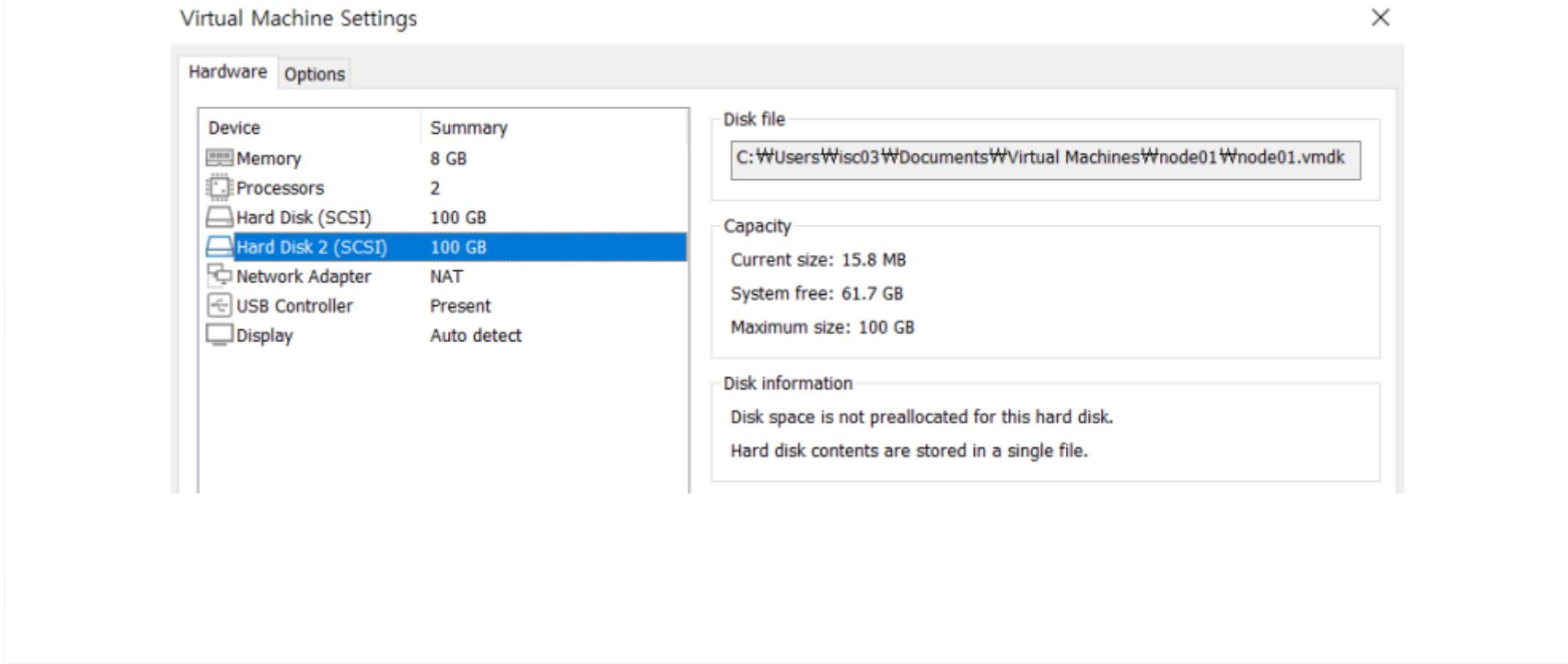


<https://danawalab.github.io/kubernetes/2020/01/28/kubernetes-rook-ceph.html>

rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

설치를 시작하기 전에

- 새 disk 추가 및 lsblk 확인 필요
- 설치를 시작하기 전에 3개의 워커 노드를 준비
- 각 워커 노드에 새로운 빈 디스크를 하나 추가



rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

설치를 시작하기 전에

- 잘 구성하고 `lsblk` 명령을 실행하면 다음과 같이 빈 디스크를 확인
- 하위에 다른 파티션이 구현되어 있어 있어서는 안됨

```
user01@node01:~$ lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0    7:0    0 55.4M  1 loop /snap/core18/2128
loop1    7:1    0 55M   1 loop /snap/core18/1705
loop2    7:2    0 72.5M  1 loop /snap/lxd/21497
loop3    7:3    0 32.3M  1 loop /snap/snapd/12883
loop4    7:4    0 69M   1 loop /snap/lxd/14804
loop5    7:5    0 61.8M  1 loop /snap/core20/1081
loop6    7:6    0 27.1M  1 loop /snap/snapd/7264
sda      8:0    0 100G  0 disk
└─sda1   8:1    0 1M    0 part
└─sda2   8:2    0 100G  0 part /
sdb      8:16   0 100G  0 disk
```

rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

▶ 루크로 ceph 설치하기

- 루크를 깃에서 다운로드하고 설치를 시작
- 한 번에 모든 명령을 실행하면 정확히 실행되지 않는 경우들이 있으니 명령어를 하나씩 실행
- rook 프로젝트에는 다양한 기능이 있으나 그중에 kubernetes ceph를 설치

```
git clone --single-branch --branch release-1.7  
https://github.com/rook/rook.git  
cd rook/cluster/examples/kubernetes/ceph  
kubectl create -f crds.yaml -f common.yaml -f operator.yaml  
kubectl create -f cluster.yaml
```

rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

▶ 툴 박스와 ceph csd를 설치

- 툴박스는 ceph의 상황을 모니터링할 수 있는 툴을 구성한 컨테이너
- csd는 Container Storage Interface라는 의미를 가짐
- csd는 각 노드에 컨테이너에서 스토리지를 사용할 수 있는 인터페이스를 제공

```
kubectl create -f toolbox.yaml  
kubectl create -f csi/rbd/storageclass.yaml
```

rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

▶ 툴박스 컨테이너로 상태 확인

- 툴박스 컨테이너로 접속해서 ceph 스토리지의 status를 확인 가능
- 다음 명령을 실행해 ceph 툴박스 컨테이너로 접속하고 상태 확인

```
$ kubectl -n rook-ceph exec -it $(kubectl -n rook-ceph get pod -l "app=rook-ceph-tools" -o jsonpath='{.items[0].metadata.name}') -- bash
```

컨테이너 내부에서 다음 명령 실행

```
ceph status
```

```
ceph osd pool stats
```

▶ 스토리지 클래스 확인

```
$ kubectl get sc
```

NAME	PROVISIONER	RECLAIMPOLICY	VOLUMEBINDINGMODE	ALLOWVOLUMEE...
rook-ceph-block	rook-ceph.rbd.csi.ceph.com	Delete	Immediate	true

rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

▶ 스토리지클래스 pv 생성 테스트

- pvc를 다음과 같이 생성

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongo-pvc
spec:
  storageClassName: rook-ceph-block
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
EOF
```

- 대응되는 pv가 자동으로 구성되는지 확인

```
$ kubectl get pvc,pv
```

NAME	MODES	STORAGECLASS	AGE	STATUS	VOLUME	CAPACITY	ACCESS
persistentvolumeclaim/mongo-pvc		rook-ceph-block	79m	Bound	pvc-a5c17990-1c5a-42e5-8642-744a1cefabd	2Gi	RWO

NAME	CAPACITY		ACCESS MODES	RECLAIM POLICY
STATUS	CLAIM	STORAGECLASS	REASON	AGE
persistentvolume/pvc-a5c17990-1c5a-42e5-8642-744a1cefabd	2Gi	rook-ceph-block	Bound	79m

스테이트풀셋

스테이트풀셋

▶ 스테이트풀셋?

- 애플리케이션의 상태를 저장하고 관리하는 데 사용되는 쿠버네티스 객체
- 디플로이먼트는 파드를 삭제하고 생성할 때 상태가 유지되지 않는 한계가 있음
- 스테이트풀셋으로 생성되는 파드는 영구 식별자를 가지고 상태를 유지
- 스테이트풀셋을 사용하는 경우
 - 안정적이고 고유한 네트워크 식별자가 필요한 경우
 - 안정적이고 지속적인 스토리지
 - 질서 정연한 배치 및 확장
 - 주문, 자동 롤링 업데이트
- 스테이트풀셋의 문제
 - 스테이트풀셋과 관련된 볼륨이 삭제되지 않음
 - 파드의 스토리지는 PV나 스토리지클래스로 프로비저닝 수행해야 함
 - 롤링업데이트를 수행하는 경우 수동으로 복구해야 할 수 있음
 - 파드 네트워크 ID를 유지하기 위해 헤드레스(headless) 서비스 필요

스테이트풀셋

▶ 헤드레스 서비스 작성 요령

- 헤드레스 서비스는 clusterIP를 None으로 지정하여 생성
- 헤드레스 서비스는 IP가 할당되지 않음
- kube-proxy가 밸런싱이나 프록시 형태로 동작하지 않음

nginx-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
```

스테이트풀셋

▶ 스테이트풀셋 작성 요령

- 디플로이먼트와 전반적으로 설정 방법은 유사
- serviceName을 통해 서비스 지정
- terminationGracePeriodSeconds
 - 파드에도 들어갈 수 있는 설정으로
 - 종료 요청(SIGTERM) 후 30초 기다림
 - 컨테이너가 종료되지 않으면 강제로 SIGKILL
- volumeMounts
 - 영구 스토리지를 연결하고자 하는 위치
- (다음장에 계속)

nginx-statefulset.yaml

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.spec.containers[0].labels.app
  serviceName: "nginx"
  replicas: 3 # by default is 1
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.selector.matchLabels.app
    spec:
      terminationGracePeriodSeconds: 10
      containers:
        - name: nginx
          image: k8s.gcr.io/nginx-slim:0.8
          ports:
            - containerPort: 80
              name: web
      volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
```

스테이트풀셋

▶ 스테이트풀셋 작성 요령

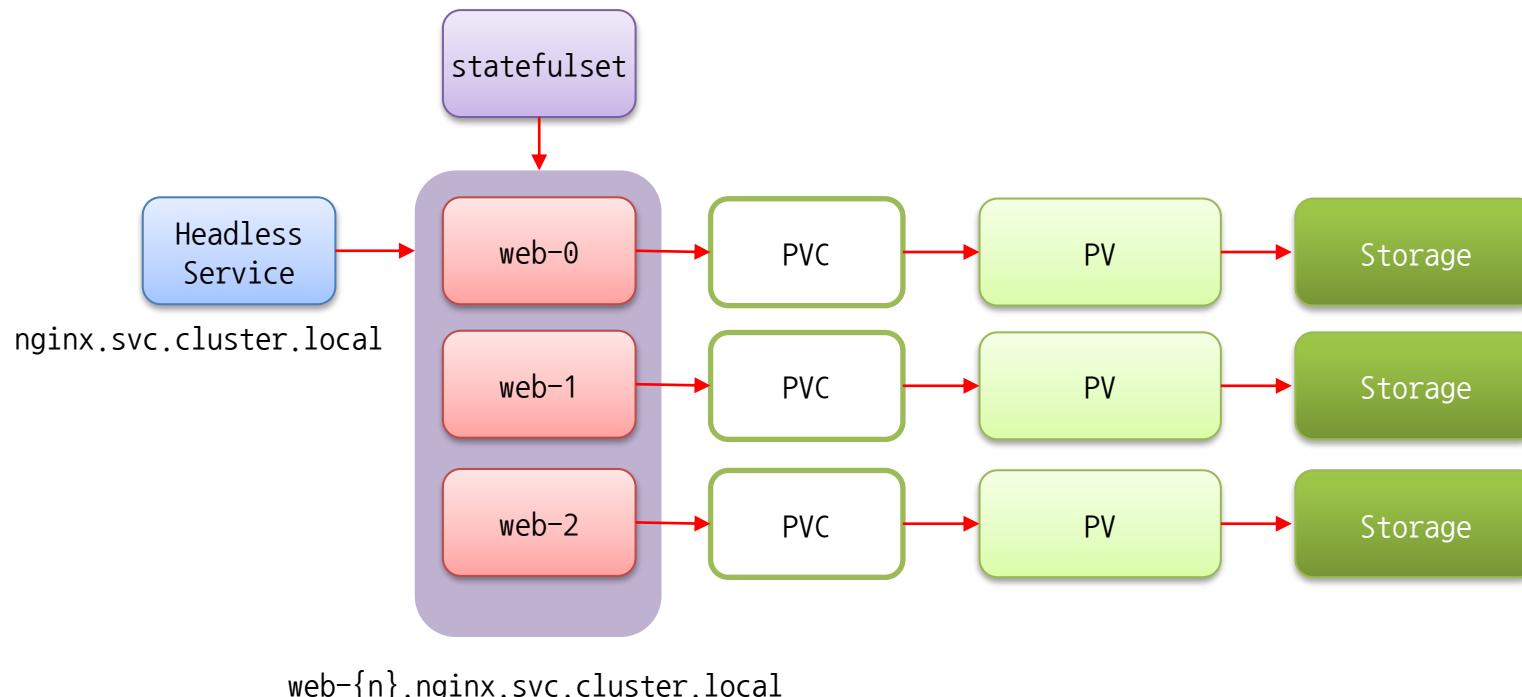
- `volumeClaimTemplates`을 사용해 안정적인 스토리지 제공

```
volumeClaimTemplates:  
- metadata:  
  name: www  
spec:  
  accessModes: [ "ReadWriteOnce" ]  
  storageClassName: "my-storage-class"  
  resources:  
    requests:  
      storage: 1Gi
```

스테이트풀셋

▶ 스테이트풀셋의 다수 파드 식별 요령

- 스테이트풀셋으로 다수의 파드를 생성할 수 있음
- 그러나 스테이트풀셋은 상태를 유지하는 파드로 각각의 파드를 인식할 수 있는 방법을 알아야 함
- 이를 사용해 안정적인 네트워크 ID와 스토리지를 식별할 수 있음
- 순차적으로 하나씩 배포하는데 앞의 파드가 준비상태가 돼야 다음 파드를 생성
- 배포 순서 0 → n-1 // 종료 순서 n-1 → 0



▶ 업데이트 전략

● OnDelete

- 파드를 자동으로 업데이트하는 기능이 아님
- 수동으로 삭제해주시면 스테이트풀 셋의 spec.template를 적용한 새로운 파드가 생성됨

● RollingUpdate

- 롤링 업데이트를 구현하면 한번에 하나씩 파드를 업데이트
- web-{n-1}부터 web-0의 순서로 진행

컨테이너 롤링 업데이트 전략 이해

▶ 디플로이먼트

▶ 애플리케이션 롤링 업데이트와 롤백

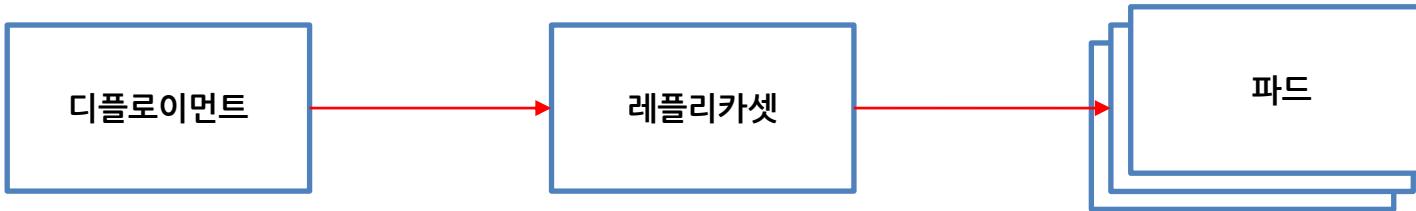


디플로이먼트

디플로이먼트

▶ 디플로이먼트

- 애플케이션을 다운 타입 없이 업데이트 가능하도록 도와주는 리소스!
- 레플리카셋과 레플리케이션컨트롤러 상위에 배포되는 리소스



● 모든 파드를 업데이트하는 방법

- 잠깐의 다운 타임 발생 (새로운 파드를 실행시키고 작업이 완료되면 오래된 파드를 삭제)
- 롤링 업데이트

디플로이먼트

▶ 디플로이먼트 작성 요령

- 파드의 metadata 부분과 spec 부분을 그대로 옮김
- Deployment의 spec.template에는 배포할 파드를 설정
- replicas에는 이 파드를 몇 개를 배포할지 명시
- label은 디플로이먼트가 배포한 파드를 관리하는데 사용됨

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

▶ 디플로이먼트 스케일링

- kubectl edit deploy <deploy name> 명령을 사용해 yaml 파일을 직접 수정하여 replicas 수를 조정
- kubectl scale deploy <deploy name> --replicas=<number> 명령을 사용해 replicas 수 조정

연습문제

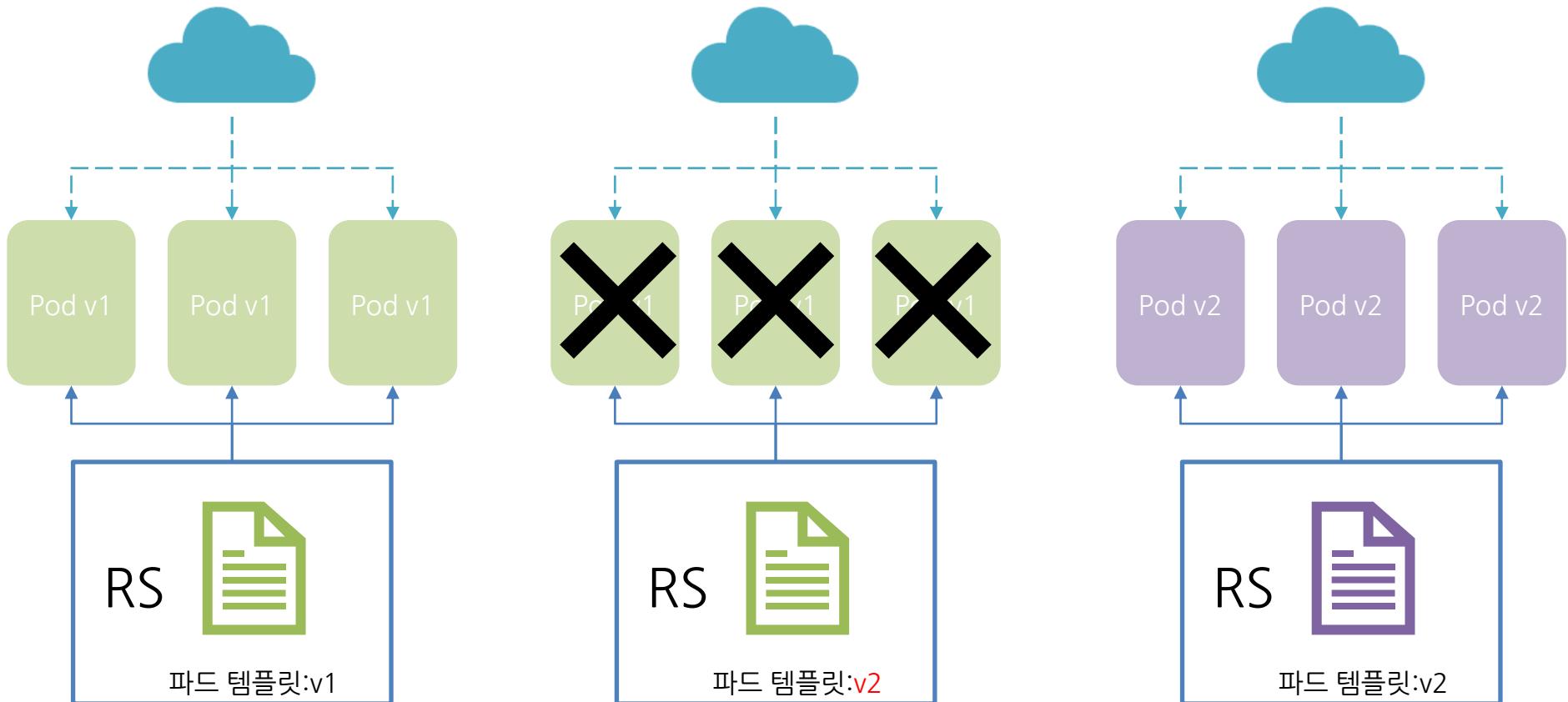
- jenkins 디플로이먼트를 deploy-jenkins를 생성하라.
- jenkins 디플로이먼트로 배포되는 앱을 app: jenkins-test로 레이블링하라.
- 디플로이먼트로 배포된 파드를 하나 삭제하고 이후 생성되는 파드를 관찰하라.
- 새로 생성된 파드의 레이블을 바꾸어 Deployment의 관리 영역에서 벗어나게 하라.
- Scale 명령을 사용해 레플리카 수를 5개로 정의한다.
- edit 기능을 사용하여 10로 스케일링하라.

애플리케이션 롤링 업데이트와 롤백

애플리케이션 롤링 업데이트와 롤백

▣ 기존 모든 파드를 삭제 후 새로운 파드 생성

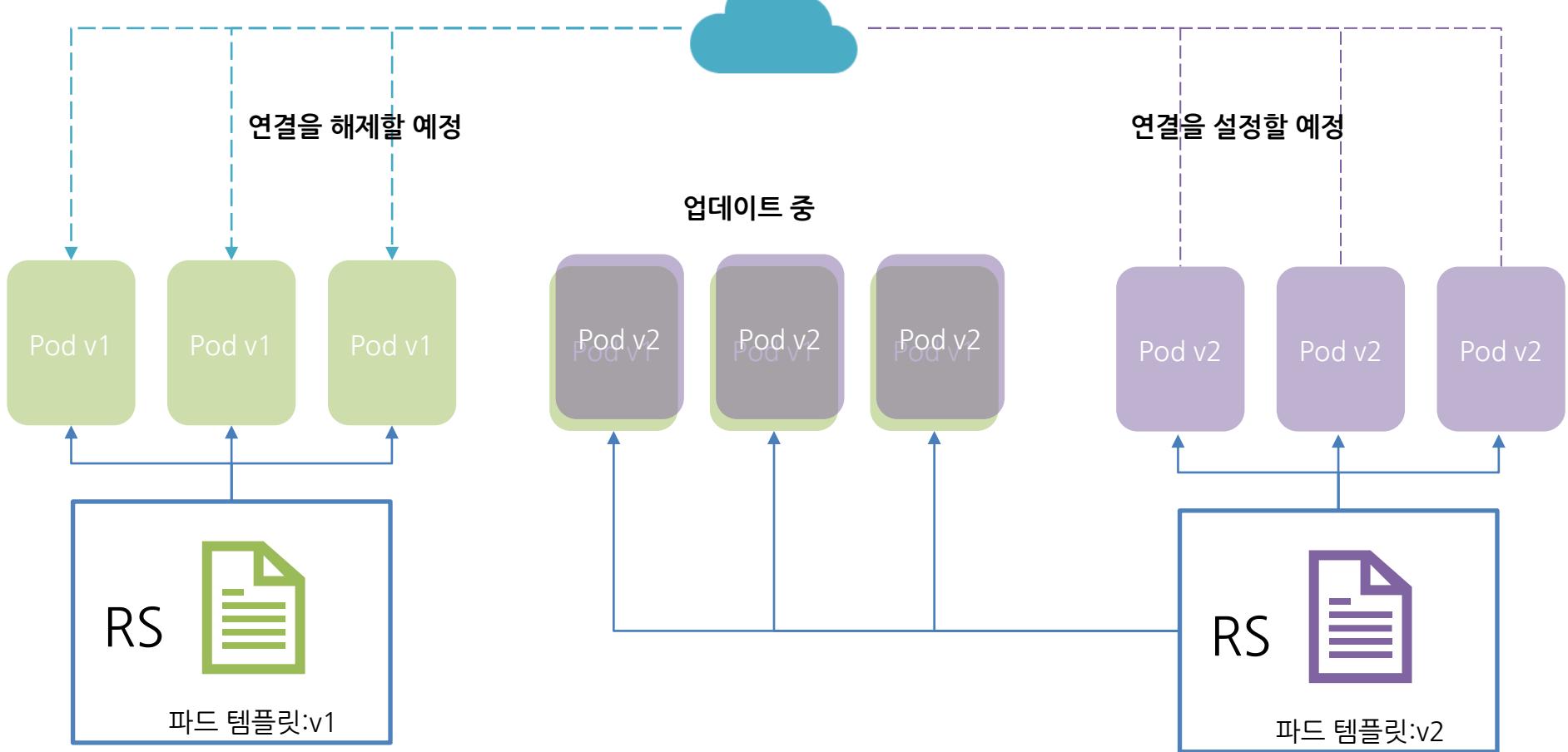
- 잠깐의 다운 타임 발생



애플리케이션 롤링 업데이트와 롤백

▶ 새로운 파드를 실행시키고 작업이 완료되면 오래된 파드를 삭제

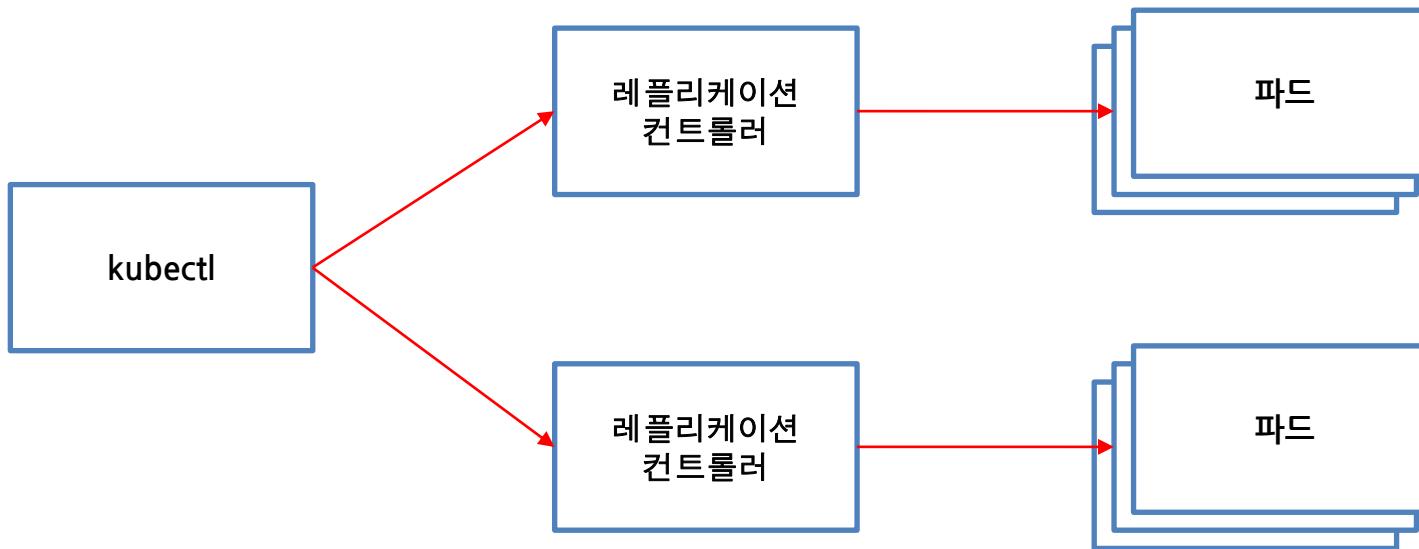
- 새 버전을 실행하는 동안 구 버전 파드와 연결
- 서비스의 레이블셀렉터를 수정하여 간단하게 수정가능



애플리케이션 롤링 업데이트와 롤백

▶ 레플리케이션컨트롤러가 제공하는 롤링 업데이트

- 이전에는 kubectl을 사용해 스케일링을 사용하여 수동으로 롤링 업데이트 진행 가능
- Kubectl 중단되면 업데이트는 어떻게 될까?
- 레플리케이션컨트롤러 또는 레플리카셋을 통제할 수 있는 시스템이 필요



애플리케이션 롤링 업데이트와 롤백

▶ 디플로이먼트 생성

- 레이블 셀렉터, 원하는 복제본 수, 파드 템플릿
- 디플로이먼트의 전략을 yaml에 지정하여 사용 가능
- 먼저 업데이트 시나리오를 위해 3개의 도커 이미지를 준비

- gasbugs/http-go:v1
- gasbugs/http-go:v2
- gasbugs/http-go:v3

dockerfiles

```
FROM golang:1.11
WORKDIR /usr/src/app
COPY main /usr/src/app
CMD ["/usr/src/app/main"]
```

```
package main
import (
    "fmt"
    "github.com/julienschmidt/httprouter"
    "net/http"
    "log"
)

func Index(w http.ResponseWriter, r *http.Request, _ httprouter.Params) {
    fmt.Fprint(w, "Welcome! v1\n")
}

func main() {
    router := httprouter.New()
    router.GET("/", Index)

    log.Fatal(http.ListenAndServe(":8080", router))
}
```

main.go

애플리케이션 롤링 업데이트와 롤백

▶ 디플로이먼트 생성 YAML 만들기

- 버전을 이름에 넣을 필요가 없음
(업데이트 되어도 동일한 디플로이먼트를 사용)

```
$ kubectl create -f http-go-deployment.yaml --record=true  
deployment.apps/http-go created
```

```
$ kubectl get deployment
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
http-go	3	3	3	3	6m15s

```
$ kubectl get rs
```

NAME	DESIRED	CURRENT	READY	AGE
http-go-6f8b8f95db	3	3	3	6m2s

```
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
http-go-6f8b8f95db-j88dn	1/1	Running	0	6m6s
http-go-6f8b8f95db-n6gdt	1/1	Running	0	6m6s
http-go-6f8b8f95db-z8pg4	1/1	Running	0	6m6s

```
$ kubectl rollout status deployment http-go  
deployment "http-go" successfully rolled out
```

http-go-deployment.yaml

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: http-go-deployment  
  labels:  
    app: http-go  
spec:  
  replicas: 3  
  template:  
    metadata:  
      name: http-go  
      labels:  
        app: http-go  
    spec:  
      containers:  
      - image: gasbugs/http-go:v1  
        name: http-go
```

rollout을 통해서도 상태 확인 가능

애플리케이션 롤링 업데이트와 롤백

▶ 디플로이먼트 업데이트 전략(StrategyType)

● Rolling Update(기본값)

- 오래된 파드를 하나씩 제거하는 동시에 새로운 파드 추가
- 요청을 처리할 수 있는 양은 그대로 유지
- 반드시 이전 버전과 새 버전을 동시에 처리 가능하도록 설계한 경우에만 사용해야 함

spec:

strategy:

type: RollingUpdate

● Recreate

- 새 파드를 만들기 전에 이전 파드를 모두 삭제
- 여러 버전을 동시에 실행 불가능
- 잠깐의 다운 타임 존재

● 업데이트 과정을 보기 위해 업데이트 속도 조절

```
$ kubectl patch deployment http-go -p '{"spec": {"minReadySeconds": 10}}'  
deployment.extensions/http-go patched
```

애플리케이션 롤링 업데이트와 롤백

▶ 디플로이먼트 업데이트 실행 준비

- 디플로이먼트를 모니터하는 프로그램 실행

```
$ while true; curl <ip>; sleep 1; done
Welcome! http-go:v1
...
...
```

애플리케이션 롤링 업데이트와 롤백

▶ 디플로이먼트 업데이트 실행

- 새로운 터미널을 열어 이미지 업데이트 실행

```
$ kubectl set image deployment http-go http-go=gasbugs/http-go:v2
deployment.extensions/http-go image updated
```

- 모니터링하는 시스템에서 관찰

```
$ while true; curl <ip>; sleep 1; done
```

```
Welcome! http-go:v1
```

```
...
```

```
Welcome! http-go:v1
```

```
Welcome! http-go:v2
```

```
Welcome! http-go:v2
```

```
Welcome! http-go:v1
```

애플리케이션 롤링 업데이트와 롤백

▶ 디플로이먼트 업데이트 실행 결과

- 업데이트한 이력을 확인

- 리비전의 개수는 디폴트로 10개까지 저장

```
$ kubectl rollout history deployment http-go
deployment.extensions/http-go
REVISION  CHANGE-CAUSE
1          <none>
2          kubectl set image deployment http-go http-go=gasbugs/http-go:v2
```

애플리케이션 롤링 업데이트와 롤백

▶ 롤백 실행하기

- 롤백을 실행하면 이전 업데이트 상태로 돌아감
- 롤백을 하여도 히스토리의 리비전 상태는 이전 상태로 돌아가지 않음

```
$ kubectl set image deployment http-go http-go=gasbugs/http-go:v3  
deployment.extensions/http-go image updated
```

```
$ kubectl rollout undo deployment http-go  
deployment.extensions/http-go
```

```
$ kubectl exec http-go-7dbcf5877-d6n6p curl 127.0.0.1:8080  
Welcome! http-go:v2
```

```
$ kubectl rollout undo deployment http-go --to-revision=1  
deployment.extensions/http-go
```

애플리케이션 롤링 업데이트와 롤백

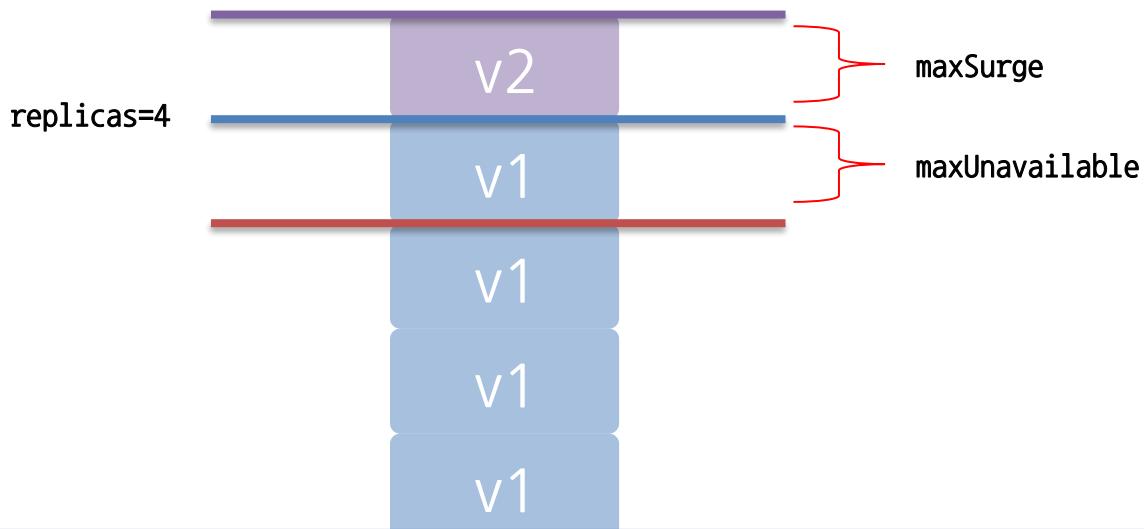
▶ 롤링 업데이터 전략 세부 설정

● maxSurge

- 기본값 25%, 개수로도 설정이 가능
- 최대로 추가 배포를 허용할 개수 설정
- 4개인 경우 25%이면 1개가 설정 (총 개수 5개까지 동시 파드 운영)

● maxUnavailable

- 기본값 25%, 개수로도 설정이 가능
- 동작하지 않는 파드의 개수 설정
- 4개인 경우 25%이면 1개가 설정(총 개수 4-1개는 운영해야 함)



```
spec:  
  strategy:  
    rollingUpdate:  
      maxSurge: 1  
      maxUnavailable: 1  
    type: RollingUpdate
```

애플리케이션 롤링 업데이트와 롤백

▶ 롤아웃 일시중지와 재시작

- 업데이트 중에 일시정지하길 원하는 경우

➤ \$ kubectl rollout pause deployment http-go

- 업데이트 일시중지 중 취소

➤ \$ kubectl rollout undo deployment http-go

- 업데이트 재시작

➤ \$ kubectl rollout resume deployment http-go

애플리케이션 롤링 업데이트와 롤백

▶ 업데이트를 실패한 경우

- 업데이트를 실패하는 케이스

- 부족한 할당량(Insufficient quota)
- 레디네스 프로브 실패(Readiness probe failures)
- 이미지 가져오기 오류(Image pull errors)
- 권한 부족(Insufficient permissions)
- 제한 범위(Limit ranges)
- 응용 프로그램 런타임 구성 오류(Application runtime misconfiguration)

- 업데이트를 실패하는 경우에는 기본적으로 600초 후에 업데이트를 중지한다.

```
spec:
```

```
  processDeadlineSeconds: 600
```

애플리케이션 롤링 업데이트와 롤백

▶ 연습문제

- 다음 mongo 이미지를 사용하여 업데이트와 롤백을 실행하라.

- 모든 revision 내용은 기록돼야 한다.
- mongo:4.2 이미지를 사용하여 deployment를 생성하라.
 - ✓ Replicas: 10
 - ✓ maxSurge: 50%
 - ✓ maxUnavailable: 50%
- mongo:4.4 롤링 업데이트를 수행하라.
- mongo:4.2로 롤백을 수행하라.

쿠버네티스 유저 관리

- 인증을 위한 다양한 리소스
- 유저와 서비스어카운트
- TLS 인증서를 활용한 통신 이해
- TLS 인증서를 활용한 유저 생성
- kube config 파일을 사용한 인증
- RBAC 기반 권한 관리



인증을 위한 다양한 리소스

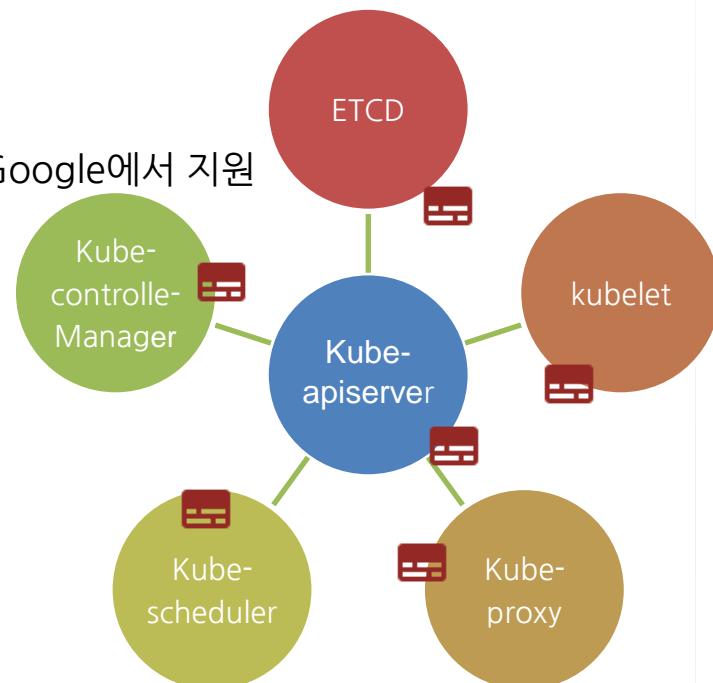
인증을 위한 다양한 리소스

▶ 쿠버네티스 인증 체계

- 모든 통신은 TLS로 대부분의 엑세스는 kube-apiserver를 통해서 통신해야 함

- 엑세스 가능한 유저

- X509 Client Certs
- Static Token File
- Putting a Bearer Token in a Request
- Bootstrap Tokens
- Service Account Tokens
- OpenID Connect Tokens: Azure Active Directory, Salesforce 및 Google에서 지원



인증을 위한 다양한 리소스

▶ 쿠버네티스 인증 체계

● 무엇을 할 수 있는가?

- RBAC Authorization: 조직 내의 개별 사용자의 역할에 따라 컴퓨터 또는 네트워크 리소스에 대한 액세스를 규제
- ABAC Authorization(속성 기반 액세스 제어): 속성을 결합하는 정책을 사용하여 사용자에게 액세스 권한을 부여
- Node Authorization: kubelets에서 만든 API 요청을 특별히 승인하는 특수 목적 권한 부여 모드
- WebHook Mode: HTTP 콜백, 특정 일이 발생할 때 URL에 메시지를 전달

유저와 서비스어카운트

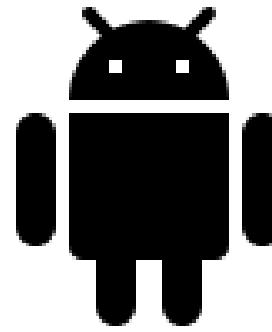
유저와 서비스어카운트

▶ 어카운츠에는 두 가지의 타입이 존재

- 유저: 일반 사용자를 위한 계정
- 서비스어카운트: 애플리케이션(파드 등)을 위한 계정



user



Service account

유저와 서비스어카운트

Static Token File

- Apiserver 서비스를 실행할 때 --token-auth-file=SOMEFILE.csv 전달 (kube-apiserver 수정 필요)
- API 서버를 다시 시작해야 적용됨
- 토큰, 사용자 이름, 사용자 uid, 선택적으로 그룹 이름 다음에 최소 3 열의 csv 파일



SOMEFILE.csv 파일 내용

```
password1,user1,uid001,"group1"  
password2,user2,uid002  
password3,user3,uid003  
password4,user4,uid004
```

유저와 서비스어카운트

Static Token File 을 적용했을 때 사용 방법

- HTTP 요청을 진행할 때 다음과 같은 내용을 헤더에 포함해야함
- Authorization: Bearer 31ada4fd-adec-460c-809a-9e56ceb75269



```
$ TOKEN=password1  
$ APISERVER=https://127.0.0.1:6443  
$ curl -X GET $APISERVER/api --header "Authorization: Bearer $TOKEN" --insecure
```

- Kubectl에 등록하고 사용하는 방법

```
$ kubectl config set-credentials user1 --token=password1  
$ kubectl config set-context user1-context --cluster=kubernetes \  
--namespace=frontend --user=user1  
$ kubectl get pod --user user1
```

유저와 서비스어카운트

연습문제

- 다음 csv 파일을 생성하고 apiserver에 토큰을 등록하여 재시작하자.

```
password1,user1,uid001,"group1"  
password2,user2,uid002  
password3,user3,uid003  
password4,user4,uid004
```

- 마스터 노드에 문제가 발생하는가? 어떤 서비스가 문제가 발생하는가?
- 문제가 발생하는 서비스의 컨테이너를 확인하기 위해 docker logs를 사용하고 그 해결 방법을 찾으라.
- 정상적으로 서비스가 시작됐다면 kubectl에 유저 정보를 등록하고 등록한 유저 권한으로 kubectl get pod 요청을 수행하라.
 - ✓ 반드시 Forbidden이라는 결과가 나와야 함 (유저는 생성됐으나 권한은 없다)

유저와 서비스어카운트

▶ 서비스 어카운트 만들기

- 명령어를 사용하여 serviceaccount sa1를 생성

```
kubectl create serviceaccount sa1
```



- sa와 함께 시크릿이 생성, 시크릿에는 토큰 값을 포함

```
$ kubectl get sa,secret
```

NAME	SECRETS	AGE
serviceaccount/default	1	14m
serviceaccount(sa1)	1	26s

NAME	TYPE	DATA	AGE
secret/default-token-n7nvk	kubernetes.io/service-account-token	3	14m
secret(sa1)-token-zgv85	kubernetes.io/service-account-token	3	26s

- 파드에 별도의 설정을 주지 않으면 default sa를 사용

유저와 서비스어카운트

파드에서 서비스 어카운트 사용하기

- Pod 에 spec.serviceAccount: <service-account-name>과 같은 형식으로 지정

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: nx
spec:
  serviceAccountName: sa1
  containers:
  - image: nginx
    name: nx
EOF
```

유저와 서비스어카운트

▶ 파드에서 서비스 어카운트 사용하기

- 생성된 파드에 sa1에 대한 시크릿에 잘 전달되었는지 확인 (자동 마운트됨)

```
$ kubectl get pod nx -o yaml
...
spec:
  containers:
    - image: nginx
      imagePullPolicy: Always
      name: nx
      resources: {}
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
        - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
          name: sa1-token-zgv85
          readOnly: true
  volumes:
    - name: sa1-token-zgv85
      secret:
        defaultMode: 420
        secretName: sa1-token-zgv85
```

유저와 서비스어카운트

파드에서 서비스 어카운트 사용하기

- 생성된 파드에 sa1에 대한 시크릿에 잘 전달되었는지 확인 (자동 마운트됨)

```
$ kubectl exec -it nx -- bash  
# ls /var/run/secrets/kubernetes.io/serviceaccount  
ca.crt  namespace  token
```

- 토큰 값을 변수에 전달하고 curl 명령을 실행

```
TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)  
curl -X GET https://$KUBERNETES_SERVICE_HOST/api --header "Authorization: Bearer $TOKEN" --insecure
```

- 실행 결과

```
{  
  "kind": "APIVersions",  
  "versions": [  
    "v1"  
  ],  
  "serverAddressByClientCIDRs": [  
    {  
      "clientCIDR": "0.0.0.0/0",  
      "serverAddress": "172.17.0.49:6443"  
    }  
  ]  
}
```

유저와 서비스어카운트

▶ 파드에서 서비스 어카운트 사용하기

- 파드에서 특정 네임스페이스에 리스트 파드 조회하기
- <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.22/#list-pod-v1-core>

Pod v1 core

- Write Operations
- Read Operations
- Read
- List
- List All Namespaces
- Watch
- Watch List
- Watch List All Namespa
- Status Operations
- EphemeralContainers
- ...
Default

HTTP Request

GET /api/v1/namespaces/{namespace}/pods

Path Parameters

Parameter	Description
namespace	object name and auth scope, such as for teams and projects

```
curl -X GET https://$KUBERNETES_SERVICE_HOST/api/v1/namespaces/default/pods -  
-header "Authorization: Bearer $TOKEN" -k
```

유저와 서비스어카운트

▶ 연습문제

- http-go라는 이름을 가진 ServiceAccount를 생성하고 http-go 파드를 생성해 http-go 서비스 어카운트를 사용하도록 설정하라.

TLS 인증서를 활용한 통신 이해

TLS 인증서를 활용한 통신 이해

SSL 통신 과정 이해

- 응용계층인 HTTP와 TCP계층사이에서 작동,
- 어플리케이션에 독립적 -> HTTP제어를 통한 유연성
- 데이터의 암호화 (기밀성), 데이터 무결성, 서버인증기능, 클라이언트 인증기능

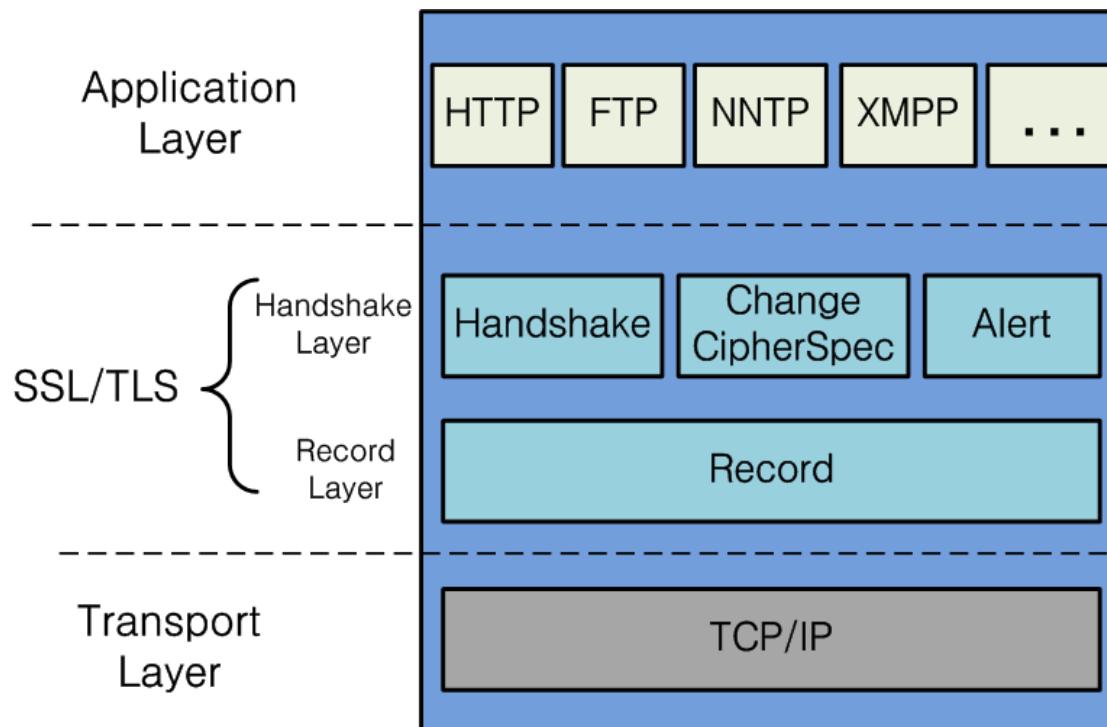


그림 출처: <https://soul0.tistory.com/510>

TLS 인증서를 활용한 통신 이해

Certificate를 보장하는 방법! 인증기관(CA)

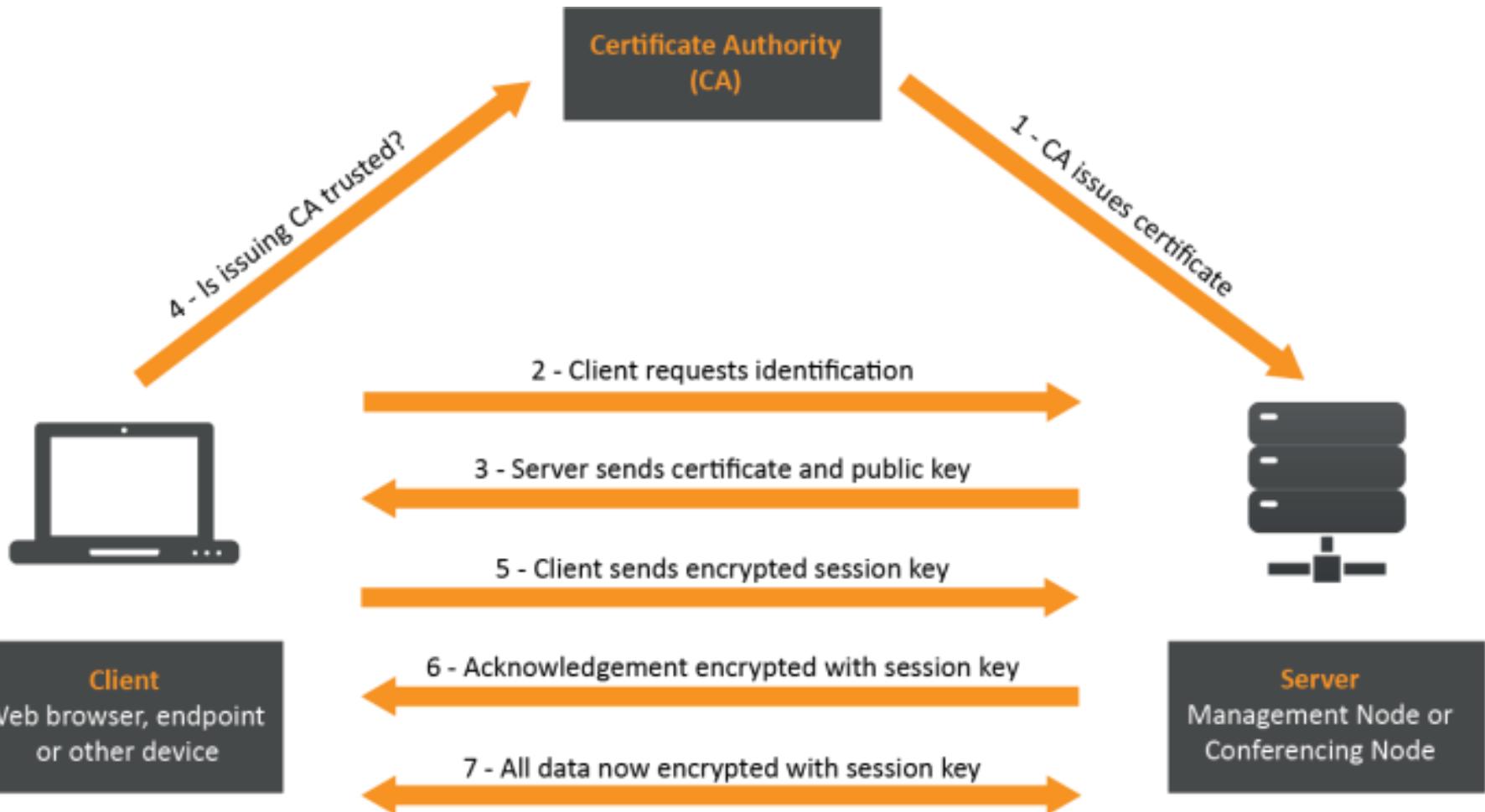


그림 출처: https://docs.pexip.com/admin/certificate_management.htm

TLS 인증서를 활용한 통신 이해

▶ kubernetes의 인증서 위치

```
$ sudo ls /etc/kubernetes/pki
apiserver-etcd-client.crt      apiserver.key      front-proxy-ca.key
apiserver-etcd-client.key      ca.crt            front-proxy-client.crt
apiserver-kubelet-client.crt   ca.key            front-proxy-client.key
apiserver-kubelet-client.key   etcd              sa.key
apiserver.crt                  front-proxy-ca.crt  sa.pub

$ sudo ls /etc/kubernetes/pki/etcd
ca.crt  healthcheck-client.crt    peer.crt  server.crt
ca.key   healthcheck-client.key   peer.key  server.key
```

TLS 인증서를 활용한 통신 이해

▶ 정확한 TLS 인증서 사용 점검

- 적절한 키를 사용하는지 확인하려면 manifests 파일에서 실행하는 certificate 확인 필요
- 적절한 키를 사용하지 않으면 에러가 발생

```
$ sudo ls /etc/kubernetes/manifests/  
etcd.yaml          kube-controller-manager.yaml  
kube-apiserver.yaml  kube-scheduler.yaml
```

TLS 인증서를 활용한 통신 이해

▶ 정확한 TLS 인증서 사용 점검

- Kubelet의 위치는 조금 다름

```
server1@MASTER:~$ sudo ls /var/lib/kubelet/pki
kubelet-client-2019-07-09-12-44-22.pem      kubelet.crt
kubelet-client-2019-07-09-12-44-54.pem      kubelet.key
kubelet-client-current.pem

server1@MASTER:~$ sudo ls /var/lib/kubelet/
config.yaml          kubeadm-flags.env   plugins           pods
cpu_manager_state    pki                  plugins_registry
device-plugins       plugin-containers  pod-resources
server1@MASTER:~$ sudo cat /var/lib/kubelet/config.yaml
address: 0.0.0.0
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 2m0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
```

TLS 인증서를 활용한 통신 이해

▶ TLS 인증서를 올바르지 않을 때 발생하는 현상 확인하기

- 먼저 현재 MASTER 노드의 상태를 스냅샷으로 저장하라.
- Kube-apiserver.yaml을 찾아 특정 경로를 수정하는 방식으로 certificate를 찾을 수 없도록 만들어라.

TLS 인증서를 활용한 통신 이해

▶ 인증서 정보 확인하기

```
$ sudo openssl x509 -in <certificate> -text
Certificate:
Data:
    Version: 3 (0x2)
    Serial Number:
        c4:70:38:32:90:74:e1:37:8a:77:36:5b:f4:39:06:a0
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN=8bc6e46e-289a-4e7a-a182-a9cd4cfa5ab8
    Validity
        Not Before: Jul  6 05:09:38 2019 GMT
        Not After : Jul  4 06:09:38 2024 GMT
    Subject: CN=8bc6e46e-289a-4e7a-a182-a9cd4cfa5ab8
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
        Modulus:
            00:b4:da:76:9f:0d:ef:08:60:32:d5:39:a6:99:51:
            8a:97:5b:7f:11:98:20:d9:b6:55:af:95:9d:aa:a7:
            fc:e1:68:a0:0d:31:64:69:d2:7f:cb:20:eb:8b:38:
            22:92:c7:e3:92:5d:b9:67:60:16:fe:2a:35:02:4d:
```

TLS 인증서를 활용한 통신 이해

▶ 인증서 목록

CERTIFICATE	RESIDUAL TIME	Certificate Path	CERTIFICATE AUTHORITY
admin.conf	365d	/etc/kubernetes	
apiserver	365d	/etc/kubernetes/pki	ca
apiserver-etcd-client	365d	/etc/kubernetes/pki/etcd	etcd-ca
apiserver-kubelet-client	365d	/etc/kubernetes/pki	ca
controller-manager.conf	365d	/etc/kubernetes/pki	
etcd-healthcheck-client	365d	/etc/kubernetes/pki/etcd	etcd-ca
etcd-peer	365d	/etc/kubernetes/pki/etcd	etcd-ca
etcd-server	365d	/etc/kubernetes/pki/etcd	etcd-ca
front-proxy-client	365d	/etc/kubernetes/pki	front-proxy-ca
scheduler.conf	365d	/etc/kubernetes	

▶ CA 목록

CERTIFICATE AUTHORITY	RESIDUAL TIME	Certificate Path
ca	10y	/etc/kubernetes/pki
etcd-ca	10y	/etc/kubernetes/pki/etcd
front-proxy-ca	10y	/etc/kubernetes/pki

참고: front-proxy 인증서는 kube-proxy에서 API 서버 확장을 지원할 때만 kube-proxy에서 필요하다.

TLS 인증서를 활용한 통신 이해

▣ 모든 인증서 갱신하기

- <https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-certs/>
- Check certificate expiration
 - kubeadm certs check-expiration
- Automatic certificate renewal
 - kubeadm은 컨트롤 플레인을 업그레이드하면 모든 인증서를 자동으로 갱신
- Manual certificate renewal
 - kubeadm certs renew all

TLS 인증서를 활용한 통신 이해

▶ 연습문제

- Apiserver가 사용하는 ca.crt 정보를 텍스트 형태로 출력하라.
- ca.crt의 CN은 무엇인가?
- ca.crt를 이슈화한 CN은 무엇인가?

TLS 인증서를 활용한 유저 생성하기

TLS 인증서를 활용한 유저 생성하기

▶ 1. ca를 사용하여 직접 csr 승인하기

- 개인 키 생성하기

- 길이 2048 만큼의 개인 키 생성

```
$ openssl genrsa -out gasbugs.key 2048
```

- private 키를 기반으로 인증서 서명 요청하기

- CN: 사용자 이름
 - O: 그룹 이름
 - CA에게 csr 파일로 인증을 요청할 수 있음!

```
$ openssl req -new -key gasbugs.key -out gasbugs.csr -subj  
"/CN=gasbugs/O=boanproject"
```

TLS 인증서를 활용한 유저 생성하기

▶ 1. ca를 사용하여 직접 csr 승인하기

- Kubernetes 클러스터 인증 기관(CA) 사용이 요청을 승인해야 함
- 내부에서 직접 승인하는 경우 pki 디렉토리에 있는 ca.key와 ca.crt를 사용하여 승인 가능
- gasbugs.csr을 승인하여 최종 인증서인 gasbugs.crt를 생성
- -days 옵션을 사용해 며칠간 인증서가 유효할 수 있는지 설정(예제에서는 500일)

```
$ openssl x509 -req -in gasbugs.csr -CA /etc/kubernetes/pki/ca.crt -CAkey /etc/kubernetes/pki/ca.key -CAcreateserial -out gasbugs.crt -days 500
```

```
Signature ok  
subject=CN = gasbugs, O = boanproject  
Getting CA Private Key
```

TLS 인증서를 활용한 유저 생성하기

2. 인증 사용을 위해 쿠버네티스에 crt를 등록

- crt를 사용할 수 있도록 kubectl을 사용하여 등록

```
$ kubectl config set-credentials gasbugs --client-certificate=.certs/gasbugs.crt --client-key=.certs/gasbugs.key  
$ kubectl config set-context gasbugs-context --cluster=kubernetes --namespace=office --user=gasbugs
```

- 다음 명령을 사용하여 사용자 권한으로 실행 가능(지금은 사용자에게 권한을 할당하지 않아 실행되지는 않는다)

```
$ kubectl --context=gasbugs-context get pods  
Error from server (Forbidden): pods is forbidden: User "gasbugs" cannot list resource "pods" in API group "" in the namespace "office"
```

TLS 인증서를 활용한 유저 생성하기

▶ 연습문제

- dev1팀에 john이 참여했다. John을 위한 인증서를 만들고 승인해보자.

kube config 파일을 사용한 인증

kube config 파일을 사용한 인증

▶ kube config 파일 확인하기

- \$ kubectl config view
- \$ kubectl config view --kube config=<config file>

```
$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://10.0.2.10:6443
    name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
    name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes # 현재 kubectl이 사용중인 쿠버네티스 계정
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

kube config 파일을 사용한 인증

▶ kube config의 구성

- ~/.kube/config 파일을 확인하면 세 가지 부분으로 작성됨
- clusters: 연결할 쿠버네티스 클러스터의 정보 입력
- users: 사용할 권한을 가진 사용자 입력
- contexts: cluster와 user를 함께 입력하여 권한 할당

```
$ curl https://kube-api-server:6443/api/v1/pods\  
--key user.key \  
--cert user.crt \  
--cacert ca.crt \  

```

Clusters

```
name: kube-cluster  
cluster:  
  certificate-authority: ca.crt  
  server: https://cluster-ip:6443
```

Contexts

```
name: user-name@kube-cluster  
context:  
  cluster: kube-cluster  
  user: user-name
```

Users

```
name: user-name  
user:  
  client-certificate: user.crt  
  client-key: user.key
```

kube config 파일을 사용한 인증

▶ 인증 사용자 바꾸기

- `kubectl config use-context user@kube-cluster`

▶ 인증 테스트

- `kubectl get pod`
- `forbidden`이 나오면 성공!

kube config 파일을 사용한 인증

연습문제

- john을 유저로 사용할 수 있도록 세팅하라.

RBAC 기반 권한 관리

RBAC 기반 권한 관리

▶ 역할 기반 액세스 제어 (RBAC)

- 기업 내에서 개별 사용자의 역할을 기반으로 컴퓨터, 네트워크 리소스에 대한 액세스를 제어
- rbac.authorization.k8s.io API를 사용하여 정의
- 권한 결정을 내리고 관리자가 Kubernetes API를 통해 정책을 동적으로 구성
- RBAC를 사용하여 룰을 정의하려면 apiserver에 --authorization-mode=RBAC 옵션이 필요
- RBAC를 다루는 이 API는 총 4가지의 리소스를 컨트롤
 - Role
 - RoleBinding
 - ClusterRole
 - ClusterRoleBinding

룰

롤바인딩

클러스터 룰

클러스터
롤바인딩

RBAC 기반 권한 관리

rbac.authorization.k8s.io API

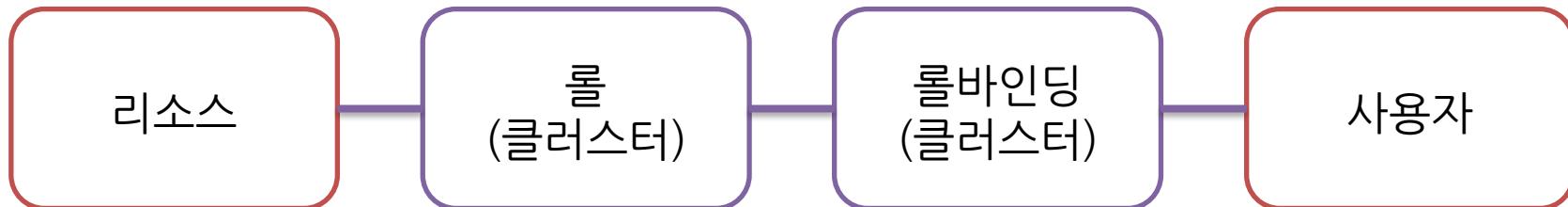
● 롤과 롤바인딩의 차이

➤ 롤

- ✓ “누가”하는 것인지는 정하지 않고 룰만을 정의
- ✓ 일반 룰의 경우에는 네임스페이스 단위로 역할을 관리
- ✓ 클러스터룰은 네임스페이스 구애 받지 않고 특정 자원을 전체 클러스터에서 자원을 관리할 룰을 정의

➤ 롤 바인딩

- ✓ “누가”하는 것인지만 정하고 룰은 정의하지 않음
- ✓ 룰을 정의하는 대신에 참조할 룰을 정의 (roleRef)
- ✓ 어떤 사용자에게 어떤 권한을 부여할지 정하는(바인딩) 리소스
- ✓ 클러스터룰에는 클러스터룰바인딩, 일반 룰에는 룰바인딩을 필요



RBAC 기반 권한 관리

▶ rbac.authorization.k8s.ioAPI

- 룰(Rule) 작성 요령

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: office
  name: deployment-manager
rules:
- apiGroups: [ "", "extensions", "apps" ]
  resources: [ "deployments", "replicasets", "pods" ]
  verbs: [ "get", "list", "watch", "create", "update", "patch", "delete" ]
```

- 룰에는 api 그룹, 리소스, 작업 가능한 동작을 작성

- API를 사용하게 할 그룹 정의

RBAC 기반 권한 관리

▶ 사용자 권한 할당

- 롤바인딩을 사용하여 office 네임스페이스 권한 할당
- kubectl create -f 를 사용하여 생성
- 해당 네임 스페이스에 모든 권한을 생성

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: deployment-manager-binding
  namespace: office
subjects:
- kind: User
  name: gasbugs
  apiGroup: ""
roleRef:
  kind: Role
  name: deployment-manager
  apiGroup: ""
```

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: office
  name: deployment-manager
rules:
- apiGroups: [ "", "extensions", "apps" ]
  resources: [ "deployments", "replicasets", "pods" ]
  verbs: [ "get", "list", "watch", "create", "update", "patch", "delete" ]
```

RBAC 기반 권한 관리

▶ 사용자 권한 테스트

- 실행돼야 하는 명령어

```
$ kubectl --context=gasbugs-context get pods -n office  
$ kubectl --context=gasbugs-context run --generator=pod-run/v1 nginx --  
image=nginx -n office
```

- 실행되면 안되는 명령어

```
$ kubectl --context=gasbugs-context get pods  
$ kubectl --context=gasbugs-context run --generator=pod-run/v1 nginx --  
image=nginx
```

RBAC 기반 권한 관리

▶ API 도큐먼트

- <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.19/>

The screenshot shows the 'API OVERVIEW' page of the Kubernetes documentation. On the left, there's a sidebar with a navigation menu:

- Overview
- API Groups
- WORKLOADS APIS
 - Container v1 core
 - CronJob v1beta1 batch
 - DaemonSet v1 apps
 - Deployment v1 apps
 - Job v1 batch
 - Pod v1 core
 - ReplicaSet v1 apps
 - ReplicationController v1 core
 - StatefulSet v1 apps
- SERVICE APIS

The main content area has the following sections:

- Copyright 2016-2020 The Kubernetes Authors.**
- Generated at: 2020-10-19 15:00:27 (CST)**
- API Version: v1.19.0**
- ## API OVERVIEW

Welcome to the Kubernetes API. You can use the Kubernetes API to read and write Kubernetes resource objects via a Kubernetes API endpoint.
- ### Resource Categories

This is a high-level overview of the basic types of resources provided by the Kubernetes API and their primary functions.

 - Workloads** are objects you use to manage and run your containers on the cluster.
 - Discovery & LB** resources are objects you use to "stitch" your workloads together into an externally accessible, load-balanced Service.
 - Config & Storage** resources are objects you use to inject initialization data into your applications, and to persist data that is external to your container.
 - Cluster** resources define how the cluster itself is configured; these are typically used only by cluster operators.

▶ 연습문제

- john 유저에게 dev1 네임스페이스에 대한 파드, 레플리카셋, 디플로이먼트를 조회, 생성하고 삭제할 수 있는 권한을 부여하라.

데이터 관리를 위한 엘라스틱서치 기초

▶ 엘라스틱서치 소개와 설치

▶ 데이터 입력과 조회

▶ 배치 프로세스

▶ 검색 API

▶ KIBANA 소개와 설치

▶ KIBANA 데이터 준비

▶ KIBANA 시각화

▶ 파일 비트를 활용한 아파치 서버 로그 수집



엘라스틱서치 소개와 설치

▶ 엘라스틱서치!

- 확장성이 뛰어난 오픈 소스 전체 텍스트 검색 및 분석 엔진
- 대량의 데이터를 신속하고 거의 실시간으로 저장, 검색 및 분석
- 일반적으로 복잡한 검색 기능과 요구 사항이 있는 응용 프로그램을 구동하는 기본 엔진 / 기술



elasticsearch.

■ 사용 사례

- 고객이 판매하는 제품을 검색 할 수 있는 **온라인 웹 스토어를 운영**
 - 이 경우 Elasticsearch를 사용하여 전체 제품 카탈로그 및 인벤토리를 저장하고 검색 및 자동 완성을 제공

- **로그 또는 트랜잭션 데이터를 수집, 분석 및 조사하여 추세, 통계, 요약 또는 예외를 탐지**
 - 이 경우 Logstash (Elasticsearch / Logstash / Kibana 스택의 일부)를 사용하여 데이터를 수집, 집계 및 구문 분석 한 다음 Logstash에 이 데이터를 Elasticsearch에 제공
 - 데이터가 Elasticsearch에 저장되면 검색 및 집계를 실행하여 관심 있는 정보를 검색



■ 사용 사례

- 가격에 정통한 고객이 "특정 전자 장치를 구입하는 데 관심이 있으며 다음 달의 모든 공급 업체 가겟 가격이 \$ X 이하로 떨어지면 알림을 받고 싶습니다"와 같은 규칙을 지정할 수 있는 **가격 알림 플랫폼을 운영**
 - 이 경우 공급 업체 가격을 깎아내어 Elasticsearch로 밀어 넣은 뒤
 - 역방향 검색 (Percolator) 기능을 사용하여 가격 변동을 고객 쿼리와 비교하고 일치 항목이 발견
 - 마지막으로 고객에게 알리미를 전송
- 분석 / 비즈니스 인텔리전스 요구 사항이 있으며 **많은 데이터에 대해 신속하게 조사, 분석, 시각화 및 임시 질문을 하고 싶습니다** (수억 또는 수십억 건의 레코드를 생각하십시오).
 - Elasticsearch 를 활용해 데이터를 저장 한 다음 Kibana (Elasticsearch / Logstash / Kibana 스택의 일부)를 사용하여 중요한 데이터를 시각화 할 수 있는 사용자 정의 대시 보드를 작성
 - 또한 Elasticsearch 집계 기능을 사용하여 데이터에 대해 복잡한 비즈니스 인텔리전스 쿼리를 수행



Elasticsearch의 핵심 개념

Near Realtime (NRT)	Elasticsearch는 거의 실시간 검색 플랫폼 문서를 색인할 때부터 검색 가능할 때까지 약간의 대기 시간 (일반적으로 1 초)이 매우 짧음
클러스터 (Cluster)	전체 데이터를 함께 보유하고 모든 노드에서 연합 인덱싱 및 검색 기능을 제공하는 하나 이상의 노드 (서버) 모음 클러스터는 기본적으로 "elasticsearch"라는 고유 한 이름으로 식별 이 이름은 노드가 이름으로 클러스터에 참여하도록 설정된 경우 노드가 클러스터의 일부일 수 있기 때문에 중요
노드 (Node)	노드는 클러스터의 일부이며 데이터를 저장하고 클러스터의 인덱싱 및 검색 기능에 참여하는 단일 서버 단일 클러스터에서 원하는 만큼의 노드를 소유 가능 또한 현재 네트워크에서 실행중인 다른 Elasticsearch 노드가 없는 경우 단일 노드를 시작하면 기본적으로 elasticsearch라는 새로운 단일 노드 클러스터가 형성
색인 (Index)	색인은 다소 유사한 특성을 갖는 문서의 콜렉션 예를 들어, 고객 데이터에 대한 색인, 제품 카탈로그에 대한 또 다른 색인 및 주문 데이터에 대한 또 다른 색인을 가질 수 있음 색인은 이름(모두 소문자 여야 함)로 식별되며 이 이름은 색인 된 문서를 색인 작성, 검색, 갱신 및 삭제할 때 색인을 참조하는 데 사용



▶ Elasticsearch의 핵심 개념

Type	사용자가 하나의 유형, 블로그 게시물을 다른 유형과 같이 여러 Type의 문서를 동일한 색인에 저장할 수 있도록 색인의 논리적 범주 / 파티션으로 사용되는 유형 더 이상 인덱스에 여러 유형을 작성할 수 없으며 이후 버전에서는 Type의 전체 개념이 제거됩니다.
Documents	문서는 색인을 생성 할 수 있는 기본 정보 단위 예를 들어, 단일 고객에 대한 문서, 단일 제품에 대한 다른 문서 및 단일 주문에 대한 문서를 보유 JSON (JavaScript Object Notation)으로 표현
RESTful API	URI를 사용한 동작이 가능 HTTP 프로토콜로 JSON 문서의 입출력과 다양한 제어 JSON 문서의 입출력과 다양한 제어



Elasticsearch 다운로드

- <https://www.elastic.co/kr/downloads/elasticsearch>

The screenshot shows the Elasticsearch Downloads page. At the top, there is a navigation bar with the Elastic logo, links for 제품 (Products), Cloud, 서비스 (Services), 고객 (Customers), and 알아보기 (Learn More). On the right side of the header are buttons for 다운로드 (Download), 문의 (Inquiry), a search icon, and a KR (Korean) language switcher.

The main content area has a heading "Downloads" and a sub-section titled "Download Elasticsearch". Below this, a call-to-action box contains the text "Want it hosted? Deploy on Elastic Cloud. [Get Started »](#)".

Below the call-to-action, detailed information is provided for version 7.1.1:

- Version: 7.1.1
- Release date: May 29, 2019
- License: [Elastic License](#)
- Downloads:
 - WINDOWS sha.asc ([Download](#))
 - MACOS sha.asc ([Download](#))
 - LINUX sha.asc ([Download](#))
 - DEB sha.asc ([Download](#))
 - RPM sha.asc ([Download](#))
 - MSI (BETA) sha.asc ([Download](#))

▶ 도커를 활용한 Elasticsearch 설치하기

- https://hub.docker.com/_/elasticsearch
- 우분투에 docker.io를 설치하고 엘라스틱서치와 키바나를 설치

```
sudo apt update && sudo apt install -y docker.io
docker network create elastic
docker run -d --name es01-test --net elastic -p 9200:9200 -p 9300:9300 -e
"discovery.type=single-node" elasticsearch:7.14.1
docker run -d --name kib01-test --net elastic -p 5601:5601 -e
"ELASTICSEARCH_HOSTS=http://es01-test:9200" kibana:7.14.1
```

- 해당 이미지의 9200번 포트로 HTTP 접속



A screenshot of a web browser window displaying the Elasticsearch version information. The URL in the address bar is 192.168.179.140:9200. The page content is a JSON object representing the Elasticsearch configuration:

```
{
  "name": "2e38049d0554",
  "cluster_name": "docker-cluster",
  "cluster_uuid": "vKtkRqQPSDWs0tp4IY6hQQ",
  "version": {
    "number": "7.14.1",
    "build_flavor": "default",
    "build_type": "docker",
    "build_hash": "66b55ebfa59c92c15db3f69a335d500018b3331e",
    "build_date": "2021-08-26T09:01:05.390870785Z",
    "build_snapshot": false,
    "lucene_version": "8.9.0",
    "minimum_wire_compatibility_version": "6.8.0",
    "minimum_index_compatibility_version": "6.0.0-beta1"
  },
  "tagline": "You Know, for Search"
}
```



▶ Elasticsearch 도커 설치

- 도커 현재 상태 및 로그 확인 방법

```
docker ps -a  
docker logs es01-test  
docker logs kib01-test
```

- 도커 컨테이너 종료 방법

```
docker stop es01-test # 시작할 때는 start 명령을 사용  
docker stop kib01-test
```

- 도커 컨테이너 삭제 방법

```
docker network rm elastic  
# 컨테이너 삭제  
docker rm es01-test  
docker rm kib01-test  
# 이미지 삭제  
docker rmi docker.elastic.co/elasticsearch/elasticsearch:7.15.0  
docker rmi docker.elastic.co/kibana/kibana:7.15.0
```



데이터 입력과 조회

▶ REST API

- 웹의 창시자 (HTTP) 중의 한 사람인 Roy Fielding 의 2000년 논문에 의해서 소개
 ”웹의 장점을 최대한 활용할 수 있는 네트워크 기반의 아키텍쳐”
- 엘라스틱서치 노드와 통신하는 방법
- 클러스터와 상호 작용하는 데 사용할 수 있는 매우 포괄적이고 강력한 REST API를 제공
- API로 수행 할 수 있는 몇 가지 작업
 - 클러스터, 노드 및 색인 상태, 상태 및 통계 확인
 - 클러스터, 노드 및 색인 데이터 및 메타 데이터 관리
 - 데이터 입력과 검색 (Create, Read, Update, Delete) 및 인덱스에 대한 검색 작업 수행
 - 페이징, 정렬, 필터링, 스크립팅, 집계 및 기타 여러 고급 검색 작업 실행



▶ REST API 구성요소 3가지

- 리소스, 메서드, 메시지



▶ 클러스터 상태(Health)

- 클러스터가 어떻게 진행되고 있는지 기본적인 확인
- 우리는 curl을 사용하여 이를 수행
- HTTP/REST 호출을 수행 할 수 있는 모든 도구를 사용 가능
- 클러스터 상태를 확인하기 위해 _cat API를 사용
- **녹색** - 모든 것이 좋음(클러스터가 완전히 작동 함).
- **노란색** - 모든 데이터를 사용할 수 있지만 일부 복제본은 아직 할당되지 않음(클러스터는 완전히 작동

```
GET /_cat/nodes?v
```

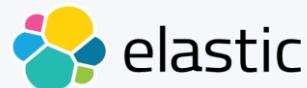
- **빨간색** - 어떤 이유로든 일부 데이터를 사용할 수 없음(클러스터가 부분적으로 작동 함).



▶ 데이터베이스가 가진 데이터 확인하기

- 갖고 있는 모든 인덱스 항목 조회
- index는 일반 RDB에서의 데이터베이스의 역할

```
GET /_cat/indices?v
```



▶ 엘라스틱 데이터베이스의 인덱싱 방식

문서	문서 내용
Doc 1	blue sky green land red sun
Doc 2	blue ocean green land
Doc 3	red flower blue sky

일반적인 관계형 DB
테이블에서 텍스트 저장

검색어	검색어가 가리키는 문서
blue	Doc1, Doc2, Doc3
sky	Doc1, Doc3
green	Doc1, Doc2
land	Doc1, Doc2
red	Doc1, Doc3
ocean	Doc2
flower	Doc3
sun	Doc1

색인 공간에 저장된
역파일 색인 데이터 구조



HTTP 메서드와 데이터 입력과 검색, SQL을 비교

HTTP 메서드	데이터 입력과 검색	SQL
GET	Read	Select
PUT	Update	Update
POST	Create	Insert
DELETE	Delete	Delete



elasticsearch.

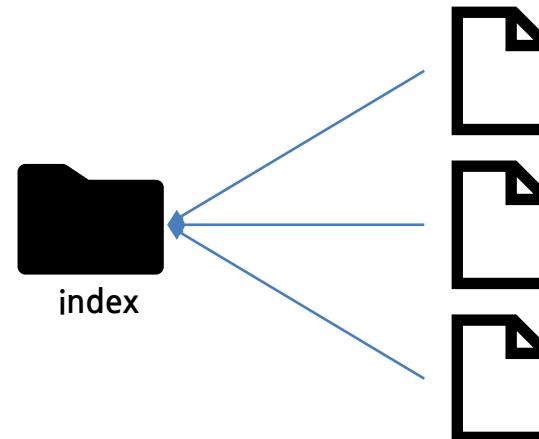


▶ 엘라스틱서치 데이터 처리

- 엘라스틱서치의 데이터 구조

- 인덱스(Index), 도큐먼트(Document)의 단위를 가짐
- 도큐먼트는 엘라스틱서치의 데이터가 저장되는 최소 단위
- 여러 개의 도큐먼트는 하나의 인덱스로 구성

관계형 DB	엘라스틱서치
데이터베이스(Database)	인덱스(Index)
테이블(Table)	타입(Type, 사라질 예정)
열(Row)	도큐먼트(Document)
행(Column)	필드(Field)
スキ마	매핑(Mapping)



▶ 엘라스틱서치의 질의 방법

- 커맨드라인의 curl 명령어 사용
- postman 응용프로그램 사용
- Kibana에서 devtool 사용

curl 도움말

```
C:\Windows\system32\cmd.exe
C:\Users\Administrator>curl --help
Usage: curl [options...] <url>
Options: (H) means HTTP/HTTPS only, (F) means FTP only
  --anyauth      Pick `any` authentication method (H)
  -a, --append    Append to target file when uploading (F/SFTP)
  --basic        Use HTTP Basic Authentication (H)
  --cacert FILE  CA certificate to verify peer against (SSL)
  --capath DIR   CA directory to verify peer against (SSL)
  -E, --cert CERT[:PASSWD] Client certificate file and password (SSL)
  --cert-status  Verify the status of the server certificate (SSL)
  --cert-type TYPE Certificate file type (DER/PEM/ENG) (SSL)
  --ciphers LIST  SSL ciphers to use (SSL)
  --compressed   Request compressed response (using deflate or gzip)
  -K, --config FILE Read config from FILE
  --connect-timeout SECONDS Maximum time allowed for connection
  --connect-to HOST1:PORT1:HOST2:PORT2 Connect to host (network level)
  -C, --continue-at OFFSET Resumed transfer OFFSET
```



Kibana의 devtool

The Console UI is split into two panes: an editor pane (left) and a response pane (right). Use the editor to type requests and submit in the response pane on the right side.

Console understands requests in a compact format, similar to cURL:

```
1 # index a doc
2 PUT index/type/1
3 {
4   "body": "here"
5 }
6
7 # and get it ...
8 GET index/type/1
```

While typing a request, Console will make suggestions which you can then accept by hitting Enter/Tab. These suggestions are made for indices and types.

Postman 프로그램

Postman

File Edit View Collection History Help

Builder Team Library IN SYNC

localhost:9200/books

GET localhost:9200/books/book/1 Send

Authorization Headers Body Pre-request Script Tests

Type No Auth

Response

데이터 입력/조회/삭제/업데이트 요약

데이터 처리	메서드	구문
입력	PUT	http://localhost:9200/index1/type1/1 -d '{"num":1, "name":"Ilson Choi"}'
조회	GET	http://localhost:9200/index1/type1/1
삭제	DELETE	http://localhost:9200/index1/type1/1
업데이트	POST	http://localhost:9200/index1/type1/1/_update -d '{doc: {"age":99} }'

* 6.x 이상에서는 POST와 PUT을 혼용



▶ 인덱스 만들기

- Customer라는 인덱스를 만들어보자.

```
PUT /customer?pretty  
GET /_cat/indices?v
```

```
<curl 명령어>  
curl -X PUT "localhost:9200/customer?pretty"  
curl -X GET "localhost:9200/_cat/indices?v"
```

- 첫 번째 명령은 PUT 동사를 사용하여 "customer"라는 색인을 작성
- JSON 응답 (있을 경우)을 예쁜 것으로 인쇄하도록 호출의 끝 부분에 간단히 추가



인덱스 만들기

- Customer라는 인덱스를 만들어보자.

health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
yellow	open	.kibana	J4MMW9DRRm6phUQH6IdDpw	1	1	1	0	3.1kb	3.1kb
yellow	open	customer	/dzyNhbpRIeNsWUE199sMQ	5	1	0	0	650b	650b

인덱스는 두 개

기본 조각 5개
복제본 1개
문서 0개

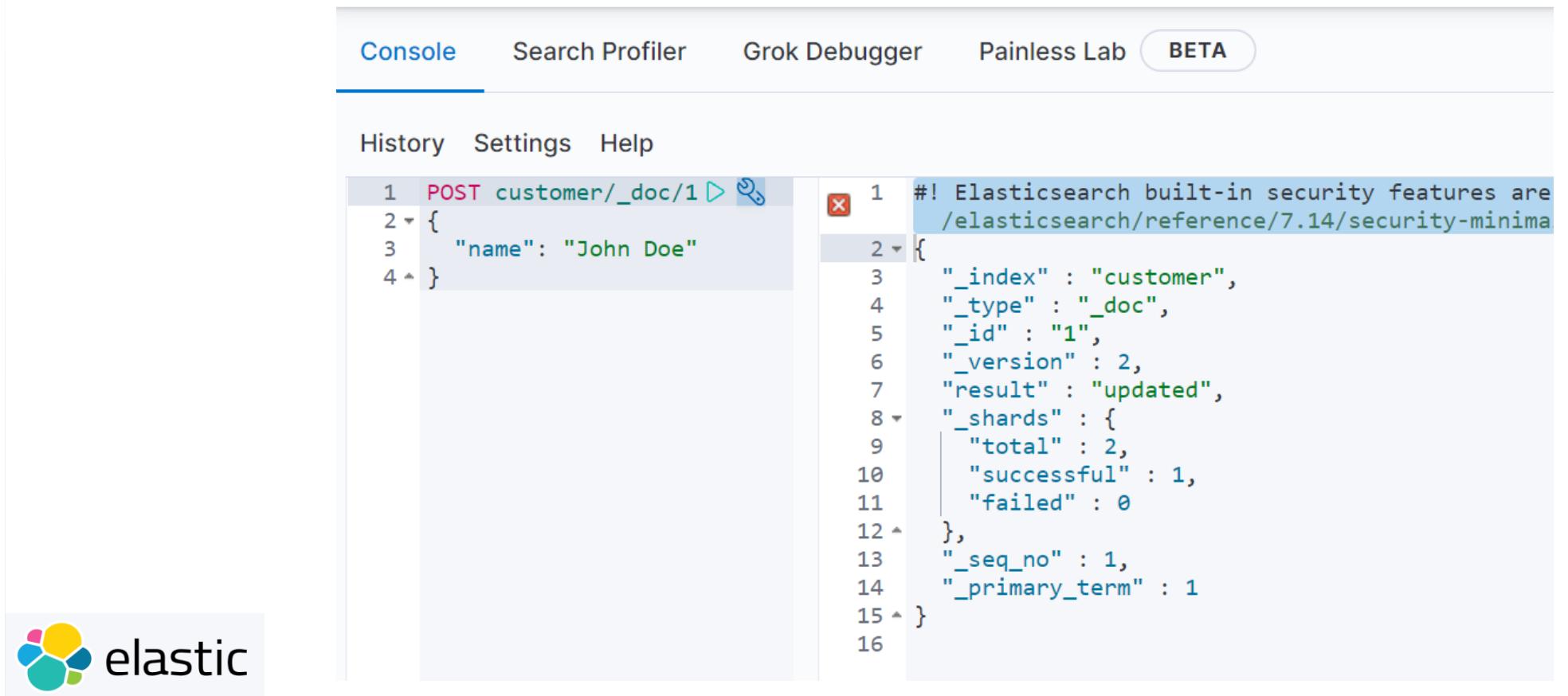
복제본이 다른 노드에 복사되면
초록으로 변경



▶ 도큐먼트 입력과 조회

- costumer 인덱스에 데이터 입력

➤ ID가 1인 간단한 고객 문서를 고객 색인으로 색인화



The screenshot shows the Elasticsearch Dev Tools interface with the 'Console' tab selected. The left panel displays a POST request to the 'customer/_doc/1' index with the following JSON payload:

```
1 POST customer/_doc/1
2 {
3   "name": "John Doe"
4 }
```

The right panel shows the response from Elasticsearch, which includes the document's metadata and the result of the update operation:

```
1 #! Elasticsearch built-in security features are
2 /elasticsearch/reference/7.14/security-minima
3 {
4   "_index" : "customer",
5   "_type" : "_doc",
6   "_id" : "1",
7   "_version" : 2,
8   "result" : "updated",
9   "_shards" : {
10     "total" : 2,
11     "successful" : 1,
12     "failed" : 0
13   },
14   "_seq_no" : 1,
15   "_primary_term" : 1
16 }
```

In the bottom left corner, there is an 'elastic' logo.

▶ 도큐먼트 입력과 조회

- 입력한 데이터 조회

History Settings Help

1 GET customer/_doc/1 ▶ 🔎

```
1 #! Elasticsearch built-in security features
  /elasticsearch/reference/7.14/security-m:
2 {
3   "_index": "customer",
4   "_type": "_doc",
5   "_id": "1",
6   "_version": 2,
7   "_seq_no": 1,
8   "_primary_term": 1,
9   "found": true,
10  "_source": {
11    "name": "John Doe"
12  }
13 }
14
```



인덱스 삭제

- 고객 색인에 뭔가 넣기

➤ 방금 작성한 색인을 삭제 한 다음 모든 색인을 다시 나열

```
DELETE /customer?pretty
GET /_cat/indices?v
```

➤ 응답은 인덱스가 성공적으로 삭제되었음을 의미

➤ 다음 단계로 넘어 가기 전에 지금까지 배웠던 몇 가지 API 명령을 확인

```
PUT /customer
PUT /customer/_doc/1
{
    "name": "John Doe"
}
GET /customer/_doc/1
DELETE /customer
```

1 {
2
3 }
"acknowledged": true

응답



▶ 여러 개의 데이터 입력

History Settings Help

```
1 POST books/_doc/1
2 {
3   "title": "Elasticsearch Guide",
4   "author": "Choi",
5   "date" : "2021-10-30",
6   "pages" : 500
7 }
```



```
1 #! Elasticsearch built-in security features are
2 #!           /guide/en/elasticsearch/reference/7.14/securi
3 {
4   "_index" : "books",
5   "_type" : "_doc",
6   "_id" : "1",
7   "_version" : 1,
8   "result" : "created",
9   "_shards" : {
10     "total" : 2,
11     "successful" : 1,
12     "failed" : 0
13   },
14   "_seq_no" : 0,
15   "_primary_term" : 1
16 }
```



데이터 조회

History Settings Help

1 GET books/_doc/1



```
1 #! Elasticsearch built-in security features are not enabled
2   /guide/en/elasticsearch/reference/7.14/security-minimal.html
3 {
4   "_index": "books",
5   "_type": "_doc",
6   "_id": "1",
7   "_version": 1,
8   "_seq_no": 0,
9   "_primary_term": 1,
10  "found": true,
11  "_source": {
12    "title": "Elasticsearch Guide",
13    "author": "Choi",
14    "date": "2021-10-30",
15    "pages": 500
16  }
17 }
```

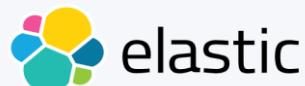


▶ 도큐먼트 삭제

- 문서를 삭제하는 것은 매우 간단
- ID가 2 인 이전 고객을 삭제하는 방법

```
DELETE /customer/_doc/2?pretty
```

- 데이터 삭제 후 데이터 조회 시 found가 false임을 확인
- 데이터 삭제 시 특징
 - 메타데이터가 그대로 유지
 - 도큐먼트 삭제 후 다시 데이터를 입력하면 _version 값이 이어서 진행
 - 버전까지 초기화하려면 인덱스를 삭제해야 함



데이터 수정

- Elasticsearch는 거의 실시간으로 데이터 조작 및 검색 기능을 제공
- 기본적으로 데이터를 색인 / 업데이트 / 삭제할 때부터 검색 결과에 표시 될 때까지 1 초의 지연 (새로 고침 간격)을 기대
- ID를 고쳐 쓰면 모든 내용이 교체됨
- ID를 쓰지 않거나 다른 ID를 사용할 경우 새롭게 저장
- 6.x부터는 POST와 PUT을 혼용

```
PUT /customer/_doc/2?pretty
{
    "name": "Jane Doe"
}
```

```
POST /customer/_doc/?pretty
{
    "name": "Jane Doe"
}
```

도큐먼트 아이디를 명시하지
않으면 랜덤하게 생성



▶ 데이터 업데이트

- 입력된 도큐먼트를 수정하는 _update API 제공
- _update API의 두 개의 매개 변수인 doc와 source를 이용해서 데이터를 제어
 - doc : 도큐먼트에 새로운 필드를 추가하거나 기존 필드 값을 변경하는 데 사용
 - script : 프로그래밍 기법을 사용. 입력된 내용에 따라 필드의 값을 변경하는 등의 처리



데이터 업데이트 예제

필드 수정 or 추가

```
1 POST customer/_doc/1
2 {
3     "name": "John Doe"
4 }
5
6 POST customer/_update/1/
7 {
8     "doc" : {
9         "category" : "IT",
10        "pages" : 50
11    }
12 }
13
14 POST customer/_update/1/
15 {
16     "doc" : {
17         "author" : "CHOI"
18    }
19 }
20
21 POST customer/_update/1/
22 {
23     "script" : "ctx._source.pages+=50"
24 }
```

필드 수정 or 추가

script 필드 수정

결과

```
{
    "_index" : "customer",
    "_type" : "_doc",
    "_id" : "1",
    "_version" : 12,
    "_seq_no" : 18,
    "_primary_term" : 2,
    "found" : true,
    "_source" : {
        "name" : "John Doe",
        "pages" : 100,
        "category" : "IT",
        "author" : "CHOI"
    }
}
```



데이터 업데이트 예제

- script의 ctx.op 명령을 사용하여 필드 조건에 따라 도큐먼트 삭제

```
1 POST customer/_update/1/
2 {
3   "script": {
4     "source": "if(ctx._source.pages <= params
5       .page_cnt){ctx.op='delete'} else{ctx.op
6         ='none'}",
7     "params": {
8       "page_cnt": 150
9     }
10   }
11 }
```



```
{
  "_index": "customer",
  "_type": "_doc",
  "_id": "1",
  "_version": 13,
  "result": "deleted",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 19,
  "_primary_term": 2
}
```



데이터 입력과 조회 연습문제

- TourCompany의 고객관리를 위해 다음 데이터를 입력하십시오.
- Index 이름은 tourcompany로 사용

Doc Id	name	phone	holiday_dest	departure_date
1	Alfred	010-1234-5678	Disneyland	2017/01/20
2	Huey	010-2222-4444	Disneyland	2017/01/20
3	Naomi	010-3333-5555	Hawaii	2017/01/10
4	Andra	010-6666-7777	Bora Bora	2017/01/11
5	Paul	010-9999-8888	Hawaii	2017/01/10
6	Colin	010-5555-4444	Venice	2017/01/16



▶ 데이터 입력과 조회 연습문제

- 다음 임무를 수행하기 위해 쿼리문을 작성하고 데이터베이스에 적용하십시오.
 - BoraBora 여행은 공항테러 사태로 취소됐습니다. BoraBora 여행자의 명단을 삭제해주십시오.
 - Hawaii 단체 관광객의 요청으로 출발일이 조정됐습니다. 2017/01/10에 출발하는 Hawaii의 출발일을 2017/01/17일로 수정해주십시오.
 - 휴일 여행을 디즈니랜드로 떠나는 사람들의 핸드폰 번호를 조회하십시오.



배치 프로세스

▶ _bulk API

- 여러 작업을 일괄적으로 수행 할 수 있는 기능
- 이 기능은 최대한 적은 네트워크 왕복으로 가능한 빠르게 여러 작업을 수행
- 다음 예제에서는 일괄 작업으로 두 개의 문서 (ID 1 - John Doe 및 ID 2 - Jane Doe)를 인덱싱



▶ _bulk API 예제

- HTTP 바디 부분 끝에 반드시 엔터 추가 입력 필요

```
POST /customer/_bulk?pretty
{"index": {"_id": "1"}}
{"name": "John Doe" }
{"index": {"_id": "2"}}
{"name": "Jane Doe" }
[엔터]
```

- 다음 예는 첫 번째 문서 (ID 1)를 업데이트 한 다음 두 번째 문서 (ID 2)를 일괄 작업에서 삭제

```
POST /customer/_bulk?pretty
{"update": {"_id": "1"}}
{"doc": { "name": "John Doe becomes Jane Doe" } }
{"delete": {"_id": "2"} }
[엔터]
```



▶ _bulk API

- Bulk API는 작업 중 하나에서 실패해도 전체 기능을 중지하지 않음
- 대량 API가 반환되면 각 액션에 대한 상태가 전송된 순서대로 제공
- 특정 액션이 실패했는지 여부를 확인 가능

```
1 { "create" : { "_index": "books", "_type": "book" } }
2 { "title": "The Tempest", "author": "William Shakespeare", "category": "Comedies", "written": "1610-03-01T11:34:00",
  "pages" : 62, "sell" : 275600000, "plot": "Magician Prospero, rightful Duke of Milan, and his daughter, Miranda,
have been stranded for twelve years on an island after Prospero's jealous brother Antonio (aided by Alonso, the
King of Naples) deposed him and set him adrift with the then-3-year-old Miranda. Gonzalo, the King's counsellor,
had secretly supplied their boat with plenty of food, water, clothes and the most-prized books from Prospero's
library. Possessing magic powers due to his great learning, Prospero is reluctantly served by a spirit, Ariel,
whom Prospero had rescued from a tree in which he had been trapped by the witch Sycorax. Prospero maintains
Ariel's loyalty by repeatedly promising to release the \"airy spirit\" from servitude. Sycorax had been banished
to the island, and had died before Prospero's arrival. Her son, Caliban, a deformed monster and the only
non-spiritual inhabitant before the arrival of Prospero, was initially adopted and raised by him. He taught
Prospero how to survive on the island, while Prospero and Miranda taught Caliban religion and their own language.
Following Caliban's attempted rape of Miranda, he had been compelled by Prospero to serve as the magician's slave.
In slavery, Caliban has come to view Prospero as a usurper and has grown to resent him and his daughter. Prospero
and Miranda in turn view Caliban with contempt and disgust."}
```



▶ 업로드할 데이터 확인

- 샘플 데이터셋

```
{  
    "account_number": 0,  
    "balance": 16623,  
    "firstname": "Bradshaw",  
    "lastname": "Mckenzie",  
    "age": 29,  
    "gender": "F",  
    "address": "244 Columbus Place",  
    "employer": "Euron",  
    "email": "bradshawmckenzie@euron.com",  
    "city": "Hobucken",  
    "state": "CO"  
}
```

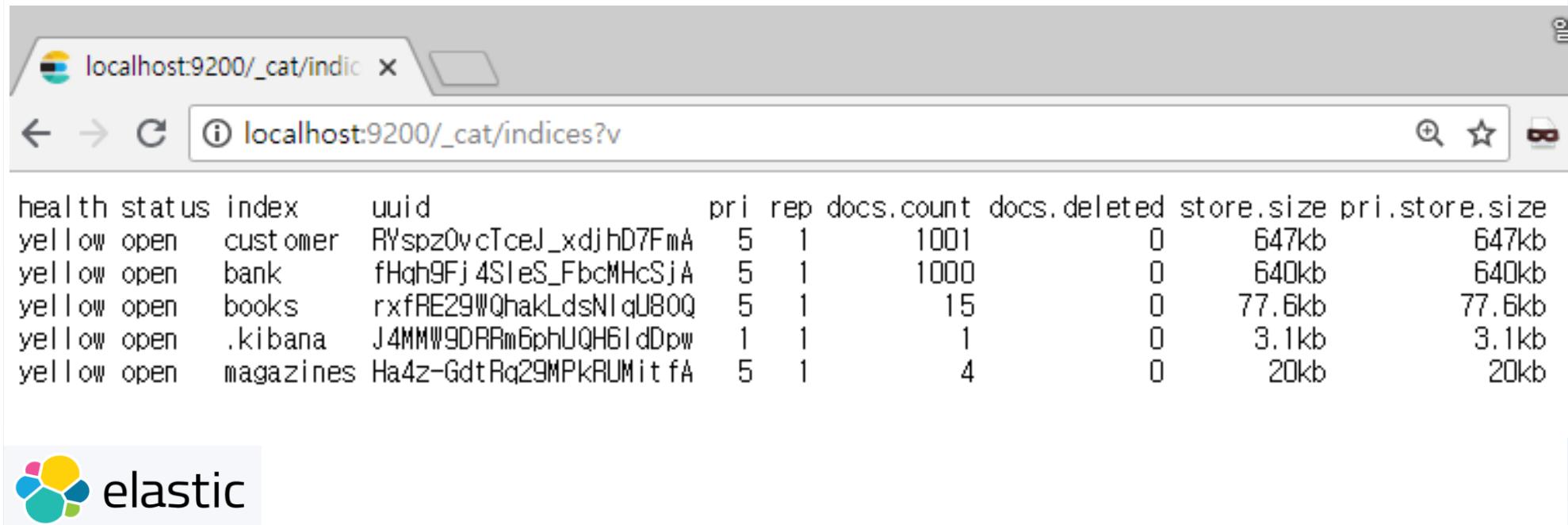


▶ 데이터셋 로드하기

- 샘플 데이터 세트(accounts.json)를 클러스터에 로드

- 리눅스 curl 명령어

```
> curl -H 'Content-Type: application/x-ndjson' -XPOST 'localhost:9200/bank/account/_bulk?pretty' --  
data-binary @accounts.json
```



health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
yellow	open	customer	RYspz0vcTceJ_xdjhD7FmA	5	1	1001	0	647kb	647kb
yellow	open	bank	fHqh9Fj4S1eS_FbcMHcSjA	5	1	1000	0	640kb	640kb
yellow	open	books	rxfrE29WQhakLdsNIqU80Q	5	1	15	0	77.6kb	77.6kb
yellow	open	.kibana	J4MMW9DRRm6phUQH6IdDpw	1	1	1	0	3.1kb	3.1kb
yellow	open	magazines	Ha4z-GdtRq29MPkRUMitfa	5	1	4	0	20kb	20kb



검색 API

▶ 검색 시작하기

- 검색 용 REST API는 _search 엔드 포인트에서 액세스
- 엘라스틱서치는 쿼리에 사용하는 JSON 스타일 도메인 관련 언어 Query DSL을 제공
- 검색을 실행하는 기본적인 두 가지 방법
 - URI: 단순한 방법이 필요한 경우에는 “URI” 방식을 사용
 - 본문: 표현력을 높여 더 많은 정보를 전달하려면 “본문” 방식으로 JSON을 사용



▶ bank 인덱스의 모든 문서를 반환하는 URI 예제

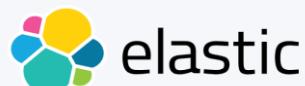
```
GET /bank/_search?q=*&sort=account_number:asc&pretty
```

- bank 인덱스에서
- q=* 매개 변수는 Elasticsearch가 인덱스의 모든 문서와 일치하도록 지시
- sort=account_number:asc는 각 문서의 account_number 필드를 사용하여 결과를 오름차순으로 정렬
- pretty 매개 변수는 다시 Elasticsearch에게 예쁜 JSON 결과를 반환하도록 지시



검색 API 결과 확인하기

항목	설명
took	Elasticsearch가 검색을 실행하는 데 걸린 시간 (밀리 초)
timed_out	검색 시간이 초과되었는지 여부를 알림
_shards	검색된 파편의 수와 성공 / 실패한 파편의 수를 알림
hits	검색 결과
hits.total	검색 조건과 일치하는 총 문서 수
hits.hits	검색 결과의 실제 배열 (기본 값은 처음 10 개)
hits.sort	결과 정렬 키 (점수순 정렬시 누락)



```

1  {
2    "took": 147,
3    "timed_out": false,
4    "_shards": {
5      "total": 5,
6      "successful": 5,
7      "failed": 0
8    },
9    "hits": {
10      "total": 1000,
11      "max_score": null,
12      "hits": [
13        {
14          "_index": "bank",
15          "_type": "type1",
16          "_id": "0",
17          "_score": null,
18          "_source": {
19            "account_number": 0,
20            "balance": 16623,
21            "firstname": "Bradshaw",
22            "lastname": "Mckenzie",
23            "age": 29,
24            "gender": "F",
25            "address": "244 Columbus Place",
26            "employer": "Euron",
27            "email": "bradshawmckenzie@euron.com",
28            "city": "Hobucken",
29            "state": "CO"
30          },
31          "sort": [
32            0
33          ]
34        },
35      ]
}

```

▶ 본문 메소드 요청 방법

- 본문 메소드를 사용하여 동일한 검색을 실행

```
POST /bank/_search
{
  "query": { "match_all": {} },
  "sort": [
    { "account_number": "asc" }
  ]
}
```

- 차이점

➤ URI에 q=*를 전달하는 대신 _search API에 JSON 스타일 쿼리 요청 본문을 제공



▶ bank 인덱스에서 전체 검색하기

- query 에 match_all을 전달하면 조건없이 모든 도큐먼트를 검색

```
POST /bank/_search
{
    "query": { "match_all": {} }
}
```

▶ 검색 사이즈 정하기

- size 속성을 사용해 쿼리되는 데이터의 크기를 지정
- from 속성을 사용해 쿼리되는 데이터의 위치를 지정

```
POST /bank/_search
{
    "query": { "match_all": {} },
    "size": 1
}
```



결과를 한 개만 검색
(Default: 10)

```
POST /bank/_search
{
    "query": { "match_all": {} },
    "from": 10,
    "size": 10
}
```

결과의 10번째 항목부터
(Default:1)

▶ 검색 결과에 대한 정렬을 수행

- sort 속성에 특정 필드(balance)를 기준으로 내림차순으로 정렬하고 상위 10개의 문서를 반환

```
GET /bank/_search
{
  "query": { "match_all": {} },
  "sort": { "balance": { "order": "desc" } }
}
```

▶ 특정 필드 내용만 검색

- 검색 조회의 _source 필드에서 원하는 필드만 지정해서 반환

```
GET /bank/_search
{
  "query": { "match_all": {} },
  "_source": [ "account_number", "balance" ]
}
```



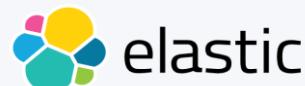
▶ 특정 필드에 문자가 일치하는 도큐먼트 검색

- 20이라는 번호가 매겨진 account_number를 반환

```
GET /bank/_search
{
  "query": { "match": { "account_number": 20 } }
```

- Address가 mill과 일치하면 반환

```
GET /bank/_search
{
  "query": { "match": { "address": "mill" } }
```



▶ 특정 필드에 문자열이 일치하는 도큐먼트 검색

- Address가 mill 또는 lane과 일치하면 반환

```
POST /bank/_search
{
  "query": { "match": { "address": "mill lane" } }
}
```

- Address에 “mill lane”이라는 문구가 일치하면 반환

```
POST /bank/_search
{
  "query": { "match_phrase": { "address": "mill lane" } }
}
```



▶ 둘 이상의 조건이 일치하는 검색

- 두 개의 일치 쿼리를 작성하고 주소에 "mill" 및 "lane"을 포함하는 모든 계정을 반환

```
POST /bank/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}
```

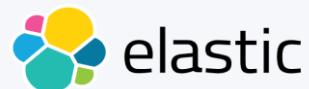
부울 쿼리	사용하는 컨텍스트	점수에 영향을 미치는가?	결과가 매칭되면 출력되는가?	정확히 매칭되는가?
must	Query	O	O	O
filter	Filter	X	O	O
should	Query	O	O	X
must_not	Filter	X	O	O



▶ 검색 연습하기

- 검색 결과에는 어떤 것들이 나올지 추측하고 검색해보자.

```
GET /bank/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "age": "40" } }
      ],
      "must_not": [
        { "match": { "state": "ID" } }
      ]
    }
  }
}
```



▶ 검색(_search) API

- 엘라스틱서치에서의 검색은 인덱스 또는 타입 단위로 수행
- _search API 사용
- 질의는 q 매개변수의 값으로 입력
- hamlet이라는 검색어로 검색

질의	질의문
books 인덱스에서 hamlet 검색	localhost:9200/books/_search?q=hamlet
전체 인덱스에서 time 검색	localhost:9200/_search?q=time



_search API 검색 결과

took에 응답시간 기록 밀리초로 표시
("took" : 7 → 0.007초)

각 검색 결과에 대한 점수 표시

hits에 결과 표시

```

GET localhost:9200/books/book/_search?q=hamlet
Params
Body Cookies Headers (3) Tests Status: 200
Pretty Raw Preview JSON ↴
1 { "took": 7,
2   "timed_out": false,
3   "_shards": {
4     "total": 5,
5     "successful": 5,
6     "failed": 0
7   },
8   "hits": {
9     "total": 1,
10    "max_score": 2.161416,
11    "hits": [
12      {
13        "_index": "books",
14        "_type": "book",
15        "_id": "AVw9yZdtB1BdYSeNM7P1",
16        "_score": 2.161416,
17        "_source": {
18          "id": 5,
19          "title": "Hamlet",
20          "author": "William Shakespeare",
21          "category": "Tragedies",
22          "written": "1599-06-01T12:34:00",
23          "pages": 172,
24          "sell": 14610000,
25          "plot": "The protagonist of Hamlet is Prince Hamlet of Denmark, son of the recently
26          nephew of King Claudius, his father's brother and successor. Claudius hastily ma
          Gertrude, Hamlet's mother. Denmark has a long-standing feud with neighbouring No
          the Norwegian prince, Fortinbras, is expected."
27        }
28      }
29    ]
30  }
31 }
```



▶ 특정 필드 검색

- q 매개변수에 <필드명: 질의> 입력

질의	질의문
전체 인덱스의 title 필드에서 time 검색	/_search?q=title:time

▶ 다중 조건 검색

- and와 or를 사용하여 다수의 조건을 검색

질의	질의문
title 필드에서 time과 machine을 검색	/_search?q=title:time AND machine



▶ explain : 점수 계산에 사용된 상세 값 출력

질의

explain 매개변수를 사용해서 검색 처리 결과 표시

질의문

/_search?q=title:time&explain

▶ 요약된 전체 hit 수와 점수(score) 등의 메타 정보를 출력

- _source false로 설정하면
- 도큐먼트 출력 생략

질의

_source 매개변수를 false로 설정해 도큐먼트 내용을 배제하고 검색

질의문

/_search?q=title:time&_source=false



▶ 출력 결과에 표시할 필드를 지정

- _source에 표시할 필드를 쉼표(,)로 구분하여 입력

질의	질의문
title, author, category 필드만 출력	/_search?q=title:time&_source=title,author,category

▶ 검색 결과의 출력 순서 정렬

- sort=필드명 형식 사용 (디폴트로 _score 값 기준)
- 내림차순 정렬 : sort=필드명:desc (디폴트로 asc(오름차순))

질의	질의문
pages 필드를 기준으로 오름차순 정렬	/_search?q=author:jules&sort=pages
pages 필드를 기준으로 내림차순 정렬	/_search?q=author:jules&sort=pages:desc



▶ 검색 API 연습문제

- TourCompany에 입력했던 데이터가 모두 날아갔다. 이런 상황을 미리 방지하기 위해 벌크 데이터를 만들고 API를 사용하여 업로드 해보자.
- Index 이름은 customer로 한다.

Doc Id	name	phone	holiday_dest	departure_date
1	Alfred	010-1234-5678	Disneyland	2017/01/20
2	Huey	010-2222-4444	Disneyland	2017/01/20
3	Naomi	010-3333-5555	Hawaii	2017/01/10
4	Andra	010-6666-7777	Borabora	2017/01/11
5	Paul	010-9999-8888	Hawaii	2017/01/10
6	Colin	010-5555-4444	Venice	2017/01/16

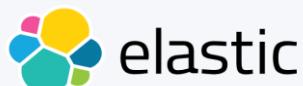


검색 API 연습문제

- 좀더 효과적인 임무 수행을 위해 검색 기능을 수행하는 쿼리를 작성하십시오.
 - tourcompany 인덱스에서 010-3333-5555를 검색하십시오.
 - 휴일 여행을 디즈니랜드로 떠나는 사람들의 핸드폰 번호를 조회하십시오(phone 필드만 출력).
 - departure date 가 2017/01/10과 2017/01/11인 사람을 조회하고 이름 순으로 출력하십시오 .
(name과 departure date 필드만 출력)
 - BoraBora 여행은 공항테러 사태로 취소됐습니다. BoraBora 여행자의 명단을 삭제해주십시오.
 - Hawaii 단체 관광객의 요청으로 출발일이 조정됐습니다. 2017/01/10에 출발하는 Hawaii의 출발일을 2017/01/17일로 수정해주십시오.

검색한 결과를 업데이트하는 API 예제

```
POST my-index-000001/_update_by_query
{
  "script": {
    "source": "ctx._source.count++", "lang": "painless"
  },
  "query": {
    "term": { "user.id": "kimchy" }
  }
}
```



▶ 해답

1. tourcompany 인덱스에서 010-3333-5555를 검색하십시오.
 - GET /tourcompany/_search?q="010-3333-5555"
2. 휴일 여행을 디즈니랜드로 떠나는 사람들의 핸드폰 번호를 조회하십시오(phone 필드만 출력).
 - GET /tourcompany/_search?q=holiday_dest: Disneyland&_source=phone,holiday_dest
3. departure date가 2017/01/10과 2017/01/11인 사람을 조회하고 이름 순으로 출력하십시오.
(name과 departure date 필드만 출력)
 - 127.0.0.1:9200/tourcompany/_search?q=departure_date:"2017/01/10" or
departure_date:"2017/01/11"&_source=name,phone,holiday_dest&sort=name.keyword:asc



▶ 해답

4. BoraBora 여행은 공항테러 사태로 취소됐습니다. BoraBora 여행자의 명단을 삭제해주십시오.

```
POST /tourcompany/_update_by_query
{
  "script": {"source": "ctx.op='delete'", "lang": "painless" },
  "query": {"match": { "holiday_dest": "Bora Bora" } }
}
```

5. Hawaii 단체 관광객의 요청으로 출발일이 조정됐습니다. 2017/01/10에 출발하는 Hawaii의 출발일을 2017/01/17일로 수정해주십시오.

```
POST tourcompany/_update_by_query
{
  "script": {"source": "ctx._source.departure_date='2017/01/17'", "lang": "painless" },
  "query": {
    "bool": {
      "must": [
        {"match": {"departure_date": "2017/01/10"}},
        {"match": {"holiday_dest": "Hawaii"}}
      ]
    }
  }
}
```

대시보드 시각화를 위한 KIBANA 기초

▶ KIBANA 소개와 설치

▶ KIBANA 데이터 준비

▶ KIBANA 시각화



KIBANA 소개와 설치

키바나 소개

- Elasticsearch와 함께 작동하도록 설계된 오픈 소스 분석 및 시각화 플랫폼
- Elasticsearch 색인에 저장된 데이터를 검색, 보기 및 상호 작용
- 고급 데이터 분석을 쉽게 수행하고 다양한 차트, 테이블 및 맵에서 데이터를 시각화
- 간단한 브라우저 기반의 인터페이스를 통해 실시간으로 Elasticsearch 쿼리의 변경 사항을 표시하는 동적 대시 보드를 신속하게 만들고 공유
- 간단한 설치!



▶ 도커로 키바나 설치하기

```
sudo apt update && sudo apt install -y docker.io
docker network create elastic
docker run -d - name es01-test - net elastic - p 9200:9200 - p 9300:9300 - e
" discovery.type=single-node " elasticsearch:7.14.1
Docker run - d - name kib01-test - net elastic - p 5601:5601 - e
"ELASTICSEARCH_HOSTS=http://es01-test:9200" kibana:7.14.1
```

▶ 엘라스틱서치 버전

- Kibana는 동일한 버전의 Elasticsearch 노드에 대해 실행되도록 구성되어야 함
- 공식적으로 지원되는 구성

▶ 엘라스틱서치 도메인 지정

- 환경변수 ELASTICSEARCH_HOSTS에 엘라스틱서치 도메인 설정하여 시작



▶ 원도우에 설치된 .zip 아카이브의 디렉토리 레이아웃

유형	설명	기본 경로
home	Kibana 홈 디렉토리 (\$KIBANA_HOME)	아카이브 압축을 풀어 생성된 디렉토리
bin	Kibana 서버를 시작하기 kibana 파일과 플러그인 설치를 위한 kibana-plugin을 포함한 바이너리 스크립트 경로	\$KIBANA_HOME\bin
config	kibana.yml을 포함한 구성 파일	\$KIBANA_HOME\config
data	Kibana 및 해당 플러그인이 디스크에 기록하는 데이터 파일의 위치입니다.	\$KIBANA_HOME\data
optimize	번역된 소스 코드 특정 관리 작업으로 인해 소스 코드가 즉석으로 변환	\$KIBANA_HOME\optimize
plugins	플러그인 파일 위치, 각 플러그인은 하위 디렉토리에 포함	\$KIBANA_HOME\plugins



▶ 키바나 접속하기

- kibana는 포트 5601을 통해 액세스하는 웹 응용 프로그램

➤ localhost : 5601 또는 http://<ip>:5601

- Kibana 서버의 상태 페이지

➤ localhost:5601/status

➤ 상태 페이지는 서버의 자원 사용에 대한 정보를 표시

➤ 설치된 플러그인을 나열

Kibana status is Green

3da0fa7e0226

1.96 GB Heap total	222.56 MB Heap used	0.06, 0.17, 0.17 Load
10.60 ms Response time avg	36.00 ms Response time max	1.00 Requests per second

Plugin status

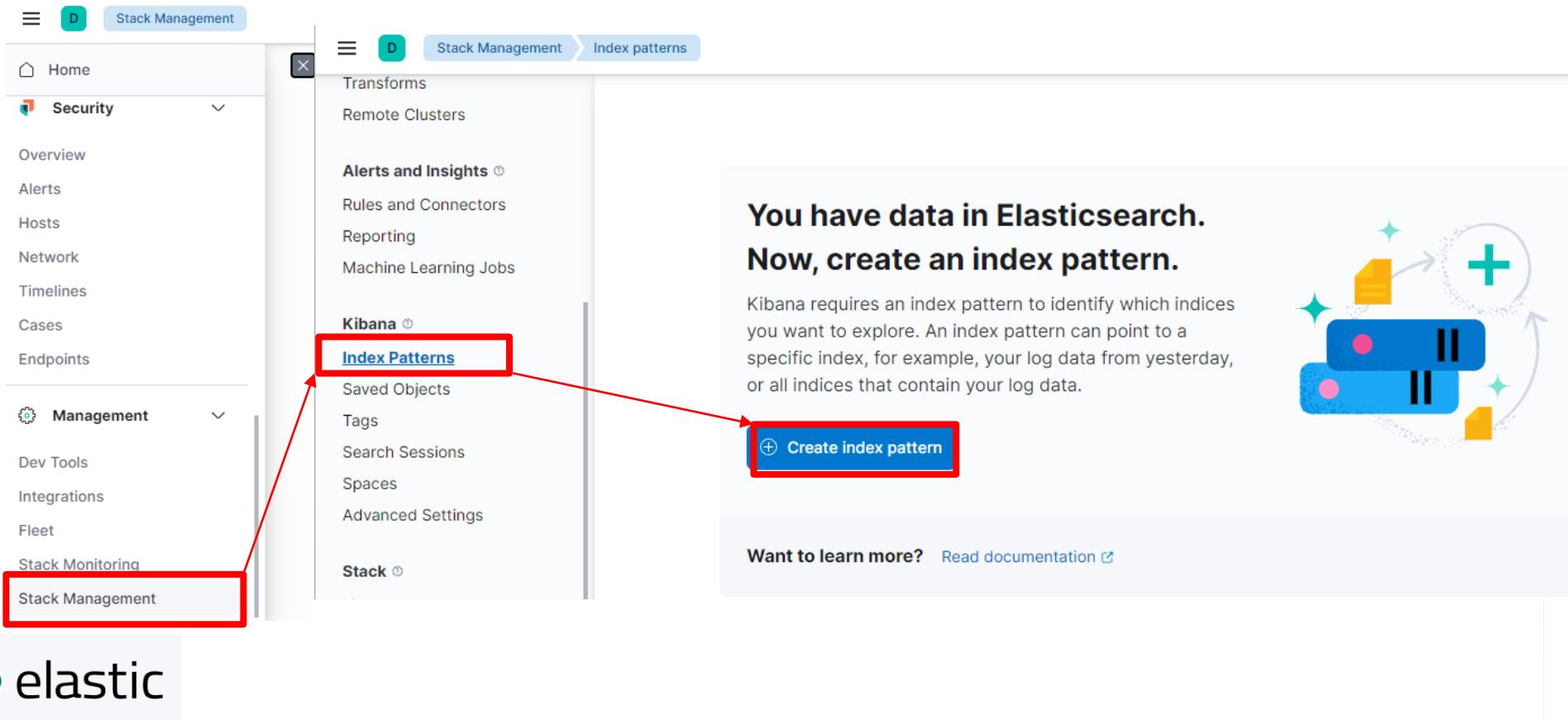
BUILD 42892 COMMIT 196ec3974d4c725a3d937725419e5ed7d8fdb104

ID	Status
core:elasticsearch@7.14.1	Elasticsearch is available
core:savedObjects@7.14.1	SavedObjects service has completed migrations and is available
plugin:advancedSettings@7.14.1	All dependencies are available
plugin:apmOss@7.14.1	All dependencies are available
plugin:bfetch@7.14.1	All dependencies are available
plugin:charts@7.14.1	All dependencies are available
plugin:console@7.14.1	All dependencies are available



▶ 엘라스틱서치 데이터 불러오기

- 브라우저에서 Kibana UI에 액세스하려면 포트 5601로 접속
- Elasticsearch 색인 이름과 일치하는 색인 패턴을 지정



The screenshot shows the Elasticsearch Stack Management interface. On the left, there's a sidebar with various management options like Home, Security, Overview, Alerts, Hosts, Network, Timelines, Cases, Endpoints, Management, Dev Tools, Integrations, Fleet, Stack Monitoring, and Stack Management. The 'Stack Management' option is highlighted with a red box. In the main content area, under the 'Kibana' section, the 'Index Patterns' option is also highlighted with a red box. To its right, there's a large callout box containing the text: 'You have data in Elasticsearch. Now, create an index pattern.' Below this text, it says 'Kibana requires an index pattern to identify which indices you want to explore. An index pattern can point to a specific index, for example, your log data from yesterday, or all indices that contain your log data.' At the bottom of this callout, there's a blue button labeled '+ Create index pattern'. A red arrow points from the 'Index Patterns' link in the sidebar to the '+ Create index pattern' button.

▶ 엘라스틱서치 데이터 불러오기

- 엘라스틱 서치에서 가져올 인덱스의 이름 패턴을 입력 logstash-*
- 시계열 데이터는 @타임스탬프로 결정

Step 1 of 2: Define an index pattern

Index pattern name
 logstash-*

Use an asterisk (*) to match multiple indices. Spaces and the characters \, /, ?, " , <, >, | are not allowed.

Include system and hidden indices

✓ Your index pattern matches 3 sources.

logstash-2015.05.18
logstash-2015.05.19
logstash-2015.05.20

Step 2 of 2: Configure settings

Specify settings for your **logstash-*** index pattern.

Select a primary time field for use with the global time filter.

Time field @timestamp Refresh ▼

> Show advanced settings

◀ Back Create index pattern



KIBANA 데이터 준비

▶ Kibana 튜토리얼 순서

- ─ Elasticsearch에 샘플 데이터 세트로드
- ─ 인덱스 패턴 정의
- ─ Discover를 사용하여 샘플 데이터 탐색
- ─ 샘플 데이터의 시각화 설정
- ─ 시각화를 대시 보드로 어셈블



▶ Kibana 튜토리얼 준비하기

- 샘플데이터 업로드하기

- 월리엄 셰익스피어 (William Shakespeare)의 전체 작품은 적절하게 필드로 파싱 - `shakespeare.json`
- 무작위로 생성 된 데이터로 구성된 가상 계정 집합 - `accounts.json`
- 임의로 생성 된 로그 파일 세트 - `logs.jsonl`

`shakespeare.json`

```
{  
  "line_id": INT,  
  "play_name": "String",  
  "speech_number": INT,  
  "line_number": "String",  
  "speaker": "String",  
  "text_entry": "String",  
}
```

`logs.jsonl`

```
{  
  "memory": INT,  
  "geo.coordinates": "geo_point",  
  "@timestamp": "date"  
}
```

`accounts.json`

```
{  
  "account_number": INT,  
  "balance": INT,  
  "firstname": "String",  
  "lastname": "String",  
  "age": INT,  
  "gender": "M or F",  
  "address": "String",  
  "employer": "String",  
  "email": "String",  
  "city": "String",  
  "state": "String"  
}
```



인덱스 매핑 설정

- 데이터를 로드하기 전에 매핑을 먼저 수행!

➤ 매핑을 수행하지 않은 경우에는 임의의 데이터 형태로 매핑

- 매핑이란?

➤ 인덱스의 문서를 논리적 그룹으로 나누고 필드의 검색 가능성 또는 토큰화되었는지 또는 별도의 단어로 분리되는 지와 같은 필드의 특성을 지정

➤ account 데이터 세트에는 매핑이 필요하지 않음

```
PUT /shakespeare
{
  "mappings" : {
    "properties" : {
      "speaker" : { "type": "keyword" },
      "play_name" : { "type": "keyword" },
      "line_id" : { "type" : "integer" },
      "speech_number" : { "type" : "integer" }
    }
  }
}
```

#18~20까지 세개의 인덱스 구성 필요

```
PUT /logstash-2015.05.18
{
  "mappings": {
    "properties": {
      "geo": {
        "properties": {
          "coordinates": { "type": "geo_point" }
        }
      }
    }
  }
}
```



▶ 엘라스틱서치의 데이터 형

- 벌크 데이터를 사용하여 Elasticsearch에 데이터 세트를 로드

```
PUT /shakespeare
{
  "mappings" : {
    "properties" : {
      "speaker" : { "type": "keyword" },
      "play_name" : { "type": "keyword" },
      "line_id" : { "type" : "integer" },
      "speech_number" : { "type" : "integer" }
    ...
  }
}
```

keyword

키워드 필드는 분석되지 않음

단일 단위로 처리 (문자열은 여러 단어가 포함)

```
#18~20까지 세개의 인덱스 구성 필요
PUT /logstash-2015.05.18
{
  "mappings": {
    "properties": {
      "geo": {
        "properties": {
          "coordinates": { "type": "geo_point" }
        }
      }
    ...
  }
}
```

integer

정수형 데이터 타입

geo_point

위도 / 경도 쌍에 지리적 위치로 레이블을 지정



▶ 샘플 데이터 업로드

- 벌크 데이터를 사용하여 Elasticsearch에 데이터 세트를 로드
- curl -H 'Content-Type: application/x-ndjson' -XPOST 'localhost:9200/bank/_bulk?pretty' --data-binary @accounts.json
- curl -H 'Content-Type: application/x-ndjson' -XPOST 'localhost:9200/_bulk?pretty' --data-binary @shakespeare_6.0.json
- curl -H 'Content-Type: application/x-ndjson' -XPOST 'localhost:9200/_bulk?pretty' --data-binary @logs.jsonl



▶ 다음 명령을 사용하여 성공적으로 로드되었는지 확인

- GET /_cat/indices?v

Pretty Raw Preview Text ⚙️

	health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
1	yellow	open	apache_elastic_example	Gr33jX5_SM2xwfvqqV1HCg	5	1	3312	0	2mb	2mb
2	yellow	open	logstash-2015.05.18	B_VNMbyATB21LavCGstmfG	5	1	4631	0	29.8mb	29.8mb
3	yellow	open	logstash-2015.05.19	SBrvUwAkTFW2GxAuL_8I2A	5	1	4624	0	30.3mb	30.3mb
4	yellow	open	bank	r3_dtMGcT8KI1J7W1yw-mA	5	1	1000	0	647.9kb	647.9kb
5	yellow	open	shakespeare	UEINXXSiR9apVrAQNSGGIA	5	1	111396	0	28.5mb	28.5mb
6	yellow	open	logstash-2015.05.20	8YeYD1N_RDWCv-xZt1BsiQ	5	1	4750	0	30mb	30mb
7	yellow	open	.kibana	a-QQKs6iTzmvANLv8109LA	1	1	2	0	11.8kb	11.8kb
8										
9										



Index 패턴 정의하기

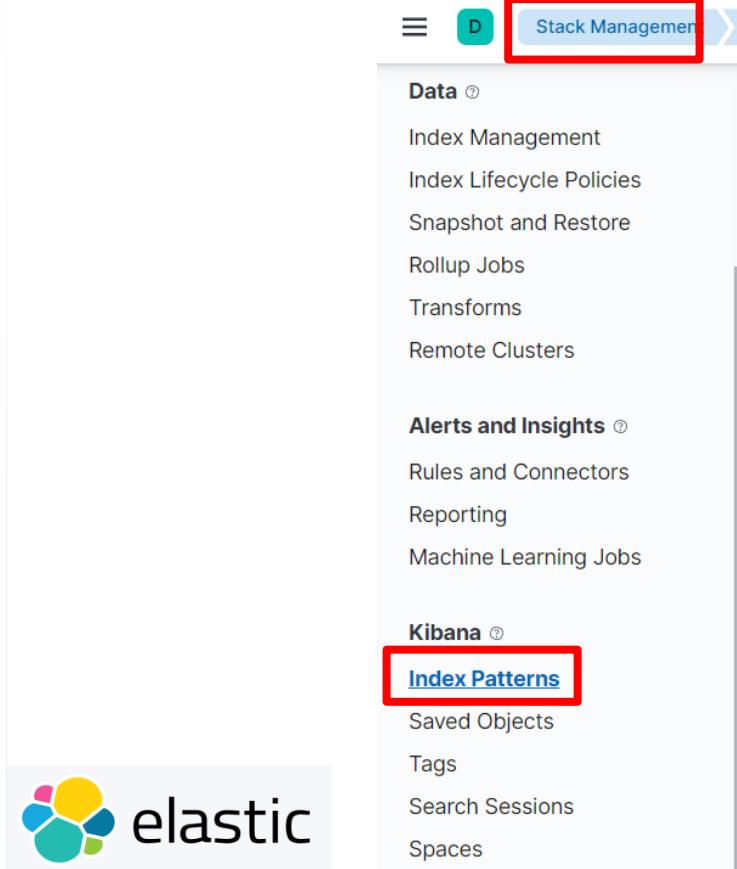
- Elasticsearch에 로드된 각 데이터 세트에는 인덱스 패턴 존재
 - Shakespeare 데이터 세트 : Shakespeare라는 인덱스(shakes*)
 - account 데이터 세트 : bank라는 인덱스(ba*)
 - logs 데이터 세트 : YYYY.MM.DD 형식의 날짜가 포함(5월에 대한 색인 패턴, logstash-2015.05*)

1. Shakespeare와 account 데이터세트에는 시계열 데이터를 포함
2. 데이터 세트에 대한 인덱스 패턴이 시간 기반 이벤트가 포함되어 있는지 확인 필요



Index 패턴 정의하기

- Management 탭을 클릭 한 다음 Index Patterns 탭을 클릭
- +Add New 새로 추가를 클릭하여 새 인덱스 패턴을 정의



The screenshot shows the Kibana Management interface. At the top, there are tabs: 'Stack Management' (highlighted with a red box), 'Index patterns', and 'Data'. The 'Data' tab has three sections: 'Index Management', 'Index Lifecycle Policies', and 'Snapshot and Restore'. Below these are 'Rollup Jobs', 'Transforms', and 'Remote Clusters'. The 'Alerts and Insights' section includes 'Rules and Connectors', 'Reporting', and 'Machine Learning Jobs'. The 'Kibana' section has 'Index Patterns' (highlighted with a red box) and other options like 'Saved Objects', 'Tags', 'Search Sessions', and 'Spaces'. To the right, a large panel displays the message: 'You have data in Elasticsearch. Now, create an index pattern.' It features a prominent red button labeled '+ Create index pattern'. At the bottom, it says 'Want to learn more? Read documentation'.



KIBANA 시작화

키바나 메뉴



Overview

Discover

Dashboard

Canvas

Maps

Machine Learning

Visualize Library



데이터베이스를 탐색하고 검색하는 기능

각종 차트를 원하는대로 배치하는 대시보드 기능

CSS 등을 사용해 자신만의 스타일을 꾸밀 수 있는 기능

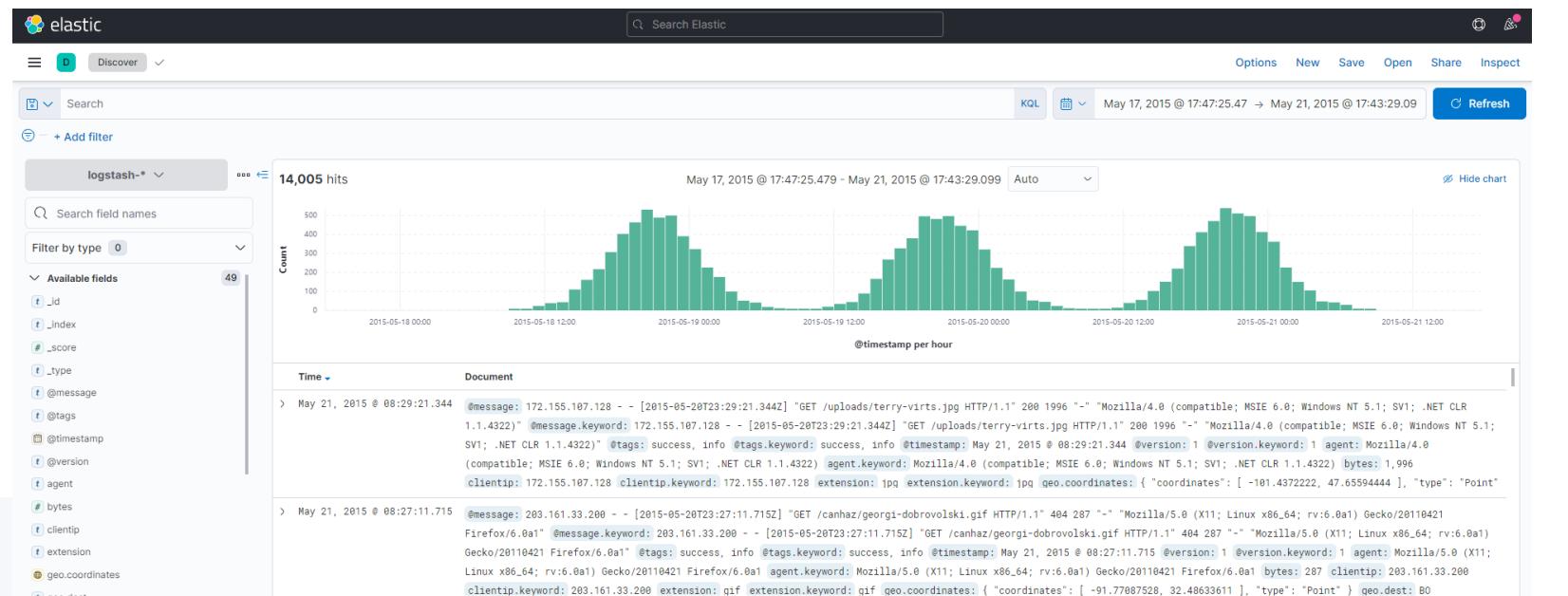
데이터를 지도에 표현할 수 있는 기능

머신러닝 기능을 사용해 이상 탐지하는 기능 (30일 trial)

데이터로 차트를 그리고 저장할 수 있는 차트 저장소

■ Kibana 시각화: Discover

- 결과를 탐색하고 Visualize에서 저장된 검색의 시각화
- 쿼리 바에서 Elasticsearch query를 입력하여 데이터를 검색
 - 관심있는 필드 이름과 값을 사용하여 검색을 구성
 - 숫자 필드에서는 큼 (>), 작음 (<) 또는 같음 (=)과 같은 비교 연산자 지원
 - 논리 연산자 AND, OR 및 NOT 등 지원



■ Kibana 시각화: Discover

- ba* 인덱스를 선택하고 다음 쿼리를 실행

- 잔액 : 47,500을 초과하는
- 0에서 99 사이의 모든 계좌 번호
- 샘플 뱅크 데이터를 검색 할 때 계정 번호 8, 32, 78, 85 및 97의 5 개의 결과가 반환됩니다.

The screenshot shows the Kibana Discover interface. On the left, there is a sidebar with the text "account_number < 100 AND balance > 47500". The main area has a search bar with the query "ba* account_number < 100 AND balance > 47500" and a "Discover" button. Below the search bar are filter options: "Search field names" (empty), "Filter by type" (0), and a list of "Available fields" including _id, _index, _score, _type, account_number, address, and age. To the right, a results panel shows "5 hits" with document details:

Document
> account_number: 32 address: 702 Quentin Street address.keyword: 702 Quentin Street age: 34 balance: 48,086 city: Veguita city.keyword: Veguita email: dillardmcphee@quailcom.com email.keyword: dillardmcphee@quailcom.com employer: Quailcom employer.keyword: Quailcom firstname: Dillard firstname.keyword: Dillard
> account_number: 78 address: 834 Amber Street address.keyword: 834 Amber Street age: 23 balance: 48,656 city: Dunbar city.keyword: Dunbar email: elvirapatterson@assistix.com email.keyword: elvirapatterson@assistix.com employer: Assistix employer.keyword: Assistix firstname: Elvira firstname.keyword: Elvira



▶ Kibana 시각화: Visualize Library

- 여러 가지 방법으로 데이터를 시각화

Home

Analytics

Overview

Discover

Dashboard

Canvas

Maps

Machine Learning

Visualize Library

New visualization

Lens
Create visualizations with our drag and drop editor. Switch between visualization types at any time. *Recommended for most users.*

TSVB
Perform advanced analysis of your time series data.

Aggregation based
Use our classic visualize library to create charts based on aggregations.
[Explore options →](#)

Want to learn more? [Read documentation ↗](#)

Maps
Create and style maps with multiple layers and indices.

Custom visualization
Use Vega to create new types of visualizations. Requires knowledge of Vega syntax.

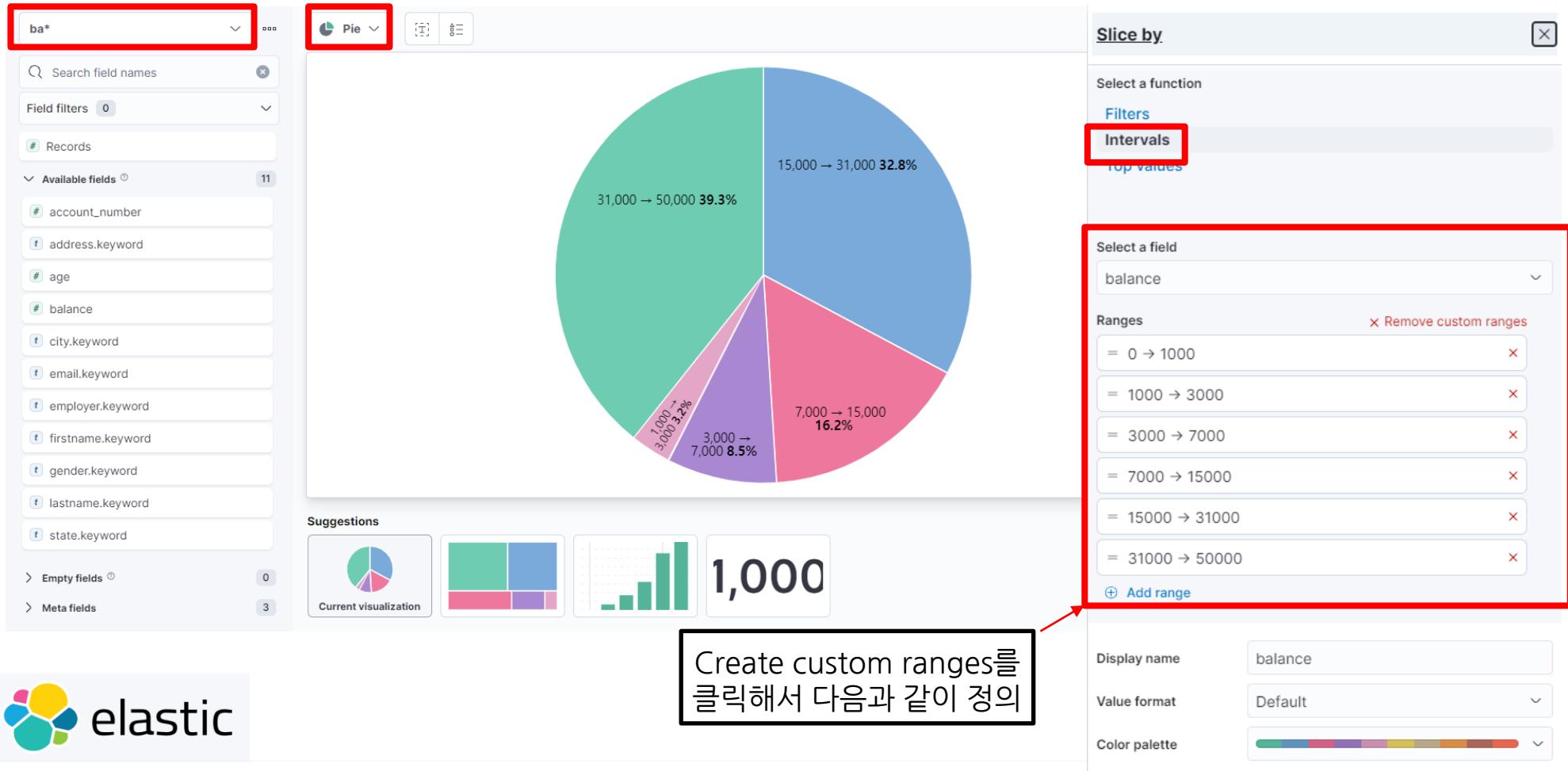
Tools

- Text**
Add text and images to your dashboard.
- Controls**
Add dropdown menus and range sliders to your dashboard.



Kibana 시각화: Visualize

- 계좌 잔액에 대한 통계 정보 만들기 - Pie 차트를 사용, ba* 인덱스 선택



KIBANA 시작화

Kibana 시작화: Visualize

- 파이 차트에 “age 변수” 추가

ba*

Slice by

≡
balance
✖

ⓘ Drop a field or click to add

Size by

Count of records
✖

⟲ [Reset visualization](#)

Slice by

Select a function

[Filters](#)

[Intervals](#)

Top values

Select a field

age

Number of values **5**

Rank by [Count of records](#)

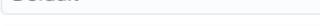
Rank direction **Descending**

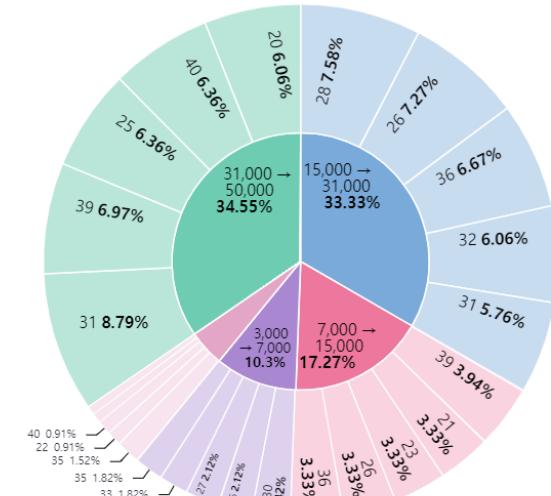
[Advanced](#)

Display name **Top values of age**

Group by this field first

Value format **Default**

Color palette 



▶ Kibana 시각화: Visualize

- 상단에 Save 버튼을 누르고 Pie Example로 저장

Download as CSVSave

KQLMay 17, 2015 @ 17:47:25.479 → ~ a year agoRefresh

Save Lens visualization

Title

Description

Tags

Add to dashboard

Existing

New

None

Add to library ①

Cancel Save and add to library

Slice by

Select a function

[Filters](#)

[Intervals](#)

Top values

Select a field

Number of values

Rank by

Rank direction

> Advanced

Display name

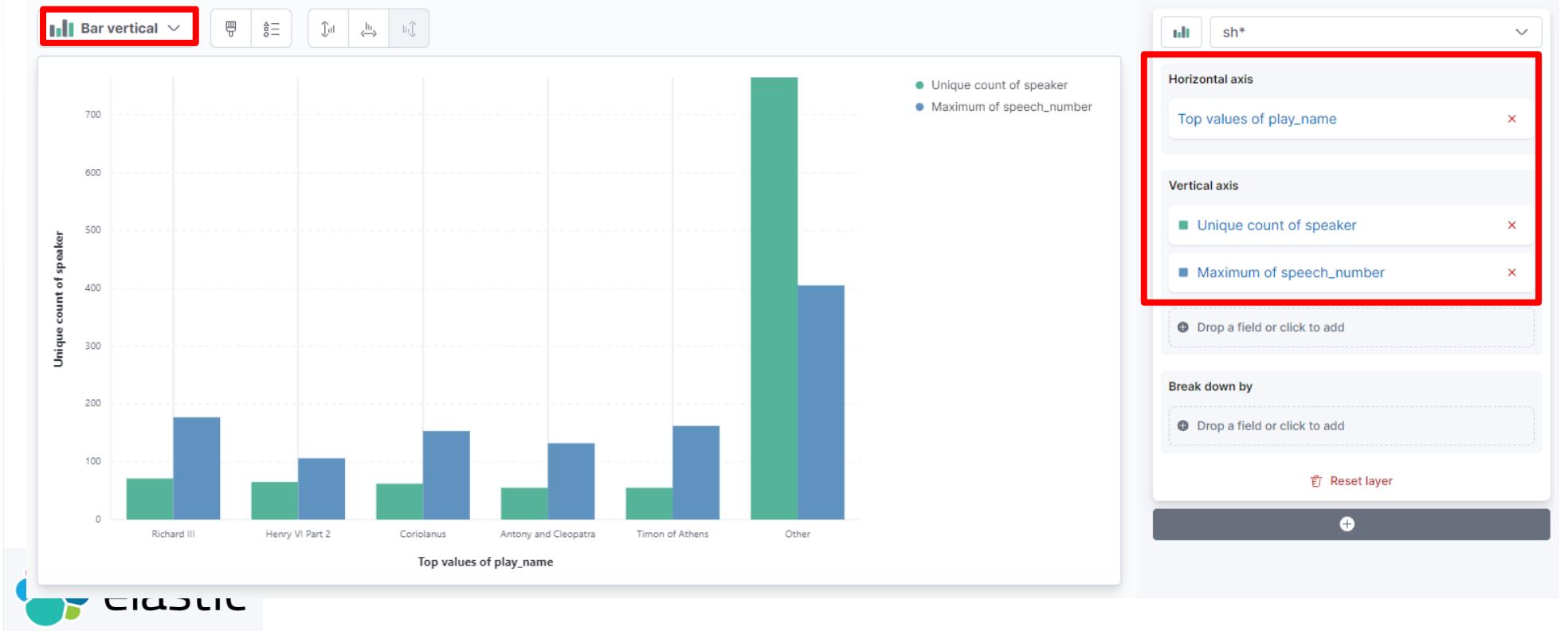
Group by this field first



▶ Kibana 시각화: Visualize

- shakespeare 관련 막대 그래프 그리기!

- 인덱스를 shakespeare로 변경하고 Bar vertical을 선택한 후 드래그엔 드롭으로 적정한 데이터를 x,y에 지정
- Bar Example로 저장



■ Kibana 시각화: Visualize

- Maps를 선택해서 logstash-*에 저장된 geo.coordinates 필드를 시각화
- Add Layer를 선택하고 clusters and grids를 선택

The screenshot shows the Kibana interface with the 'Maps' tab selected. The main area displays a map of the Middle East with various locations labeled in multiple languages. A red box highlights the 'Add layer' button in the top right corner of the map panel. Another red box highlights the 'Clusters and grids' section in the bottom right corner, which contains icons for clusters and grids and a descriptive text.

Analytics

Overview

Discover

Dashboard

Canvas

Maps

Machine Learning

Visualize Library

Map settings **Inspect** **Full screen** **Save**

KQL May 17, 2015 @ 17:47:25.47 → May 21, 2015 @ 17:04:26.96 Refresh

Nicosia Λευκωσία SYRIA IRAQ Asqabat

Road map

Add layer

Clusters and grids

Geospatial data grouped in grids with metrics for each gridded cell

■ Kibana 시각화: Visualize

- logstash-*의 geo.coordinates 필드 클릭한 후 Add Layer를 선택
- Map Example로 저장

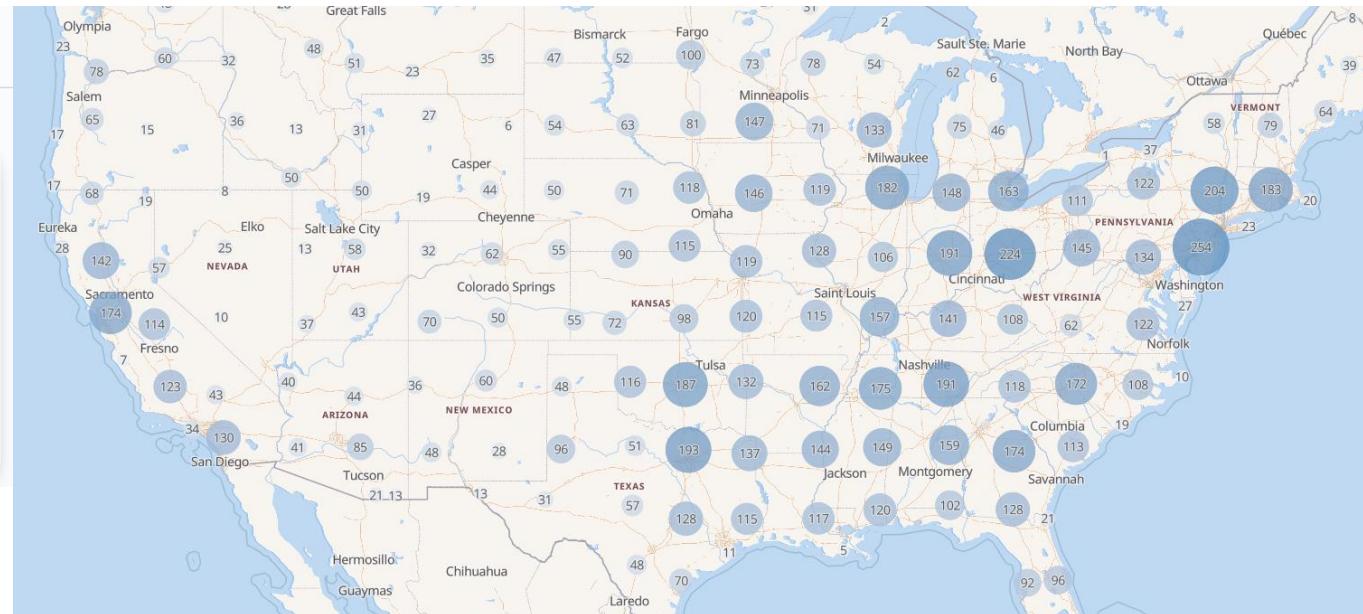
Add layer

< Change layer

Index pattern
logstash-*

Geospatial field
geo.coordinates

Show as
clusters



▶ Kibana 시각화: Dashboard

- 대시보드 탭으로 이동하여 생성한 라이브러리를 가져오고 적절히 배치 후 Example로 저장

The screenshot shows the Kibana Dashboard editor interface. On the left, there is a sidebar with a placeholder for "Add your first visualization" and an "elastic" logo at the bottom. The main area contains three visualizations:

- Bar Example:** A bar chart titled "Top values of play_name" showing the unique count of speakers. The data is as follows:

Category	Value
Richard II	~150
Henry VI Part 2	~100
Concordia	~150
Antony and Cleopatra	~100
Timon of Athens	~150
Other	~450
- Pie Example:** A pie chart showing the distribution of values across different categories. The data is as follows:

Category	Percentage
31,000 - 50,000	~34.5%
10,000 - 20,000	~23.6%
3,000 - 7,000	~16.3%
1,000 - 3,000	~26.7%
- Map Example:** A map of North America showing data points for various cities. Each city has a blue circle with a numerical value indicating a specific metric. Some labeled values include: Vancouver (26), Victoria (5), Portland (34), Salt Lake City (85), Las Vegas (254), Denver (164), Cheyenne (107), Omaha (213), Dallas (44), Fort Worth (222), Oklahoma City (211), Memphis (161), Jackson (137), Atlanta (137), Tampa (99), and Miami (125).

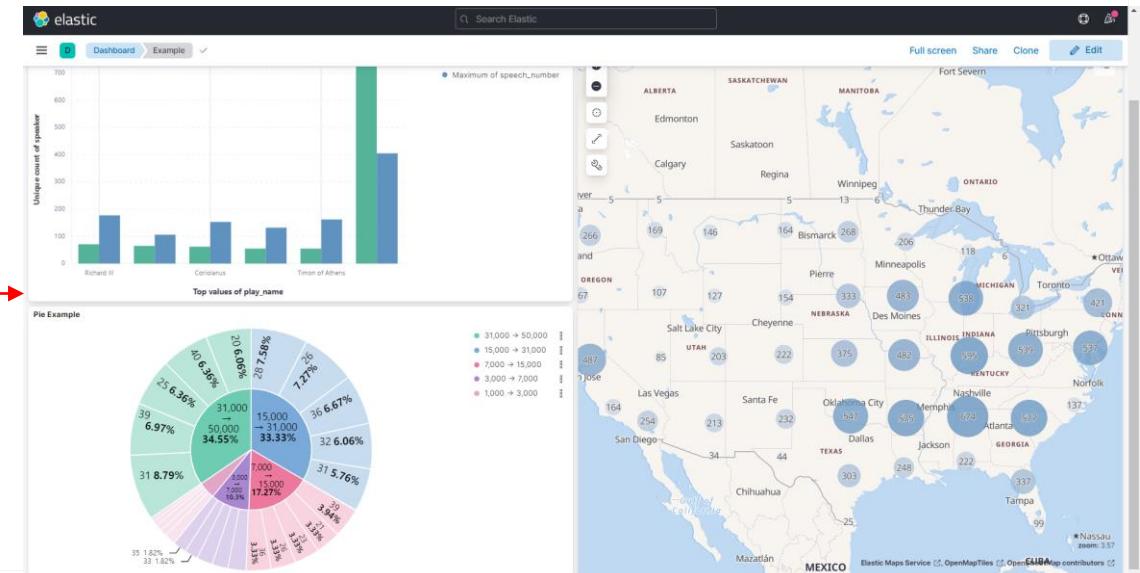
The top navigation bar includes tabs for "Dashboard" (selected) and "Editing New Dashboard". Below the dashboard tabs are buttons for "Create visualization", "All types", and "Add from library" (which is highlighted with a red box). The bottom right corner of the dashboard area shows a "Save" button.

Kibana 시각화: Dashboard

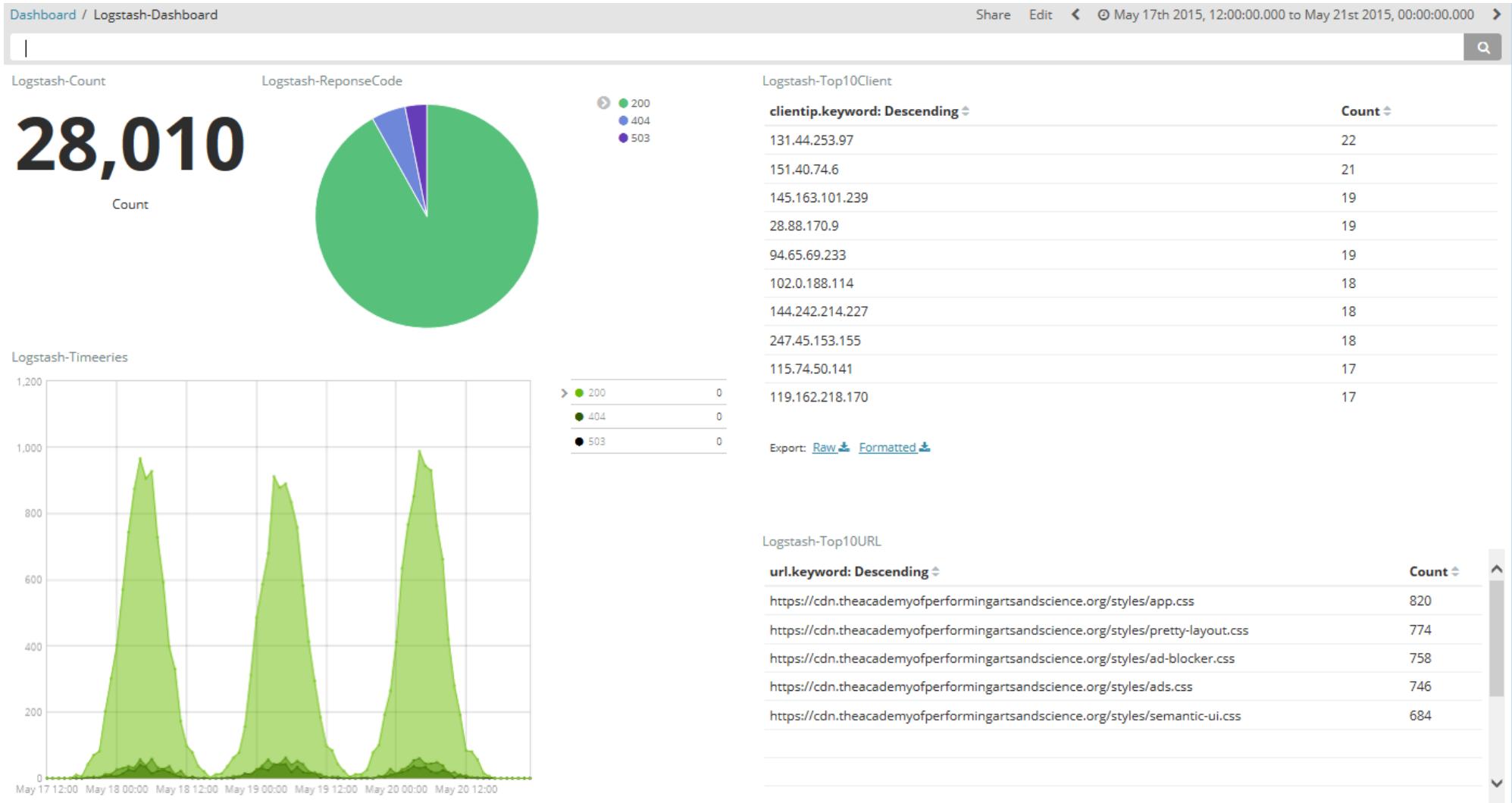
- 대시보드 탭으로 이동하여 생성한 라이브러리를 가져오고 적절히 배치 후 Example로 저장

The screenshot shows the 'Share this dashboard' section of the Kibana interface. It includes options for 'Embed code' and 'Permalinks'. A red box highlights the 'Permalinks' button, and another red box highlights the 'Copy link' button below it.

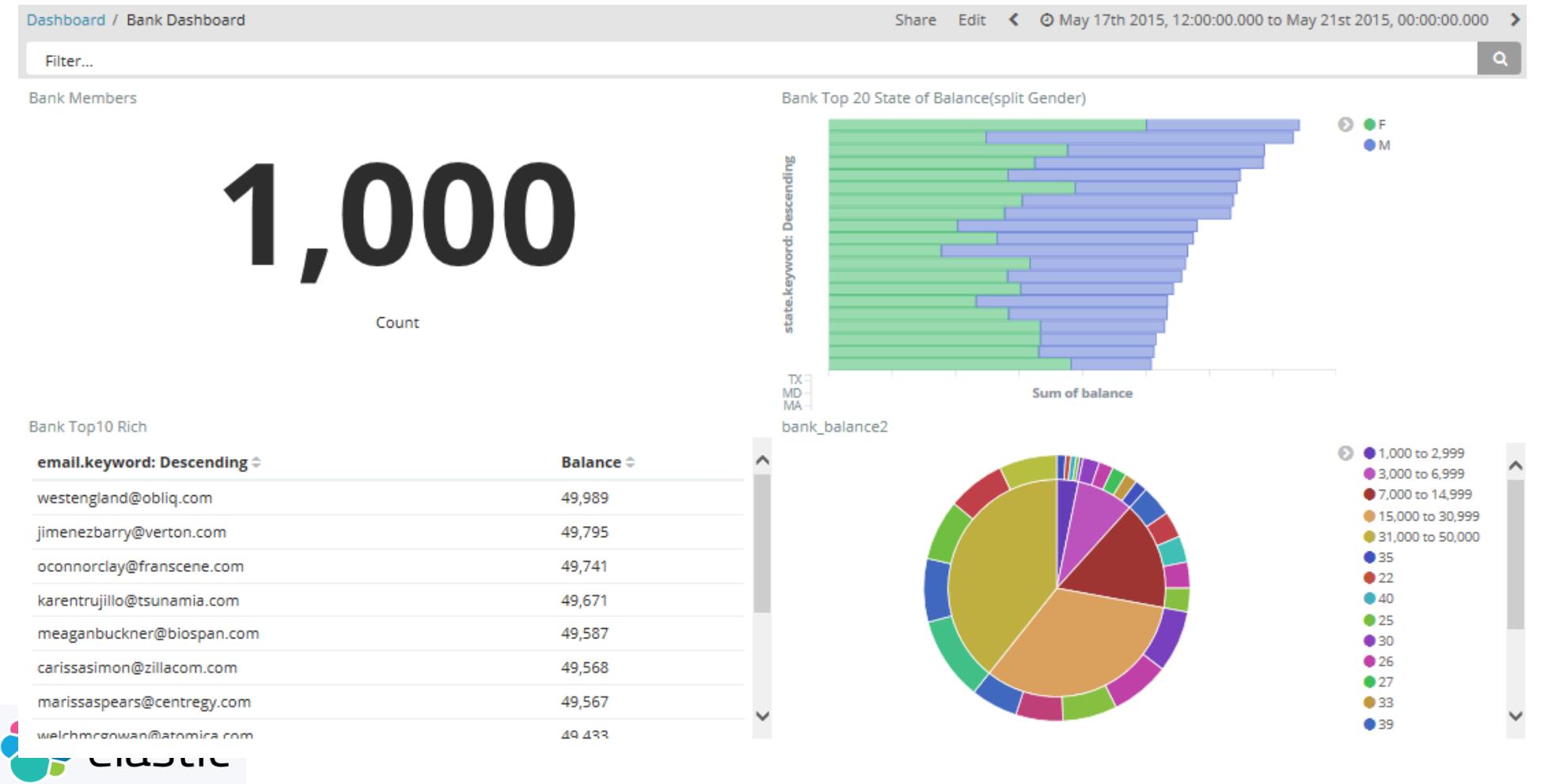
외부로 바로
공유 가능



Kibana 시각화: 연습문제 Logstash



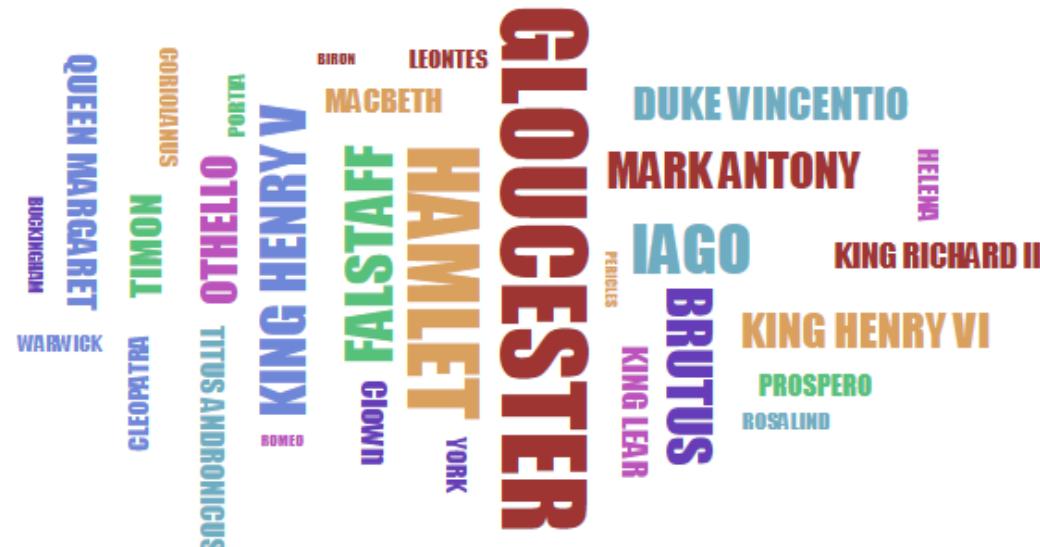
Kibana 시각화: 연습문제 Bank



KIBANA 시작화

Kibana 시각화: 연습문제 Bank

Shakes Count - speaker: Descending

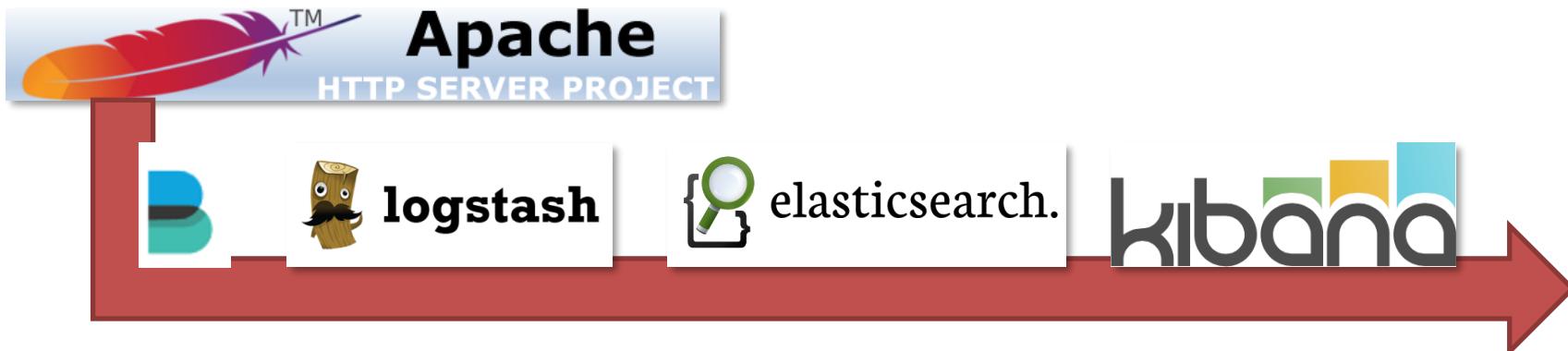


파일 비트를 활용한 아파치 서버 로그 수집

파일 비트를 활용한 아파치 서버 로그 수집

Logstash/Filebeat webservice 로그 자동 수집

- Ubuntu환경의 아파치2 설치 후 액세스 로그 저장
- Filebeat의 기능을 사용하여 파일을 긁어옴
- Logstash는 apache 파싱 기능을 사용해 데이터의 필드를 각각 분할하여 엘라스틱 서치에 전송



파일 비트를 활용한 아파치 서버 로그 수집

파일비트와 apache2 설치

- 아파치 설치하기

- apt install apache2 -y

- 파일비트 다운로드 및 설치

- 파일비트 7.15.1 deb 32비트 다운로드 명령어 실행
 - wget https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-7.15.1-amd64.deb --no-check-certificate
 - sudo dpkg -i filebeat-7.15.1-amd64.deb

파일 비트를 활용한 아파치 서버 로그 수집

▶ API 키 발급하기

- Stack Management - API keys - Create에서 API 키 생성

The screenshot shows the Elastic Stack Management interface for creating an API key. On the left, there's a sidebar with various management options like Ingest Node Pipelines, Logstash Pipelines, and Data management (Index Management, Index Lifecycle Policies, etc.). The main area is titled 'API Keys' and contains a table of existing keys. A red box highlights the 'Create API key' button. A modal window titled 'Create API key' is open on the right, showing fields for 'User' (set to 'elastic'), 'Name' (set to 'filebeat-http-logs'), and three optional checkboxes: 'Restrict privileges', 'Expire after time', and 'Include metadata'. Another red box highlights the success message 'Created API key 'filebeat-http-logs'' and the JSON key details at the bottom of the main screen.

elasticsearch

Stack Management API keys Create

Ingest Node Pipelines Logstash Pipelines

Data

- Index Management
- Index Lifecycle Policies
- Snapshot and Restore
- Rollup Jobs
- Transforms
- Cross-Cluster Replication
- Remote Clusters

API Keys

View and delete API keys. An API key sends requests on behalf of a user.

Name	User	Realm	Created
filebeat-access-log	elastic	reserved	7 minutes

+ Create API key

✓ Created API key 'filebeat-http-logs'

Copy this key now. You will not be able to view it again.

JSON { "id": "2bI0knwBLv_M-o02JoWT", "name": "filebeat-http-logs" }

Create API key

Cancel

Create API key

User
elastic

Name
filebeat-http-logs

What is this key used for?

Restrict privileges

Expire after time

Include metadata

파일 비트를 활용한 아파치 서버 로그 수집

▶ 엘라스틱서치 접속 정보 입력

- /etc/filebeat/filebeat.yml 파일 수정

- sudo vim /etc/filebeat/filebeat.yml
- output.elasticsearch를 찾아서 호스트 위치가 정확한지 확인
- https 프로토콜을 사용하도록 체크
- 앞서 생성한 id와 apikey를 입력
- 안전하지 않은 SSL 통신을 위해 mode를 none으로 설정

```
# ----- Elasticsearch Output -----
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["localhost:9200"]

  # Protocol - either `http` (default) or `https`.
  protocol: "https"

  # Authentication credentials - either API key or username/password.
  api_key: "yrIGknwBLv_M-o02UYUC:t8slUwl2RCewwRHRXsi3wg"
  ssl.verification_mode: none
```

파일 비트를 활용한 아파치 서버 로그 수집

파일비트 모듈 설정으로 apache2 로그 수집

- 모듈 디렉토리에 apache.yml을 활성화하고 필요한 데이터를 입력

- cd /etc/filebeat/modules.d
- cp apache.yml.disabled apache.yml
- vim apache.yml

```
- module: apache
access:
  enabled: true
  var.paths: ["/var/log/apache2/access.log*"]
error:
  enabled: true
  var.paths: ["/var/log/apache2/error.log*"]
```

- 설정 파일 적용을 위해 재시작

```
sudo /etc/init.d/filebeat restart
journalctl -u filebeat # 로그 확인
```

- 오류 발생 시 확인 및 조치

파일 비트를 활용한 아파치 서버 로그 수집

▶ 대시보드에 파일비트

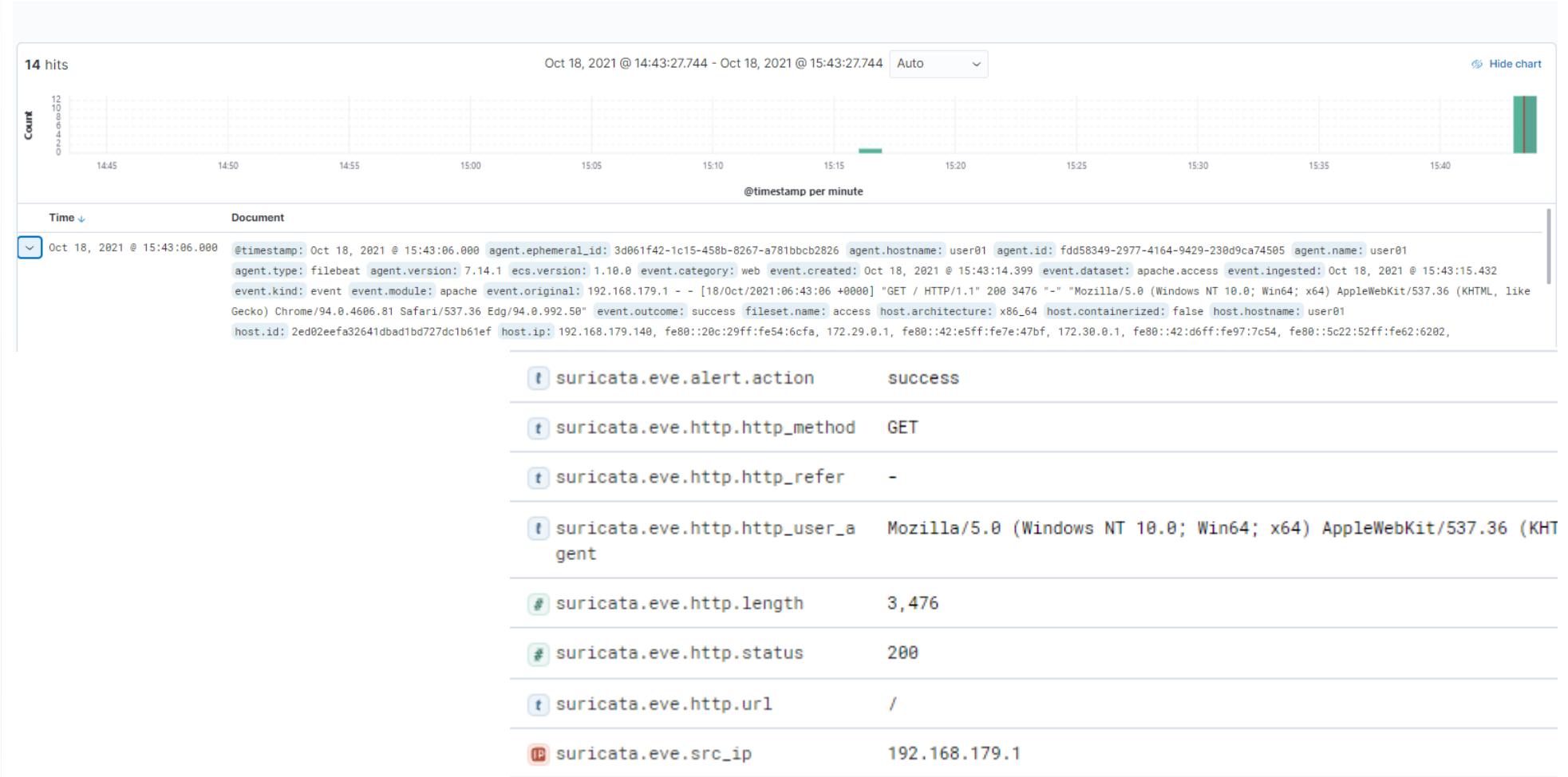
- 인덱스 패턴 확인 및 추가

The screenshot shows the Elastic Stack Management interface. On the left, there's a sidebar with categories like Ingest Node Pipelines, Logstash Pipelines, Data (Index Management, Index Lifecycle Policies, Snapshot and Restore, Rollup Jobs, Transformations, Cross-Cluster Replication, Remote Clusters), Alerts and Insights (Rules and Connectors, Reporting, Machine Learning Jobs, Watcher), and Security (Users, Roles, API keys, Role Mappings). The main area is titled 'Create index pattern'. It has fields for 'Name' (set to 'filebeat-7.14.1-*') and 'Timestamp field' (set to '@timestamp'). A note says 'Use an asterisk (*) to match multiple characters. Spaces and the characters , /, ?, ", <, >, | are not allowed.' Below these are 'Show advanced settings' and 'Create index pattern' buttons. To the right, a message says 'Your index pattern matches 1 source.' followed by 'filebeat-7.14.1-2021.10.18-000001' with an 'Index' button. There's also a 'Rows per page: 10' dropdown.

파일 비트를 활용한 아파치 서버 로그 수집

▶ 디스커버에서 정보 확인

● Suricata 필드에서 파싱된 로그 확인



쿠버네티스 보안

▶ 시큐리티 콘텍스트

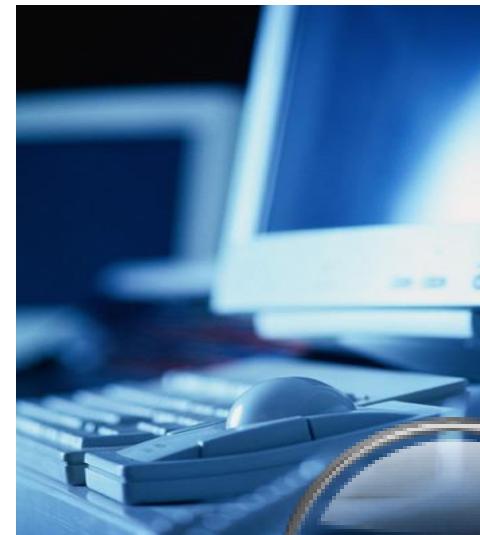
▶ 네트워크 정책 적용

▶ 쿠버네티스 감사(Audit) 기능 활성화

▶ Trivy를 활용한 컨테이너 취약점 진단

▶ kube-bench를 활용한 쿠버네티스 보안 점검

▶ falco를 활용한 쿠버네티스 컨테이너 보안
모니터링



시큐리티 콘텍스트

▶ 보안 컨텍스트

- 서버 침해사고 발생 시 침해사고를 당한 권한을 최대한 축소하여 그 사고에 대한 확대를 방지
- 침해사고를 당한 서비스가 모든 권한(root나 노드 커널 기능)으로 동작하는 경우 서비스를 탈취한 공격자는 그대로 컨테이너의 권한을 사용할 수 있음
- 최소 권한 정책에 따른 취약점 감소

- 보안 컨텍스트 설정에는 다음 사항을 포함
 - 권한 상승 가능 여부
 - 프로세스 기본 UID/GID를 활용한 파일 등의 오브젝트의 액세스 제어
 - Linux Capabilities를 활용한 커널 기능 추가
 - 오브젝트에 보안 레이블을 지정하는 SELinux (Security Enhanced Linux) 기능
 - AppArmor : 프로그램 프로필을 사용하여 개별 프로그램의 기능 제한
 - Seccomp : 프로세스의 시스템 호출을 필터링

시큐리티 콘텍스트

파드에 시큐리티 콘텍스트 설정하기

- 파드에 UID를 설정하면 모든 컨테이너에 적용됨

- 컨테이너 UID/GID 설정하기
- 권한상승 제한하기

```
$ kubectl exec -it security-context-demo /bin/bash  
/ $ id  
uid=1000 gid=3000 groups=2000  
/ $ ps -eaf  
PID  USER      TIME  COMMAND  
 1 1000      0:00  sleep 1h  
10 1000      0:00  sh  
16 1000      0:00  ps -eaf
```

```
apiVersion: v1          security-context.yaml  
kind: Pod  
metadata:  
  name: security-context-demo  
spec:  
  securityContext:  
    runAsUser: 1000  
    runAsGroup: 3000  
    fsGroup: 2000  
  volumes:  
  - name: sec-ctx-vol  
    emptyDir: {}  
  containers:  
  - name: sec-ctx-demo  
    image: busybox  
    command: [ "sh", "-c", "sleep 1h" ]  
    volumeMounts:  
    - name: sec-ctx-vol  
      mountPath: /data/demo  
    securityContext:  
      allowPrivilegeEscalation: false
```

시큐리티 콘텍스트

▶ 컨테이너에 시큐리티 콘텍스트 설정하기

- 컨테이너에 내부에 새로운 유저를 사용하면 그 설정이 우선됨
 - 컨테이너 UID/GID 설정하기

security-context-2.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo-2
spec:
  securityContext:
    runAsUser: 1000
  containers:
    - name: sec-ctx-demo-2
      image: gcr.io/google-samples/node-hello:1.0
      securityContext:
        runAsUser: 2000
        allowPrivilegeEscalation: false
```

시큐리티 콘텍스트

▣ 캐퍼빌리티 적용

- 캐퍼빌리티를 적용하면 리눅스 커널에서 사용할 수 있는 권한을 추가 설정 가능
- 캐퍼빌리티를 활용한 리눅스 커널 권한 변수
➤ <https://github.com/torvalds/linux/blob/master/include/uapi/linux/capability.h>
- date +%T -s "12:00:00" 명령은 SYS_TIME 커널 권한이 있어야만 가능

security-context-4.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo-4
spec:
  containers:
  - name: sec-ctx-4
    image: gcr.io/google-samples/node-hello:1.0
    securityContext:
      capabilities:
        add: [ "NET_ADMIN", "SYS_TIME" ]
```

시큐리티 콘텍스트

▶ SELinux 권한

- 모든 프로세스와 객체에 시큐리티콘텍스트(또는 레이블)을 달아 관리하는 형태
- 넷필터와 더불어 리눅스의 핵심 보안 기능을 담당
- 그러나 실무에서는 복잡도가 높아 오히려 소외...
- SELinux에 대한 자세한 정보 사이트
 - <https://www.lesstif.com/ws/selinux/selinux>

- 다음과 같은 형태로 취급

```
...
securityContext:
  selinuxOptions:
    level: "s0:c123,c456"
```

자료출처: <https://www.lesstif.com/ws/selinux/selinux>

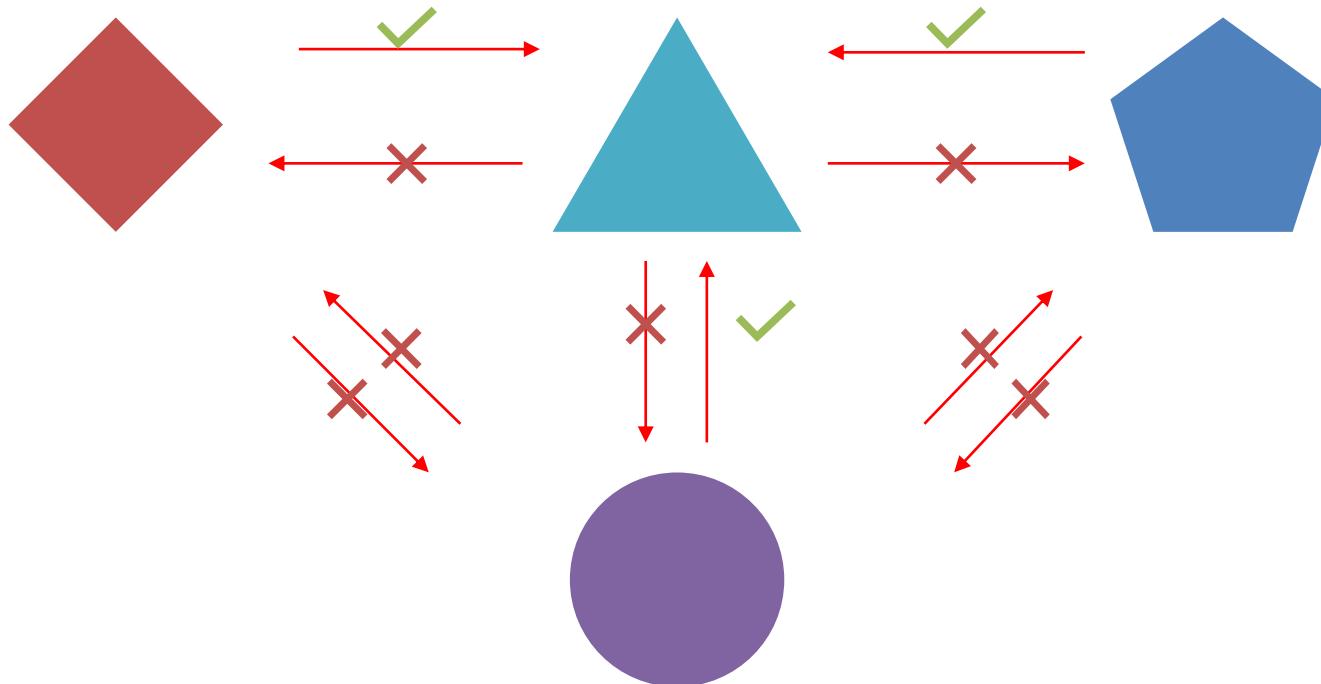
요소	설명
사용자	시스템의 사용자와는 별도의 SELinux 사용자로 역할이나 레벨과 연계하여 접근 권한을 관리하는데 사용.
역할(Role)	하나 혹은 그 이상의 타입과 연결되어 SELinux 의 사용자의 접근을 허용할 지 결정하는데 사용.
타입(Type)	Type Enforcement의 속성중 하나로 프로세스의 도메인이나 파일의 타입을 지정하고 이를 기반으로 접근 통제를 수행.
레벨(Label)	레벨은 MLS(Multi Level System)에 필요하며 강제 접근 통제보다 더 강력한 보안이 필요할 때 사용하는 기능으로 정부나 군대등 최고의 기밀을 취급하는 곳이 아니라면 사용하지 않습니다.

네트워크 정책 적용

네트워크 정책 적용

▶ 네트워크 정책(Network Policy)

- 파드 그룹이 서로 및 다른 네트워크 끝점과 통신하는 방법을 지정
- NetworkPolicy 리소스는 레이블을 사용하여 파드를 선택
- 선택한 파드에 허용되는 트래픽을 지정하는 규칙을 정의
- 특정 파드를 선택하는 네임 스페이스에 NetworkPolicy가 있으면 해당 파드는 NetworkPolicy에서 허용하지 않는 연결을 거부



네트워크 정책 적용

▶ 이그레스

- 선택된 파드에서 나가는 트래픽에 대한 정책 설정
- ipBlock을 통해 허용하고자 하는 ip 대역 설정 가능
- Ports에는 어떤 포트를 허용하는지 명시

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Egress
  egress:
    - to:
        - ipBlock:
            cidr: 10.0.0.0/24
  ports:
    - protocol: TCP
      port: 5978
```

네트워크 정책 적용

▶ 인그레스

- 선택된 파드로 들어오는 트래픽에 대한 정책 설정
- IpBlock을 통해 허용하고자 하는 ip 대역 설정 가능
- Except를 사용하여 예외 항목 설정 가능
- NamespaceSelector와 podSelector를 사용
 - 그룹별 더욱 상세한 정책 설정 가능

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Ingress
  ingress:
    - from:
        - ipBlock:
            cidr: 172.17.0.0/16
            except:
              - 172.17.1.0/24
        - namespaceSelector:
            matchLabels:
              project: myproject
        - podSelector:
            matchLabels:
              role: frontend
  ports:
    - protocol: TCP
      port: 6379
```

네트워크 정책 적용

▶ 인그레스 정책 만들어보기

- 앱 생성

```
kubectl run hello-web --labels app=hello \
--image=gcr.io/google-samples/hello-app:1.0 --port 8080 --expose
```

- 네트워크 폴리시 적용

```
# hello-allow-from-foo
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: hello-allow-from-foo
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: hello
  ingress:
  - from:
    - podSelector:
      matchLabels:
        app: foo
```

네트워크 정책 적용

▶ 인그레스 정책 만들어보기

- 서로 다른 레이블을 가진 파드를 만들고 네트워크 폴리시 동작 확인

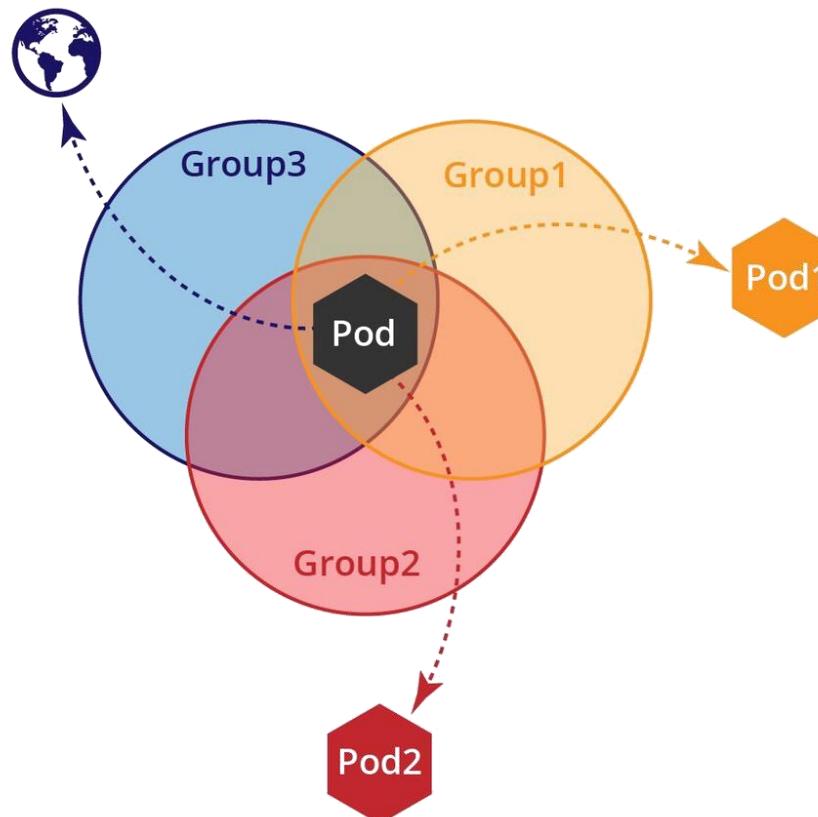
```
kubectl run -l app=foo --image=alpine --restart=Never --rm -i -t test-1  
wget -qO- --timeout=2 http://hello-web:8080
```

```
kubectl run -l app=other --image=alpine --restart=Never --rm -i -t test-1  
wget -qO- --timeout=2 http://hello-web:8080
```

네트워크 정책 적용

▶ 여러 개의 정책이 겹치면 어떻게 될까?

- OR 연산을 수행해 하나라도 허용하면 허용

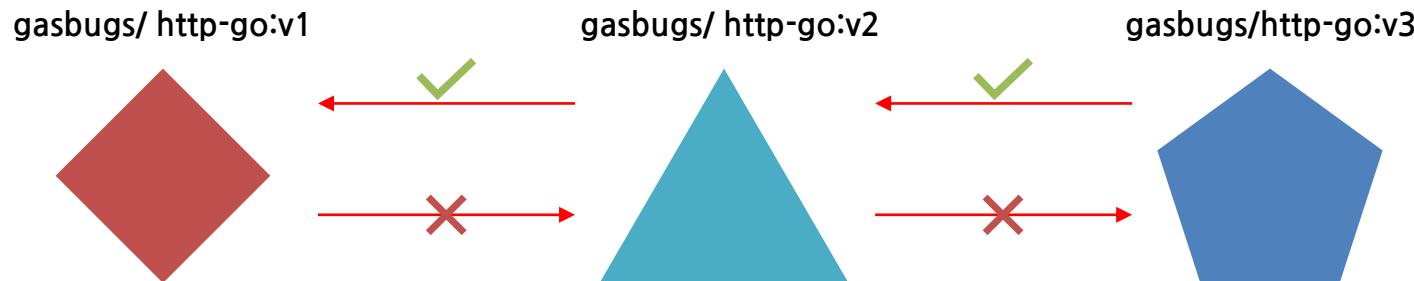


이미지 출처: <https://medium.com/@reuvenharrison/an-introduction-to-kubernetes-network-policies-for-security-people-ba92dd4c809d>

네트워크 정책 적용

연습 문제

- 다음과 같이 통신이 가능하도록 구성하라(v 표시는 통신 가능, x 표시는 통신 불가).



쿠버네티스 감사(Audit) 기능 활성화

쿠버네티스 감사(Audit) 기능 활성화

▶ 쿠버네티스 감사 기능

- 클러스터에서 발생하는 다양한 작업들에 대해 관리자가 모니터링을 하기 위해 audit 기능을 활성화
- 쿠버네티스 audit 기능을 사용해 실시간으로 각종 유저들이 사용하는 API 정보를 확인 가능
- 쿠버네티스 audit는 보안 관련 레코드 세트를 시간순으로 저장
- 클러스터는 사용자 정보와 Kubernetes API를 사용하는 애플리케이션 및 컨트롤플레인 자체에서 발생하는 활동을 감사

▶ 감사 기능을 통해 클러스터 관리자가 확인할 수 있는 정보들

- 무슨 일이 일어났는가?
- 언제 일어났는가?
- 누가 그것을 시작했는가?
- 과거에 무슨 일이 있었는가?
- 어디에서 관찰되었는가?
- 어디에서 시작되었는가?
- 결과는 어디로 흘러가는가?

쿠버네티스 감사(Audit) 기능 활성화

▶ kube-apiserver를 통해 실행

- 감사 레코드는 kube-apiserver를 통해서 실행
- 실행의 각 단계에 대한 각 요청은 감사 이벤트를 생성하고
- 특정 정책에 따라 사전 처리되고 백엔드에 기록
- 각 요청은 연결된 단계에 따라 기록하는 정보가 다름
- 요청에 따른 기록 정보

단계	설명
RequestReceived	감사 핸들러가 요청을 받는 즉시 그리고 핸들러 체인 아래로 위임되기 전에 생성된 이벤트에 대한 단계다.
ResponseStarted	응답 헤더가 전송되었거나 응답 본문이 전송되기 전 상태다. 이 단계는 장기 실행 요청(예: watch)에 대해서만 생성된다.
ResponseComplete	응답 본문이 완료되었으며 더 이상 바이트가 전송되지 않는 단계다.
Panic	패닉이 발생했을 때 생성되는 이벤트다.

쿠버네티스 감사(Audit) 기능 활성화

▣ 감사 정책

- 감사 정책은 기록해야 하는 이벤트와 포함해야 하는 데이터에 대한 규칙을 정의
 - 감사 정책 객체 구조는 audit.k8s.io API 그룹에 정의
 - 이벤트가 처리되면 규칙 목록과 순서대로 비교
 - 첫 번째로 일치하는 규칙은 이벤트의 감사 수준을 설정한다
-
- 정의된 감사 수준

감사 수준	설명
None	이 규칙과 일치하는 이벤트를 기록하지 않습니다.
Metadata	요청 메타데이터(요청 사용자, 타임스탬프, 리소스, 동사 등)를 기록하지만 요청 또는 응답 본문은 기록하지 않습니다.
Request	이벤트 메타데이터 및 요청 본문을 기록하지만 응답 본문은 기록하지 않습니다. 리소스가 아닌 요청에는 적용되지 않습니다.
RequestResponse	이벤트 메타데이터, 요청 및 응답 본문을 기록합니다. 리소스가 아닌 요청에는 적용되지 않습니다.

쿠버네티스 감사(Audit) 기능 활성화

▶ 정책 파일 구성하기

- 다음은 감사 정책 파일의 예시: <https://blog.naver.com/isc0304/222509921722>
- 일을 마스터 노드의 /etc/kubernetes/audit-policy.yaml에 작성
- 각 룰에 대한 설명은 주석을 참조

```
apiVersion: audit.k8s.io/v1 # This is required.

kind: Policy

# RequestReceived 단계의 모든 요청에 대해 감사 이벤트를 생성하지 말아야 한다.

omitStages:
  - "RequestReceived"

rules:
  # RequestResponse 수준에서 포드 변경 사항 기록
  - level: RequestResponse
    resources:
      - group: ""
        # 리소스 "포드"는 RBAC 정책과 일치하는 포드의 하위 리소스에 대한 요청과 일치하지 않습니다.
        resources: ["pods"]
  # 메타데이터 수준에서 "pods/log", "pods/status"를 기록합니다.
  - level: Metadata
    resources:
      - group: ""
        resources: ["pods/log", "pods/status"]
```

쿠버네티스 감사(Audit) 기능 활성화

▶ kube-apiserver 파드 설정 변경과 재시작

- 마스터 노드에 --audit-policy-file 플래그를 설정
- 플래그를 생략하면 이벤트가 기록되지 않음
- /etc/kubernetes/manifests/kube-apiserver.yaml 파일을 열고 다음 작업을 진행
- .spec.containers[0].command에 다음 인자를 추가

```
- --audit-policy-file=/etc/kubernetes/pki/audit.conf  
- --audit-log-path=/var/log/audit.log
```

- .spec.container[0].volumeMounts와 .spec.volumes에 다음 내용 추가

```
volumeMounts:  
- mountPath: /etc/kubernetes/audit-policy.yaml  
  name: audit  
  readOnly: true  
- mountPath: /var/log/audit.log  
  name: audit-log  
  readOnly: false
```

```
volumes:  
- name: audit  
  hostPath:  
    path: /etc/kubernetes/audit-policy.yaml  
    type: File  
- name: audit-log  
  hostPath:  
    path: /var/log/audit.log  
    type: FileOrCreate
```

쿠버네티스 감사(Audit) 기능 활성화

▣ 감사 로그 확인

- api 서버가 잘 올라오는지 확인

- (오류 발생 시 트러블슈팅 필요)

```
$ kubectl get pod -n kube-system
```

- 마스터 노드에서 /etc/logs/audit.log 로그 확인

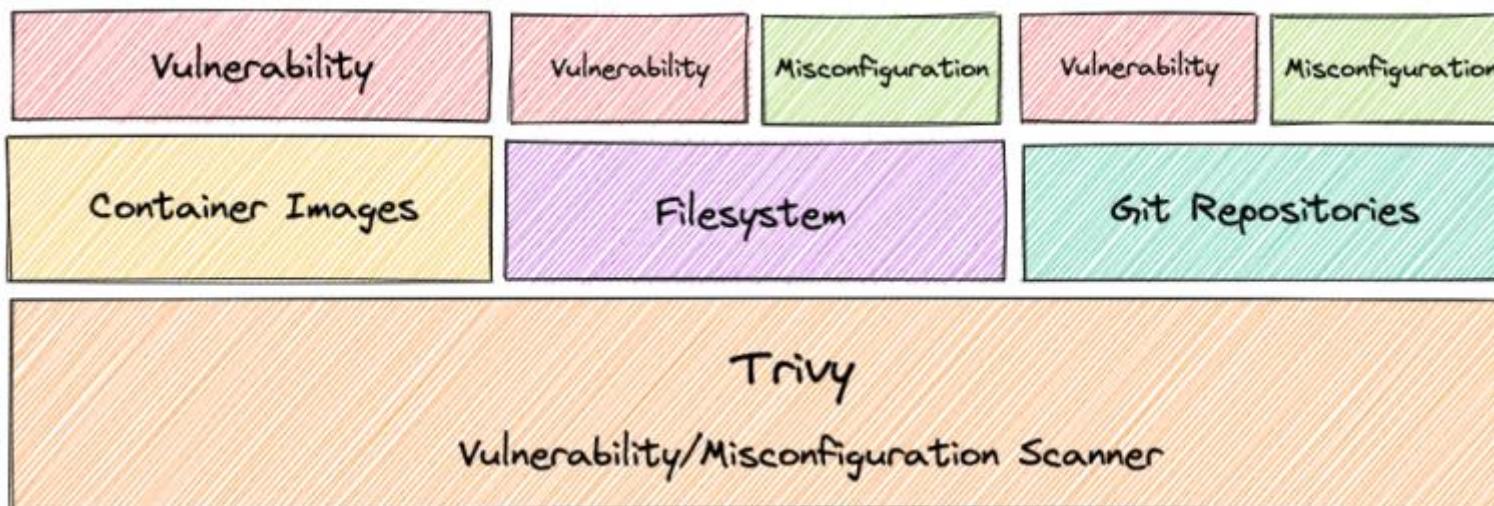
```
{
    "kind": "Event",
    "apiVersion": "audit.k8s.io/v1",
    "level": "Request", // 로그 수준, 요청을 기록
    "auditID": "d8d2a777-d6f9-4b29-b8f4-70cba6b98986",
    "stage": "ResponseStarted",
    "requestURI": "/api/v1/services?allowWatchBookmarks=true\u0026resourceVersion=59106",
    "verb": "watch", // api 실행
    "user": {
        "username": "system:kube-scheduler", // 요청을 수행한 유저
        "groups": [
            "system:authenticated" # 유저가 속한 그룹
        ]
    },
    "sourceIPs": [
        "10.0.2.15" // 요청한 IP
    ],
    "userAgent": "kube-scheduler/v1.22.1 (linux/amd64) kubernetes/632ed30/scheduler",
    "objectRef": {
        "resource": "services",
        "apiVersion": "v1"
    },
    "responseStatus": {
        "metadata": {
            "status": "Success",
            "message": "Connection closed early",
            "code": 200 // 응답 코드
        },
        "requestReceivedTimestamp": "2021-09-02T03:39:34.261603Z",
        "stageTimestamp": "2021-09-02T03:45:11.263737Z",
        "annotations": {
            "authorization.k8s.io/decision": "allow",
            "authorization.k8s.io/reason": "RBAC: allowed by ClusterRoleBinding \\"system:kube-scheduler\\"
        }
    }
}
```

Trivy를 활용한 컨테이너 취약점 진단

Trivy를 활용한 컨테이너 취약점 진단

▶ Trivy 소개

- 취약점을 간단히 스캔할 수 있는 도구
- 컨테이너 이미지, 파일 시스템 및 Git 리포지토리의 취약점, 미설정 구성 문제에 대한 다양한 진단
- OS 패키지(알파인, RHEL, CentOS 등)와 언어별 패키지(Bundler, Composer, npm, yarn 등)의 취약점을 탐지
- 또한 테라폼, Dockerfile 및 Kubernetes와 같은 IaC(코드) 파일로 인프라를 스캔하여 배포를 공격 위험에 노출시키는 잠재적 구성 문제를 검색하는 기능
- 바이너리를 설치하거나 컨테이너를 사용하는 등 다양하고도 간단한 방법으로 취약점을 진단



Trivy를 활용한 컨테이너 취약점 진단

▶ 도커 컨테이너를 활용한 Trivy 설치

- 컨테이너를 사용해서 설치하면 호환성 등의 문제 없이 1분내로 Trivy를 구성
- -v 옵션을 사용해 캐시 디렉토리를 마운트
- socket을 공유해 현재 호스트에 구성된 도커 소켓을 통해 이미지를 컨트롤
- --rm 옵션이 설정되어 있어 실행 후에는 컨테이너가 자동으로 삭제
- 다음 명령을 사용해 다운로드 후에 trivy가 자동으로 실행되며 도움말이 출력

```
$ docker run --rm -v trivy-cache:/root/.cache/ \
-v /var/run/docker.sock:/var/run/docker.sock aquasec/trivy:latest
```

NAME:

trivy - A simple and comprehensive vulnerability scanner for containers

USAGE:

trivy command [command options] target

COMMANDS:

image, i	scan an image
filesystem, fs	scan local filesystem
repository, repo	scan remote repository
client, c	client mode
server, s	server mode
config, conf	scan config files
plugin, p	manage plugins

Trivy를 활용한 컨테이너 취약점 진단

▶ Trivy를 활용한 도커 이미지 취약점 진단

- Trivy에 image 명령을 사용하면 도커 이미지의 취약점을 진단
- 알려진 컴포넌트에 대한 버전별 취약점 DB를 다운로드
- 현재 구성되어 있는 다양한 모듈들의 취약점을 도출

```
$ docker run --rm -v trivy-cache:/root/.cache/ \
-v /var/run/docker.sock:/var/run/docker.sock \
aquasec/trivy:latest \
image gasbugs/http-go
```

Trivy를 활용한 컨테이너 취약점 진단

▶ Trivy 결과 확인

```
2021-09-19T05:57:18.809Z [34mINFO[0m Detected OS: debian
2021-09-19T05:57:18.809Z [34mINFO[0m Detecting Debian vulnerabilities...
2021-09-19T05:57:18.855Z [34mINFO[0m Number of language-specific files: 0
```

gasbugs/http-go (debian 10.0)

```
=====
Total: 1217 (UNKNOWN: 5, LOW: 65, MEDIUM: 644, HIGH: 451, CRITICAL: 52)
```

LIBRARY	VULNERABILITY ID	SEVERITY	INSTALLED VERSION	FIXED VERSION	TITLE
apt	CVE-2020-27350	MEDIUM	1.8.2	1.8.2.2	apt: integer overflows and underflows while parsing .deb packages --> avd.aquasec.com/nvd/cve-2020-27350
	CVE-2020-3810			1.8.2.1	Missing input validation in the ar/tar implementations of APT before version 2.1.2... --> avd.aquasec.com/nvd/cve-2020-3810
	CVE-2011-3374	LOW			It was found that apt-key in apt, all versions, do not correctly... --> avd.aquasec.com/nvd/cve-2011-3374
bash	CVE-2019-18276	HIGH	5.0-4		bash: when effective UID is not equal to its real UID the... --> avd.aquasec.com/nvd/cve-2019-18276

kube-bench를 활용한 쿠버네티스 보안 점검

kube-bench를 활용한 쿠버네티스 보안 점검

▶ 쿠버네티스 CIS 벤치마크 소개

- CIS에서는 클라우드에서 준수해야하는 다양한 보안 점검 리스트를 benchmark로 제공한다. 다음 사이트를 통해 상세히 명시된 pdf 파일을 다운로드할 수 있다. (개인정보 입력 필요)
- <https://www.cisecurity.org/benchmark/kubernetes/>

The screenshot shows the CIS Benchmarks page for Kubernetes. At the top, there's a green banner with a 'Limited Time Offer' for a new CIS SecureSuite Membership. The CIS logo and name are on the left, along with a 'Confidence in the Connected World' tagline. On the right, there are links for 'CIS Hardened Images', 'Support', 'CIS WorkBench Sign-in', and an 'Alert Level: Guarded'. Below the banner, there are navigation links for 'Why CIS', 'Solutions', 'Join CIS', and 'Resources'. The main content area features a large image of two people working at desks with multiple monitors. To the left of the image, the CIS Benchmarks logo is displayed, followed by the title 'Securing Kubernetes' and a subtitle 'An objective, consensus-driven security guideline for the Kubernetes Server Software'. Below this, there's a section for a step-by-step checklist with a 'Download Latest CIS Benchmark' button. Another section discusses the history of the benchmark and links to the community. At the bottom, there's information about other CIS Benchmark versions and a complete archive.

Limited Time Offer: Save up to 20% on a new CIS SecureSuite Membership | Learn more X

CIS Center for Internet Security™ Confidence in the Connected World

CIS Hardened Images Support CIS WorkBench Sign-in Alert Level: Guarded

Why CIS Solutions Join CIS Resources

Home • Resources • Platforms • Kubernetes

CIS Benchmarks™

Securing Kubernetes
An objective, consensus-driven security guideline for the Kubernetes Server Software.

A step-by-step checklist to secure Kubernetes:

Download Latest CIS Benchmark →
Free to Everyone

For Kubernetes 1.0.0 (CIS Alibaba Cloud Container Service For Kubernetes (ACK) Benchmark version 1.0.0)

CIS has worked with the community since 2017 to publish a benchmark for Kubernetes
Join the Kubernetes community →

Other CIS Benchmark versions:
For Kubernetes (CIS Kubernetes Benchmark version 1.6.0)
Complete CIS Benchmark Archive →

kube-bench를 활용한 쿠버네티스 보안 점검

▶ kube-bench?

- aquasecurity에서는 쿠버네티스에서 사용할 수 있는 kube-bench라는 오픈소스를 구성해 많은 사용자들이 쿠버네티스에 설정 취약성을 파악할 수 있도록 구성
- 벤치마크에서 모든 부분을 구현하지는 않았지만 그래도 상당한 부분을 자동화
- 깃헙의 릴리즈 페이지를 통해 다양한 플랫폼에서 컴파일된 파일을 구성
- <https://github.com/aquasecurity/kube-bench/releases>

release v0.6.4 downloads 39k docker pulls / kube-bench 20M go report A Build passing License Apache 2.0
 Docker image Source commit codecov 65%



kube-bench is tool that checks whether Kubernetes is deployed securely by running the checks documented in the [Kubernetes Benchmark](#).

Tests are configured with YAML files, making this tool easy to update as test specifications evolve.

```
[INFO] 1 Master Node Security Configuration
[INFO] 1.1 API Server
[FAIL] 1.1.1 Ensure that the --allow-privileged argument is set to false (Scored)
[FAIL] 1.1.2 Ensure that the --anonymous-auth argument is set to false (Scored)
[PASS] 1.1.3 Ensure that the --basic-auth-file argument is not set (Scored)
[PASS] 1.1.4 Ensure that the --insecure-allow-any-token argument is not set (Scored)
[FAIL] 1.1.5 Ensure that the --kubelet-https argument is set to true (Scored)
[PASS] 1.1.6 Ensure that the --insecure-bind-address argument is not set (Scored)
[PASS] 1.1.7 Ensure that the --insecure-port argument is set to 0 (Scored)
[PASS] 1.1.8 Ensure that the --secure-port argument is not set to 0 (Scored)
[FAIL] 1.1.9 Ensure that the --profiling argument is set to false (Scored)
[FAIL] 1.1.10 Ensure that the --repair-malformed-updates argument is set to false (Scored)
[PASS] 1.1.11 Ensure that the admission control policy is not set to AlwaysAdmit (Scored)
[FAIL] 1.1.12 Ensure that the admission control policy is set to AlwaysPullImages (Scored)
[FAIL] 1.1.13 Ensure that the admission control policy is set to DenyEscalatingExec (Scored)
[FAIL] 1.1.14 Ensure that the admission control policy is set to SecurityContextDeny (Scored)
[PASS] 1.1.15 Ensure that the admission control policy is set to NamespaceLifecycle (Scored)
[FAIL] 1.1.16 Ensure that the --audit-log-path argument is set as appropriate (Scored)
[FAIL] 1.1.17 Ensure that the --audit-log-maxage argument is set to 30 or as appropriate (Scored)
[FAIL] 1.1.18 Ensure that the --audit-log-maxbackup argument is set to 10 or as appropriate (Scored)
[FAIL] 1.1.19 Ensure that the --audit-log-maxsize argument is set to 100 or as appropriate (Scored)
[PASS] 1.1.20 Ensure that the --authorization-mode argument is not set to AlwaysAllow (Scored)
[PASS] 1.1.21 Ensure that the --token-auth-file parameter is not set (Scored)
[FAIL] 1.1.22 Ensure that the --kubelet-certificate-authority argument is set as appropriate (Scored)
```

kube-bench를 활용한 쿠버네티스 보안 점검

▶ 클러스터 진단

- kubectl이 실행한 가능한 환경에서 kube-bench를 설치
- 릴리즈로부터 파일을 다운로드 받고 실행

릴리즈 파일 다운로드

```
wget https://github.com/aquasecurity/kube-bench/releases/download/v0.6.3/kube-bench\_0.6.3\_linux\_amd64.tar.gz
```

```
tar - xf kube-bench_0.6.3_linux_amd64.tar.gz  
sudo mv kube-bench /usr/bin/ # 설치 완료
```

깃헙 프로젝트 다운로드

```
git clone https://github.com/aquasecurity/kube-bench  
cd kube-bench  
kube-bench --config-dir `pwd`/cfg --config `pwd`/cfg/config.yaml
```

```
gasbugs21c@cloudshell:~/kube-bench (gkesecurity)$ kube-bench --config-dir `pwd`/cfg --config `pwd`/cfg/config.yaml  
[INFO] 4 Worker Node Security Configuration  
[INFO] 4.1 Worker Node Configuration Files  
[FAIL] 4.1.1 Ensure that the kubelet service file permissions are set to 644 or more restrictive (Automated)  
[FAIL] 4.1.2 Ensure that the kubelet service file ownership is set to root:root (Automated)  
[PASS] 4.1.3 If proxy kubeconfig file exists ensure permissions are set to 644 or more restrictive (Manual)  
[PASS] 4.1.4 Ensure that the proxy kubeconfig file ownership is set to root:root (Manual)  
[FAIL] 4.1.5 Ensure that the --kubeconfig kubelet.conf file permissions are set to 644 or more restrictive (Automated)  
[WARN] 4.1.6 Ensure that the --kubeconfig kubelet.conf file ownership is set to root:root (Manual)  
[WARN] 4.1.7 Ensure that the certificate authorities file permissions are set to 644 or more restrictive (Manual)  
[WARN] 4.1.8 Ensure that the client certificate authorities file ownership is set to root:root (Manual)  
[FAIL] 4.1.9 Ensure that the kubelet --config configuration file has permissions set to 644 or more restrictive (Automated)  
[FAIL] 4.1.10 Ensure that the kubelet --config configuration file ownership is set to root:root (Automated)  
[INFO] 4.2 Kubelet  
[FAIL] 4.2.1 Ensure that the anonymous-auth argument is set to false (Automated)  
[FAIL] 4.2.2 Ensure that the --authorization-mode argument is not set to AlwaysAllow (Automated)  
[FAIL] 4.2.3 Ensure that the --client-ca-file argument is set as appropriate (Automated)  
[WARN] 4.2.4 Ensure that the --read-only-port argument is set to 0 (Manual)  
[WARN] 4.2.5 Ensure that the --streaming-connection-idle-timeout argument is not set to 0 (Manual)  
[FAIL] 4.2.6 Ensure that the --protect-kernel-defaults argument is set to true (Automated)  
[FAIL] 4.2.7 Ensure that the --make-iptables-util-chains argument is set to true (Automated)  
[WARN] 4.2.8 Ensure that the --hostname-override argument is not set (Manual)  
[WARN] 4.2.9 Ensure that the --event-qps argument is set to 0 or a level which ensures appropriate event capture (Manual)  
[WARN] 4.2.10 Ensure that the --tls-cert-file and --tls-private-key-file arguments are set as appropriate (Manual)  
[WARN] 4.2.11 Ensure that the --rotate-certificates argument is not set to false (Manual)  
[WARN] 4.2.12 Verify that the RotateKubeletServerCertificate argument is set to true (Manual)
```

```
== Summary total ==  
2 checks PASS  
10 checks FAIL  
35 checks WARN  
0 checks INFO
```

kube-bench를 활용한 쿠버네티스 보안 점검

▶ 노드 진단

- GKE에서 실행하는 경우 SSH로 접속해서 실행
- 클라우드 쿠버네티스에서는 COS를 사용하므로 읽기 전용 구간이 많음
- GKE에서는 /home/Kubernetes/bin에 설정이 가능

```
# 릴리즈 파일 다운로드  
wget https://github.com/aquasecurity/kube-bench/releases/download/v0.6.3/kube-bench\_0.6.3\_linux\_amd64.tar.gz  
tar -xf kube-bench_0.6.3_linux_amd64.tar.gz  
sudo mv kube-bench /home/kubernetes/bin
```

```
# 깃헙 프로젝트 다운로드  
git clone https://github.com/aquasecurity/kube-bench  
cd kube-bench  
kube-bench --config-dir `pwd`/cfg --config `pwd`/cfg/config.yaml
```

```
== Summary total ==  
9 checks PASS  
5 checks FAIL  
33 checks WARN  
0 checks INFO
```

falco를 활용한 쿠버네티스 컨테이너 보안 모니터링

falco를 활용한 쿠버네티스 컨테이너 보안 모니터링

▶ falco 개요

- 클라우드 네이티브 런타임 보안 프로젝트인 Falco는 사실상 Kubernetes 위협 탐지 엔진
- 2016년 Sysdig에 의해 만들어졌으며 인큐베이션 수준 프로젝트
- CNCF에 합류한 최초의 런타임 보안 프로젝트
- Falco는 예기치 않은 애플리케이션 동작을 감지하고 런타임 시 위협에 대해 경고
- CKS 시험에서도 나올 정도로 CNCF에서 밀고 있는 프로그램



Falco, the cloud-native runtime security project, is the de facto **Kubernetes threat detection engine**

Falco was created by Sysdig in 2016 and is the first runtime security project to join CNCF as an incubation-level project. Falco detects unexpected application behavior and alerts on threats at runtime.



<https://falco.org/>

falco를 활용한 쿠버네티스 컨테이너 보안 모니터링

▶ 호스트에 falco 설정 정보 확인

- log_syslog: true 옵션은 /var/log/syslog에 falco 관련 로그를 함께 로깅한다는 것을 의미
- rules_file 섹션에는 실행할 때 참조하는 룰의 위치가 명시됨

```
$ sudo vim /etc/falco/falco.yaml
# 룰 파일 목록

rules_file:
  - /etc/falco/falco_rules.yaml
  - /etc/falco/falco_rules.local.yaml
  - /etc/falco/k8s_audit_rules.yaml
  - /etc/falco/rules.d

...
# 로그 저장 형식

log_stderr: true
log_syslog: true
```

falco를 활용한 쿠버네티스 컨테이너 보안 모니터링

▶ falco 로그

- 다음 명령으로 /var/log/syslog에서 확인

```
$ sudo cat /var/log/syslog | grep falco

Sep 22 03:33:40 node01 kernel: [11782.615640] falco: loading out-of-tree module taints
Sep 22 03:33:40 node01 kernel: [11782.616590] falco: module verification failed: signat
Sep 22 03:33:40 node01 kernel: [11782.618203] falco: driver loading, falco 17f5df52a7d9
Sep 22 03:33:40 node01 falco: Falco version 0.29.1 (driver version 17f5df52a7d9ed6bb12c
Sep 22 03:33:40 node01 falco[85088]: Wed Sep 22 03:33:40 2021: Falco version 0.29.1 (dr
Sep 22 03:33:40 node01 falco: Falco initialized with configuration file /etc/falco/falc
Sep 22 03:33:40 node01 falco[85088]: Wed Sep 22 03:33:40 2021: Falco initialized with c
Sep 22 03:33:40 node01 falco: Loading rules from file /etc/falco/falco_rules.yaml:
Sep 22 03:33:40 node01 falco[85088]: Wed Sep 22 03:33:40 2021: Loading rules from file
Sep 22 03:33:40 node01 falco: Loading rules from file /etc/falco/falco_rules.local.yaml
Sep 22 03:33:40 node01 falco[85088]: Wed Sep 22 03:33:40 2021: Loading rules from file
Sep 22 03:33:40 node01 falco: Loading rules from file /etc/falco/k8s_audit_rules.yaml:
Sep 22 03:33:40 node01 falco[85088]: Wed Sep 22 03:33:40 2021: Loading rules from file
```

falco를 활용한 쿠버네티스 컨테이너 보안 모니터링

➤ falco에서 지원하는 필드

- 생성하는 로그를 어떤 정보를 매칭해 어떤 데이터를 저장할지 필드
- 필드의 개수가 워낙 많고, 많은 정보를 내포
- 너무 많아 모든 것을 다 외우기는 어려움

다양한 클래스를 지원

Name	Type	Description
evt.num	UINT64	event number.
evt.time	CHARBUF	event timestamp as a time string that includes the nanosecond part.
evt.time.s	CHARBUF	event timestamp as a time string with no nanoseconds.
evt.time.iso8601	CHARBUF	event timestamp in ISO 8601 format, including nanoseconds and time zone offset (in UTC).
evt.datetime	CHARBUF	event timestamp as a time string that includes the date.
evt.rawtime	ABSTIME	absolute event timestamp, i.e. nanoseconds from epoch.
evt.rawtime.s	ABSTIME	integer part of the event timestamp (e.g. seconds since epoch).
evt.rawtime.ns	ABSTIME	fractional part of the absolute event timestamp.
evt.reltime	RELTIME	number of nanoseconds from the beginning of the capture.
evt.reltimes	RELTIME	number of seconds from the beginning of the capture.

클래스 내부에는 다수의 필드를 포함

Create child page
 Create documentation issue
 Create project issue
System Calls (source syscall)
Field Class: evt
Field Class: process
Field Class: user
Field Class: group
Field Class: container
Field Class: fd
Field Class: syslog
Field Class: fdlist
Field Class: k8s
Field Class: mesos
Field Class: span
Field Class: evtin
Kubernetes Audit Events (source k8s_audit)

falco를 활용한 쿠버네티스 컨테이너 보안 모니터링

▶ falco 룰 예제

- falco 사이트에 몇가지 예제를 제공
- <https://falco.org/docs/examples/>

A shell is run in a container Rule

```
- macro: container
  condition: container.id != host

- macro: spawned_process
  condition: evt.type = execve and evt.dir=<

- rule: run_shell_in_container
  desc: a shell was spawned by a non-shell program in a container. Container entrypoints are excluded.
  condition: container and proc.name = bash and spawned_process and proc.pname exists and not proc.pname in (bash, docker)
  output: "Shell spawned in a container other than entrypoint (user=%user.name container_id=%container.id
  container_name=%container.name shell=%proc.name parent=%proc.pname cmdline=%proc.cmdline) "
  priority: WARNING
```

룰 조건

- 1) container and spawned_process and:
앞서 생성한 두 macro는 참이어야 하고
- 2) proc.name = bash and: 프로세스 이름은 bash여야 하며,
- 3) proc.pname exists and: 부모 프로세스 이름이 있어햐하고
- 4) not proc.pname in (bash, docker): 부모 프로세스 이름은 bash나 docker여서는 안된다.

falco를 활용한 쿠버네티스 컨테이너 보안 모니터링

간단한 로그 생성과 탐지

- python 이미지 컨테이너를 구성하고 패키지 매니저를 통해 vim을 설치

```
kubectl run py --image=python:3.7 -- sleep infinity  
kubectl exec py -- apt update  
kubectl exec py -- apt install -y vim
```

- syslog에 falco를 grep하면 관련 로그 확인 가능

```
# cat /var/log/syslog | grep falco
```

```
Sep 22 05:44:13 node01 falco[123799]: 05:44:13.306534965: Error Package management process launched  
in container (user=root user_loginuid=-1 command=apt update container_id=4831e292a3c8  
container_name=k8s_py_py_default_13877cb8-d825-4831-a8ca-f21a9d79b7f0_0 image=python:3.7)  
Sep 22 05:44:13 node01 falco: 05:44:13.306534965: Error Package management process launched in  
container (user=root user_loginuid=-1 command=apt update container_id=4831e292a3c8  
container_name=k8s_py_py_default_13877cb8-d825-4831-a8ca-f21a9d79b7f0_0 image=python:3.7)  
Sep 22 05:45:02 node01 falco[123799]: 05:45:02.431342562: Error Package management process launched  
in container (user=root user_loginuid=-1 command=apt install -y vim container_id=48f80132919f  
container_name=k8s_py_py_default_2293f44d-dfd3-4508-b001-9fc0d43a4c9e_0 image=python:3.7)  
Sep 22 05:45:02 node01 falco: 05:45:02.431342562: Error Package management process launched in  
container (user=root user_loginuid=-1 command=apt install -y vim container_id=48f80132919f  
container_name=k8s_py_py_default_2293f44d-dfd3-4508-b001-9fc0d43a4c9e_0 image=python:3.7)
```

리소스 로깅과 모니터링

▶ 클러스터 컴포넌트 모니터링

▶ 애플리케이션 로그 확인

▶ 큐브 대시보드 설치와 사용

▶ 프로메테우스 그라파나를 활용한 리소스 모니터링

▶ Istio를 활용한 네트워크 메시 모니터링

▶ EFK를 활용한 k8s 로그 모니터링

▶ 오토 스케일링 HPA 워크스루

▶ Jaeger 트레이싱 튜토리얼



쿠버네티스 모니터링 시스템과 아키텍처

쿠버네티스 모니터링 시스템과 아키텍처

▣ 모니터링 서비스 플랫폼

- 쿠버네티스를 지원하는 다양한 모니터링 플랫폼
- 쿠버네티스의 메트릭 수집 모니터링 아키텍처에서 코어메트릭 파이프라인 경량화
- 힙스터를 deprecated하고 모니터링 표준으로 메트릭서버(metrics-server) 도입

```
$ kubectl top node
```

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
gke-standard-cluster-1-default-pool-0cadf7f5-25rl	42m	4%	572Mi	21%
gke-standard-cluster-1-default-pool-0cadf7f5-jt4b	48m	5%	612Mi	23%
gke-standard-cluster-1-default-pool-0cadf7f5-tssr	49m	5%	618Mi	23%

```
$ kubectl top pod
```

NAME	CPU(cores)	MEMORY(bytes)
envar-demo	0m	9M

Heapster
(deprecated)

Metrics
Service

cAdvisor



쿠버네티스 모니터링 시스템과 아키텍처

▶ 리소스 모니터링 도구

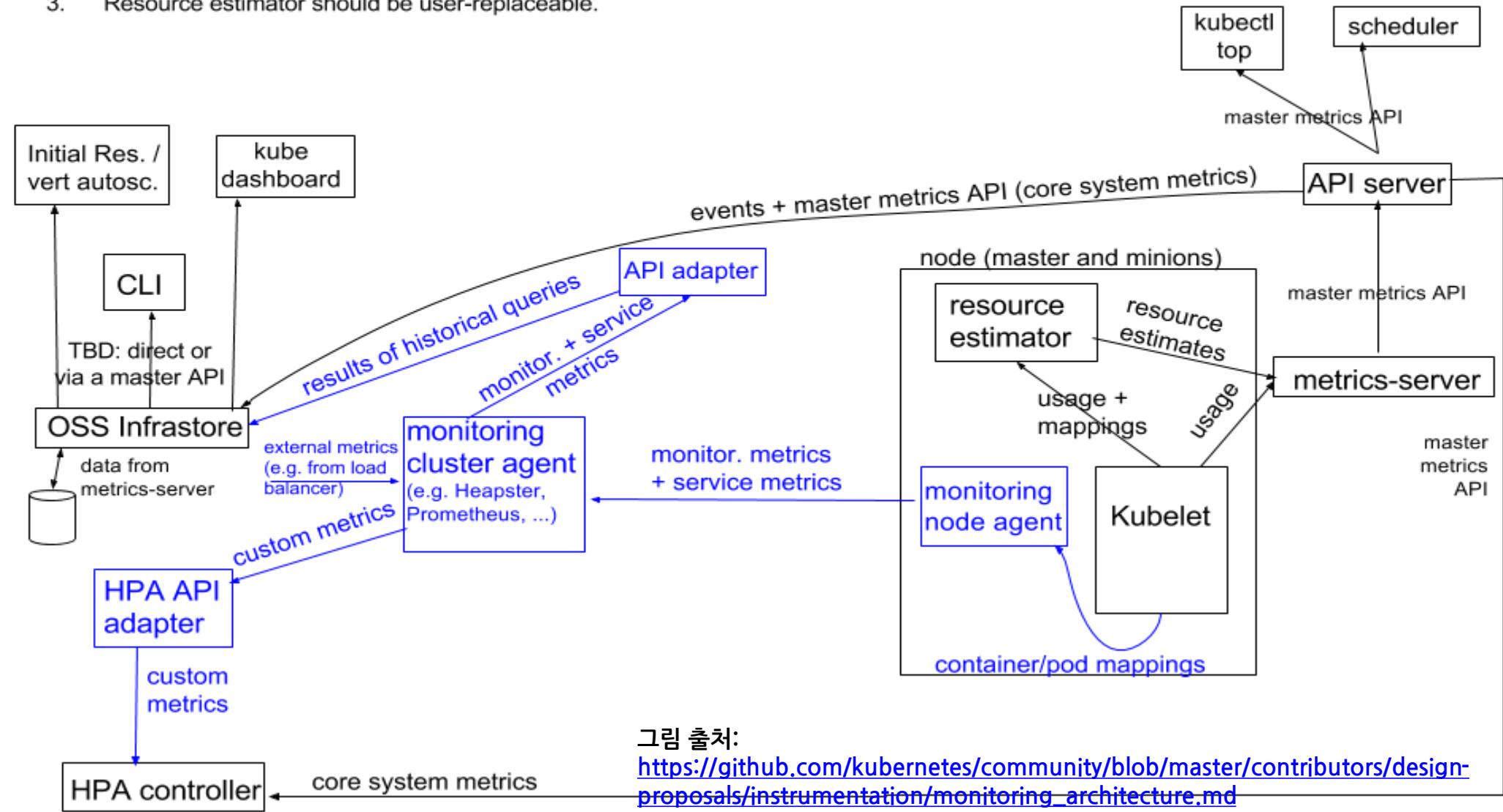
- 쿠버네티스 클러스터 내의 애플리케이션 성능을 검사
- 쿠버네티스는 각 레벨에서 애플리케이션의 리소스 사용량에 대한 상세 정보를 제공
- 애플리케이션의 성능을 평가하고 병목 현상을 제거하여 전체 성능을 향상을 도모
- 리소스 메트릭 파이프라인
 - kubectl top 등의 유틸리티 관련된 메트릭들로 제한된 집합을 제공
 - 단기 메모리 저장소인 metrics-server에 의해 수집
 - metrics-server는 모든 노드를 발견하고 kubelet에 CPU와 메모리를 질의
 - kubelet은 kubelet에 통합된 cAdvisor를 통해 레거시 도커와 통합 후 metric-server 리소스 메트릭으로 노출
 - /metrics/resource/v1beta1 API를 사용
- 완전한 메트릭 파이프라인
 - 보다 풍부한 메트릭에 접근
 - 클러스터의 현재 상태를 기반으로 자동으로 스케일링하거나 클러스터를 조정
 - 모니터링 파이프라인은 kubelet에서 메트릭을 가져옴
 - CNCF 프로젝트인 프로메테우스가 대표적
 - custom.metrics.k8s.io, external.metrics.k8s.io API를 사용

Monitoring architecture proposal: OSS

(arrows show direction of metrics flow)

Notes

1. Arrows show direction of metrics flow.
2. Monitoring pipeline is in blue. It is user-supplied and optional.
3. Resource estimator should be user-replaceable.



쿠버네티스 모니터링 시스템과 아키텍처

Metrics-server 설치 방법

- 메트릭스 서버는 쿠버네티스에서 리소스 메트릭 파이프라인을 구성하는 가장 기본 형태
- 그러나 쿠버네티스를 설치한다고해서 메트릭 서버가 자동으로 설치되지는 않음
- 다음 명령을 실행해 공개된 yaml 파일을 사용하여 metrics-server를 설치

```
$ kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.5.1/components.yaml
```

- yaml 파일 내부에서 args를 찾아가서 다음 설정 두 개를 추가

```
kubectl edit deployments.apps -n kube-system metrics-server
```

아규먼트	설명
- --kubelet-insecure-tls	인증서가 인증 기관에 승인 받지 않은 안전하지 않기 때문에 보안적으로 취약하지만 무시하겠다는 의미
- --kubelet-preferred-address-types=InternalIP	kubelet 연결에 사용할 때 사용하는 주소 타입을 지정

쿠버네티스 모니터링 시스템과 아키텍처

Metrics-server 설치 방법

- yaml 파일 내부에서 args를 찾아가서 다음 설정 두 개를 추가

```
$ kubectl edit deployments.apps -n kube-system metrics-server
```

아규먼트	설명
- --kubelet-insecure-tls	인증서가 인증 기관에 승인 받지 않은 안전하지 않기 때문에 보안적으로 취약하지만 무시하겠다는 의미
- --kubelet-preferred-address-types=InternalIP	kubelet 연결에 사용할 때 사용하는 주소 타입을 지정

〈앞 부분 생략〉

spec: containers:

- args:

- --cert-dir=/tmp

- --secure-port=4443

- --kubelet-insecure-tls # 추가된 옵션

- --kubelet-preferred-address-types=InternalIP # 추가된 옵션

〈뒷 부분 생략〉

쿠버네티스 모니터링 시스템과 아키텍처

Metrics-server 설치 방법

- 정보 수집에 시간이 걸리므로 1분 정도 지난 후 명령어 실행
- 파드와 노드의 리소스를 요청하면 정상적으로 리소스를 모니터링

```
$ k top pod -n kube-system
  NAME           CPU(cores)   MEMORY(bytes)
coredns-6955765f44-cfn8r      2m          21Mi
coredns-6955765f44-qvw44      1m          16Mi
etcd-master                  9m          47Mi
kube-apiserver-master        19m         325Mi
kube-controller-manager-master 4m          54Mi
```

```
$ k top nodes
  NAME     CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
master    175m       8%     2880Mi        75%
work1     29m        1%     2653Mi        69%
work2     21m        1%     2319Mi        60%
```

쿠버네티스 모니터링 시스템과 아키텍처

▶ 연습문제

- 현재 가장 많은 CPU를 사용하는 파드는 무엇인가?
- 현재 가장 낮은 MEM을 사용하는 파드는 무엇인가?
- 각각의 파드 이름을 /tmp/maxcpu와 /tmp/minmem에 저장하라.

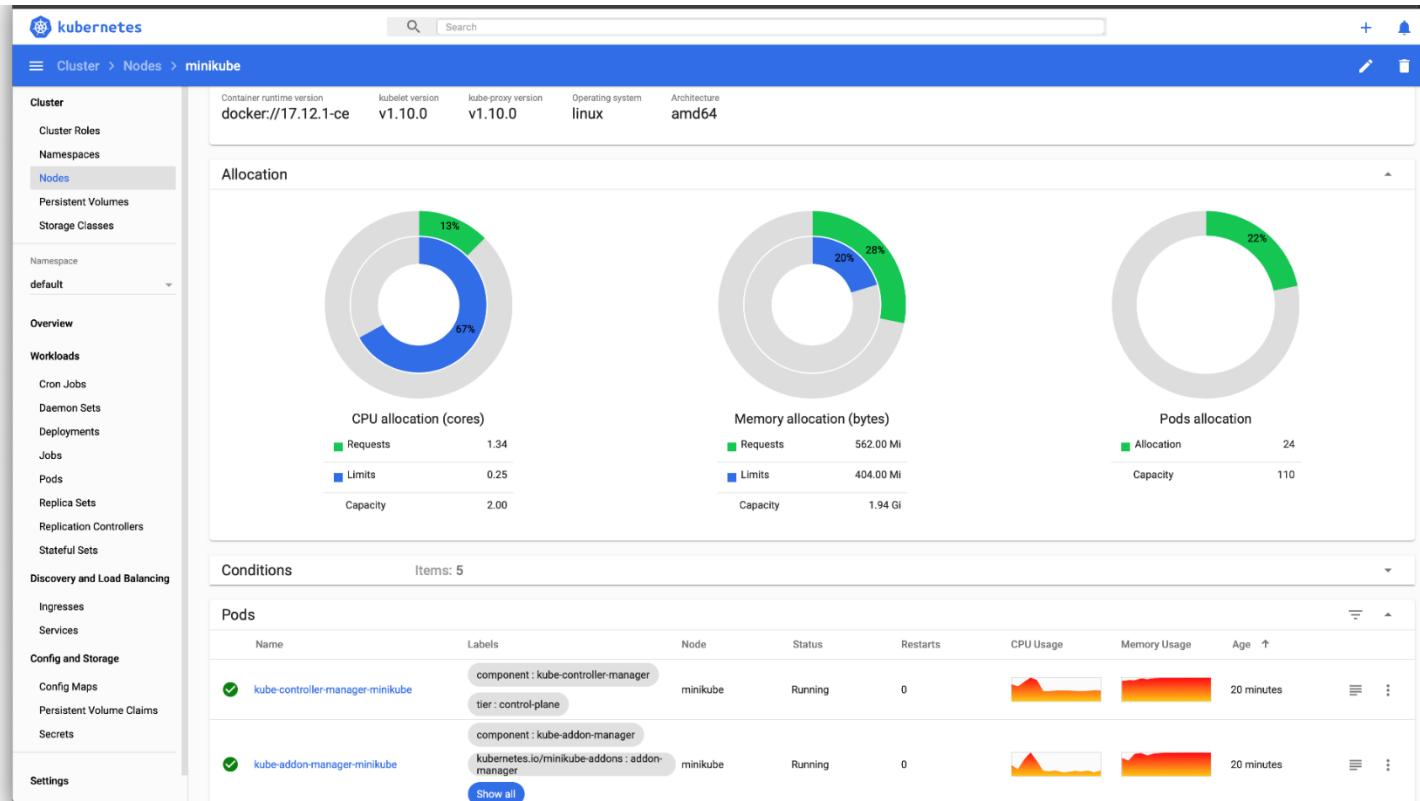
큐브 대시보드 설치와 사용

큐브 대시보드 설치와 사용

Kubernetes Dashboard

- Kubernetes 클러스터용 범용 웹 기반 UI
- 사용자는 클러스터에서 실행중인 응용 프로그램을 관리하고 문제를 해결, 클러스터 자체를 관리
- <https://github.com/kubernetes/dashboard>

thisisunsafe



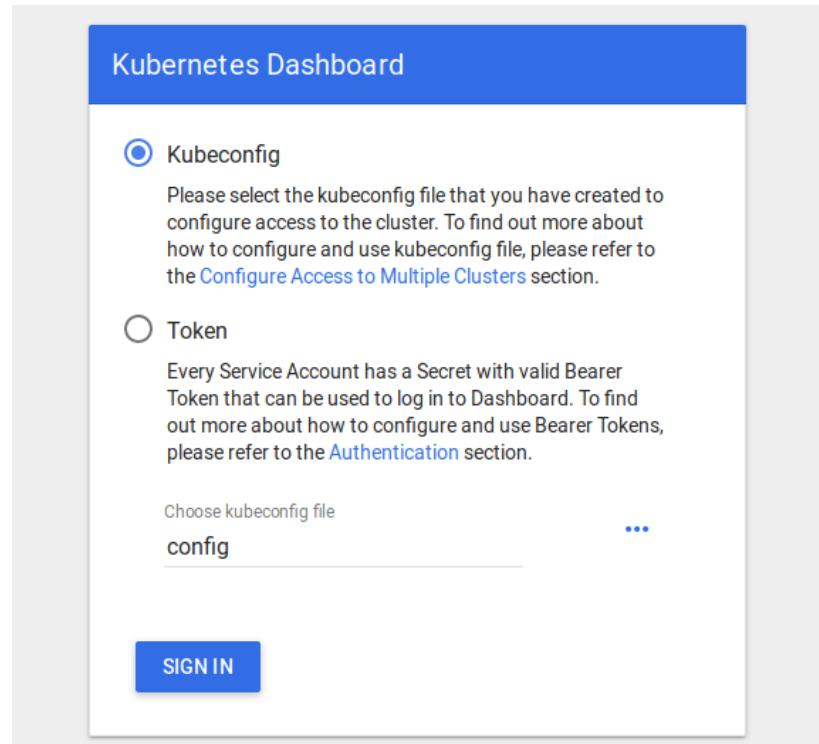
큐브 대시보드 설치와 사용

▶ Kubernetes Dashboard 설치

- <https://github.com/kubernetes/dashboard>
- 다음 명령을 사용하여 git에 올라온 yaml 파일을 바로 적용

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta8/aio/deploy/recommended.yaml
```

- 외부로 443 포트를 열고 접속



큐브 대시보드 설치와 사용

Kubernetes Dashboard 토큰 확인

```
$ kubectl create -f user.yaml
```

```
$ kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes-dashboard get secret | grep admin-user | awk '{print $1}')
```

<출력되는 토큰을 복사>

```
yJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3Mi0iJrdWJlc...  
JuZXRLcy5pb...  
2Ui0iJrdWJ1LXN5c3R1bSIsImt1YmVybmV0ZXMuaw8vc2VydmljZW...  
ppi11c2V...  
5pb...  
nQubmFtZSI6ImFkbWluLXVzZXIiLCJrdWJlc...  
LmlvL3Nlc...  
...
```

admin-user.yaml

```
apiVersion: v1  
kind: ServiceAccount  
metadata:  
  name: admin-user  
  namespace: kubernetes-dashboard  
  
  변경 필요!  
  
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
  name: admin-user  
roleRef:  
  apiGroup: rbac.authorization.k8s.io  
  kind: ClusterRole  
  name: cluster-admin  
subjects:  
- kind: ServiceAccount  
  name: admin-user  
  namespace: kubernetes-dashboard
```

큐브 대시보드 설치와 사용

Kubernetes Dashboard 접속

The screenshot shows the Kubernetes Dashboard interface. The top navigation bar includes a logo, the text "kubernetes", a search bar, and a "생성" (Create) button. The left sidebar has sections for "클러스터" (Cluster), "네임스페이스" (Namespace), "노드" (Nodes), "퍼시스턴트 볼륨" (Persistent Volumes), "롤" (Roles), and "스토리지 클래스" (Storage Classes). A dropdown for "네임스페이스" is set to "default". The main content area is divided into two tabs: "워크로드" (Workload) and "디플로이먼트" (Deployment). The "워크로드" tab displays four pie charts under "워크로드 상태" (Workload Status):

워크로드 타입	상태	비율 (%)
디플로이먼트	100.00%	100.00%
파드	43.33% 46.67% 10.00%	46.67% 43.33% 10.00%
레플리카 셋	100.00%	100.00%
레플리케이션 컨트롤러	66.67% 16.67% 16.67%	66.67% 16.67% 16.67%

The "디플로이먼트" tab shows a list of deployments:

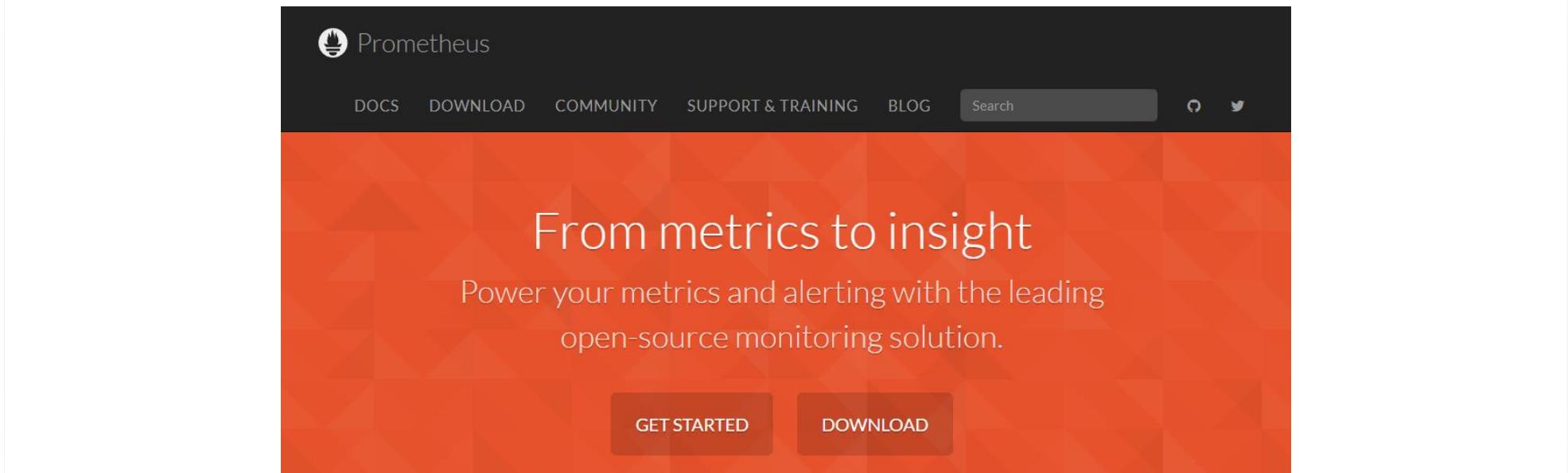
이름	레이블	파드	기간	이미지
load-generator	run: load-gen...	1 / 1	한 시간	busybox
php-apache	run: php-ap...	1 / 1	한 시간	k8s.gcr.io/h...

프로메테우스 그라파나를 활용한 리소스 모니터링

프로메테우스 그라파나를 활용한 리소스 모니터링

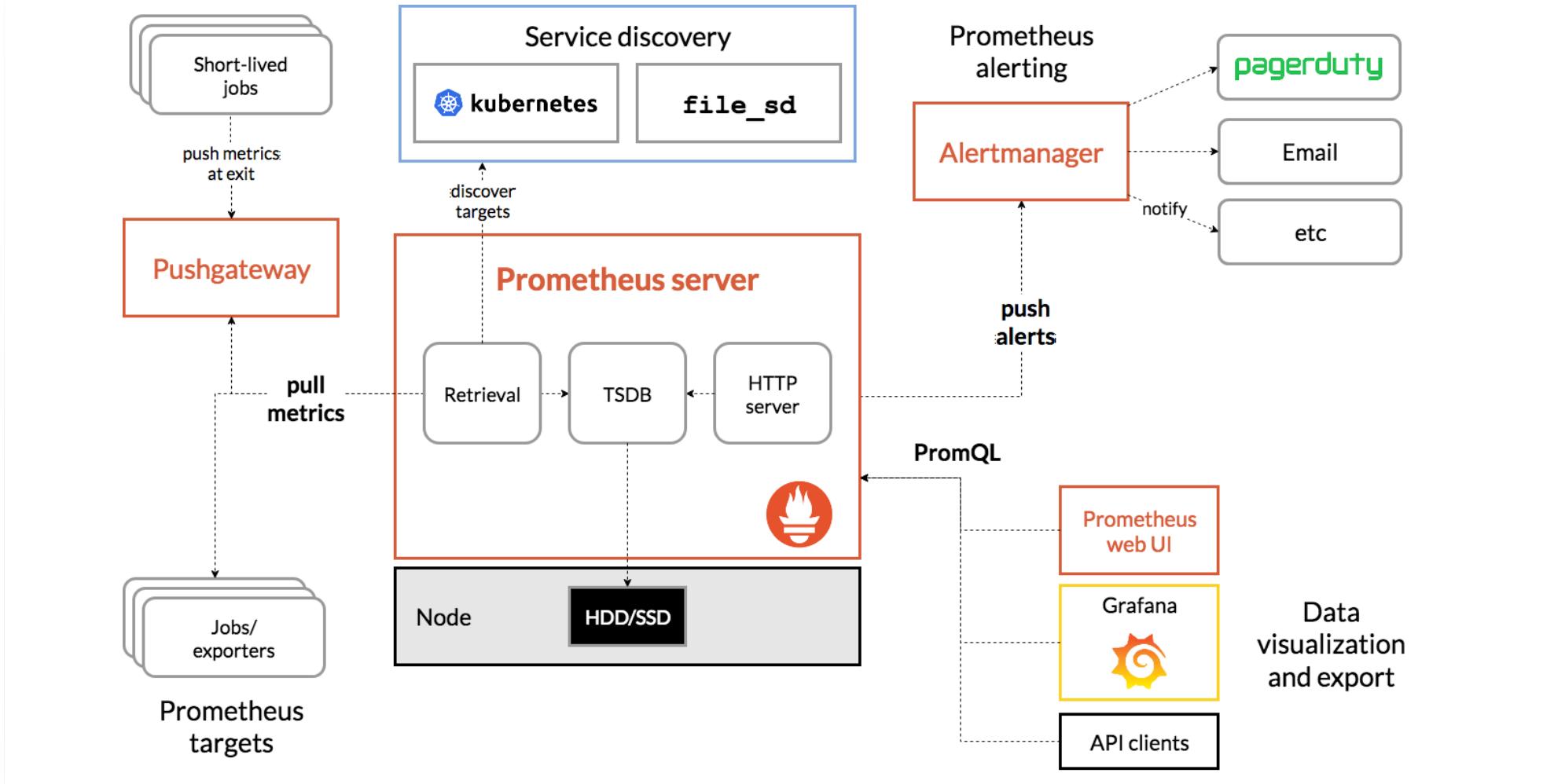
▶ 프로메테우스란?

- 프로메테우스는 원래 SoundCloud에서 구축된 오픈 소스 시스템 모니터링 및 경고 도구 키트
- 2012년에 설립된 이래, 많은 기업과 조직이 Prometheus를 채택했으며, 이 프로젝트는 매우 활발한 개발자와 사용자 커뮤니티를 보유
- 지금은 독립형 오픈 소스 프로젝트이며 모든 회사와 독립적으로 유지관리
- 이를 강조하고 프로젝트의 거버넌스 구조를 명확히 하기 위해 프로메테우스는 2016년 쿠베르네츠에이어 두 번째 호스팅 프로젝트로 클라우드 네이티브 컴퓨팅 재단에 가입
- 프로메테우스는 메트릭을 타임시리즈 데이터(예: 메트릭 정보)로 기록
- 레이블이라고 하는 선택적 키, 벨류 쌍과 함께 기록된 타임스탬프와 함께 저장



프로메테우스 그라파나를 활용한 리소스 모니터링

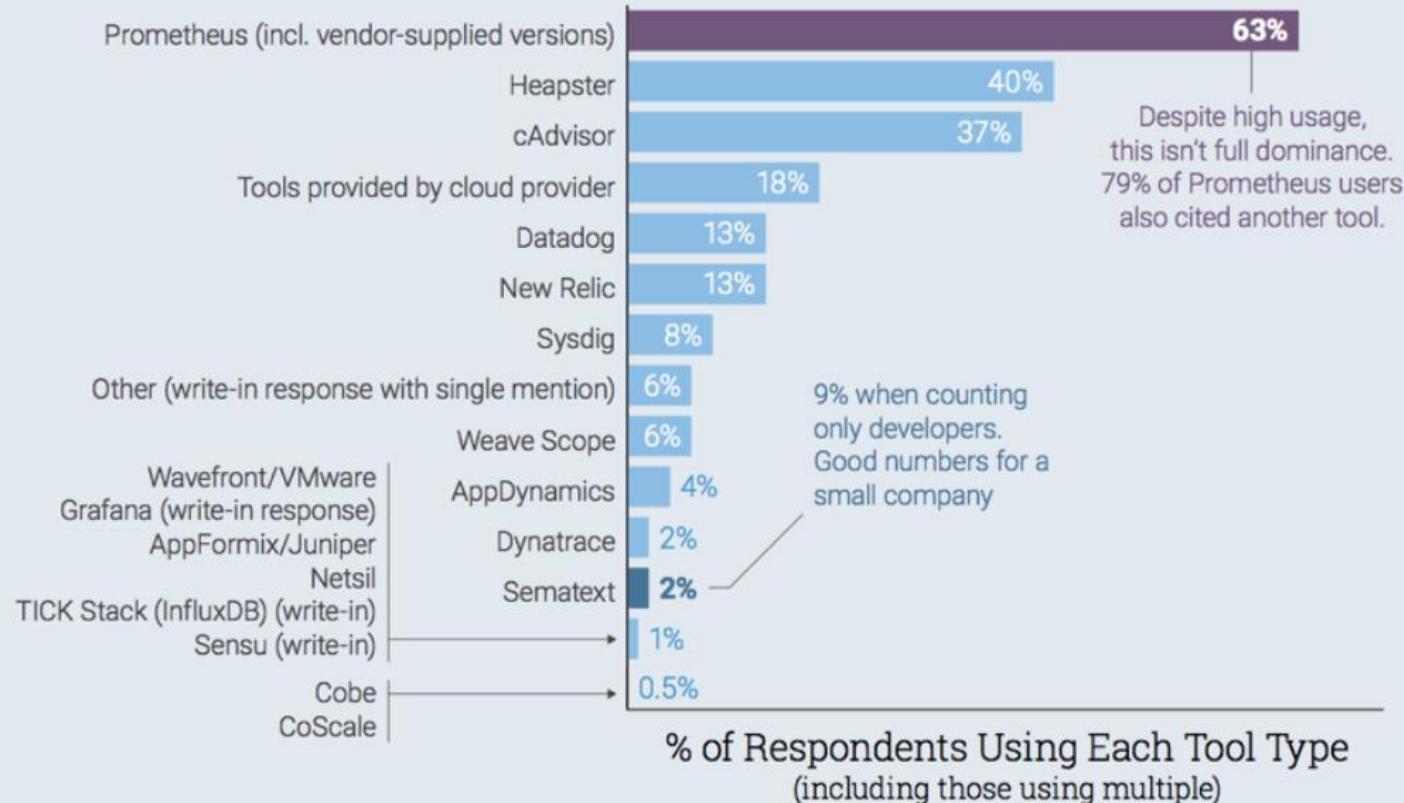
▶ 프로메테우스 아키텍처 예제



프로메테우스 그라파나를 활용한 리소스 모니터링

▣ 쿠버네티스 클러스터 모니터링 툴과 서비스 사용 현황

Tools/Services Used to Monitor Kubernetes Clusters



Source: The New Stack 2017 Kubernetes User Experience Survey.
Q. What tools, products and services are being used to monitor Kubernetes clusters? n=208.

THE NEW STACK

프로메테우스 그라파나를 활용한 리소스 모니터링

▶ 헬름 레파지토리 추가

- <https://blog.naver.com/isc0304/222515904650>
- 프로메테우스와 그라파나를 위한 헬름 저장소를 추가

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts  
helm repo add grafana https://grafana.github.io/helm-charts  
helm repo update
```

▶ 그라파나와 프로메테우스 배포

- 헬름 배포를 위해 그라파나와 프로메테우스의 values.yaml을 구성할 디렉토리를 하나 구성

```
mkdir grafana_prometheus  
cd grafana_prometheus
```

프로메테우스 그라파나를 활용한 리소스 모니터링

▶ 프로메테우스 values 파일 구성

- 다음 명령을 실행해 values-prometheus.yaml를 생성
- pv를 구성하여 스토리지를 구성하고 15일간 데이터를 보존하도록 구성

```
cat <<EOF > values-prometheus.yaml
server:
  enabled: true

persistentVolume:
  enabled: true
  accessModes:
    - ReadWriteOnce
  mountPath: /data
  size: 100Gi
  replicaCount: 1

## Prometheus data retention period (default if not specified is 15 days)
##
  retention: "15d"
EOF
```

프로메테우스 그라파나를 활용한 리소스 모니터링

▶ 그라파나 values 파일 구성

- 다음 명령을 실행해 values-grafana.yaml를 생성
- pvc를 구성하여 스토리지를 구성하여 설정정보를 유지할 수 있도록 구성
- 아이디 패스워드는 admin//test1234!234로 구성
- 서비스가 생성될 때 접속하기 쉽도록 로드 밸런서로 구성
- 주의: 실무에서는 모니터링 서비스가 외부로 노출되지 않도록 조심해야 함

```
cat << EOF > values-grafana.yaml
replicas: 1

service:
  type: LoadBalancer

persistence:
  type: pvc
  enabled: true
  # storageClassName: default
  accessModes:
    - ReadWriteOnce
  size: 10Gi
  # annotations: {}
  finalizers:
    - kubernetes.io/pvc-protection

# Administrator credentials when not using an existing secret (see below)
adminUser: admin
adminPassword: test1234!234
EOF
```

프로메테우스 그라파나를 활용한 리소스 모니터링

▶ 헬름 차트 배포

- 앞서 생성한 values 파일들을 사용해 배포를 시작

```
kubectl create ns prometheus
```

```
helm install prometheus prometheus-community/prometheus -f values-prometheus.yaml -n prometheus
```

```
helm install grafana grafana/grafana -f values-grafana.yaml -n prometheus
```

- 배포 확인

```
$ kubectl get pod,svc -n prometheus
```

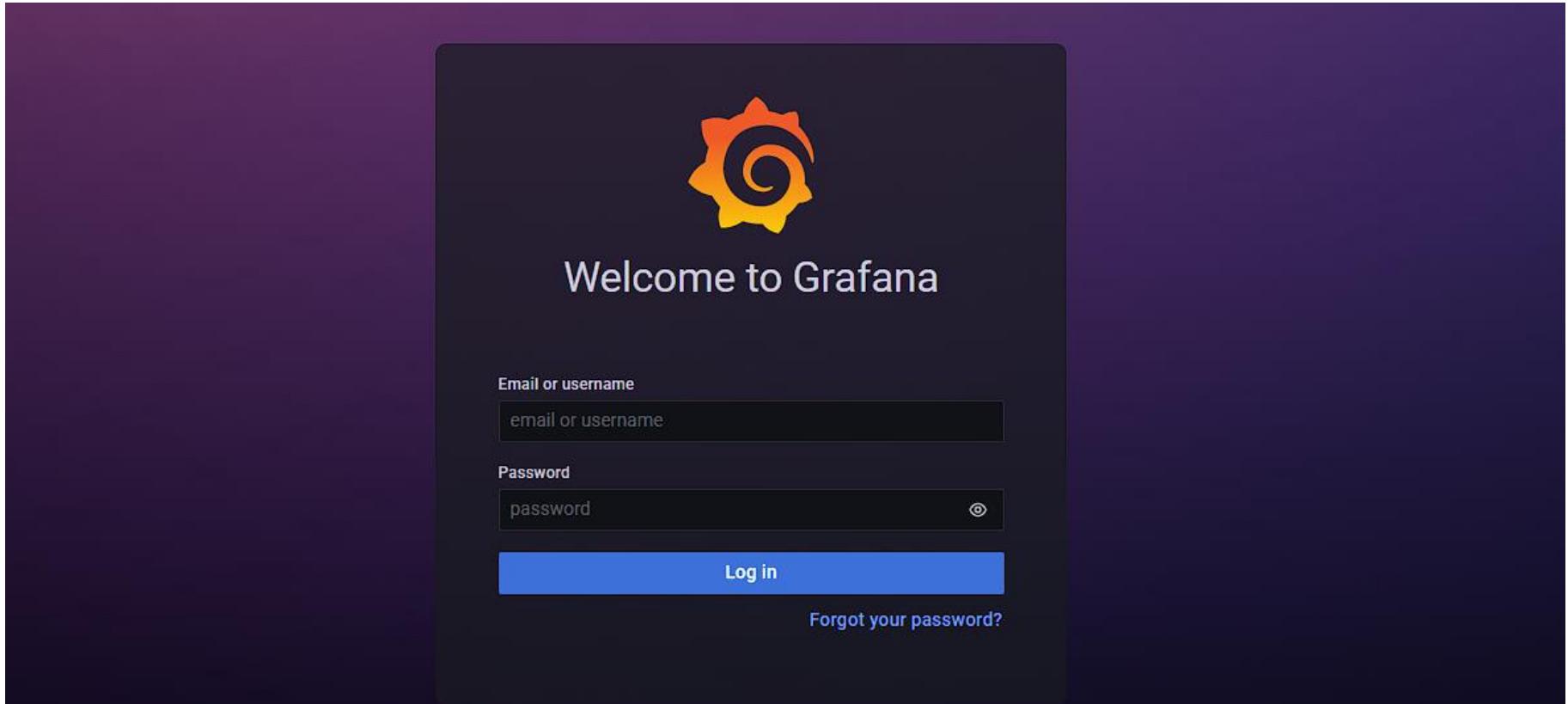
NAME	READY	STATUS	RESTARTS	AGE
pod/grafana-b64fbdc-b-qdxnm	1/1	Running	0	49s
pod/prometheus-alertmanager-6755b9794f-thq5l	1/2	Running	0	57s
pod/prometheus-kube-state-metrics-696cf79768-xkjlk	1/1	Running	0	58s
pod/prometheus-node-exporter-bxvbw	1/1	Running	0	58s
pod/prometheus-node-exporter-dlmnl	1/1	Running	0	58s
pod/prometheus-node-exporter-qgjpc	1/1	Running	0	58s
pod/prometheus-pushgateway-898d5bdb9-7bh7h	1/1	Running	0	58s
pod/prometheus-server-c68ccc7ff-54jzw	2/2	Running	0	58s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT
service/grafana	LoadBalancer	10.8.0.77	34.70.251.10	80:
service/prometheus-alertmanager	ClusterIP	10.8.14.141	<none>	80/
service/prometheus-kube-state-metrics	ClusterIP	10.8.6.161	<none>	808
service/prometheus-node-exporter	ClusterIP	None	<none>	910
service/prometheus-pushgateway	ClusterIP	10.8.13.17	<none>	909
...

프로메테우스 그라파나를 활용한 리소스 모니터링

▶ 그라파나 접속하기

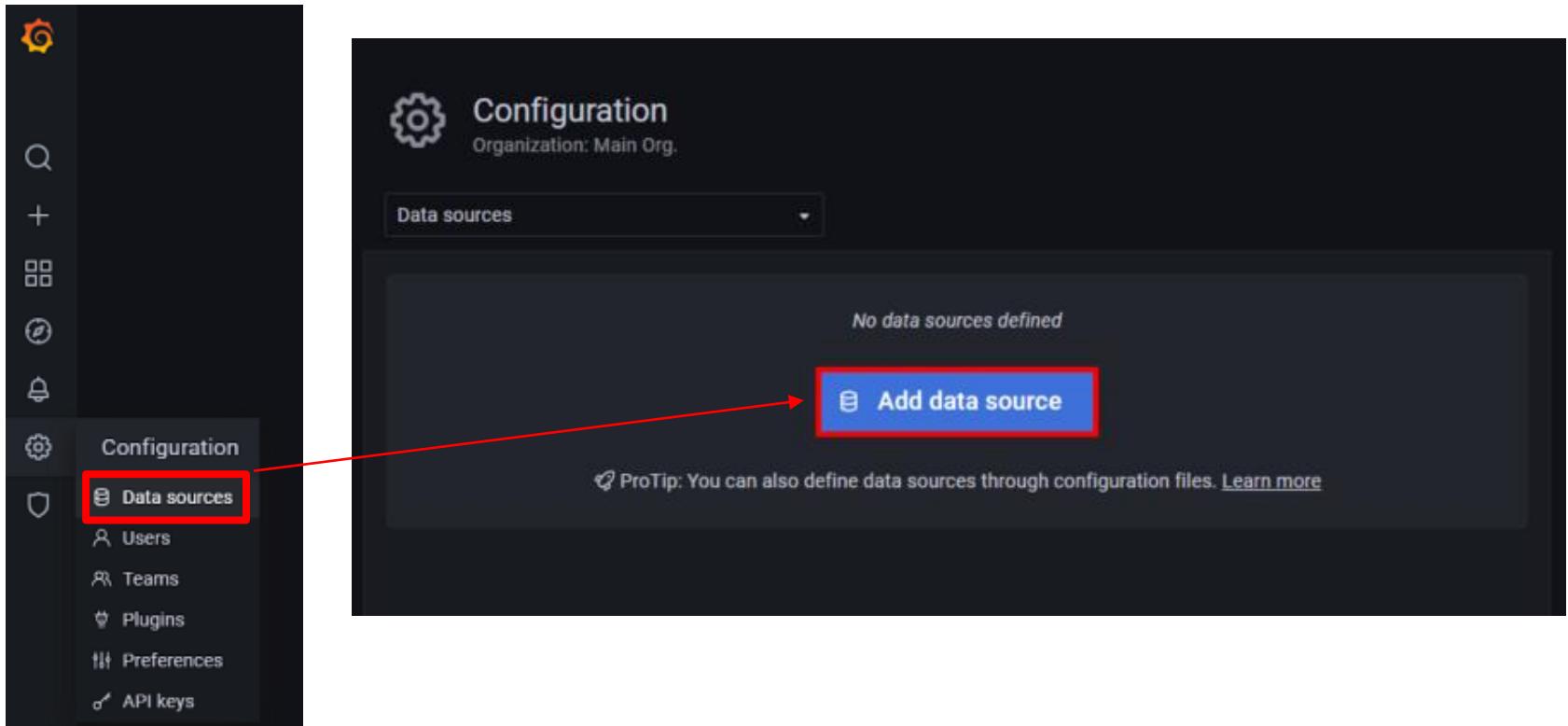
- 외부 IP로 노출된 grafana의 IP로 접속을 수행
- 아이디와 패스워드는 admin//test1234!234



프로메테우스 그라파나를 활용한 리소스 모니터링

▶ 프로메테우스 데이터 가져오기

- 왼쪽 메뉴에서 Configuration - Data Sources 메뉴에서 데이터 소스 추가



프로메테우스 그라파나를 활용한 리소스 모니터링

▶ 프로메테우스 데이터 가져오기

- 프로메테우스를 선택하고 URL에 앞서 자동 생성된 service의 이름을 입력
- <http://prometheus-server> 을 입력하고 save & test 클릭

The image shows two screenshots of the Grafana interface. On the left, the 'Add data source' screen displays options for 'Time series databases' including 'Prometheus', 'Graphite', and 'OpenTSDB'. The 'Prometheus' option is highlighted with a red box. On the right, the 'Data Sources / Prometheus' configuration screen is shown. It has a 'Settings' dropdown, a central info icon, and a message about using Grafana Cloud. Below this, it shows a table with one row for 'Prometheus'. The 'Name' column is set to 'Prometheus', the 'Default' toggle is turned on, and the 'URL' field contains the value 'http://prometheus-server'. The 'Access' field is set to 'Server (default)'. There are also sections for 'Whitelisted Cookies' and 'Timeout'.

프로메테우스 그라파나를 활용한 리소스 모니터링

▶ 대시보드 구성하기

- 대시보드를 구성하기 위해 +버튼 (Create)의 Import를 선택
- Import는 외부에 사용자들이 미리구성해 놓은 다양한 대시보드를 바로 가져와서 사용
- 다음 링크에서 대시보드 검색 가능: <https://grafana.com/grafana/dashboards/315>

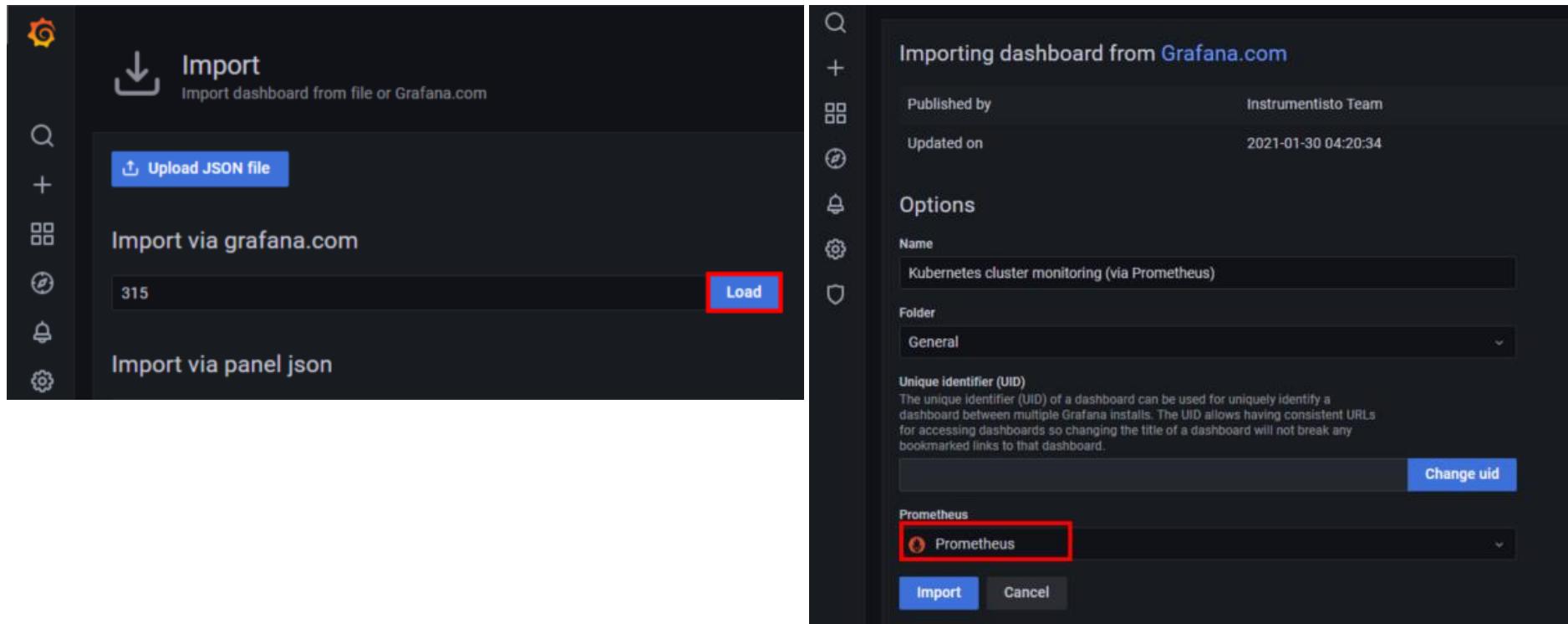
The screenshot shows the Grafana Labs website with the following details:

- Header:** Grafana Labs logo, navigation links for Grafana, Products, Open Source, Learn, Downloads, Contact us (highlighted in blue), and Login.
- Breadcrumbs:** All dashboards > Kubernetes cluster monitoring (via Prometheus)
- Dashboard Preview:** A circular icon with a flame inside, labeled "Kubernetes cluster monitoring (via Prometheus)" by Instrumentisto Team. Below it, a description states: "Monitors Kubernetes cluster using Prometheus. Shows overall cluster CPU / Memory / Filesystem usage as well as individual pod, containers, systemd services statistics. Uses cAdvisor metrics only." It also mentions "Last updated: 8 months ago".
- Metrics:** Downloads: 182409, Reviews: 7, and a 5-star rating.
- Actions:** A blue button labeled "Add your review!"
- Navigation:** Overview (selected), Revisions, Reviews.
- Visuals:** Five small screenshots of the dashboard panels.
- Download Options:** A red box highlights a "Get this dashboard:" section with a "315" button and a "Copy ID to Clipboard" button.
- Dependencies:** GRAFANA 3.1.1
- Footer:** Grafana Cloud free tier information, license (MIT), dependencies (node-exporter, kubernetes, prometheus, grafana), and a note about the initial idea being taken from another dashboard.

프로메테우스 그라파나를 활용한 리소스 모니터링

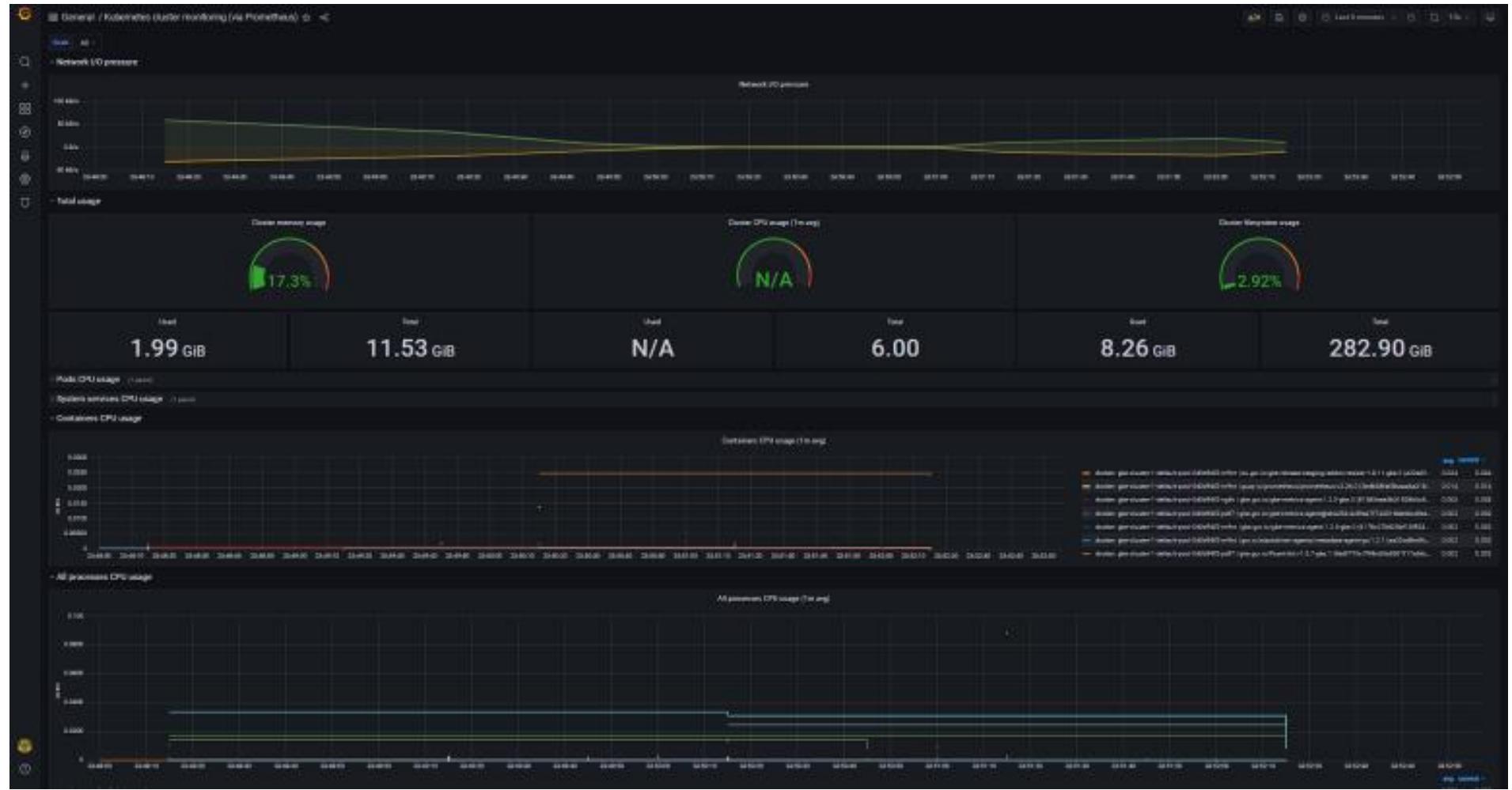
▶ 대시보드 구성하기

- 획득한 ID 정보를 입력하고 Load를 클릭
- Prometheus 데이터 소스를 선택하고 Import 클릭



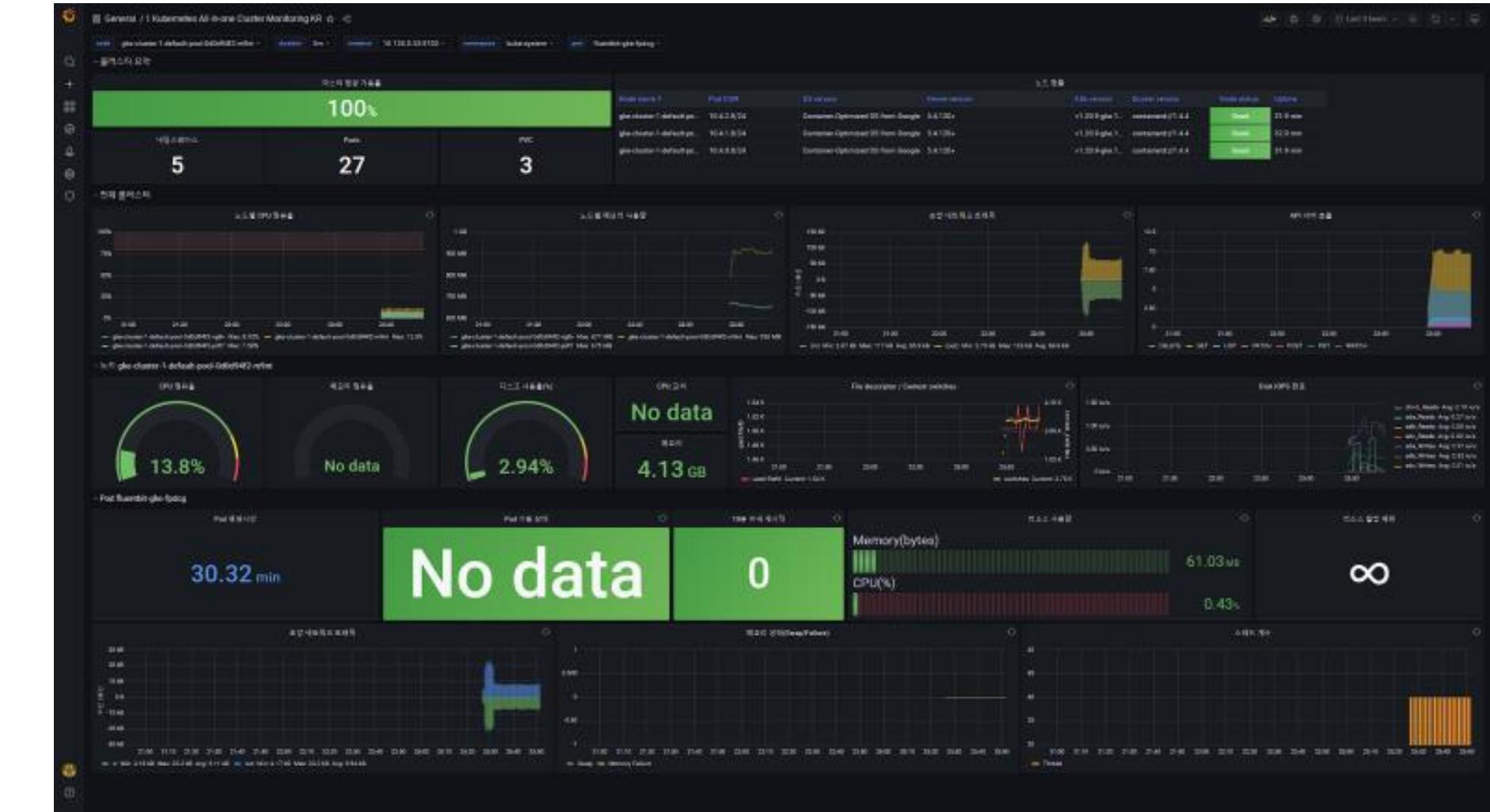
프로메테우스 그라파나를 활용한 리소스 모니터링

▶ 315번 대시보드



프로메테우스 그라파나를 활용한 리소스 모니터링

▶ 13770번 대시보드

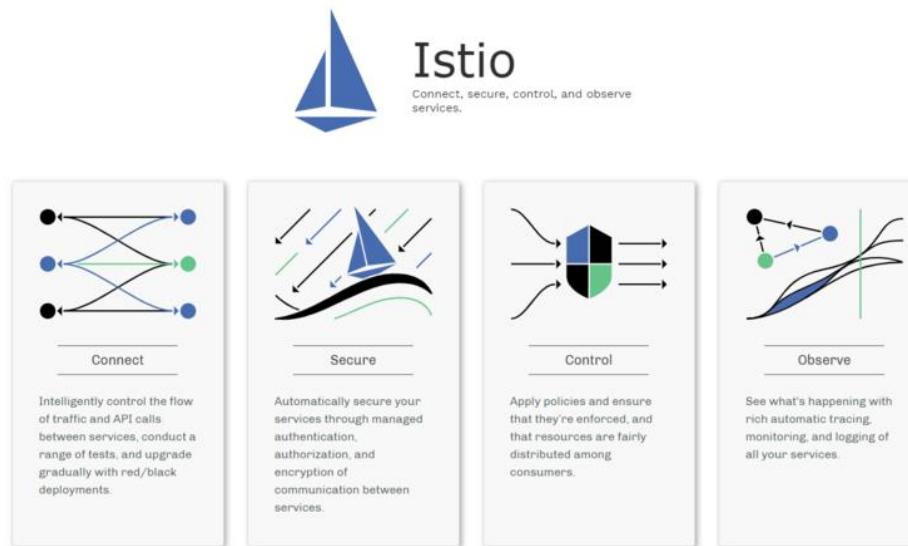


Istio를 활용한 네트워크 메시 모니터링

Istio를 활용한 네트워크 메시 모니터링

Istio는 무엇인가?

- <https://blog.naver.com/isc0304/221892105612>
- 다수의 컨테이너가 동작하는 경우에는 각 컨테이너의 트래픽을 관찰하고 정상 동작하는지 모니터링하기가 어렵기 때문에 DevOps 팀에 부담
- 개발자는 이식성을 위해 마이크로서비스를 사용하여 아키텍처를 설계하고 운영자는 이 컨테이너들을 다양한 클러스터에 배포하고 관리
- 서비스 메시의 크기와 복잡성이 커짐에 따라 이해하고 관리하기가 더 어려워짐
(예: 로드 밸런싱, 장애 복구, 메트릭 및 모니터링)
- Istio는 쿠버네티스 환경의 네트워크 메시 이슈를 보다 간편하게 해결하기 위해 지원하는 환경



Istio를 활용한 네트워크 메시 모니터링

Istioctl 설치

- istio 최신 버전 다운로드 및 설치

```
curl -L https://istio.io/downloadIstio | sh -
cd istio-1.10.2
export PATH=$PWD/bin:$PATH # 실행 경로를 환경 변수에 추가
istioctl # kubectl 설정을 사용
```

- 이스티오 배포 형태

	default	demo	minimal	external	empty	preview
Core components						
istio-egressgateway		✓				
istio-ingressgateway	✓	✓				✓
istiod	✓	✓	✓			✓

Istio를 활용한 네트워크 메시 모니터링

▶ 쿠버네티스에 이스티오 구성하기

- istioctl을 사용해 데모 버전으로 설치

```
$ istioctl install --set profile=demo --skip-confirmation
```

- 네임스페이스 레이블을 이스티오 인젝션이 수행되도록 설정

```
kubectl label namespace default istio-injection=enabled
```

Istio를 활용한 네트워크 메시 모니터링

▶ 애플리케이션 배포하기

- 북 인포에 대한 프로젝트 배포 (istio와 무관한 기능)

```
kubectl delete all --all # 잘못 설치한 경우 삭제
```

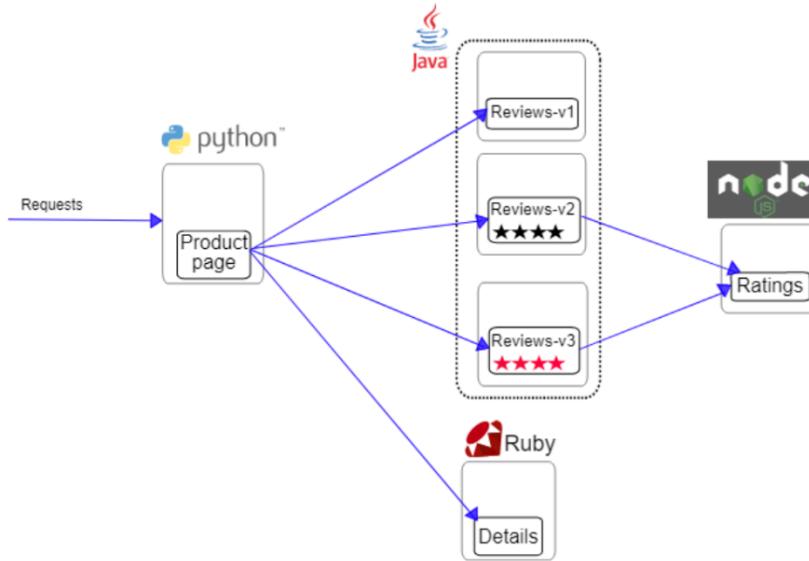
```
kubectl delete limitrange default-limit-range # 잘못 설치한 경우 삭제
```

```
kubectl delete -f samples/bookinfo/platform/kube/bookinfo.yaml # 잘못 설치한 경우 삭제
```

```
kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml
```

- 북인포를 외부로 서비스할 수 있도록 게이트웨이 생성(istio의 기능)

```
kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml
```



Istio를 활용한 네트워크 메시 모니터링

▶ 애플리케이션 배포하기

- 게이트웨이를 생성하면서 만들어진 서비스를 확인

```
$ kubectl get svc -n istio-system -l istio=ingressgateway --all-namespaces
NAMESPACE      NAME           TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)
istio-system    istio-ingressgateway   LoadBalancer  10.8.11.193   35.226.15.253   15021:3...
```

- 배포한 북인포 프로젝트 페이지에 접속
- <http://35.226.15.253/productpage> 반드시 **productpage**라는 곳으로 접속

The Comedy of Errors

Summary: Wikipedia Summary: The Comedy of Errors is one of William Shakespeare's early plays. It is his shortest and one of his most farcical comedies, with a major part of the humour coming from slapstick and mistaken identity, in addition to puns and word play.

Type: paperback
Pages: 200
Publisher: PublisherA
Language: English
ISBN-10: 1234567890
ISBN-13: 123-1234567890

Book Details

An extremely entertaining play by Shakespeare. The slapstick humour is refreshing!
— Reviewer1

Absolutely fun and entertaining. The play lacks thematic depth when compared to other plays by Shakespeare.
— Reviewer2

Book Reviews

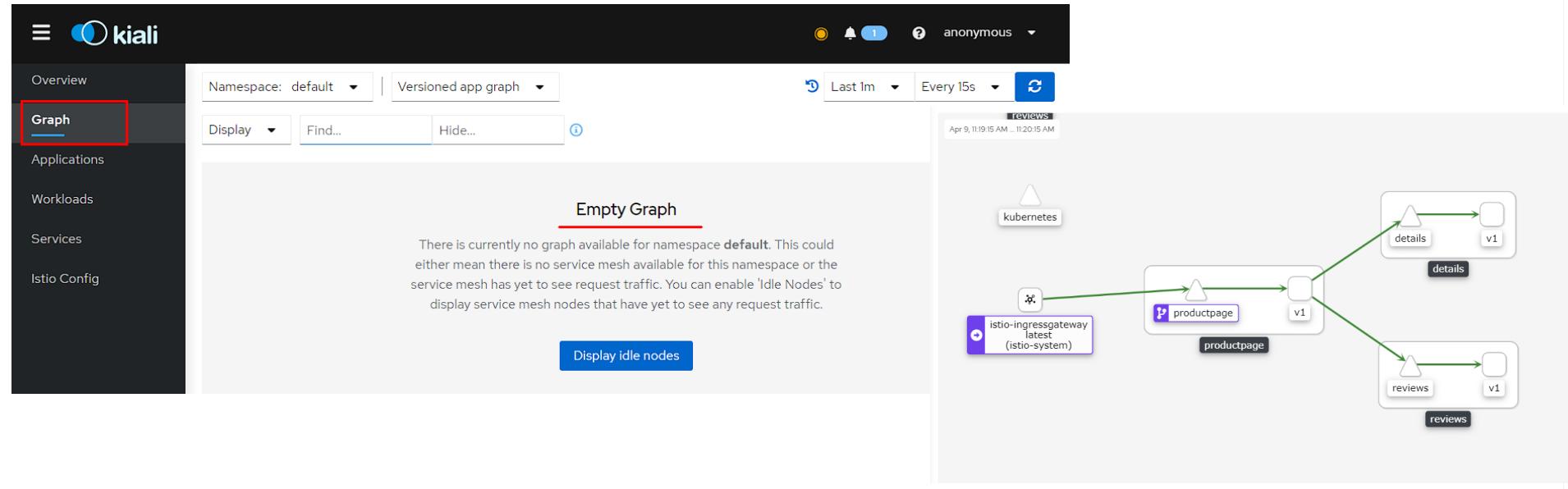
Istio를 활용한 네트워크 메시 모니터링

Kiali와 프로메테우스 구성

- kiali 대시보드와 데이터베이스역할을 하는 프로메테우스를 띄우고서 대시보드로 접근

```
kubectl apply -f samples/addons/kiali.yaml  
kubectl apply -f samples/addons/prometheus.yaml  
istioctl dashboard kiali # localhost:20001 서비스를 오픈
```

- 로컬 호스트의 20001번 포트로 접근하고(웹 미리보기 기능 사용) 그래프 확인



EFK를 활용한 k8s 로그 모니터링

EFK를 활용한 k8s 로그 모니터링

▶ EFK 설치 환경

- 쿠버네티스에서 EFK를 사용하면 도커의 각 컨테이너 로그를 수집하고 시각화, 분석 가능
- 쿠버네티스 메모리 8GB 이상 필요
- 엘라스틱스택은 주로 비츠나 로그스태시를 사용하는 것이 일반적
- 쿠버네티스에서는 fluentd를 사용해서 수집하는 것이 유행
 - E: Elasticsearch 데이터베이스 & 검색엔진
 - L: Logstash 데이터 수집기, 파이프라인(여기서는 F: Fluentd를 대신 사용한다.)
 - K: Kibana 그라파나의 역할, 대시보드 (SIEM, 머신러닝, Alert)

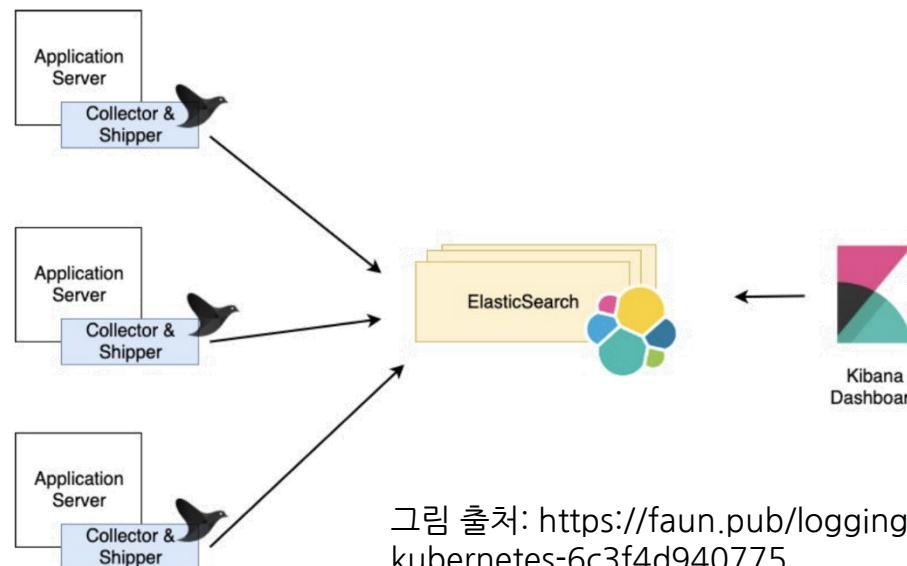


그림 출처: <https://faun.pub/logging-architectures-efk-stack-and-kubernetes-6c3f4d940775>

EFK를 활용한 k8s 로그 모니터링

▶ EFK 설치 환경

- <https://blog.naver.com/isc0304/221860255105>
- <https://blog.naver.com/isc0304/221879552183>
- 예제 파일 다운로드

 elasticsearch.yaml	2021-09-23 오후 8:33	YAML 파일
 fluentd.yaml	2021-09-23 오후 8:47	YAML 파일
 kibana.yaml	2021-09-23 오후 8:33	YAML 파일
 ns.yaml	2021-09-23 오후 10:20	YAML 파일

- 이 파일은 kubectl이 사용 가능한 곳으로 옮겨서 kubectl을 사용해 실행하여 배포

```
kubectl apply -f efk/
```

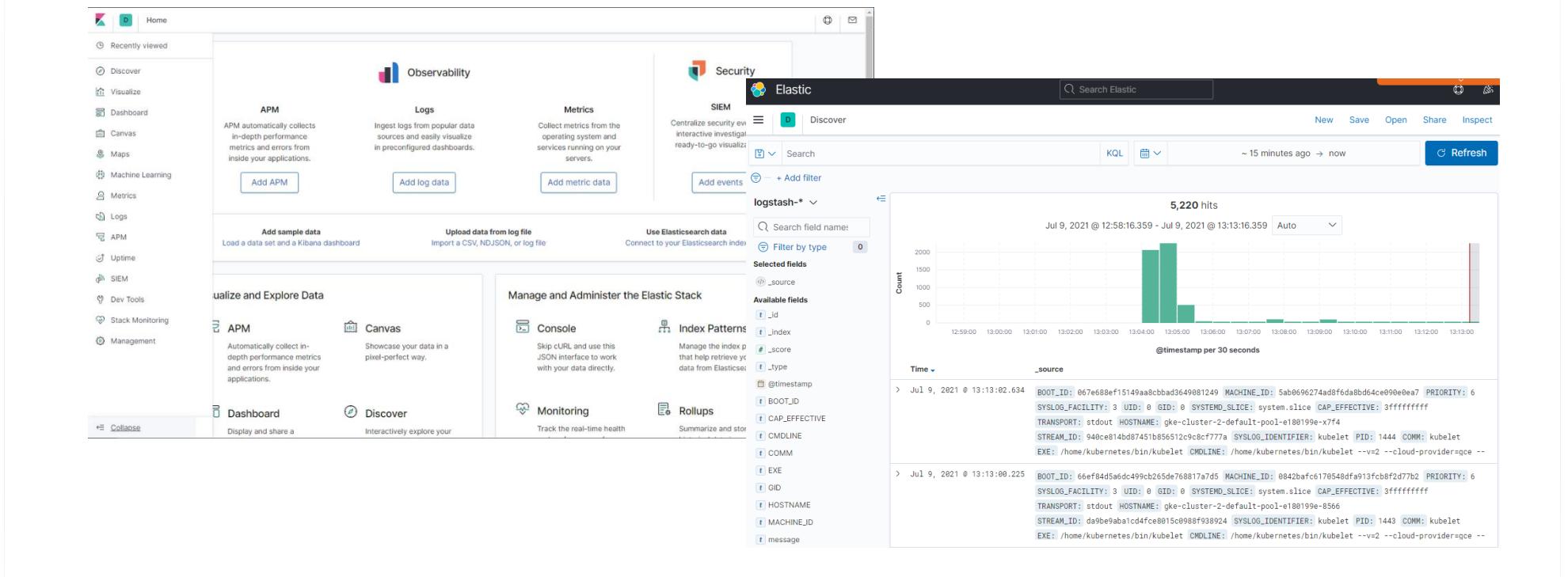
EFK를 활용한 k8s 로그 모니터링

▶ EFK 설치 환경

- 다음 명령어로 포트포워딩 기능을 사용하여 접속 - `http://<IP>:5601`

```
$ kubectl port-forward kibana-kibana-7db5d4964f-8qdt6 5601:5601
```

- Kibana - Discover에 가면 인덱스를 설정하라고 나옴
- 인덱스 설정에서 logstash-*을 입력하고 시간 설정은 @timestamp로 설정



오토 스케일링 HPA 워크스루

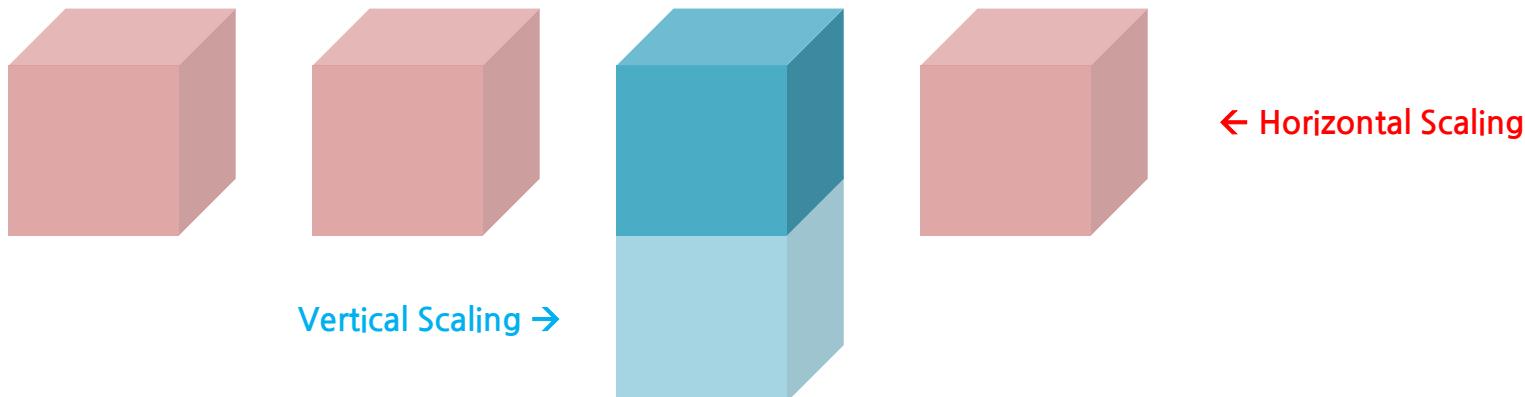
오토 스케일링 HPA 워크스루

▶ 파드 스케일링의 두 가지 방법

- HPA: 파드 자체를 복제하여 처리할 수 있는 파드의 개수를 늘리는 방법
- VPA: 리소스를 증가시켜 파드의 사용 가능한 리소스를 늘리는 방법
- CA: 번외로 클러스터 자체를 늘리는 방법(노드 추가)

▶ HPA(Horizontal Pod Autoscaler)

- 쿠버네티스에는 기본 오토스케일링 기능 내장
- CPU 사용률을 모니터링하여 실행된 파드의 개수를 늘리거나 줄임



오토 스케일링 HPA 워크스루

▶ VPA와 CA는 어떻게!?

- 공식 쿠버네티스에서 제공하지는 않으나 클라우드 서비스에서 제공

- 클러스터 자동 확장 처리(CA)

➤ <https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-autoscaler>

```
gcloud container clusters create example-cluster \
    --zone us-central1-a \
    --node-locations us-central1-a,us-central1-b,us-central1-f \
    --num-nodes 2 --enable-autoscaling --min-nodes 1 --max-nodes 4
```

- 수직형 pod 자동 확장(VPA)

➤ https://cloud.google.com/kubernetes-engine/docs/how-to/vertical-pod-autoscaling?authuser=1&_ga=2.95397596.-1524441062.1575378592&_gac=1.156945225.1583154351.CjwKCAjA-vLyBRBWEiwAzOkGVAWhHWKuFV0kIRfpVDRA5yaGh1wIUZOHQZ6Z9IZYS2lZtzlyC5IBbh0CT3AQAvDBwE

```
gcloud container clusters create [CLUSTER_NAME] --enable-vertical-pod-autoscaling
gcloud container clusters update [CLUSTER-NAME] --enable-vertical-pod-autoscaling
```

오토 스케일링 HPA 워크스루

▶ HPA(Horizontal Pod Autoscaler) 설정 방법

- 명령어를 사용하여 오토 스케일 저장하기

```
$ kubectl autoscale deployment my-app --max 6 --min 4 --cpu-percent 50
```

autoscaling.yaml

- HPA yaml을 작성하여 타겟 파드 지정

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: myapp-hpa
  namespace: default
spec:
  maxReplicas: 10
  minReplicas: 1
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: myapp
  targetCPUUtilizationPercentage: 30
```

오토 스케일링 HPA 워크스루

▶ php-apache 서버 구동 및 노출

- <https://kubernetes.io/ko/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/>
- 이미지는 이미 구축돼 있으므로 바로 kubectl 명령어를 실행

```
$ kubectl apply -f  
https://k8s.io/examples/application/php-apache.yaml  
deployment.apps/php-apache created  
service/php-apache created
```

```
$ kubectl autoscale deployment php-apache --cpu-  
percent=50 --min=1 --max=10  
horizontalpodautoscaler.autoscaling/php-apache  
autoscaled
```

```
$ kubectl get hpa  
NAME      REFERENCE          TARGETS  
MINPODS   MAXPODS   REPLICAS   AGE  
php-apache Deployment/php-apache <unknown>/50%   1  
10        0           5s
```

```
FROM php:5-apache  
ADD index.php /var/www/html/index.php  
RUN chmod a+r index.php
```

dockerfile

```
<?php  
$x = 0.0001;  
for ($i = 0; $i <= 1000000; $i++) {  
    $x += sqrt($x);  
}  
echo "OK!";  
?>
```

index.php

오토 스케일링 HPA 워크스루

 오토스케일링을 적용한 서비스에 무한 루프 쿼리를 통해 공격 수행

```
$ kubectl run --generator=run-pod/v1 -it --rm load-generator --image=busybox /bin/sh  
Hit enter for command prompt
```

 1분 후 확인(중간에 unknown이 발생할 수도 있음, 수집까지 시간이 걸림)

```
$ kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	458%/50%	1	10	1	75s

```
$ kubectl get hpa
NAME          REFERENCE           TARGETS      MINPODS     MAXPODS   REPLICAS   AGE
php-apache    Deployment/php-apache  123%/50%    1           10         10         4m29s`
```

오토 스케일링 HPA 워크스루

▶ 연습문제

- 이미지 nginx를 생성하고 HPA가 적용하라.

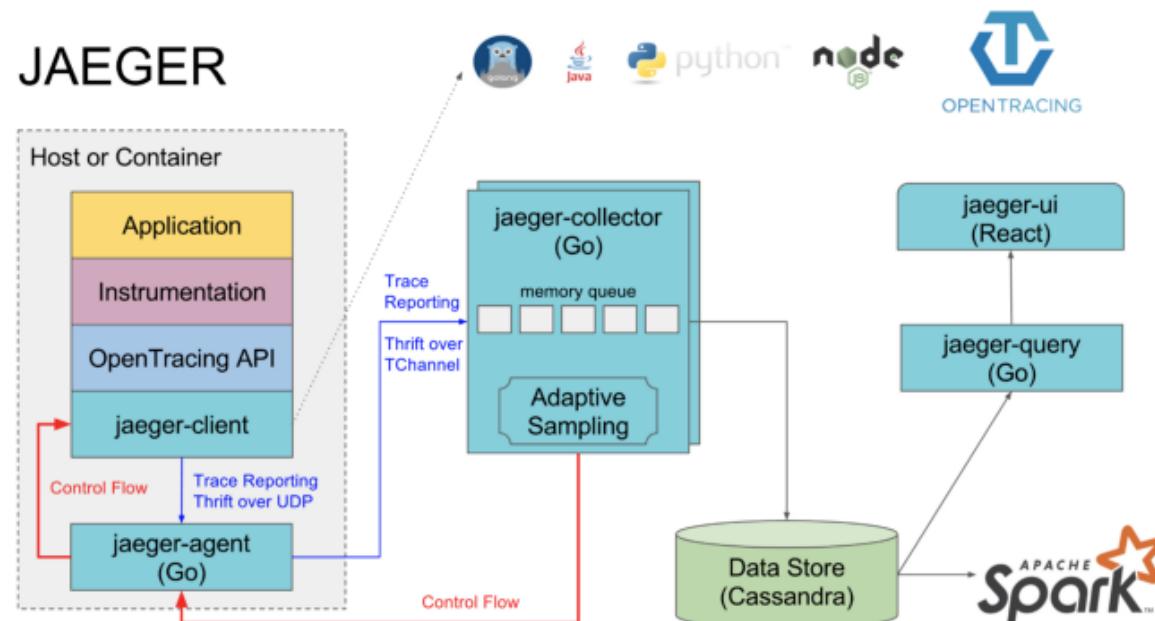
- 최대 스케일링 개 수: 10
- 최소 스케일링 개 수: 2
- 스케일링을 수행할 CPU 활동량: 30%

Jaeger 트레이싱 튜토리얼

Jaeger 트레이싱 튜토리얼

▶ Jaeger 트레이싱 시스템

- 마이크로 서비스를 위한 오픈 소스 추적 시스템이며 OpenTracing 표준을 지원
- 처음에 Uber Technologies에 의해 오픈 소스로 공개
- 예거는 추적, 근본 원인 분석, 서비스 종속성 분석 등을 배포
- 추적이 활성화된 소규모 서비스를 빌드해 튜토리얼을 실습
- Go, 자바, 자바 스크립트, 노드.js, 파이썬, C++에 위한 라이브러리를 포함
- 헬름 차트나 istio 등을 활용해 편리하게 설치 가능



▶ 예거 트레이싱 용어집

- 에이전트 - 사용자 데이터그램 프로토콜을 통해 전송된 범위에 대해 듣는 네트워크 데몬
- 클라이언트 - 분산 추적을 위해 OpenTracing API를 구현하는 구성 요소
- 컬렉터 - 범위를 수신하고 처리할 큐에 추가하는 구성 요소
- 콘솔 - 사용자가 분산 추적 데이터를 시각화할 수 있는 UI
- 쿼리 - 저장소에서 추적을 가져오는 서비스
- 범위 - 이름, 시작 시간 및 작업의 기간을 포함하는 예거에서 작업의 논리적 단위
- 추적 - 예거가 실행 요청을 제시하는 방식, 추적은 하나 이상의 범위로 구성

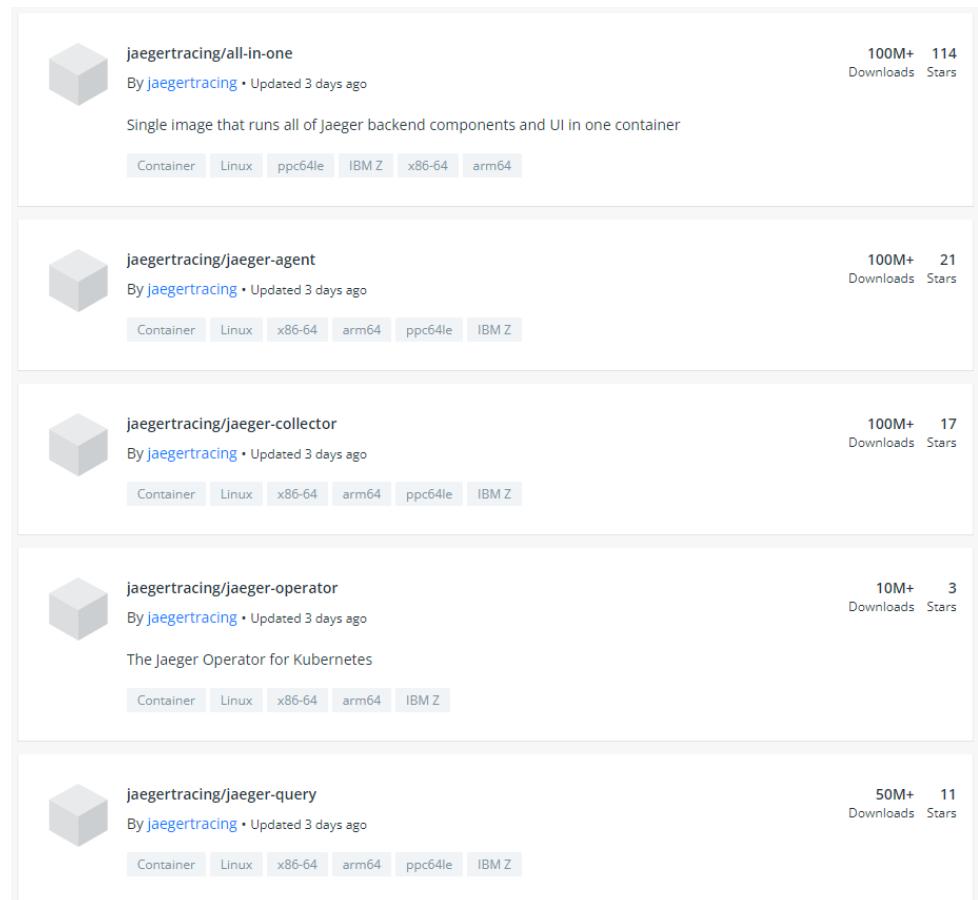
Jaeger 트레이싱 튜토리얼

▶ 예거 설치

- 추적 정보를 수집, 저장 및 표시하기 위한 분산 구성 요소 집합
- 전체 예거 시스템을 실행하는 “올인원” 이미지로 설치

```
docker run -d --name jaeger -p 16686:16686 \
-p 6831:6831/udp \
jaegertracing/all-in-one:1.22
```

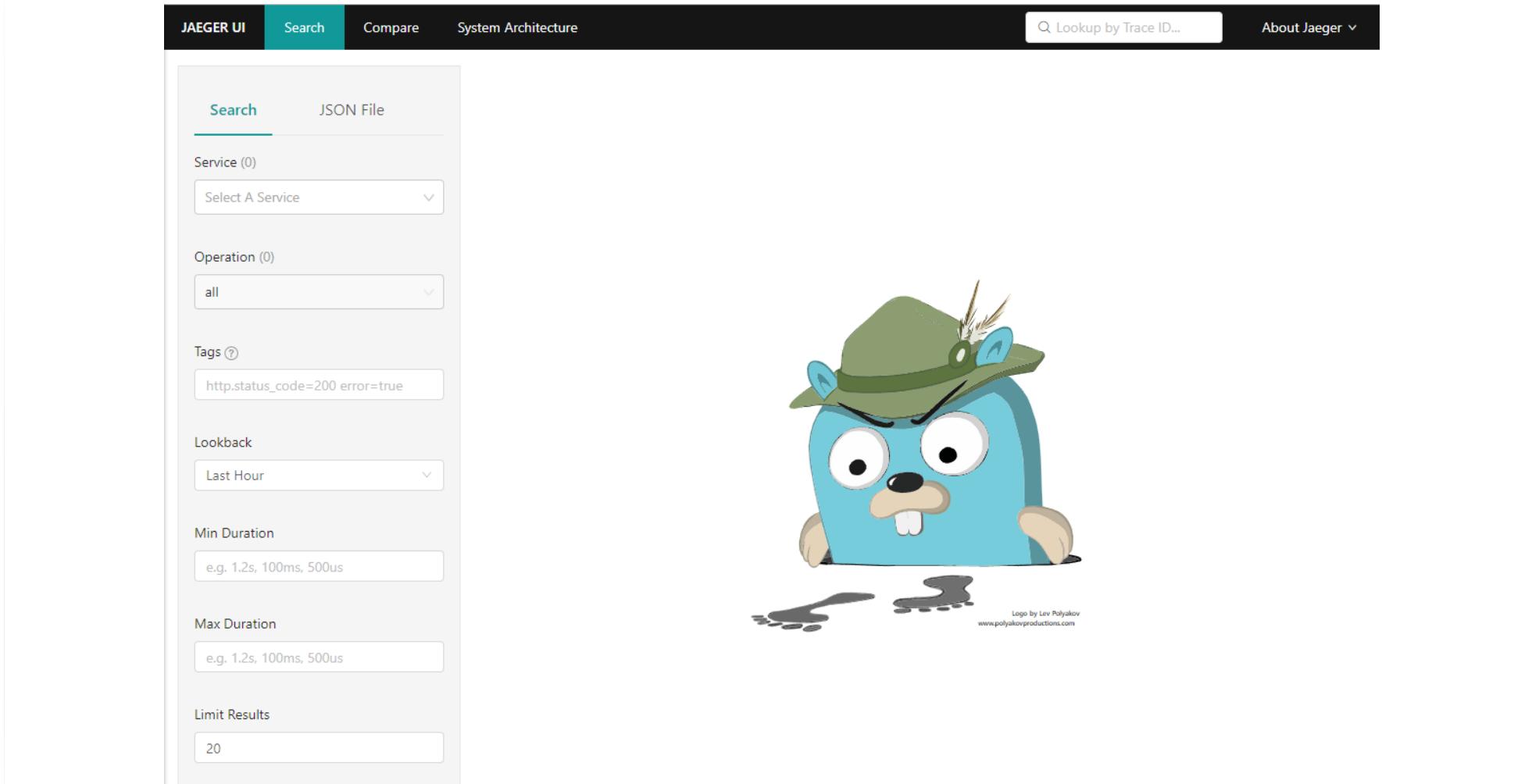
```
pip3 install jaeger-client
pip3 install requests
```



Jaeger 트레이싱 튜토리얼

▶ 예거 시스템에 접속하기

- 16686 포트로 접속하기



The image shows the Jaeger UI search interface. On the left, there is a sidebar with various search filters: Service (0), Operation (0), Tags (http.status_code=200 error=true), Lookback (Last Hour), Min Duration (e.g. 1.2s, 100ms, 500us), Max Duration (e.g. 1.2s, 100ms, 500us), and Limit Results (20). On the right, there is a search bar with the placeholder "Lookup by Trace ID..." and a "About Jaeger" link. A large, cartoonish blue gopher character wearing a green hat is overlaid on the interface, appearing to peek out from behind the search bar.

Jaeger 트레이싱 튜토리얼

파이썬 프로그램에 트레이싱 예제

- Init tracer 함수는 db에 등록할 정보와 로깅 레벨 등을 설정한 추적 객체를 구성해 반환

```
import logging
from jaeger_client import Config
import requests

def init_tracer(service):
    logging.getLogger('').handlers = []
    logging.basicConfig(format='%(message)s', level=logging.DEBUG)

    config = Config(
        config={
            'sampler': {
                'type': 'const',
                'param': 1,
            },
            'logging': True,
        },
        service_name=service,
    )

    # this call also sets opentracing.tracer
    return config.initialize_tracer()
```

Jaeger 트레이싱 튜토리얼

파이썬 프로그램에 트레이싱 예제

- First-service 트레이서를 구성
- 트레이서를 활용해 get-ip-api-jobs 라는 span을 구성
- 웹 요청 결과를 span에 데이터 추가

```
tracer = init_tracer('first-service')

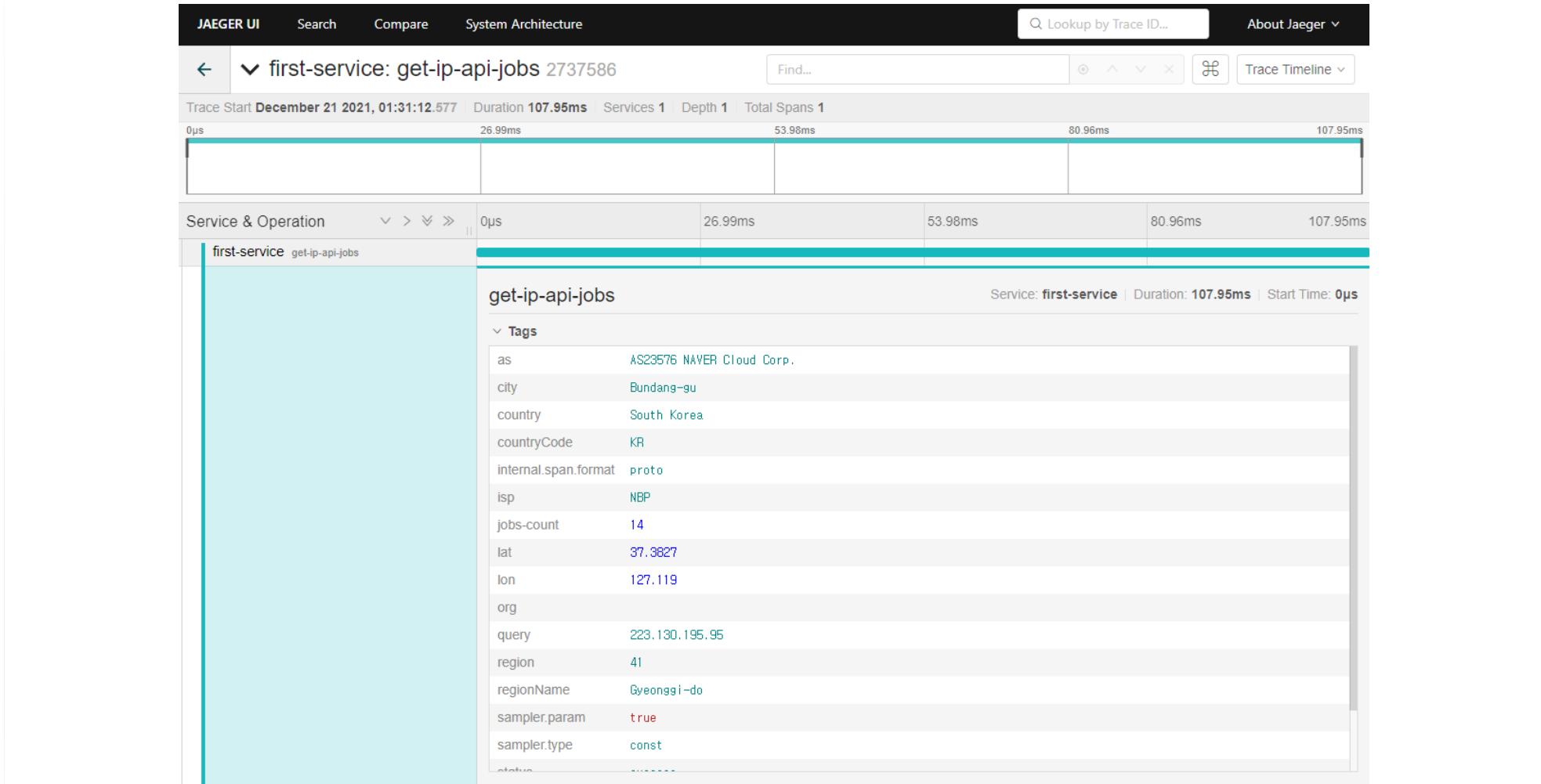
with tracer.start_span('get-ip-api-jobs') as span:
    try:
        res = requests.get('http://ip-api.com/json/naver.com')
        result = res.json()
        print('Getting status %s' % result['status'])
        span.set_tag('jobs-count', len(res.json()))
        for k in result.keys():
            span.set_tag(k, result[k])

    except:
        print('Unable to get site for')
```

Jaeger 트레이싱 튜토리얼

파이썬 프로그램에 트레이싱 예제

- 추가된 스펜 데이터를 다음과 같이 jaeger-tracer UI에서 확인 가능



쿠버네티스 CI/CD 환경 구성과 활용

▶ CI/CD 이해

- ▶ harbor를 활용한 컨테이너 레지스트리 구축
- ▶ 도커를 활용한 Jenkins 설치
- ▶ 깃헙 Webhook 구성과 젠킨스 파이프라인 설정
- ▶ Argo를 활용한 CD 환경 구축



CI/CD 이해

▶ CI/CD의 필요

출처: www.jetbrains.com

● 기존 개발 프로세스

- 소프트웨어를 출시하기까지는 힘든 과정과 오랜 시간이 필요
- 버그가 발견될 위험을 안고 몇 주의 시간을 들여 수작업으로 통합과 구성 및 테스트 작업을 수행
- 늘 출발점으로 되돌아가야 할 수도 있다는 두려움을 가짐
- 코드 릴리스를 위해서는 오랜 준비 과정이 필요하기 때문에 새로운 릴리스를 내놓기까지 적어도 몇 달은 소요

● CI/CD의 개념

- 통합, 전달 및 배포 도구를 의미
- CI/CD의 도움을 받아 많은 조직들이 이미 품질 저하 없이 릴리스 간격을 단축
- CI/CD를 사용하면 반복적 빌드, 테스트 및 배포 작업을 처리하고, 문제가 있을 때 경고
- 자동화된 파이프라인을 통해 코드 변경을 원활하게 진행

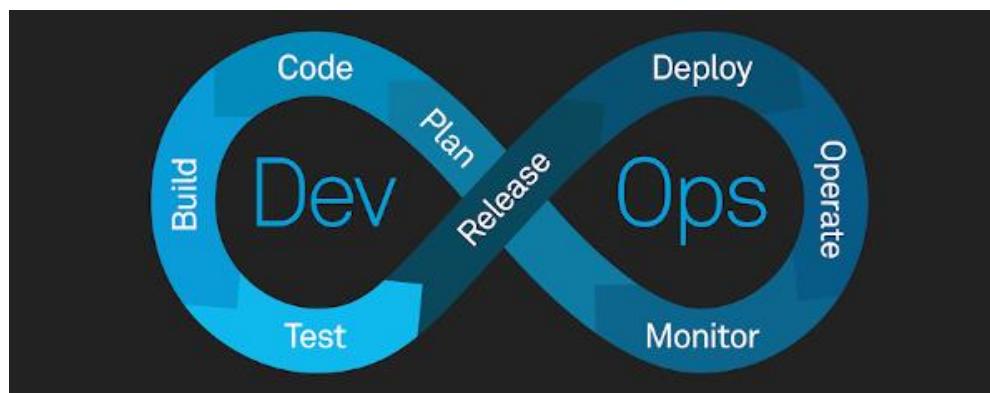


그림 출처: <https://cd.foundation/>

▶ CI/CD의 장점

출처: www.jetbrains.com

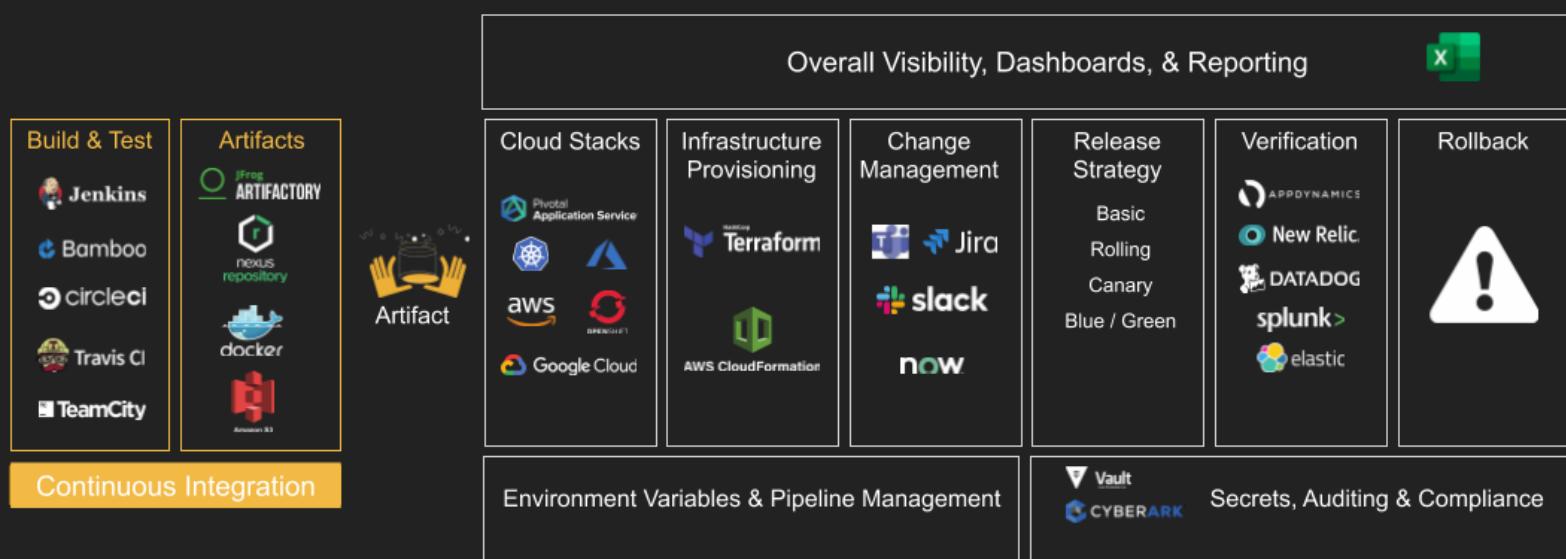
- **시장 출시 기간 단축:** Agile 및 DevOps 기술을 채택하여 개발 프로세스에 혁신을 이루고 사용자에게 지속적인 개선을 제공
 - 위험 감소: 사전 프로덕션 환경에 테스터 또는 실제 사용자를 참여시켜 초기에 자주 사용자와 함께 혁신적 기능을 테스트
 - 검토 시간 단축: 통합 도구를 이용하는 개발자는 자연스럽게 더 자주 코드를 변경하고 싶은 마음이 생기게 되는데, 경험적으로 최소 하루에 한 번은 코드를 변경, 동일한 기반 위에서 작업할 수 있을 뿐만 아니라 코드 검토 속도가 빨라지고 변경 사항을 더 쉽게 통합
- **코드 품질 개선:** CI/CD 파이프라인의 핵심 기능으로, 빌드할 때마다 실행되는 일련의 자동화 테스트
 - 프로덕션 환경으로 원활한 전환: 빌드 자동화, 테스트, 환경 생성 및 배포를 추가하면 각 단계의 일관성과 반복성을 높임, 지속적으로 최적화
 - 더 빠른 버그 수정: 변경 사항을 정기적으로 커밋하고 자주 제공하면 프로덕션으로 전환하는 릴리스 사이에 코드 변경이 상대적으로 적기 때문에 문제의 원인을 찾기가 훨씬 쉬움
 - 효율적인 인프라: 지속적 배포 단계가 더 빠르고 강력해질 뿐만 아니라 개발 작업에 대한 중단을 최소화하면서 추가적인 미리보기 및 교육 환경에 대한 요청에 신속하게 대응
 - 진행상황에 대한 평가 가능: 테스트 커버리지, 결함률, 테스트 수정 시간에 이르는 전체 평가 지표가 제공, 데이터를 확보하면 주의가 필요한 부분을 확인하여 파이프라인을 지속적으로 개선
 - 더 긴밀한 피드백 루프: 지속적 배포 주기마다 통찰력을 적용하면 변경을 수행하는 즉시 그 효과를 확인
 - 협업 및 커뮤니케이션: CI/CD를 시작하려면 팀 사이의 장벽을 허물고 더 많은 의사 소통의 분위기를 마련
- **창의력 극대화:** 컴퓨터를 사용하여 반복적인 작업을 수행함으로써 프로세스를 자동화하면 보다 창의적인 작업에 열중

CI/CD 이해

▶ CI/CD에 사용되는 도구들

그림 출처: <https://cd.foundation/>

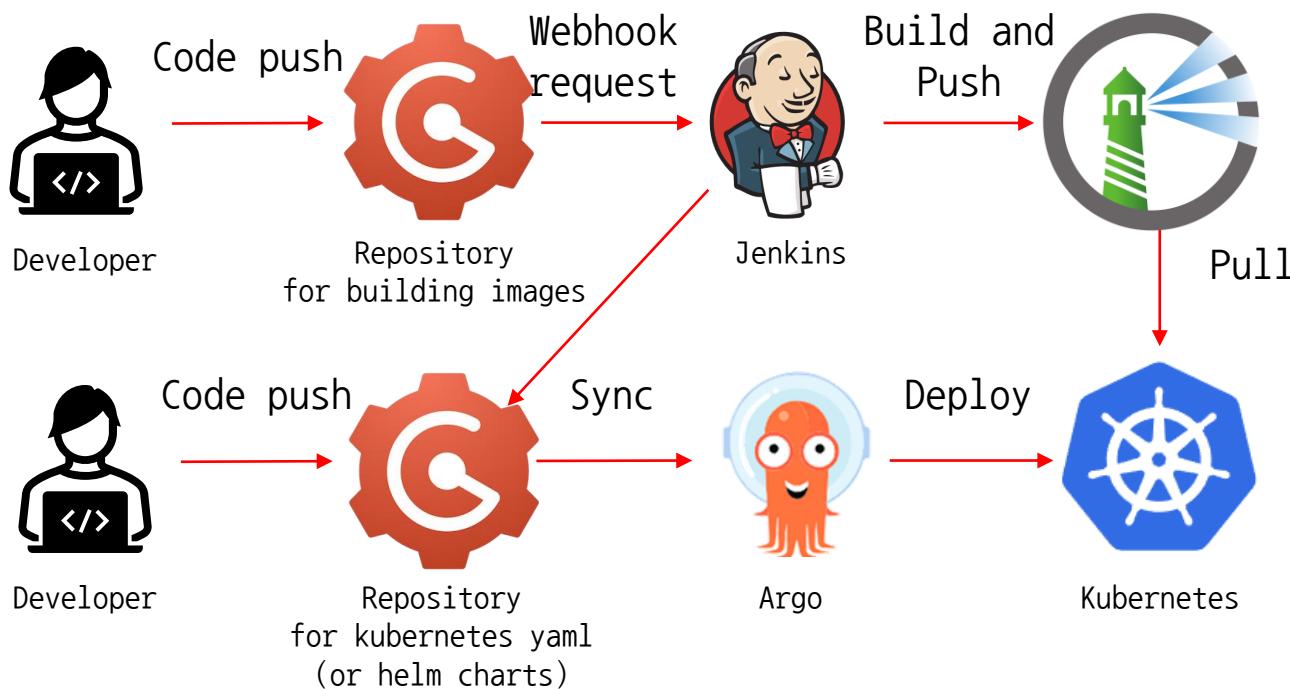
Continuous Integration != Continuous Delivery



CI/CD 이해

도커와 쿠버네티스 CI/CD 플랫폼

- 개발팀에서 코드를 작성하여 깃헙 레파지토리에 코드 푸시
- 코드 푸시 이벤트를 감지해 Webhook Request를 사용하여 Jenkins에 코드 전송
- Jenkins에서 코드를 사용해 빌드, 테스트한 후 도커 레지스트리에 업로드
- 쿠버네티스 배포를 위한 매니페스트를 위한 두 번째 깃헙 레파지토리 구성
- Argo를 사용해 실시간으로 현재 레파지토리와 클러스터의 상태를 모니터링하고 싱크 상태를 조정



harbor를 활용한 컨테이너 레지스트리 구축

harbor를 활용한 컨테이너 레지스트리 구축

▶ Harbor란?

- CNCF 출업 프로젝트로 오픈 소스 레지스트리
- 정책 및 역할 기반 액세스 제어를 제공
- 아티팩트를 보호하고, 이미지가 스캔, 취약성 확인, 이미지를 신뢰할 수 있도록 서명
- 웹 대시보드 제공



harbor를 활용한 컨테이너 레지스트리 구축

▶ Harbor를 설치할 인스턴스 생성

- GCP를 사용해 우분투 인스턴스를 구성, 디스크는 100GB

부팅 디스크 ?

유형 새로운 균형 있는 영구 디스크

크기 100GB

이미지 Ubuntu 20.04 LTS

변경

방화벽 ?

태그 및 방화벽 규칙을 추가하여 인터넷에서 들어오는 특정 네트워크 트래픽을 허용합니다.

HTTP 트래픽 허용

HTTPS 트래픽 허용

harbor를 활용한 컨테이너 레지스트리 구축

▶ docker-compose를 사용해서 배포

- 도커 설치가 필요
- 관리자 권한으로 넘어와서 도커 설치를 진행

```
sudo -i
```

```
# 도커 설치
```

```
apt update && apt install docker.io -y
```

```
# 도커 컴포즈 설치
```

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

harbor를 활용한 컨테이너 레지스트리 구축

▶ Harbor 설치

- harbor를 설치하기 위해 harbor 사이트에서 명령을 가져와 셀스크립트를 다운로드하고 실행

```
wget
```

```
https://gist.githubusercontent.com/kacole2/95e83ac84fec950b1a70b0853d6594dc/raw/ad6d65  
d66134b3f40900fa30f5a884879c5ca5f9/harbor.sh
```

```
bash harbor.sh
```

- 설치를 시작하면 IP와 도메인을 선택하는 메뉴가 나오는데 IP를 사용할 예정이므로 1번을 선택

```
1) IP
```

```
2) FQDN
```

```
Would you like to install Harbor based on IP or FQDN?
```

```
> 1번 선택
```

harbor를 활용한 컨테이너 레지스트리 구축

▶ Harbor 설치

- HTTPS 통신을 위한 인증서를 구성하고 설정하여 설치를 진행
- 다음 스크립트를 실행해 CA와 Harbor에서 사용할 Certificate를 생성

```
#!/bin/bash
#
mkdir pki
cd pki

# ca 키와 인증서 생성
openssl req -x509 -nodes -days 3650 -newkey rsa:2048 \
    -out ca.crt \
    -keyout ca.key \
    -subj "/CN=ca"

# harbor server 키와 인증서 생성
openssl genrsa -out server.key 2048
openssl req -new -key server.key -out server.csr -subj "/CN=harbor-server"
openssl x509 -req -in server.csr -CA ca.crt \
    -CAkey ca.key \
    -CAcreateserial -out server.crt -days 365

# 키와 인증서 복제
mkdir -p /etc/docker/certs.d/server
cp server.crt /etc/docker/certs.d/server/
cp server.key /etc/docker/certs.d/server/
cp ca.crt /etc/docker/certs.d/server/

cp ca.crt /usr/local/share/ca-certificates/harbor-ca.crt
cp server.crt /usr/local/share/ca-certificates/harbor-server.crt
update-ca-certificates
```

harbor를 활용한 컨테이너 레지스트리 구축

▶ Harbor 설치

- harbor.yml 템플릿을 사용해 harbor의 설정을 구성

```
cd ~/harbor  
cp harbor.yml.tpl harbor.yml  
vim harbor.yml
```

- harbor.yml 파일의 내부에서 호스트 이름과 키, 인증서의 경로만 바꿈
- 여기서 초기 패스워드와 유저 아이디도 확인 가능
- 호스트 이름의 IP는 현재 인스턴스의 공인 IP를 입력

hostname: 34.134.229.160

<중략>

```
# https related config  
https:  
  # https port for harbor, default is 443  
  port: 443  
  # The path of cert and key files for nginx  
  certificate: /etc/docker/certs.d/server/server.crt  
  private_key: /etc/docker/certs.d/server/server.key
```

harbor를 활용한 컨테이너 레지스트리 구축

▶ Harbor 설치

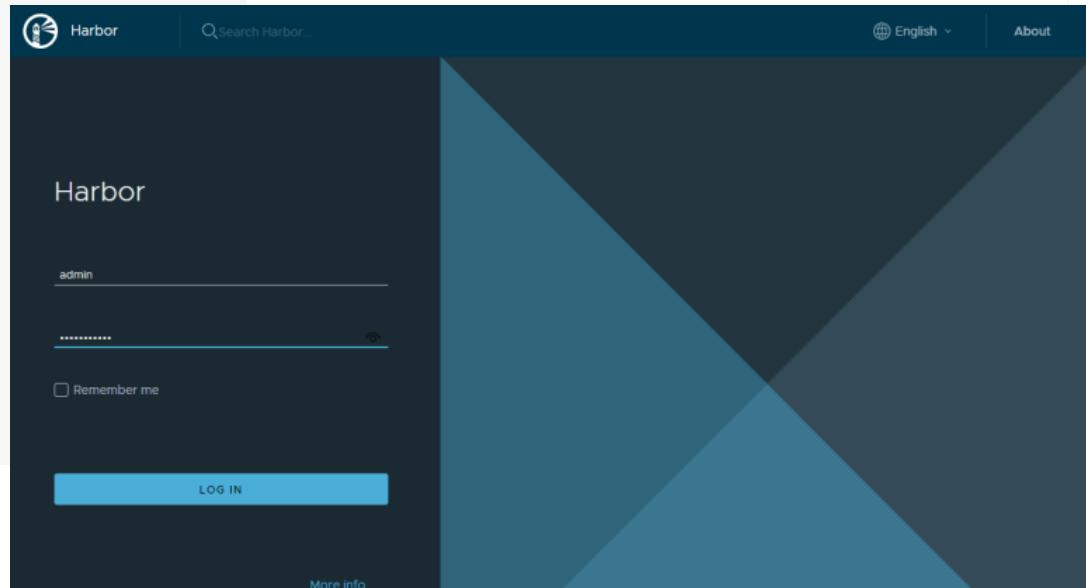
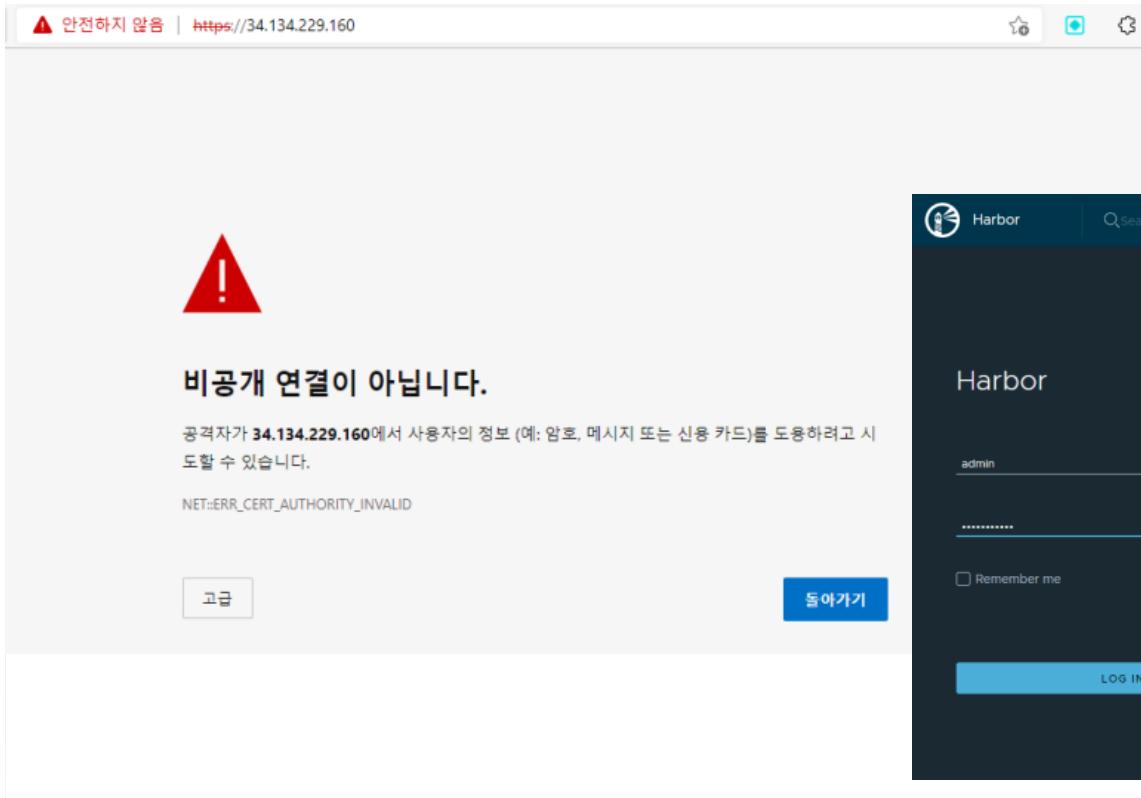
- 끝으로 준비 및 설치 스크립트를 실행
- 준비 스크립트는 이미지를 준비하고 인증서 파일을 위한 설정을 구성
- install.sh 파일은 도커 컴포즈를 사용해 harbor 실행에 필요한 컨테이너들을 배포

```
./prepare  
./install.sh
```

harbor를 활용한 컨테이너 레지스트리 구축

▶ 웹 Harbor 접속

- 웹브라우저로 인스턴스 IP로 접속
- 80포트는 443포트로 리다이렉션
- 고급을 누르고 안전하지 않음을 선택
- Harbor의 초기 ID와 패스워드는 admin//Harbor12345다. 접속을 수행한다.



harbor를 활용한 컨테이너 레지스트리 구축

▶ 웹 Harbor 패스워드 변경

- Admin 계정의 패스워드를 변경

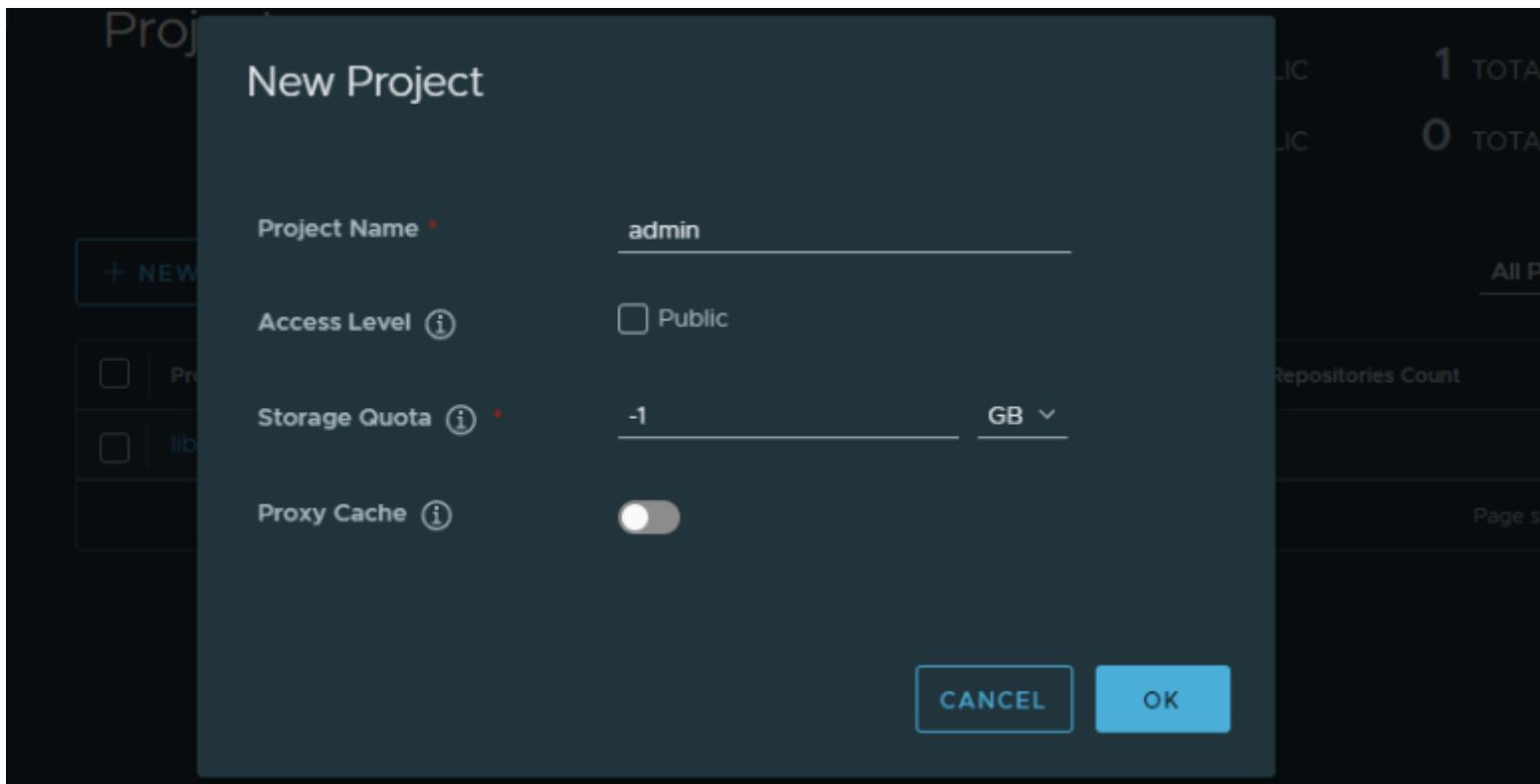
The screenshot shows the Harbor web interface with the following details:

- Header:** Harbor logo, Search bar, English language selection, and a dropdown for the user "admin". The "Change Password" option in this dropdown is highlighted with a red box.
- Sidebar:** Projects (selected), Logs, Administration (with sub-options: Users, Robot Accounts, Registries, Replications, Distributions, Labels, Project Quotas, Interrogation Services, Garbage Collection, Configuration, LIGHT, and Harbor API V2.0).
- Main Content:**
 - Projects Summary:** PROJECTS: 0 PRIVATE, 1 PUBLIC, 1 TOTAL; REPOSITORIES: 0 PRIVATE, 0 PUBLIC, 0 TOTAL.
 - Project List:** A table with columns: Project Name, Access Level, Role, Type, Repositories Count, and Creation Time. One row is visible: library (Public, Project Admin, Project, 10/26/21, 1:18 AM).
 - Buttons:** + NEW PROJECT, X DELETE.
 - Filters:** All Projects, search bar, and a clear button.
 - Pagination:** Page size: 15, 1 - 1 of 1 items.

harbor를 활용한 컨테이너 레지스트리 구축

▶ 웹 Harbor 프로젝트 생성

- 처음 로그인하면 기본 프로젝트인 Library가 보임
- 이 프로젝트는 누구나 엑세스해서 사용할 수 있는 Public 모드로 구성
- 여기에 New Project 버튼을 눌러 새 프로젝트를 생성



harbor를 활용한 컨테이너 레지스트리 구축

▶ 프로젝트 멤버 설정

- admin 프로젝트로 가면 해당 기능을 사용할 수 있는 유저를 정할 수 있는 기능
- 프로젝트 단위대로 권한을 부여해 액세스할 수 있는 유저 구성
- 유저는 왼쪽 유저 탭에서 새로운 유저를 구성 가능

The screenshot shows the 'Members' tab selected in the Harbor project interface. The top navigation bar includes tabs for Summary, Repositories, Members (highlighted with a red box), Labels, Scanner, P2P Preheat, Policy, Robot Accounts, Webhooks, Logs, and more. Below the tabs are buttons for '+ USER' and '+ GROUP'. A search bar with a magnifying glass icon and a refresh/clear button are also present. The main table displays one member entry:

<input type="checkbox"/>	Name	Member Type	Role
<input type="checkbox"/>	admin	User	Project Admin

At the bottom right of the table, there are 'Page size' and '1 - 1 of 1 items' indicators.

harbor를 활용한 컨테이너 레지스트리 구축

▶ docker CLI를 활용한 Harbor 사용

- 도커 CLI에서 하버로 접속하는 설정
- admin 유저를 사용해 원하는 이미지를 업로드하기 위해 로그인

```
docker login 34.134.229.160 -u admin -p Harbor12345
```

- admin 권한으로 업로드를 진행

- nginx를 pulling하고 nginx에 태그를 추가
- 전달되는 IP는 Harbor의 IP, 그리고 추가된 태그로 푸시를 진행

```
docker pull nginx
docker tag nginx 34.134.229.160/admin/nginx
docker push 34.134.229.160/admin/nginx
```

도커를 활용한 Jenkins 설치

도커를 활용한 Jenkins 설치

▶ 젠킨스란?

- 어떤 규모로든 훌륭한 시스템을 구축하기 위한 도구
- 선도적인 오픈 소스 자동화 서버
- 모든 프로젝트를 빌드, 배포 및 자동화하는 데 지원
- 수백 개의 플러그인 제공



지속적인 통합 및 지속적인 납품



확장 가능한 자동화 서버인 Jenkins는 간단한 CI 서버로 사용하거나 모든 프로젝트의 연속 배달 허브로 전환할 수 있습니다.

간편한 설치



Jenkins는 독립형 자바 기반 프로그램으로, 윈도우, 리눅스, 맥OS 및 기타 유닉스와 같은 운영 체제에 대한 패키지와 함께 즉시 실행 될 준비가되어 있습니다.

쉬운 구성



Jenkins는 웹 인터페이스를 통해 쉽게 설정하고 구성할 수 있으며, 여기에는 즉석 오류 검사 및 기본 제공 도움말이 포함됩니다.

플러그인



업데이트 센터에 수백 개의 플러그인을 갖춘 Jenkins는 지속적인 통합 및 지속적인 배달 도구 체인의 거의 모든 도구와 통합됩니다.

확장



Jenkins는 플러그인 아키텍처를 통해 확장 될 수 있습니다. Jenkins 무엇을 할 수 있는 거의 무한 한 가능성을 제공합니다.

분산



Jenkins는 여러 컴퓨터에 작업을 쉽게 배포할 수 있으며 여러 플랫폼에서 빌드, 테스트 및 배포를 더 빠르게 구축할 수 있습니다.

도커를 활용한 Jenkins 설치

Jenkins 서버 생성

- Jenkins 서버 구성을 시작
- Jenkins의 서버는 외부의 공개된 IP 필요
- 공용 IP를 편하게 구성하기 위해 GCP에 인스턴스를 구성하여 Jenkins를 구축
- GCP에서 인스턴스를 생성
- 인스턴스 이름은 jenkins-ci로 구성하고 CPU는 2개 메모리는 4GB를 선택

The screenshot shows the Google Cloud Platform Compute Engine VM Instances creation interface. A red box highlights the '인스턴스 만들기' (Create Instance) button. On the right, detailed configuration options are shown:

- 이름 ***: jenkins-ci (highlighted by a red box)
- 라벨**: + ADD LABELS
- 리전 ***: us-central1 (아이오와) (highlighted by a red box)
- 영역 ***: us-central1-a (highlighted by a red box)
- 활별 예상 가격**: US\$25.46
- 시간당 약**: US\$0.03
- 사용한 만큼만 비용 지불**: 선불 비용 없이 초당 청구
- 세부정보**
- 머신 구성**
- 마신 계열**: 일반 용도 (highlighted by a red box)
- E2** (highlighted by a red box)
- 마신 유형**: e2-medium(vCPU 2개, 4GB 메모리) (highlighted by a red box)
- vCPU**: 공유 코어 1개
- Memory**: 4GB

도커를 활용한 Jenkins 설치

Jenkins 서버 생성

- 인스턴스의 부팅 디스크에서 변경을 누르고 Ubuntu 20.04 100GB 디스크로 구성
- 방화벽은 뒤에 세팅할 다른 서비스를 위해 HTTP/HTTPS 트래픽을 허용

부팅 디스크

유형	새로운 균형 있는 영구 디스크
크기	100GB
이미지	 Ubuntu 20.04 LTS

변경

공개 이미지 커스텀 이미지 스냅샷 기존 디스크

운영체제
Ubuntu

버전 *
Ubuntu 20.04 LTS

amd64 focal image built on 2021-09-27, supports Shielded VM features

부팅 디스크 유형 *
균형 있는 영구 디스크

크기(GB) *
100

▼ 고급 구성 표시

선택

취소

도커를 활용한 Jenkins 설치

▶ Jenkins 서버 구성

- 빠르고 쉬운 설치를 위해 도커를 설치하고 도커 이미지를 사용해 Jenkins를 배포
- Jenkins를 배포할 때는 일부 디렉토리를 공유하도록 설정
- 도커 소켓 또한 공유하도록 구성
- 이 소켓을 사용해 Jenkins는 호스트에 설치된 도커 기능을 사용

```
# 관리자 권한  
sudo -i
```

```
# docker 설치  
apt update && apt install -y docker.io
```

```
# 도커를 사용해 jenkins 구성 및 도커 소켓 공유  
docker run -d -p 8080:8080 --name jenkins -v /home/jenkins:/var/jenkins_home -v  
/var/run/docker.sock:/var/run/docker.sock -u root jenkins/jenkins:lts
```

도커를 활용한 Jenkins 설치

▶ Jenkins 서버 구성

- 젠키스 서비스를 위해 방화벽을 8080 포트 허용

```
gcloud compute firewall-rules create jenkins-ci --allow=tcp:8080
```

- 젠킨스 도커 클라이언트 설치

```
# jenkins에 docker client 설치  
docker exec jenkins apt update  
docker exec jenkins apt install -y docker.io
```

- 젠킨스 패스워드 확인

```
# 젠킨스 초기패스워드 조회  
docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

도커를 활용한 Jenkins 설치

Jenkins 서버 구성

- 젠킨스 설치를 위해 웹으로 접근해 플러그인 설치

The image consists of three vertically stacked screenshots of the Jenkins 'Getting Started' configuration wizard.

- Unlock Jenkins:** This step guides the user through unlocking Jenkins. It includes instructions to find the initial admin password in the log or at `/var/jenkins_home/secrets/initialAdminPassword`, and a field for entering the administrator password.
- Customize Jenkins:** This step allows users to extend Jenkins with plugins. It offers two options:
 - Install suggested plugins:** A red box highlights this option, which installs recommended plugins for common use cases.
 - Select plugins to install:** This option allows users to manually choose specific plugins.
- Instance Configuration:** This step configures the Jenkins instance URL. It shows the current value as `http://34.135.119.94/` and provides a detailed explanation of what the Jenkins URL is used for, including its role in email notifications and build steps.

도커를 활용한 Jenkins 설치

Jenkins 서버 구성

- 젠킨스 구성이 완료되면 Jenkins 관리 - 플러그인 관리로 접근
- docker pipeline을 검색해서 설치 진행

The screenshot shows the Jenkins Management interface. On the left, there's a sidebar with links like 'Dashboard', '새로운 Item', '사람', '빌드 기록', 'Jenkins 관리' (which is selected), 'My Views', 'Lockable Resources', and 'New View'. Below the sidebar is a section for 'Build Queue'. The main area is titled 'Jenkins 관리' and contains a 'System Configuration' section. It includes links for '시스템 설정' (System Configuration), 'Global Tool Configuration', and a red-highlighted '플러그인 관리' (Plugin Management) section. A search bar at the top right contains the text 'docker pipeline'. Below it, a table lists the 'Docker Pipeline' plugin. The table has columns for 'Name', 'Version', and 'Release Date'. The 'Name' column shows 'Docker Pipeline', the 'Version' column shows '1.26', and the 'Release Date' column shows '8 months ago'. At the bottom of the table, there are buttons for 'Install without restart' (which is highlighted with a red border), 'Download now and install after restart', and '지금 확인' (Check Now). A status message at the bottom says 'Update information obtained: 11 min ago'.

Name	Version	Release
Docker Pipeline <input checked="" type="checkbox"/> Deployment DevOps docker pipeline Build and use Docker containers from pipelines.	1.26	8 months ago

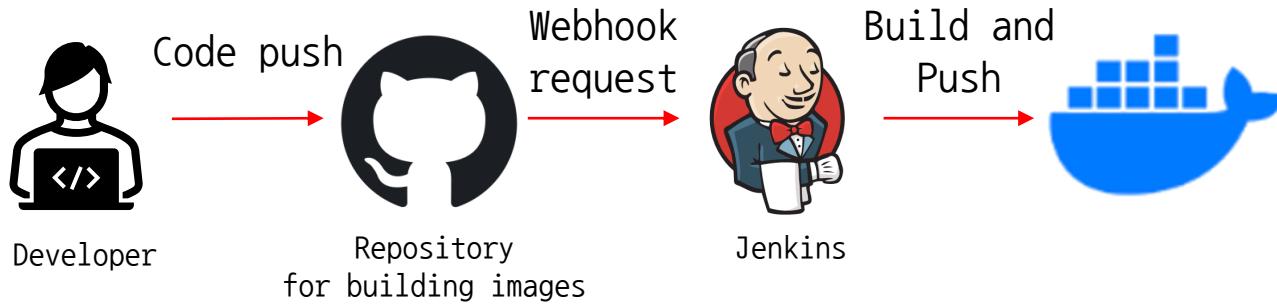
Install without restart Download now and install after restart Update information obtained: 11 min ago 지금 확인

깃헙 레파지토리 Webhook 구성과 jenkins 파이프라인 설정

깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

▶ github 레파지토리 구성

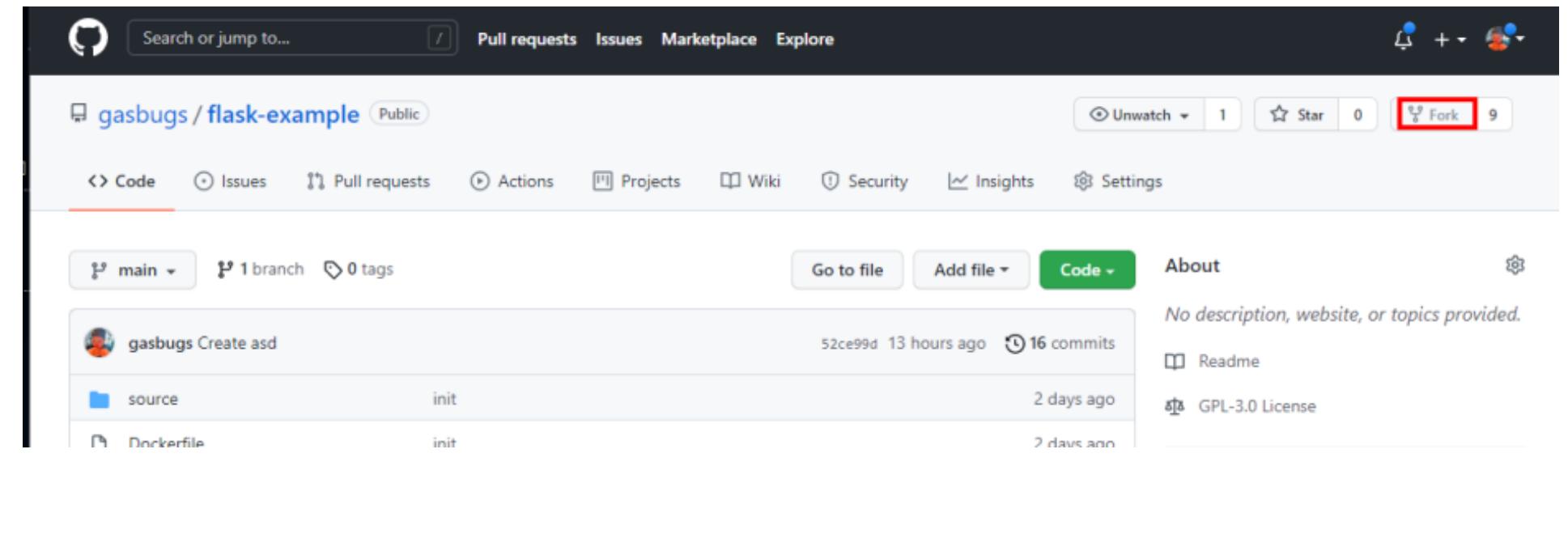
- 필자가 미리 구성해둔 flask-example 프로젝트를 활용
- <https://github.com/gasbugs/flask-example>
- 이 프로젝트를 fork해서 내 레파지토리로 가져와야 함
- 이 예제에는 Docker 이미지를 구성하는 Dockerfile과 Jenkins 파이프라인에 사용할 Jenkinsfile이 함께 첨부



깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

▶ github 레파지토리

- github 레파지토리를 구성
- 필자가 미리 구성해둔 flask-example 프로젝트를 활용
- <https://github.com/gasbugs/flask-example>
- 이 프로젝트를 fork해서 내 레파지토리로 가져와야 함
- 이 예제에는 Docker 이미지를 구성하는 Dockerfile과 Jenkins 파이프라인에 사용할 Jenkinsfile이 함께 첨부



깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

▶ github 레파지토리 웹훅 설정

- 변경사항이 있을 때마다 jenkins에 알릴 수 있도록 Webhook을 설정
- 레파지토리의 Settings - Webhooks에 구축된 젠킨스 인스턴스 URL 주소와 /github-webhook을 함께 작성하고 저장

The screenshot shows the GitHub repository settings page for 'gasbugs/flask-example'. The 'Webhooks' tab is selected in the sidebar. The main area displays the 'Manage webhook' configuration. A red box highlights the 'Payload URL' field containing 'http://35.193.232.22/github-webhook/'. Another red box highlights the 'Content type' dropdown set to 'application/json'. Below these, the 'Secret' field is empty. At the bottom, a red box highlights the 'Just the push event.' radio button, which is selected.

gasbugs / flask-example Public

Unwatch 1 Star 0 Fork 9

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Options

Manage access

Security & analysis

Branches

Webhooks

Notifications

Integrations

Deploy keys

Actions

Environments

Secrets

Pages

Webhooks / Manage webhook

Settings Recent Deliveries

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our developer documentation.

Payload URL *

http://35.193.232.22/github-webhook/

Content type

application/json

Secret

Which events would you like to trigger this webhook?

Just the push event.

Send me everything.

www.dooriproject.com

깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

▶ Jenkins에서 파이프라인 구성

- Jenkins에서 Webhook 이벤트를 받아 빌드를 시작할 수 있도록 구성
- 새로운 Item을 클릭
- 이름을 적절하게 구성하고 pipeline을 선택
- 이름: flask-example-docker-pipeline

The screenshot shows the Jenkins dashboard with a red box highlighting the 'Pipeline' option under the 'Freestyle project' section. The 'Pipeline' section is described as orchestrating long-running activities across multiple build agents, suitable for building pipelines or organizing complex activities.

Jenkins

Dashboard

새로운 Item

사람

빌드 기록

Enter an item name

flask-example-docker-pipeline

» Required field

Freestyle project

이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.

Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

▶ Jenkins에서 파이프라인 구성

- General에는 Github프로젝트를 선택하고 프로젝트 URL 정보를 입력
- 포크한 본인의 레파지토리를 입력

The screenshot shows the 'General' configuration tab for a Jenkins job. The 'GitHub project' checkbox is checked, and the 'Project url' field contains the URL <https://github.com/gasbugs/flask-example>. Below the project URL, there is a section with several checkboxes:

- Do not allow concurrent builds
- Do not allow the pipeline to resume if the controller restarts
- GitHub project
- Pipeline speed/durability override
- Preserve stashes from completed builds
- Throttle builds
- 오래된 빌드 삭제
- 이 빌드는 매개변수가 있습니다

깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

▶ Jenkins에서 파이프라인 구성

- 빌드 트리거는 Webhook을 통해서 시작할 수 있도록 설정

The screenshot shows the 'Build Triggers' section of a Jenkins project configuration. It lists several options with checkboxes:

- Build after other projects are built
- Build periodically
- GitHub hook trigger for GITScm polling
- Poll SCM
- 빌드 안함
- Quiet period
- 빌드를 원격으로 유발 (예: 스크립트 사용)

- 파이프라인 정보로 사용할 flask-example의 데이터를 설정
- Credentials는 Add 버튼을 눌러 새로 생성

The screenshot shows the 'Pipeline' configuration page under 'Definition'. It includes sections for 'Pipeline script from SCM' (SCM), 'Git' repository, and 'Repository URL' (set to <https://github.com/gasbugs/flask-example>). In the 'Credentials' section, there is a dropdown menu set to '- none -' and a red box highlights the 'Add' button.

깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

▶ Jenkins에서 파이프라인 구성

- 깃헙에서 토큰을 발급 받아 다음과 같이 입력
- 토큰 발급에 대한 상세 절차는 생략
 - username: github ID
 - password: Token 값
 - github_cred: 젠킨스에서 사용할 Credential ID

The screenshot shows the Jenkins Credentials Provider: Jenkins interface. It is a modal window titled 'Jenkins Credentials Provider: Jenkins'. The 'Add Credentials' section is active. The 'Domain' dropdown is set to 'Global credentials (unrestricted)'. The 'Kind' dropdown is set to 'Username with password'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' field contains 'gasbugs'. The 'Password' field is filled with a series of dots. The 'ID' field is set to 'github_cred'. There is a 'Description' field which is currently empty. At the bottom left are 'Add' and 'Cancel' buttons.

깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

▶ Jenkins에서 파이프라인 구성

- main 정보를 입력하고 Jenkinsfile에 대한 위치 정보는 그대로 유지
- Flask-example/Jenkinsfile의 파이프라인 정보를 읽어서 빌드 절차를 진행
- 모든 것이 완료되면 “저장” 버튼

The screenshot shows the Jenkins Pipeline configuration page. The 'Branches to build' section has a 'Branch Specifier' input containing '/main'. There is an 'Add Branch' button and an 'X' button. Below this is the 'Repository browser' section with a dropdown set to '(자동)'. Under 'Additional Behaviours', there is an 'Add' button. The 'Script Path' section shows 'Jenkinsfile' in the input field. A checked checkbox for 'Lightweight checkout' is present. At the bottom, there is a link for 'Pipeline Syntax'.

깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

▶ Jenkins에 도커 허브 인증 정보 입력

- 이미지 빌드를 수행한 뒤 docker hub로 푸시를 진행하려면 docker hub에 대한 인증정보가 필요
- Jenkins 관리에 Manage Credentials로 진입

The screenshot shows the Jenkins System Configuration page. On the left sidebar, the 'Jenkins 관리' (Jenkins Management) option is highlighted with a red box. In the main content area, there are several configuration sections: 'System Configuration' (with a gear icon), 'Global Tool Configuration' (with a wrench and screwdriver icon), '노드 관리' (Node Management) (with a computer monitor icon), and 'Security' (with a padlock icon). The 'Manage Credentials' link under the Security section is also highlighted with a red box.

building on the controller node can be a security issue. You should set up distributed buildsl. See the documentation.

System Configuration

시스템 설정
환경변수 및 경로 정보 등을 설정합니다.

Global Tool Configuration
Configure tools, their locations and automatic installers.

노드 관리
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Security

Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.

Manage Credentials
Configure credentials

깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

▶ Jenkins에 도커 허브 인증 정보 입력

- Jenkins 스토어로 진입한 후에 Global credentials로 계속 접근

Stores scoped to Jenkins

The screenshot shows the Jenkins Global credentials configuration interface. It consists of two main parts: a left sidebar and a right content area.

Left Sidebar:

- A navigation bar at the top with tabs: 'P' (selected), 'Store ↓', and 'Domains'.
- An icon for 'Jenkins' (a house with a Jenkins logo) is highlighted with a red box.
- An icon for '(global)' is also present.
- A section titled 'System' follows, with an icon for 'Domain'.
- A table below it lists 'Global credentials (unrestricted)' with a description: 'Credentials that should be available irrespective of domain specification to requirements matching.' An icon for a castle is next to the table.
- At the bottom of the sidebar, there are icons for 'S' (Small), 'M' (Medium), and 'L' (Large).

Right Content Area:

- A header 'Jenkins' with a search bar.
- A breadcrumb navigation: Dashboard > Credentials > System > Global credentials (unrestricted).
- A link 'Back to credential domains' with a green arrow icon.
- A button 'Add Credentials' with a key icon, highlighted with a red box.
- A title 'Global credentials (unrestricted)' with a castle icon.
- A description: 'Credentials that should be available irrespective of domain specification to requirements matching.'
- A table with columns: ID, Name, and Kind.
- A single row in the table:

ID	Name	Kind
github_cred	gasbugs/*****	Username with password
- At the bottom of the table, there are icons for 'S' (Small), 'M' (Medium), and 'L' (Large).

깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

▶ Jenkins에 도커 허브 인증 정보 입력

- username과 password에 도커 허브의 ID와 패스워드를 차례로 입력
- ID는 반드시 docker-hub로 입력
- Jenkinsfile에서 이 ID를 참조하도록 구성

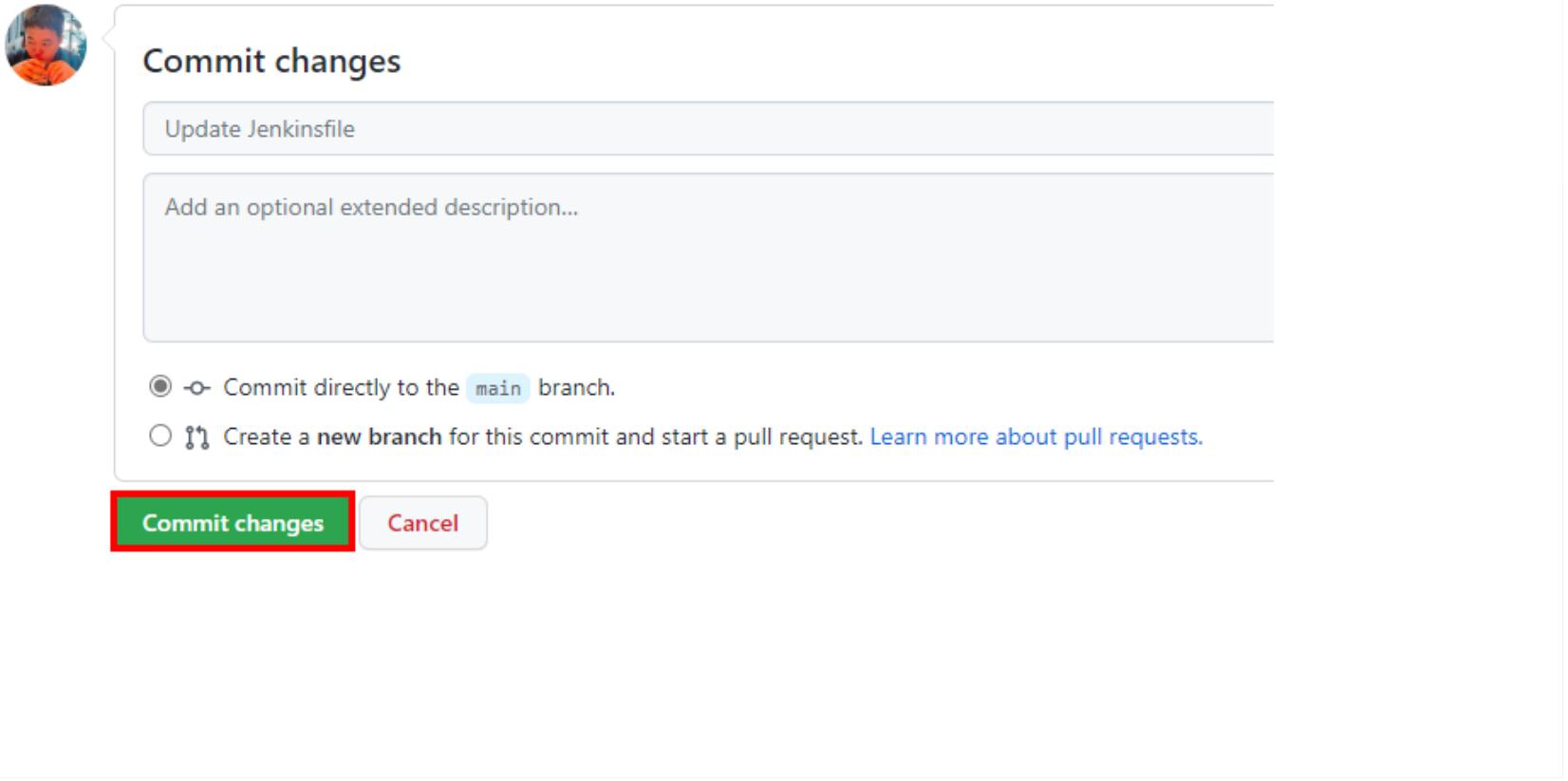
The screenshot shows the Jenkins 'Credentials' configuration page for a 'Username with password' entry. The 'Kind' dropdown is set to 'Username with password'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' field contains 'gasbugs'. The 'Password' field contains a redacted password. The 'ID' field contains 'docker-hub', which is highlighted with a red box. A checkbox labeled 'Treat username as secret' is unchecked.

Kind	Scope	Username	Password	ID
Username with password	Global (Jenkins, nodes, items, all child items, etc)	gasbugs	docker-hub

깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

▶ 빌드와 푸시를 위한 Jenkinsfile 설정 변경

- 코드를 변경해서 Github의 웹훅이 적절히 전달되는지 확인
- Github 사이트에서 flask-example/Jenkinsfile를 선택하고 수정 버튼을 누르면 푸시 이벤트 발생
- gasbugs라는 필자의 ID를 당신의 docker hub ID로 변경하고 수정 완료



깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

▶ Github 웹훅 테스트

- 코드를 바꿨지만 젠킨스로 가보면 빌드가 진행되지 않음
- 첫 빌드는 jenkins에서 Build now 버튼을 눌러 직접 진행 주어야 함
- 첫 빌드 후에는 푸시 이벤트 발생 시 자동으로 빌드 진행

Jenkins

Dashboard > flask-example-docker-pipeline >

Back to Dashboard

Status

Changes

Build Now (highlighted with a red box)

구성

Pipeline 삭제

Full Stage View

Github

Rename

Pipeline Syntax

GitHub Hook Log

Build History

Stage View

Average stage times:
(Average full run time: ~1min 3s)

Clone repository	Build Image	Push image	Build Image	Push image
904ms	21s	7s	0ms	0ms
Oct 23 11:53	No Changes			
904ms	42s	9s	390ms	4s

고정링크

- Last build. (#1), 26 sec 전

Argo를 활용한 CD 환경 구축

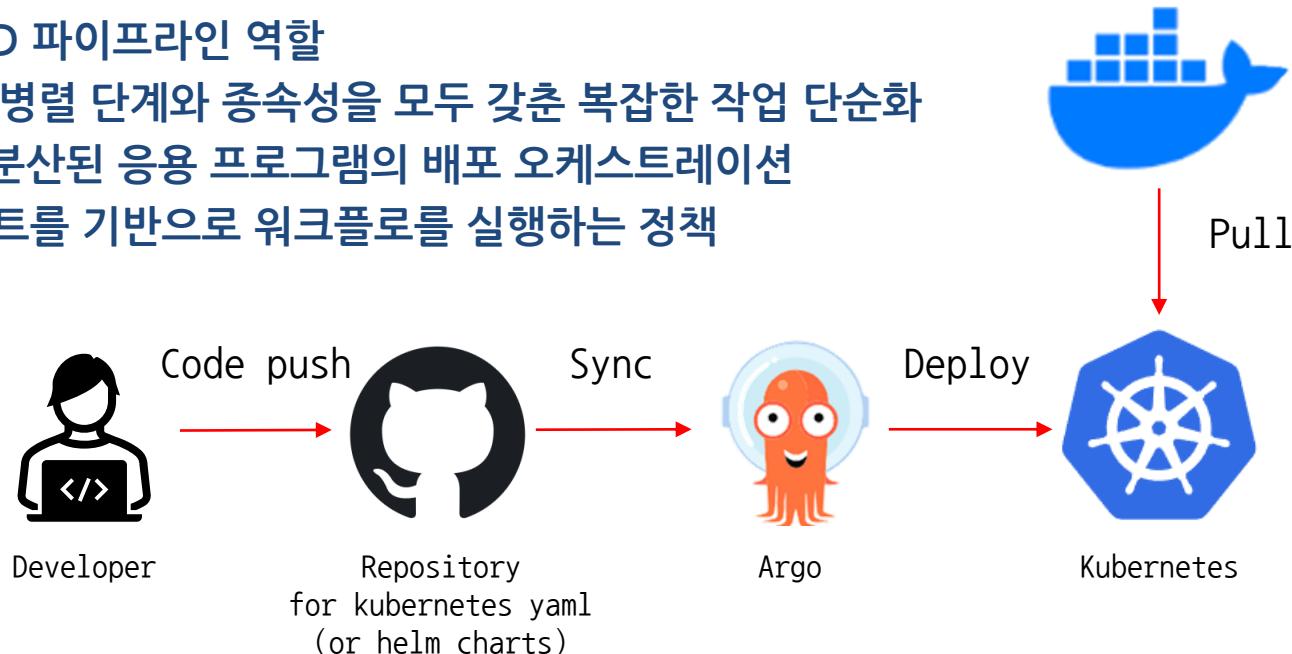
Argo를 활용한 CD 환경 구축

■ argo의 소개

- argo는 Kubernetes용 오픈 소스 도구로 워크플로를 실행하고 클러스터를 관리 GitOps를 수행
- 쿠버네티스의 CD를 담당하는 도구로 쿠버네티스의 컨테이너 네이티브 워크플로우 엔진
- Kubernetes에서 복잡한 워크플로 및 응용 프로그램의 실행을 쉽게 지정하고 예약, 조정

■ argo의 워크플로의 역할

- 기존 CI/CD 파이프라인 역할
- 순차적 및 병렬 단계와 종속성을 모두 갖춘 복잡한 작업 단순화
- 복잡하고 분산된 응용 프로그램의 배포 오케스트레이션
- 시간/이벤트를 기반으로 워크플로를 실행하는 정책

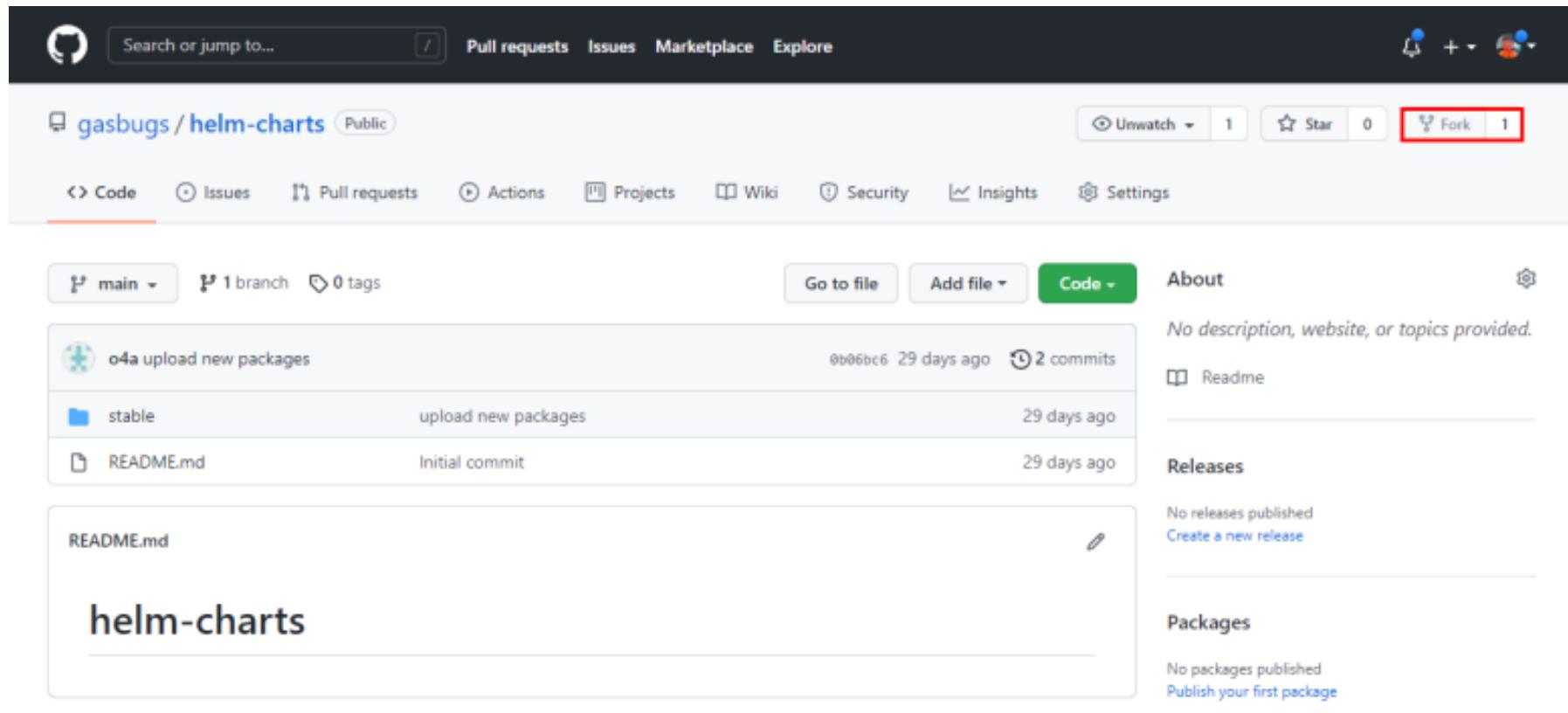


Argo를 활용한 CD 환경 구축

▶ github 레파지토리 포크

● 깃헙 레파지토리를 구성

- 필자가 미리구성해 둔 레파지토리를 포크
- <https://github.com/gasbugs/flask-example-apps>
- <https://github.com/gasbugs/helm-charts>



The screenshot shows the GitHub repository page for 'gasbugs/helm-charts'. At the top, there's a navigation bar with links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the header, the repository name 'gasbugs / helm-charts' is displayed, along with a 'Public' badge. To the right of the repository name are buttons for 'Unwatch', 'Star', and 'Fork'. The 'Fork' button is highlighted with a red box. Below the header, there's a toolbar with links for 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. Under the 'Code' tab, it shows 'main' branch, '1 branch', and '0 tags'. On the left, there's a list of commits:

- o4a upload new packages (commit 0b06bc6, 29 days ago, 2 commits)
- stable upload new packages (29 days ago)
- README.md Initial commit (29 days ago)

On the right side of the page, there are sections for 'About', 'Readme', 'Releases', and 'Packages'. The 'About' section notes 'No description, website, or topics provided.' The 'Readme' section has a link to 'Create a new release'. The 'Releases' section notes 'No releases published'. The 'Packages' section notes 'No packages published' and 'Publish your first package'.

Argo를 활용한 CD 환경 구축

Argo 설치

- 로드밸런서로 변경하고 서비스의 IP를 확인
- 변경되는데 1분 정도 소요

```
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'  
kubectl get svc argocd-server -n argocd -w
```

// 출력

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
argocd-server	LoadBalancer	10.8.9.12	<pending>	80:32524/TCP,443:31837/TCP	35m
argocd-server	LoadBalancer	10.8.9.12	35.222.254.32	80:32524/TCP,443:31837/TCP	36m

- 시크릿 정보를 통해 argo 패스워드 확인

```
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath=".data.password" | base64 -d
```

Argo를 활용한 CD 환경 구축

Argo 접속

- 로드밸런서로 할당 받은 IP로 https로 접속
- username: admin
- password: <secret을 통해 확인한 값>

The image shows two screenshots of the Argo UI. On the left, a large orange cartoon octopus character with a smiling face is standing on a grey gear, with the text "Let's get stuff deployed!" above it. On the right, the Argo login screen has the word "argo" in orange at the top. It features two input fields: "Username" and "Password". Below the login form, the Argo application dashboard is visible. It includes a sidebar with icons for Applications (v2.1.5+), + NEW APP, SYNC APPS, and a search bar. The main area shows a large circular icon with three stacked squares, and the text "No applications yet" followed by "Create new application to start managing resources in your cluster" and a "CREATE APPLICATION" button. The top right corner of the dashboard shows "APPLICATIONS", a grid icon, a user icon, and a "Log out" link.

Argo를 활용한 CD 환경 구축

Argo 앱 생성

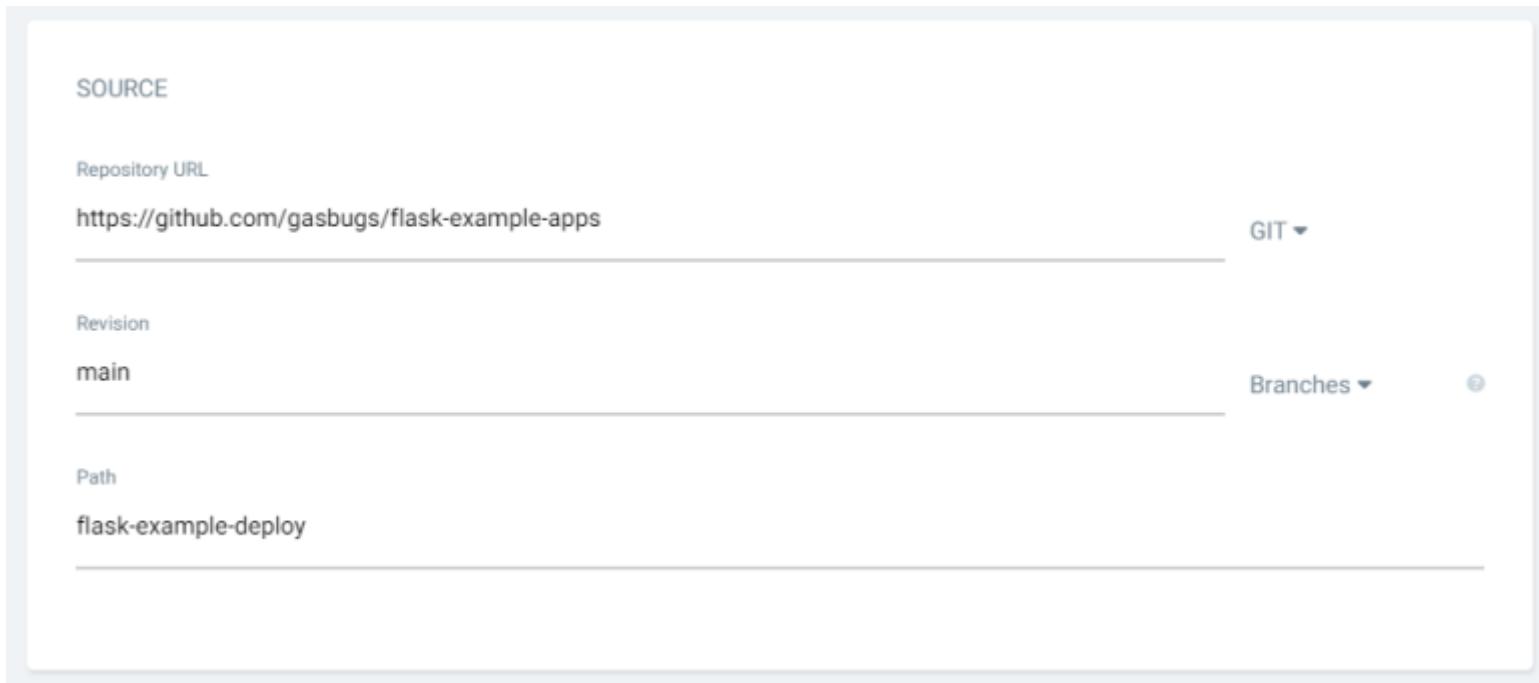
- 애플리케이션 이름: flask-example
- 프로젝트: default (Argo의 기본 프로젝트)
- 싱크 방식: Manual
 - 자동 동기화: Git에 설정된 매니페스트 내용과 현재 애플리케이션의 상태가 다르거나 정상 동작하지 않으면 자동으로 동기화해 애플리케이션을 다시 배포하거나 재구성하는 기능
- AUTO CREATE NAMESPACE: Enable

The screenshot shows the Argo UI interface for creating a new application named 'flask-example'. The 'GENERAL' tab is selected, displaying fields for 'Application Name' (flask-example) and 'Project' (default). Under 'SYNC POLICY', 'Manual' is selected. In the 'SYNC OPTIONS' section, the 'AUTO-CREATE NAMESPACE' checkbox is checked, while 'SKIP SCHEMA VALIDATION', 'PRUNE LAST', 'APPLY OUT OF SYNC ONLY', and 'REPLACE' options are unchecked. The 'PRUNE PROPAGATION POLICY' is set to 'foreground'. The 'EDIT AS YAML' button is located in the top right corner of the configuration panel.

Argo를 활용한 CD 환경 구축

Argo 앱 생성

- 매니페스트 파일의 소스의 위치를 지정
- flask-example-apps 프로젝트 링크를 전달
- main 브랜치를 사용하도록 설정
- Path는 배포할 yaml 파일이 있는 디렉토리를 지정
- 우리의 프로젝트에는 flask-example-deploy 아래에 필요한 yaml이 구성



Argo를 활용한 CD 환경 구축

Argo 앱 생성

- 쿠버네티스 API 서버에 대한 정보를 입력
- 우리가 구성한 argo는 쿠버네티스 내에 구성
- kube-apiserver에 대한 URL 정보를 도메인 주소로 입력
- 네임스페이스는 애플리케이션을 배포할 공간을 의미
- jenkins-ns 네임스페이스에 배포

The screenshot shows the 'DESTINATION' section of the Argo app creation interface. It includes fields for 'Cluster URL' (set to 'https://kubernetes.default.svc') and 'Namespace' (set to 'flask-ns'). A 'URL ▾' button is also visible.

DESTINATION	
Cluster URL	<input type="text" value="https://kubernetes.default.svc"/> URL ▾
Namespace	<input type="text" value="flask-ns"/>

Argo를 활용한 CD 환경 구축

Argo 앱 싱크

- 모든 설정이 완료되면 상단에 CREATE 버튼을 클릭
- 새로 생성된 app인 flask-example을 확인
- SYNC 버튼을 눌러서 배포 진행

The screenshot shows the Argo UI interface. On the left, there's a sidebar with icons for Applications, New App, Sync Apps, and various status filters like Sync Status and Health Status. The main area displays a list of applications, with one entry for 'flask-example' highlighted. This entry shows details such as Project: default, Label: missing, Status: OutOfSync, Repo: https://github.com/gasbugs/flask-example-apps, Target: main, Path: flask-example-deploy, Destination: in-cluster, and Namespace: flask-ns. Below this entry are three buttons: SYNC (highlighted in blue), CANCEL, and O. To the right of the application list is a large modal window titled 'SYNCHRONIZE'. The modal contains the following information:

- Synchronizing application manifests from <https://github.com/gasbugs/flask-example-apps>
- Revision: main
- Sync Options:
 - PRUNE
 - DRY RUN
 - APPLY ONLY
 - FORCE
- Sync Options (checkboxes):
 - SKIP SCHEMA VALIDATION
 - AUTO-CREATE NAMESPACE
 - PRUNE LAST
 - APPLY OUT OF SYNC ONLY
- Prune Propagation Policy: foreground
- Sync Resources:
 - /SERVICE/FLASK-NS/FLASK
 - APPS/DEPLOYMENT/FLASK-NS/FLASK

Argo를 활용한 CD 환경 구축

▶ 헬름차트를 활용한 배포

- 앞서 구성한 방법과 동일하게 애플리케이션 이름과 몇 정보를 입력

The screenshot shows a configuration interface for a Helm chart. At the top right is a button labeled "EDIT AS YAML".

GENERAL

Application Name: mychart

Project: default

SYNC POLICY

Manual

SYNC OPTIONS

SKIP SCHEMA VALIDATION AUTO-CREATE NAMESPACE
 PRUNE LAST APPLY OUT OF SYNC ONLY

PRUNE PROPAGATION POLICY: foreground

REPLACE ⚠

Argo를 활용한 CD 환경 구축

▶ 헬름차트를 활용한 배포

- Github의 helm-charts 레파지토리에서 index.yaml 파일의 링크 확인

The screenshot shows a GitHub repository page for `gasbugs/helm-charts`. The main navigation bar includes `Code`, `Issues`, `Pull requests`, `Actions`, `Projects`, `Wiki`, `Security`, `Insights`, and `Settings`. Below the navigation, it shows the `main` branch, the `helm-charts / stable / index.yaml` file, and a commit message `o4a upload new packages` from 29 days ago.

In the center, there's a `SOURCE` section with fields for `Repository URL` (`https://github.com/gasbugs/helm-charts/raw/main/stable/`) and `Chart` (`mychart2`). To the right, there's a `HELM` dropdown set to `0.1.0`.

On the right side, a context menu is open over the file content area. The menu items include `Raw` (highlighted with a red box), `새 탭에서 링크 열기`, `새 창에서 링크 열기`, `InPrivate 창에서 링크 열기`, `장치에 링크 보내기`, `(으)로 링크 저장`, `링크 복사` (highlighted with a red box), `컬렉션에 추가`, `공유`, and `웹 캡처`.

```
13     version: 0.1.0
14   mychart2:
15     - apiVersion: v2
```

Argo를 활용한 CD 환경 구축

▶ 헬름차트를 활용한 배포

- 앞서 작성한 것과 동일하게 쿠버네티스 API의 도메인과 배포할 네임스페이스 정보 입력

The screenshot shows the Argo UI interface for deploying Helm charts to a Kubernetes cluster. The left side displays the 'DESTINATION' configuration, including the Cluster URL (`https://kubernetes.default.svc`) and Namespace (`mychart`). The right side shows two Helm chart configurations: `mychart` and `flask-example`. Each configuration has its details listed, such as Project, Labels, Status, Repository, Target, Chart, Destination, and Namespace. The `mychart` configuration has a red box around the 'SYNC' button, indicating it is the active or selected chart for deployment.

Chart	Project	Labels	Status	Repository	Target	Chart	Destination	Namespace
mychart	default		Missing OutOfSync	https://github.com/gasbugs/...	0.1.0	mychart2	in-cluster	mychart
flask-example	default		Healthy Synced	https://github.com/gasbugs/...	main	flask-example-deploy	in-cluster	flask-ns

쿠버네티스 마이크로서비스아키텍처 프로젝트

▶ MSA 아키텍처 소개

▶ flask를 활용한 웹페이지 렌더링

▶ RestfulAPI 설계하기

▶ flask를 활용한 RestfulAPI 구성



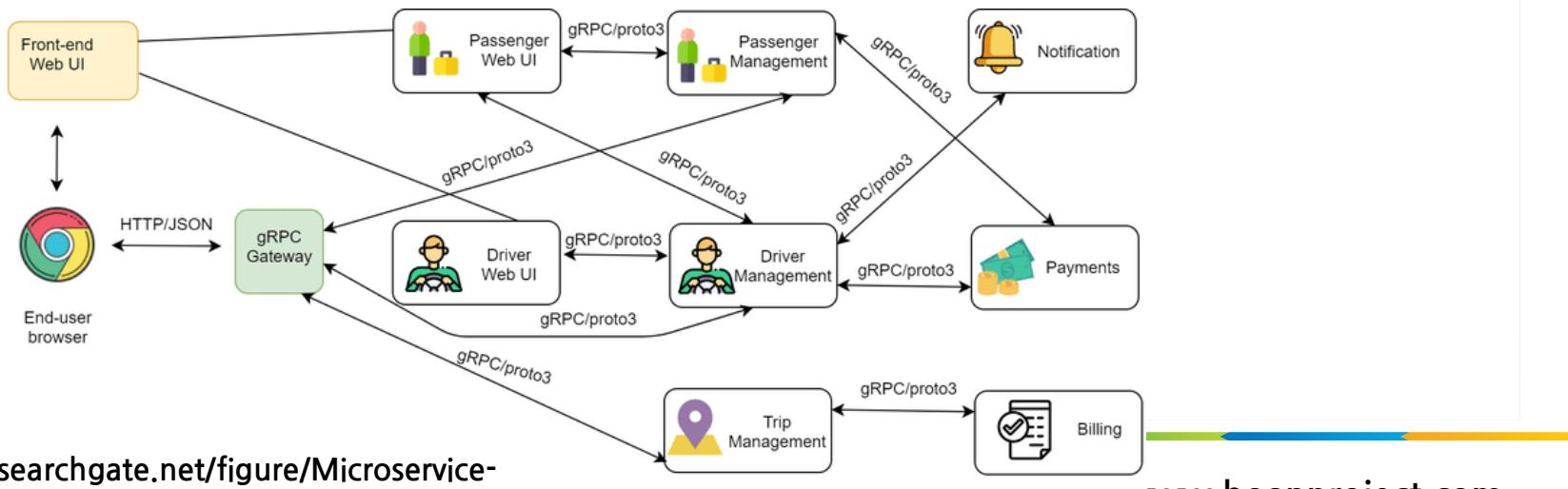
MSA 아키텍처 소개

MSA 아키텍처 소개

■ MSA의 특징

출처: <https://waspro.tistory.com/429>

- ① 애플리케이션 로직을 각자 책임이 명확한 작은 컴포넌트들로 분해하고 이들을 조합해서 솔루션 제공
- ② 각 컴포넌트는 작은 책임 영역을 담당하고 완전히 상호 독립적으로 배포
마이크로서비스는 비즈니스 영역의 한 부분에서만 책임을 담당
여러 애플리케이션에서 재사용할 수 있어야 함
- ③ 마이크로서비스는 몇가지 기본 원칙에 기반
소비자와 서비스 제공자 사이의 데이터 교환을 위해 HTTP와 JSON 같은 경량 통신 프로토콜 사용
- ④ 애플리케이션은 항상 기술 중립적 프로토콜을 사용해 통신하므로 서비스 구현 기술과는 무관
마이크로서비스 기반의 애플리케이션을 다양한 언어와 기술로 구축 가능
- ⑤ 작고 독립적이며 분산된 마이크로서비스를 사용해 조직
명확히 정의된 책임 영역을 담당하는 소규모 팀을 보유
이 팀들은 애플리케이션 출시처럼 하나의 목표를 향해 일함//자기가 개발하는 서비스만 책임



MSA 아키텍처 소개

■ MSA의 목적

- 마이크로서비스를 달성하기 위해서는 많은 노력과 비용이 수반
- 따라서 MSA를 적용하고자 할 경우 명확한 목적을 갖고 고려해야 함
- 1) 마이크로서비스 아키텍처를 통해 달성하고자 하는 목표는 무엇인가?
 - 명백한 목표 예를 들어 기존 대비 1.5배 빠른 성능, 이벤트에도 중단되지 않는 가용성 높은 서비스 등 결과를 기준으로 설명할 수 있어야 하며, 시스템의 end-user에게 이점을 제공하는 방식으로 설명될 수 있어야 한다.
- 2) 마이크로 서비스 사용에 대한 대안을 고려했습니까?
 - 마이크로서비스가 제공하는 것과 동일한 이점을 얻을 수 있는 다른 많은 방법이 있다. 때로는 MSA를 고려하지 않고도 적용 가능한 방법이 다양하게 존재하므로, 이에 대한 고려를 선행한 후 MSA를 선택하는 것이 좋다.

출처: <https://waspro.tistory.com/429>

MSA 아키텍처 소개

■ MSA의 장점

출처: <https://waspro.tistory.com/429>

- ① 작은 서비스들로 나누고, 각 서비스를 독립적으로 만듦 → loosely-coupled(약결합)
- ② 대용량 분산 환경에 적합
- ③ 복잡도 감소
- ④ 유연한 배포
- ⑤ 재사용성 → 확장성
- ⑥ 서비스별 hw/sw 플랫폼/기술의 도입 및 확장이 자유로움
- ⑦ 개발자가 이해하기 쉽고 개발/운영 매 단계의 생산성이 높음
- ⑧ 지속적인 개발/디플로이가 biz capability 단위로 관련된 소수의 인원의 책임하에 이뤄짐
- ⑨ Fault isolation (장애 허용 시스템) 특성이 좋음

MSA 아키텍처 소개

■ MSA의 단점

출처: <https://waspro.tistory.com/429>

단점	보완방법
장애추적, 모니터링, 메시징 처리가 어렵다.	Sleuth 등과 ELK, EFK, Splunk 등의 서비스를 연동하여 사용하는 방안 고려
여러 서비스에 걸쳐져 있는 Feature의 경우, 트랜잭션을 다루기 어렵다.	보상 트랜잭션 또는 부분적으로 composite 서비스로의 병합 고려
여러 서비스에 걸쳐져 있는 Feature의 경우, 테스팅이 복잡하다.	테스팅 계획 및 방법에 노력 투자
서비스 간 Dependency가 있는 경우 릴리즈가 까다롭다.	관련 개발조직 간 roll-out 계획 마련 및 dependency의 관리체계 수립
서비스 개수가 많고 유동적이기 때문에 CI/CD 및 서비스 관리 상의 문제가 발생할 수 있다.	서비스 레지스트리, 모니터링, 개발/디플로이 자동화 기술 고려 (PaaS 고려)
모놀리스 시스템을 마이크로서비스 전환할 때 큰 고동이 수반될 수 있다.	B2C 웹 또는 SaaS 어플리케이션은 처음부터 마이크로서비스 및 서비스 별 조직을 고려하여 구성

flask를 활용한 웹페이지 렌더링

flask를 활용한 웹페이지 렌더링

Flask?

- 파이썬의 대표적인 웹 개발 프레임워크 중 하나
- 마이크로 프레임워크, 가볍고 간단
- 지정한 라이브러리와 패키지만 설치됨 => 효율성, 자유도가 높음



공식 홈페이지

<https://flask.palletsprojects.com/en/1.1.x/>

“Micro” does not mean that your whole
web application has to fit into a single
Python file

,,

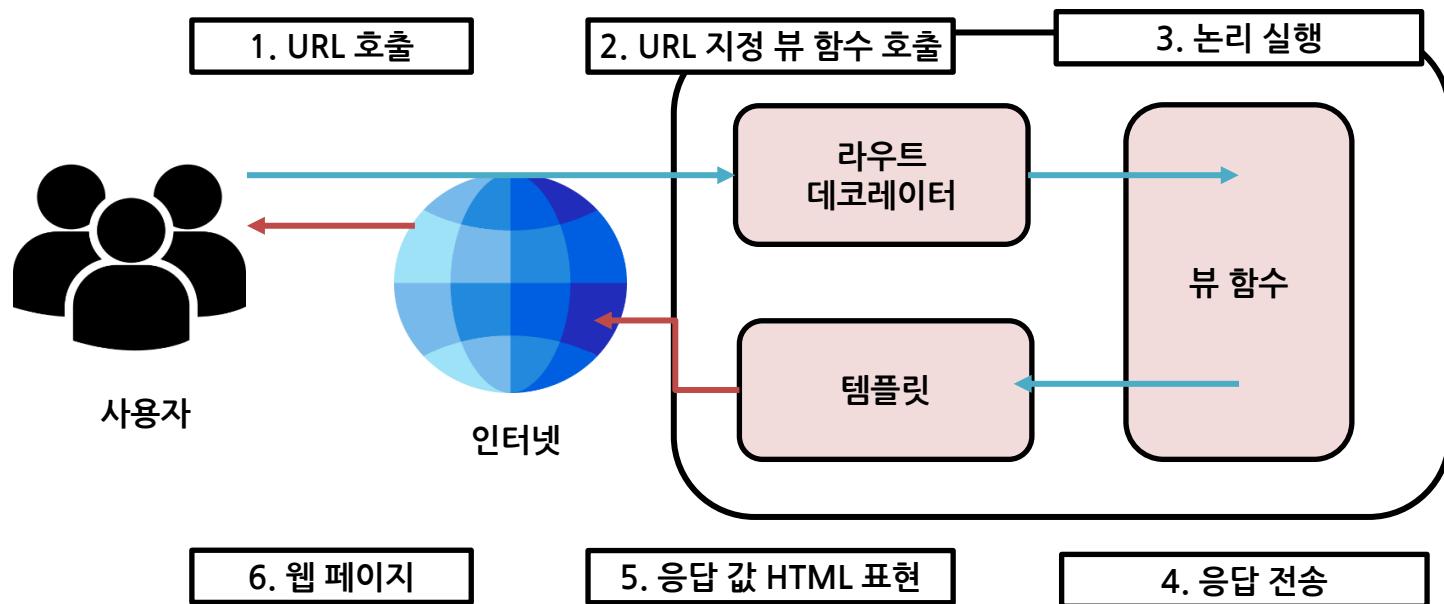


flask를 활용한 웹페이지 렌더링

▶ 플라스크 앱 구조

- 플라스크 앱은 다음 과정을 통해 호출

1. 특정 URL 호출
2. 뷰 함수 호출
3. 논리 실행
4. 논리 결과 응답 전송
5. 응답 값 HTML 표현
6. 클라이언트 전달



flask를 활용한 웹페이지 렌더링

▶ 플라스크 시작하기

● 파일웹 마이크로 프레임워크, 백엔드 서버 기능

- WSGI : Web Server Gateway Interface, CGI의 업그레이드형 (성능 업!)
- Werkzeug + Jinja2 템플릿 엔진 구성
- 기본 5000번 포트 사용
- 웹 페이지 문자열 출력 예제

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "<h1>Hello World!</h1>

if __name__ == "__main__":
    app.run()
```



사용하기 쉬운 REST API

사용하기 쉬운 REST API

▶ REST API 기본 개념

- Representational State Transfer의 약자로 자원을 이름으로 구분하여 주고 받는 모든 것을 의미
 - HTTP URL로 자원을 명시하고 HTTP Method로 자원의 CRUD 오퍼레이션을 적용
-
- 장점
 - HTTP 표준 프로토콜에 따르는 모든 프로토콜에서 사용 가능하며 디자인 문제 최소화, 서버와 클라이언트 역할 분리 등이 있음
 - 단점
 - 구형 브라우저가 모든 메서드를 지원하지는 않아 사용할 수 있는 메서드가 제한적임
 - 특징
 1. 서버와 클라이언트 구조
 2. 무상태, 클라이언트 context를 서버에 저장하지 않아 구현이 단순하며 클라이언트 요청만 단순 처리
 3. 캐시 처리가 가능
 4. 계층화, 클라이언트는 REST API 서버만 호출하며 다중 계층으로 구성

사용하기 쉬운 REST API

▶ REST API 설계 기본 규칙

● REST API는 도큐먼트, 컬렉션, 스토어로 구성

- 도큐먼트 : 데이터베이스 레코드,
- 컬렉션 : 서버에서 관리하는 리소스, 디렉터리
- 스토어 : 클라이언트에서 관리하는 리소스 저장소

● REST API 설계 규칙

1. 슬래시 구분자(/)는 계층 관계를 나타내는데 사용한다.
2. URI 마지막 문자로 슬래시(/)를 포함하지 않으며 URI 경로의 마지막에는 슬래시(/)를 사용하지 않는다.
3. 하이픈(-)은 URI 가독성을 높이는데 사용한다.
4. 언더바(_)는 URI에 사용하지 않는다.
5. URI 경로에는 소문자가 적합하다.
6. 파일확장자는 URI에 포함하지 않는다.
7. 리소스 간에는 연관 관계가 있는 경우 /리소스명/리소스 ID/관계가 있는 다른 리소스명으로 표기한다.

사용하기 쉬운 REST API

▶ REST API 잘 설계하기

- 처음부터 설계를 잘해야 함
- 팀 멤버와 유저들도 제대로 활용할 수 있음
- API는 기본적인 CRUD를 활용
- 명확한 패턴이 필요
- URL에서 동사를 사용하지 않아야 함(대신에 HTTP request method를 사용)
- 자동차 판매점 API 예제
 - 자동차를 추가
 - 자동차를 검색
 - 자동차 엔진이나 옵션, 소개글 등 조회

잘못된 설계

/createCar
/seeCars
/getCar/Benz
/getEngines/Benz/e-class
/deleteCar/Benz
/updateCar/Benz
/getTopRatedCars
/findCarsFromThisYear

제대로된 설계

POST /cars
GET /cars
GET /cars/benz
GET /cars/benz/e-class
DELETE /cars/benz
PUT /cars/benz
GET /cars?min_rating=9.8
GET /cars?release_date=2021

압축

GET
POST
PUT
DELETE

/cars
/cars/benz
/cars/benz/e-class

flask를 활용한 REST API 구성

사용하기 쉬운 REST API

▶ REST API 잘 설계하기

GET /cars
POST /cars/benz
PUT /cars/benz/e-class
DELETE /cars

```
from flask_restful import Resource
from flask_restful import reqparse

class Minus(Resource):
    def get(self):
        try:
            parser = reqparse.RequestParser()
            parser.add_argument('x', required=True, type=int, help='x cannot be blank')
            parser.add_argument('y', required=True, type=int, help='y cannot be blank')
            args = parser.parse_args()
            result = args['x'] - args['y']
            return {'result': result}
        except Exception as e:
            return {'error': str(e)}

from flask import Flask
from flask_restful import Api

app = Flask('My First App')
api = Api(app)
api.add_resource(Minus, '/minus')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000, debug=True)
```



Thank You !