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

www.boanproject.com

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.

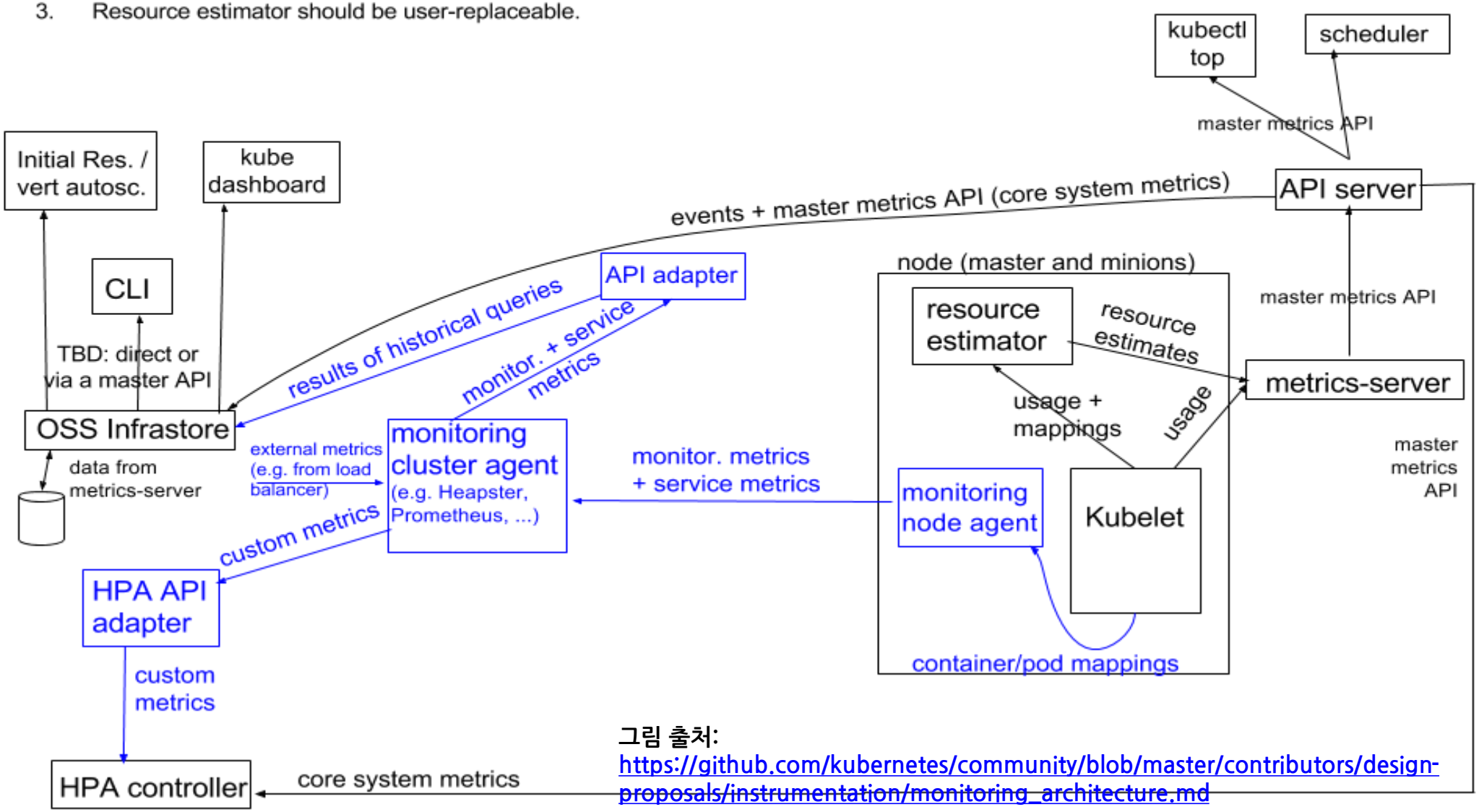


그림 출처:
https://github.com/kubernetes/community/blob/master/contributors/design-proposals/instrumentation/monitoring_architecture.md

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

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〉
```

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%
```

연습문제

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

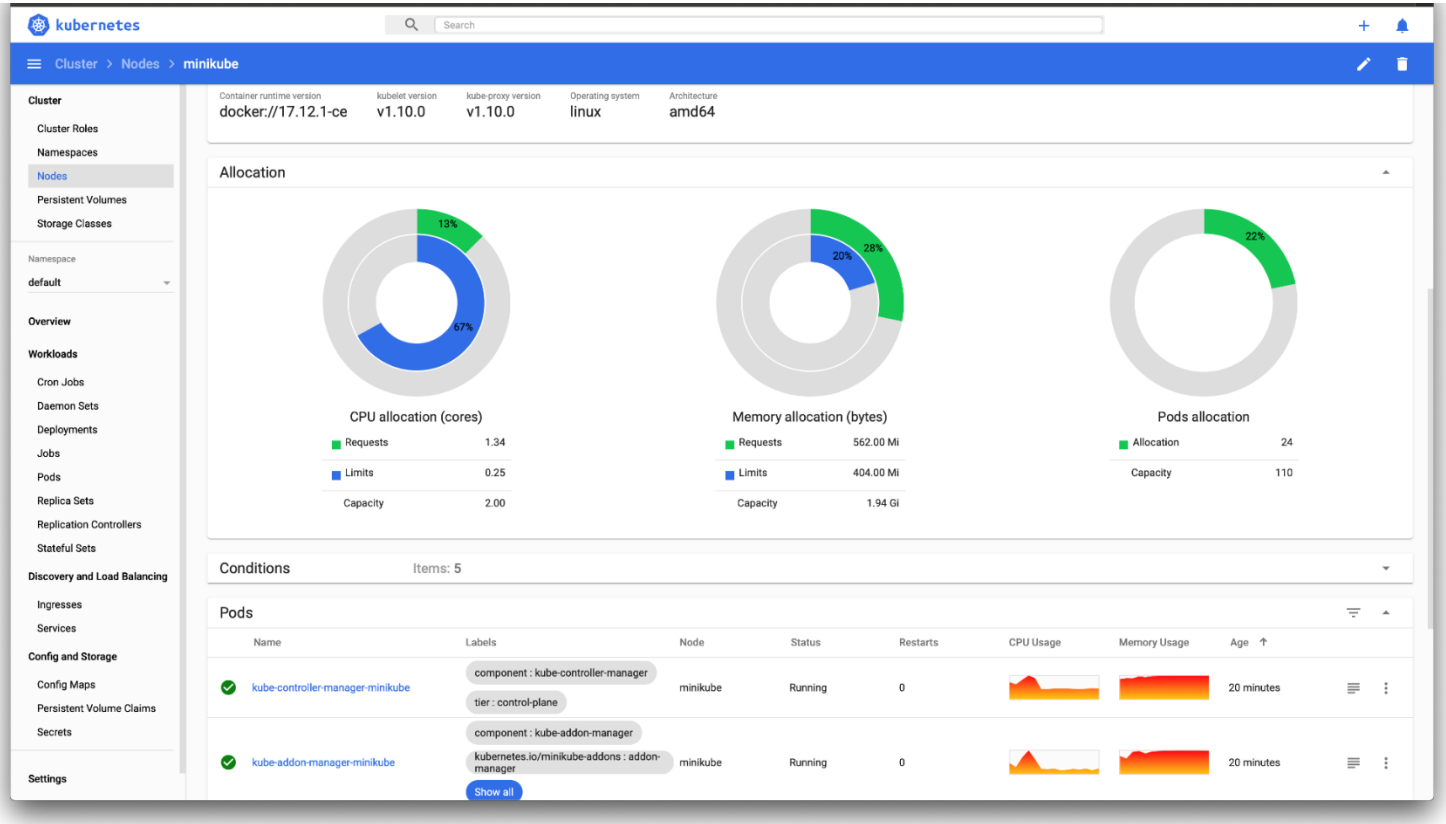
큐브 대시보드 설치와 사용

큐브 대시보드 설치와 사용

Kubernetes Dashboard

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

thisisunsafe



www.boanproject.com

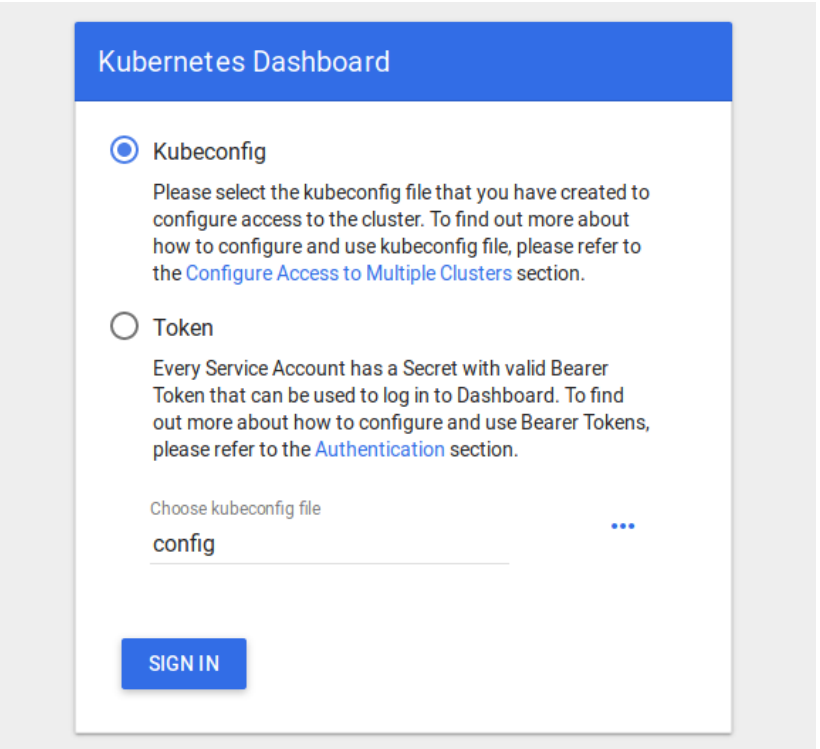
큐브 대시보드 설치와 사용

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



www.boanproject.com

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.eyJpc3MiOiJ
rdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZX
JuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY
2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8v
c2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1
pbi11c2VyLXRva2VuLTVrMjlrIiwia3ViZXJuZXRlcy
5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291b
nQubmFtZSI6ImFkbWluLXVzZXIiLCJrdWJlcm5ldGVz
LmlvL3NlcnZpY2VhY2NvdW50…
```

admin-user.yaml

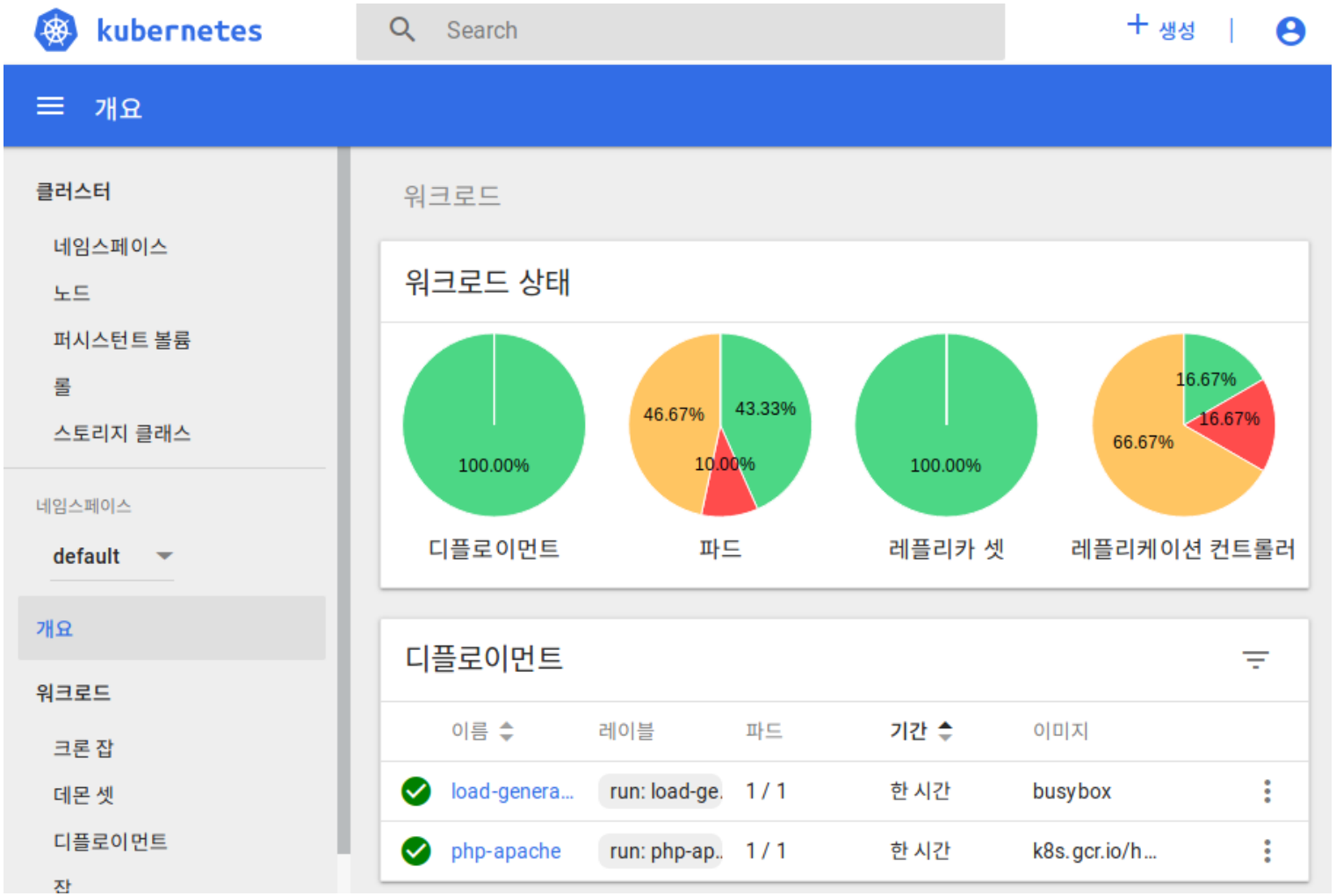
```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
```

변경 필요!

```yaml
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



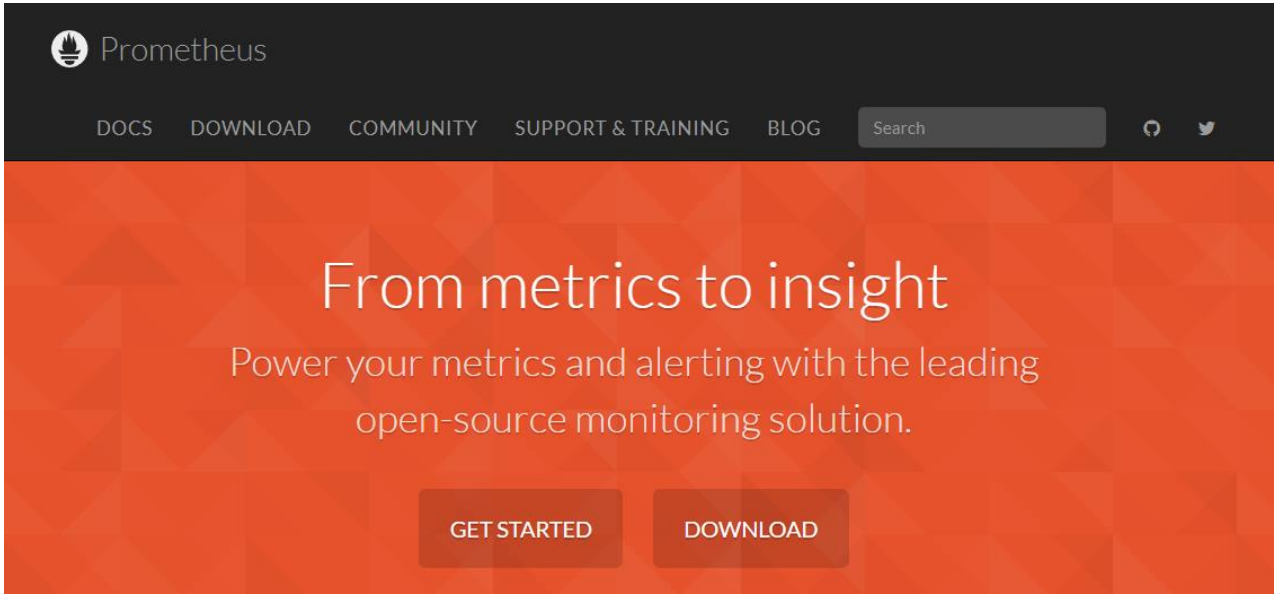
www.boanproject.com

프로메테우스 그라파나를 활용한 리소스 모니터링

프로메테우스 그라파나를 활용한 리소스 모니터링

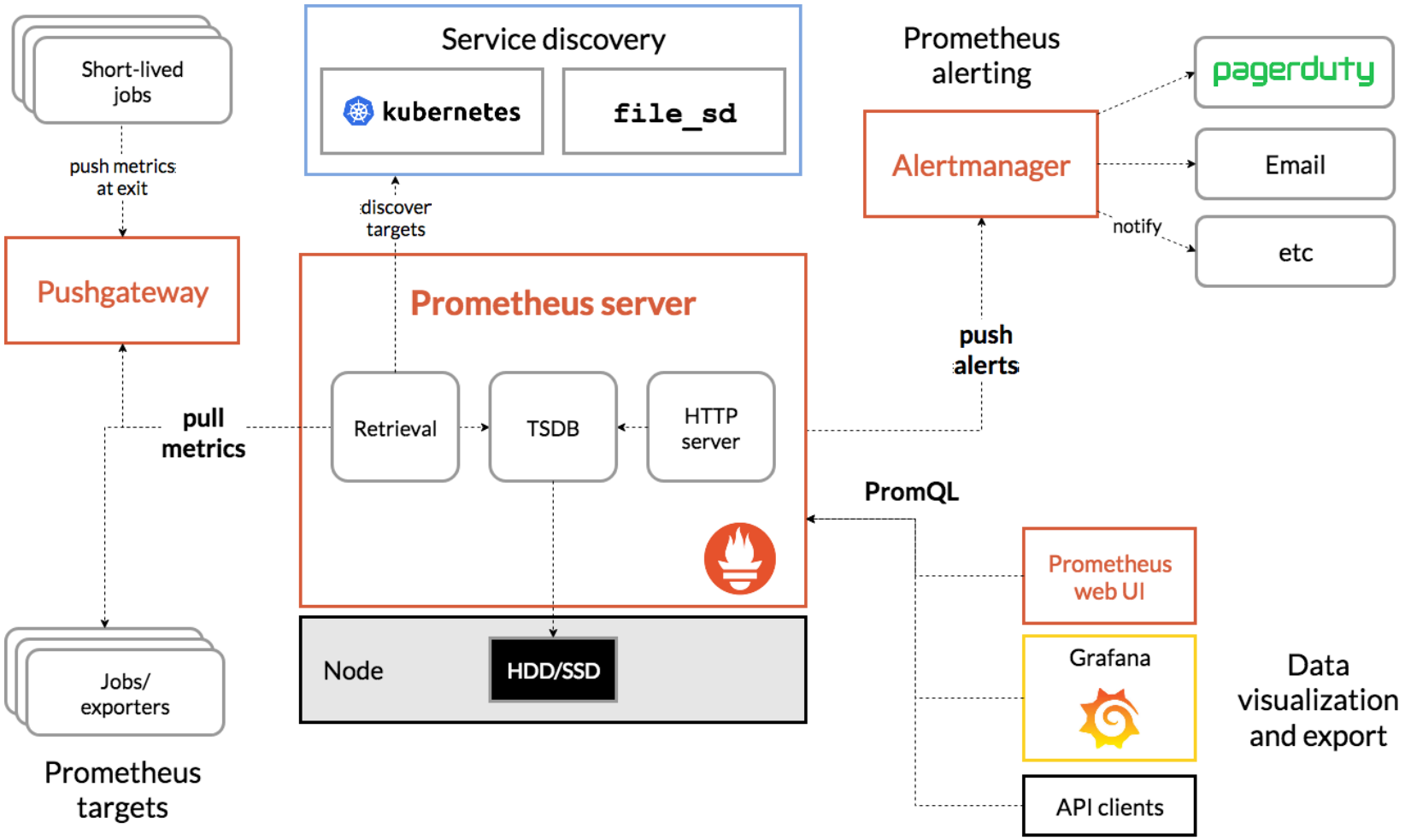
프로메테우스란?

- 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

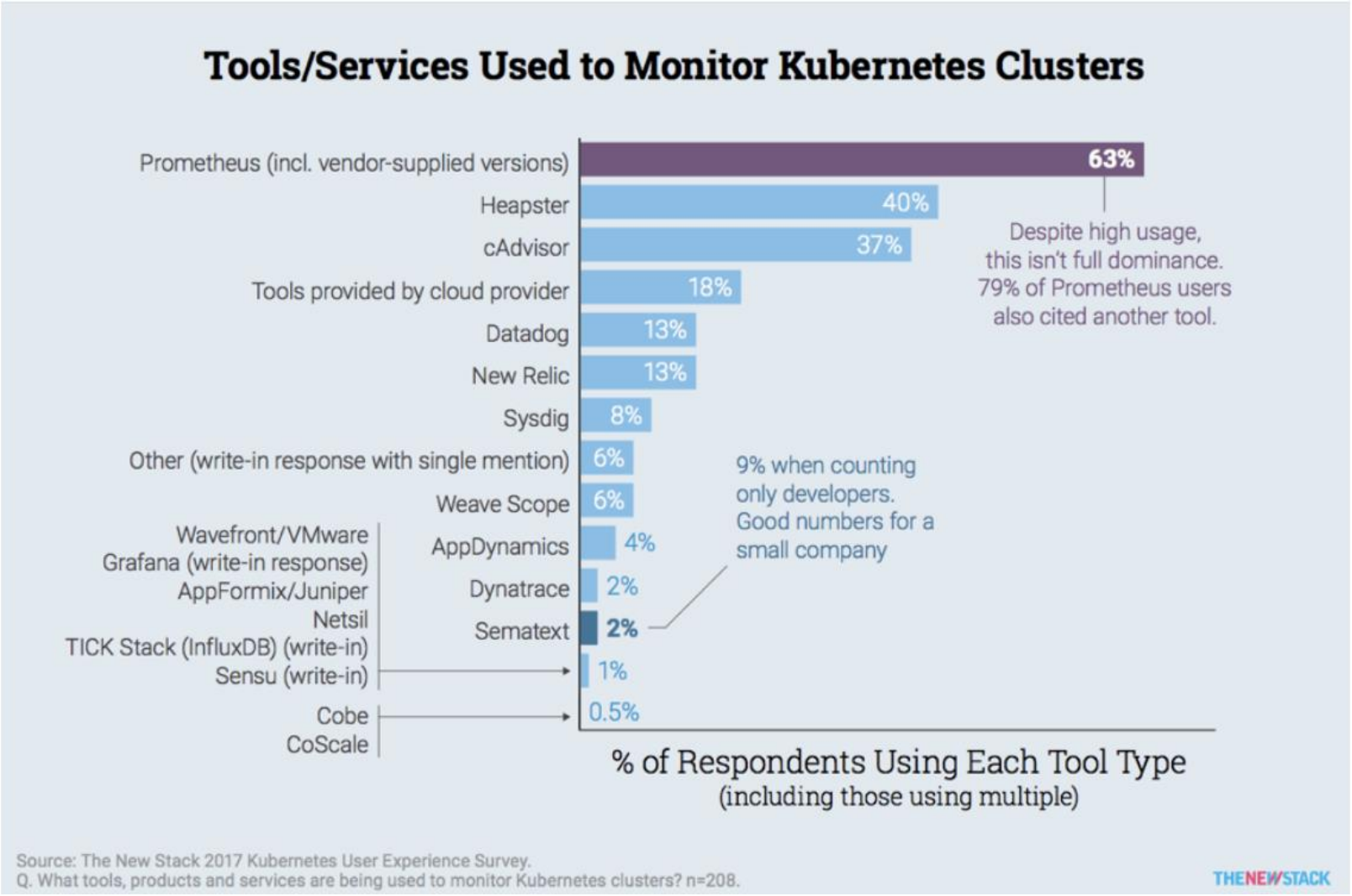
프로메테우스 아키텍처 예제



www.boanproject.com

프로메테우스 그라파나를 활용한 리소스 모니터링

쿠버네티스 클러스터 모니터링 툴과 서비스 사용 현황



www.boanproject.com

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
```


www.boanproject.com

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
```


www.boanproject.com

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

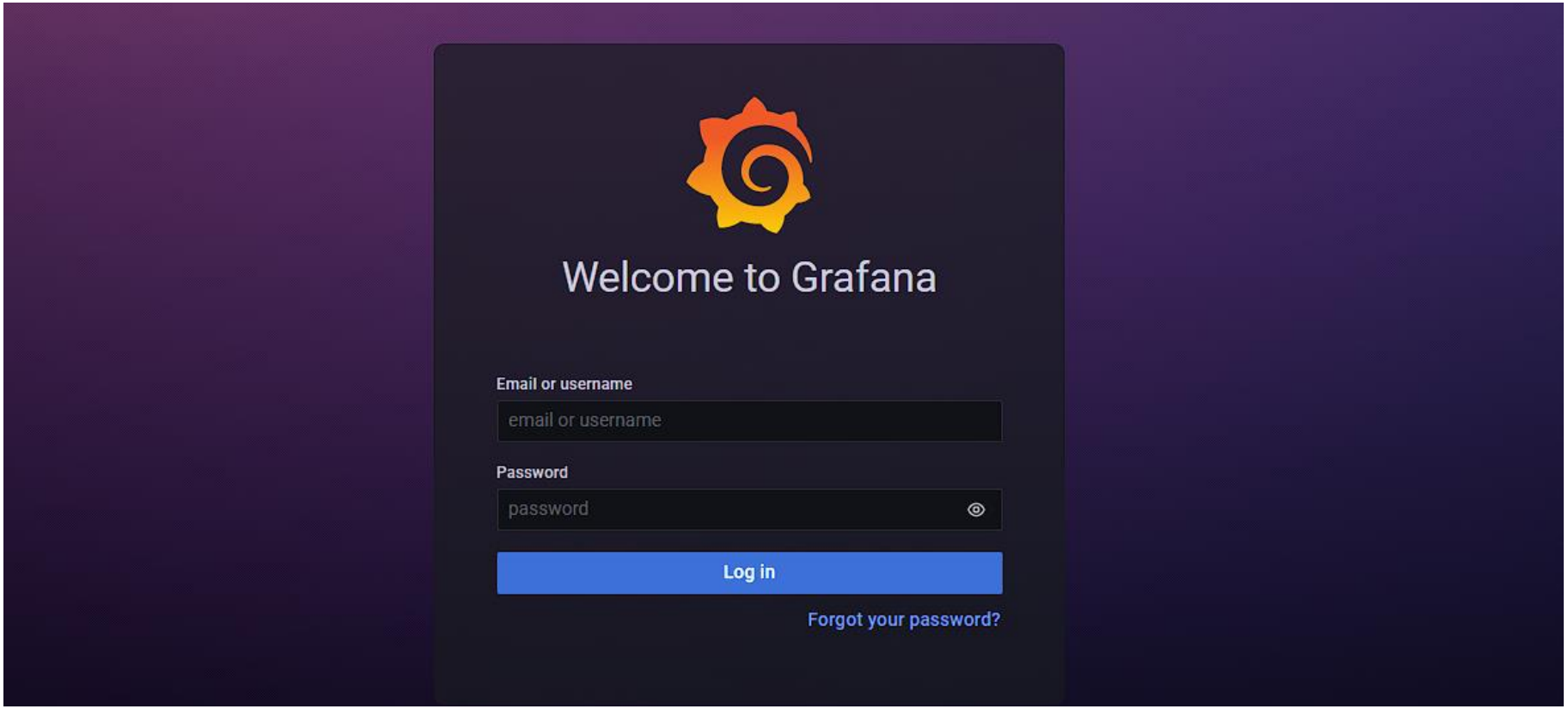
```
$ kubectl get pod,svc -n prometheus
NAME                                             READY   STATUS    RESTARTS   AGE
pod/grafana-b64fbdcb-qdxnm                       1/1     Running   0          49s
pod/prometheus-alertmanager-6755b9794f-thq5l     1/2     Running   0          57s
pod/prometheus-kube-state-metrics-696cf79768-xkjlk   1/1     Running   0          58s
pod/prometheus-node-exporter-bxvbv               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     POR
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
```


www.boanproject.com

프로메테우스 그라파나를 활용한 리소스 모니터링

그라파나 접속하기

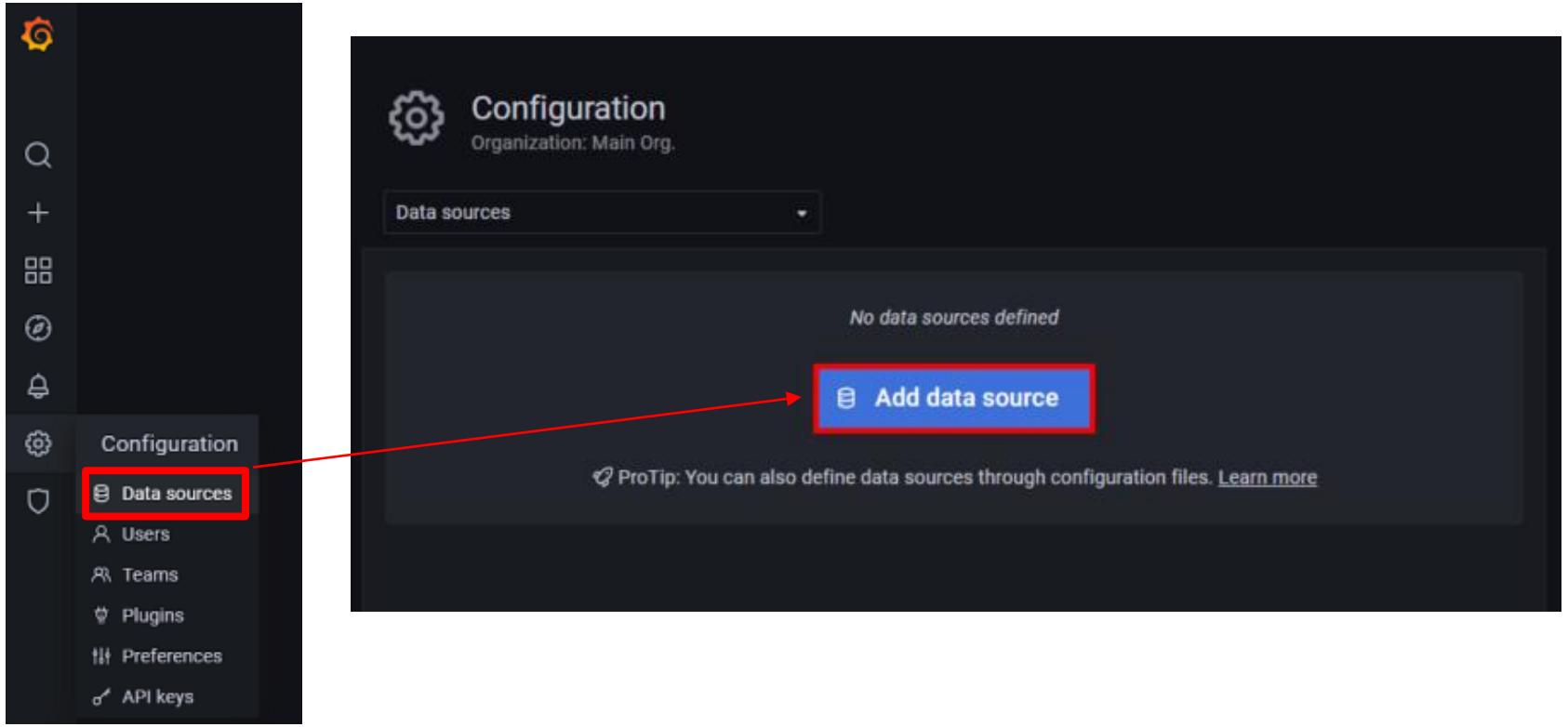
- 외부 IP로 노출된 grafana의 IP로 접속을 수행
- 아이디와 패스워드는 admin//test1234!234



프로메테우스 그라파나를 활용한 리소스 모니터링

프로메테우스 데이터 가져오기

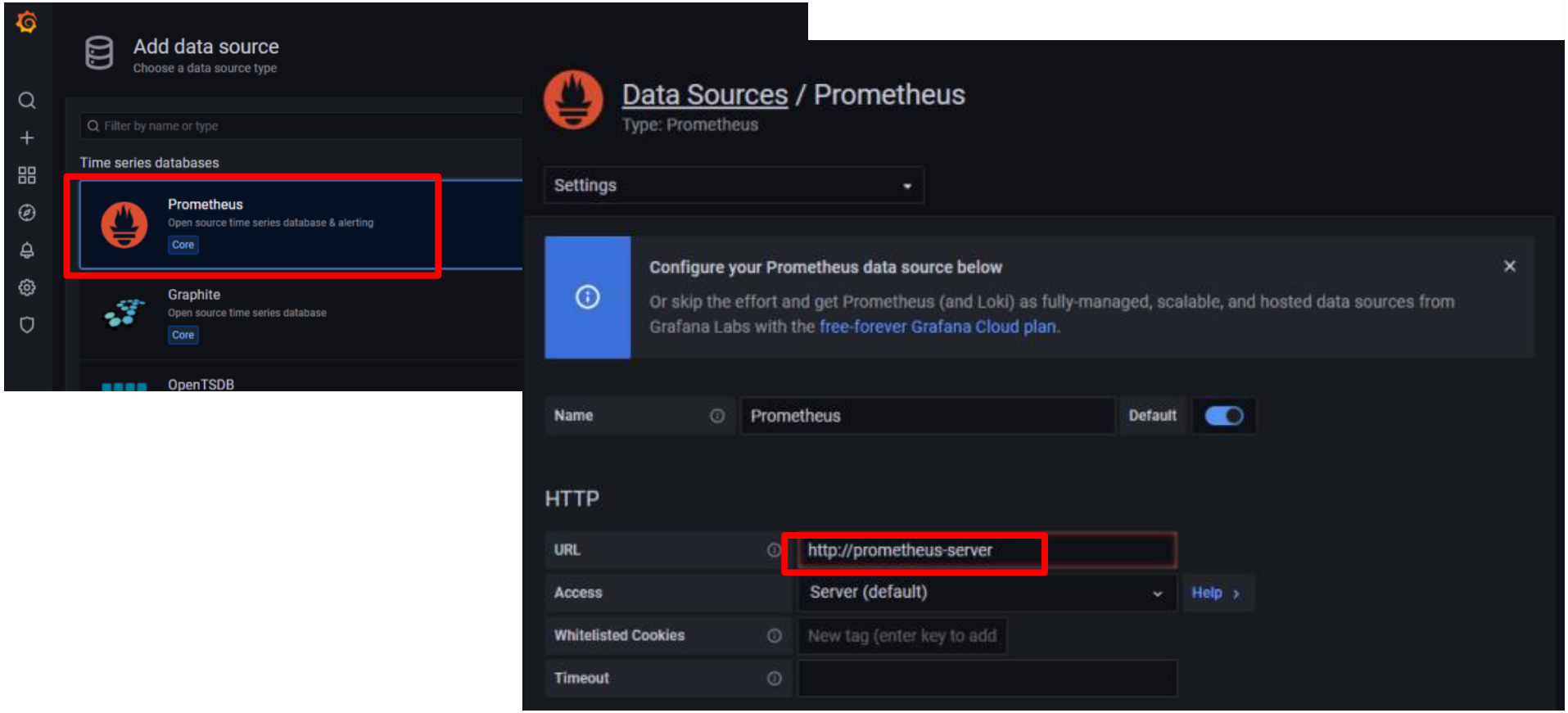
- 왼쪽 메뉴에서 Configuration - Data Sources 메뉴에서 데이터 소스 추가



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

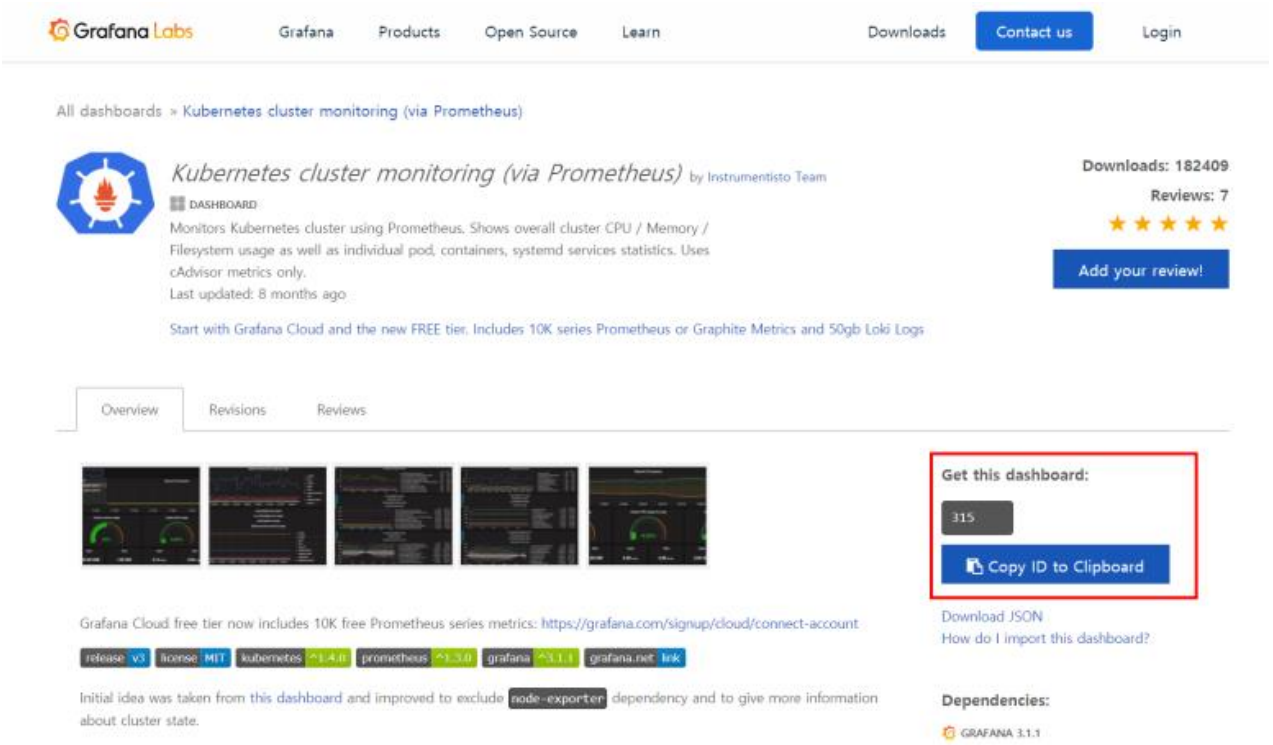


www.boanproject.com

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

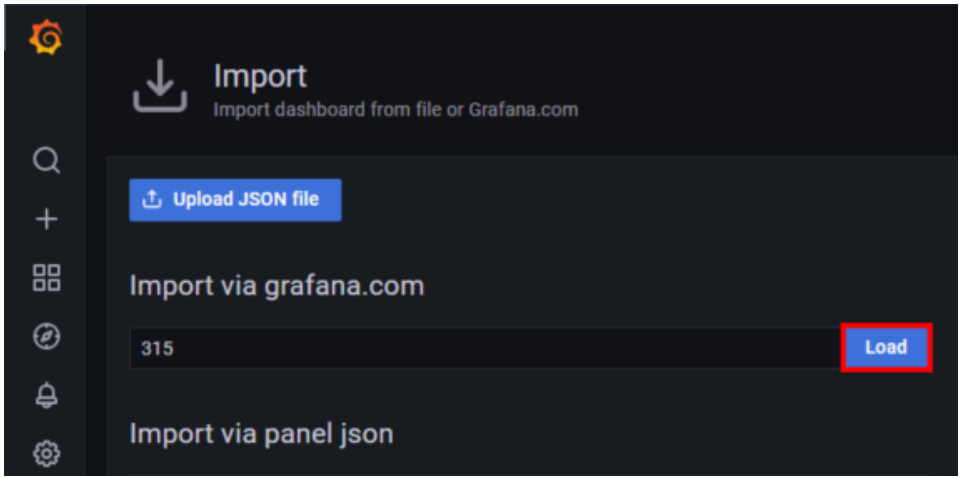


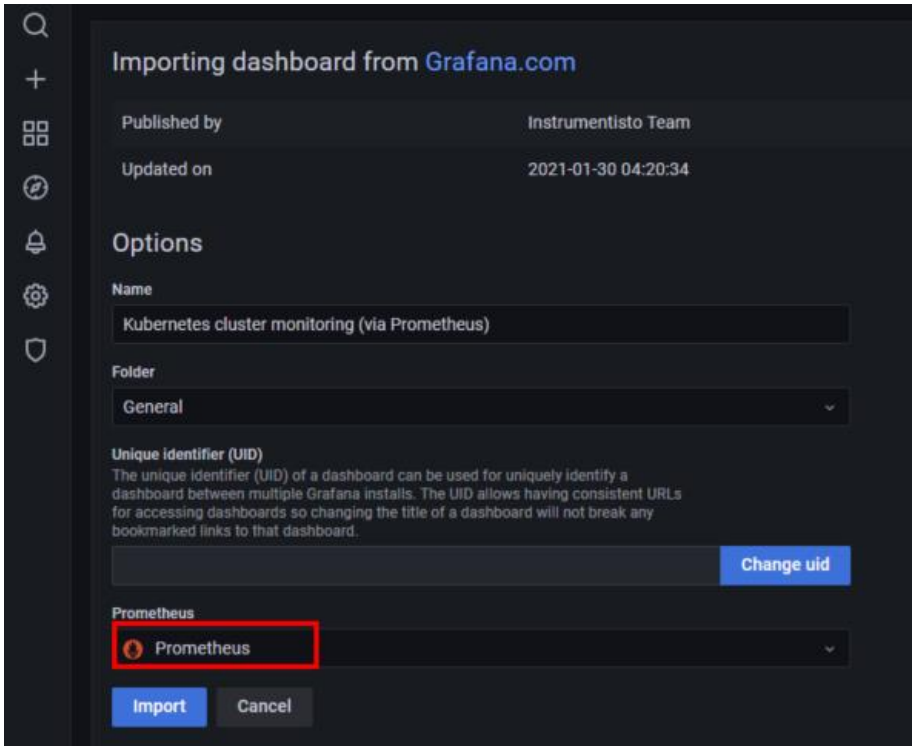
www.boanproject.com

프로메테우스 그라파나를 활용한 리소스 모니터링

대시보드 구성하기

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

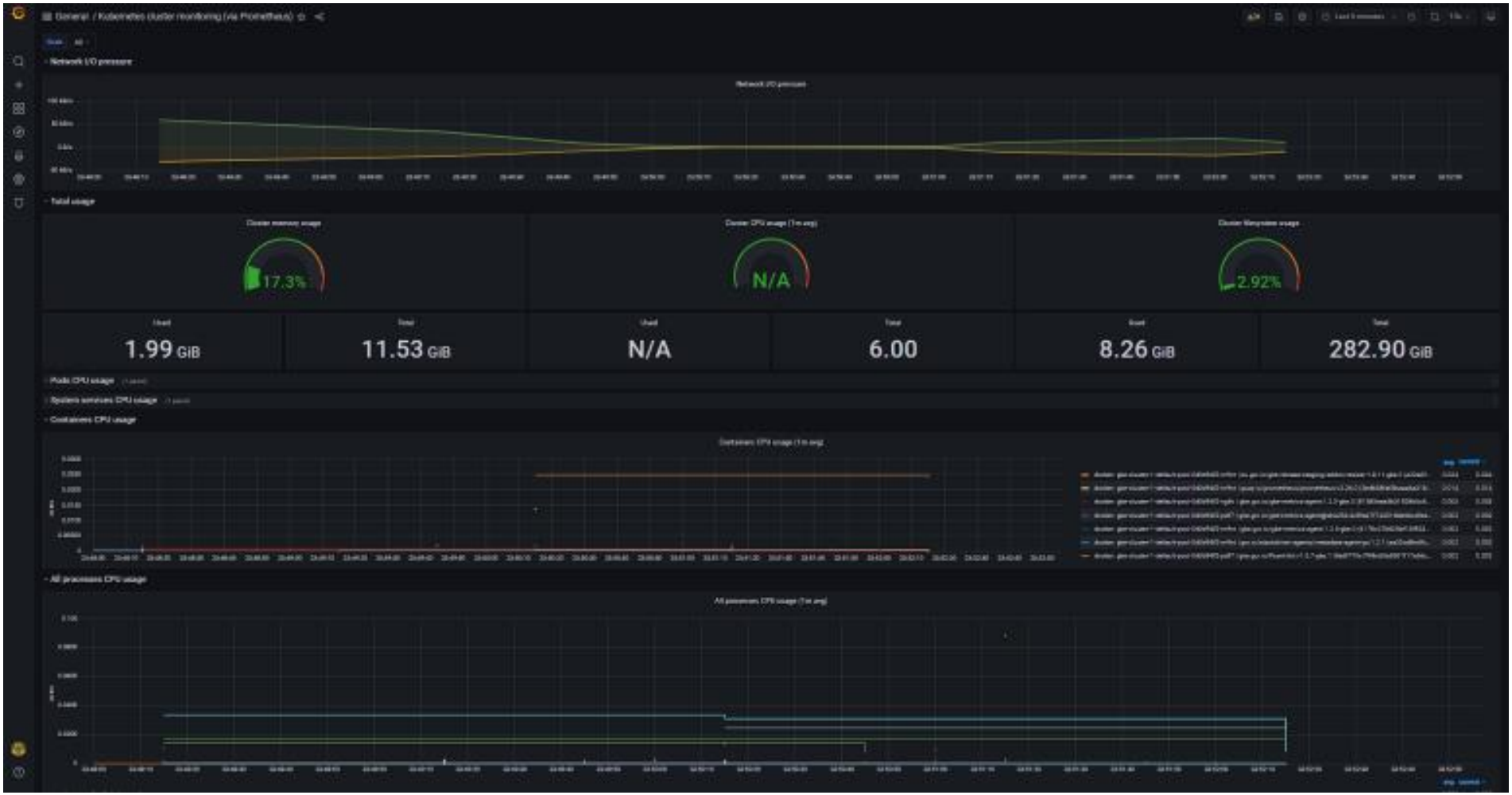




www.boanproject.com

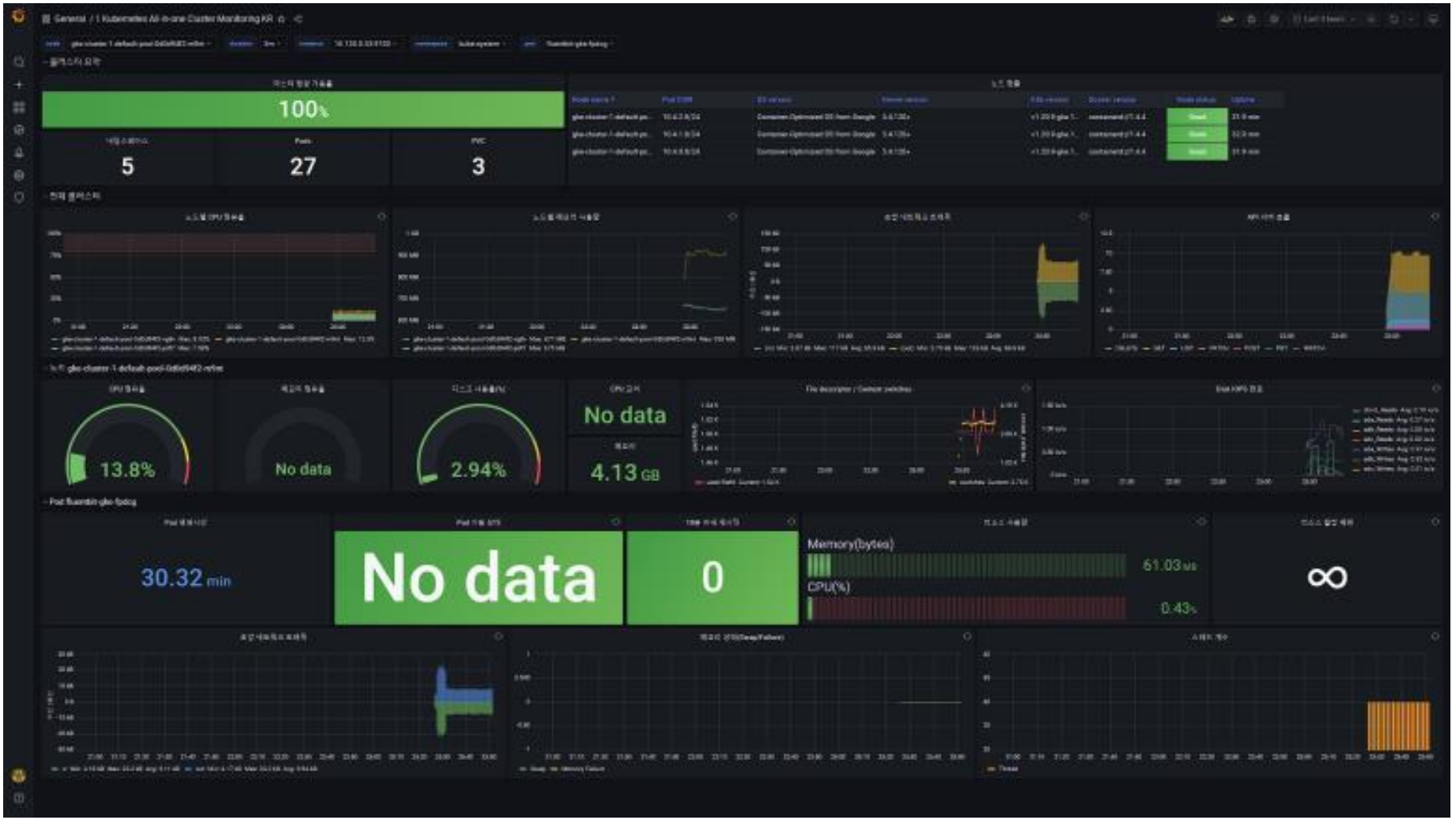
프로메테우스 그라파나를 활용한 리소스 모니터링

315번 대시보드



프로메테우스 그라파나를 활용한 리소스 모니터링

13770번 대시보드

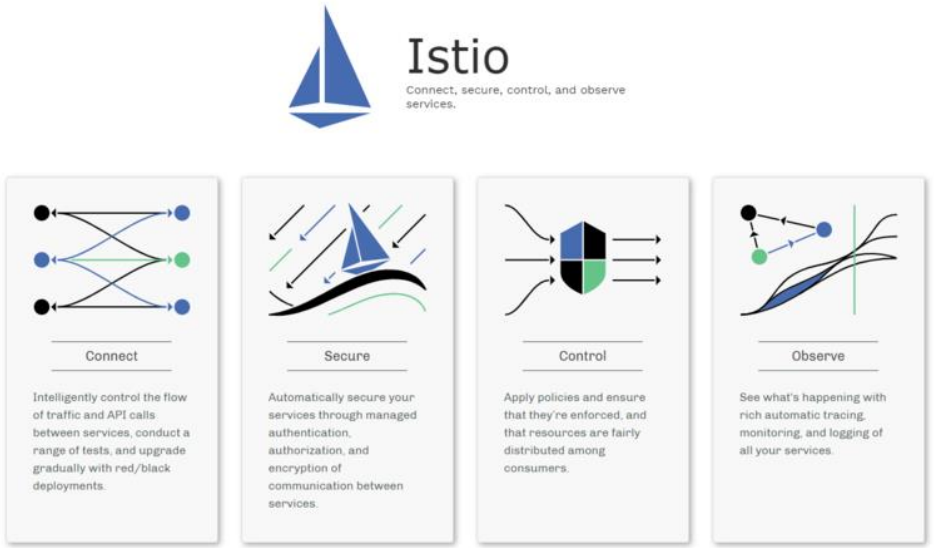


Istio를 활용한 네트워크 메시 모니터링

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



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	✓	✓	✓			✓

쿠버네티스에 이스티오 구성하기

- istioctl을 사용해 데모 버전으로 설치

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

- 네임스페이스 레이블을 이스티오 인젝션이 수행되도록 수정

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

Istio를 활용한 네트워크 메시 모니터링

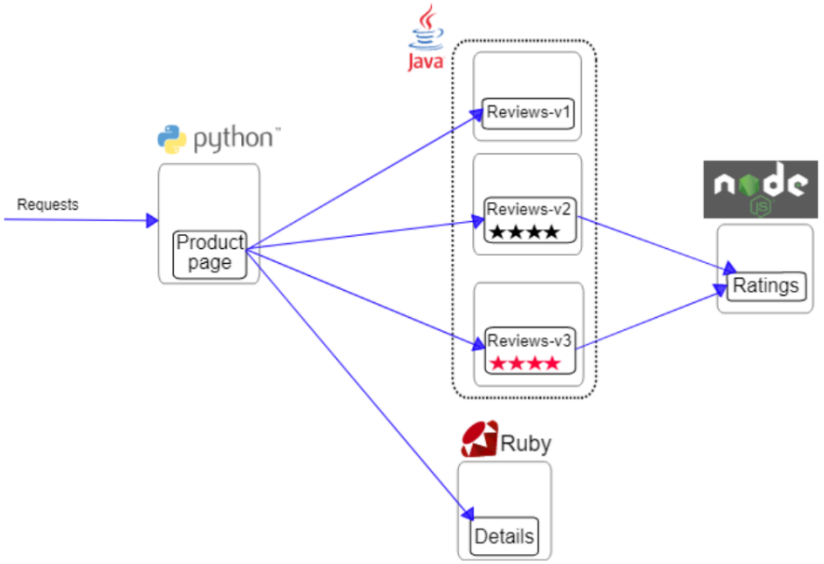
애플리케이션 배포하기

● 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
```



www.boanproject.com

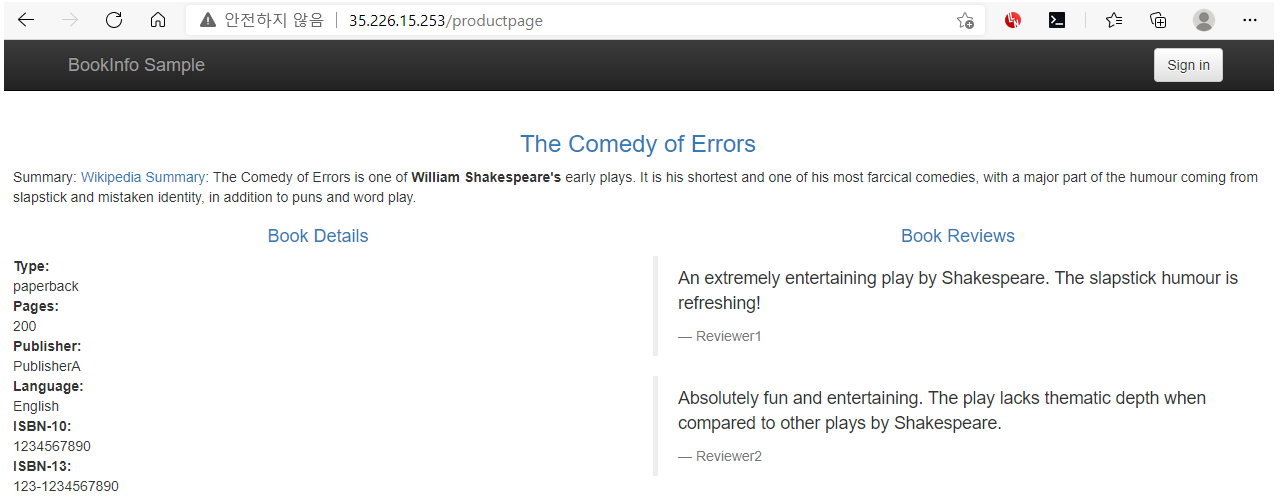
Istio를 활용한 네트워크 메시 모니터링

애플리케이션 배포하기

- 게이트를 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 productpage라는 곳으로 접속



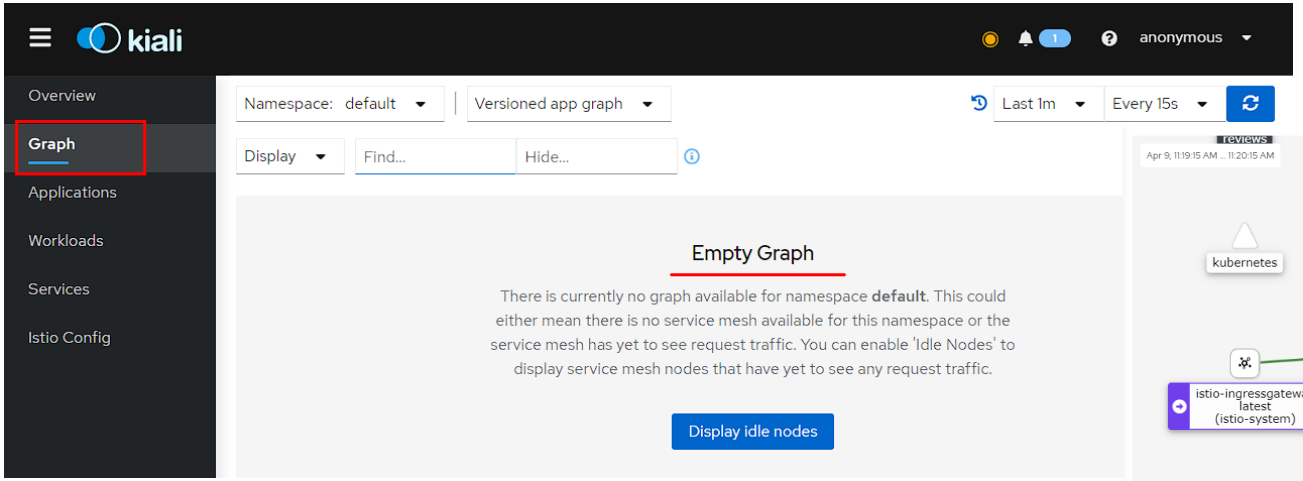
Istio를 활용한 네트워크 메시 모니터링

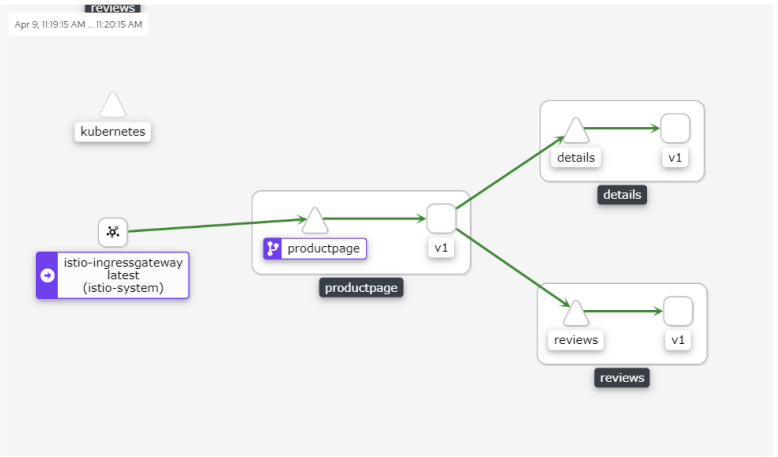
Kiali와 프로메테우스 구성

- 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





www.boanproject.com

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)

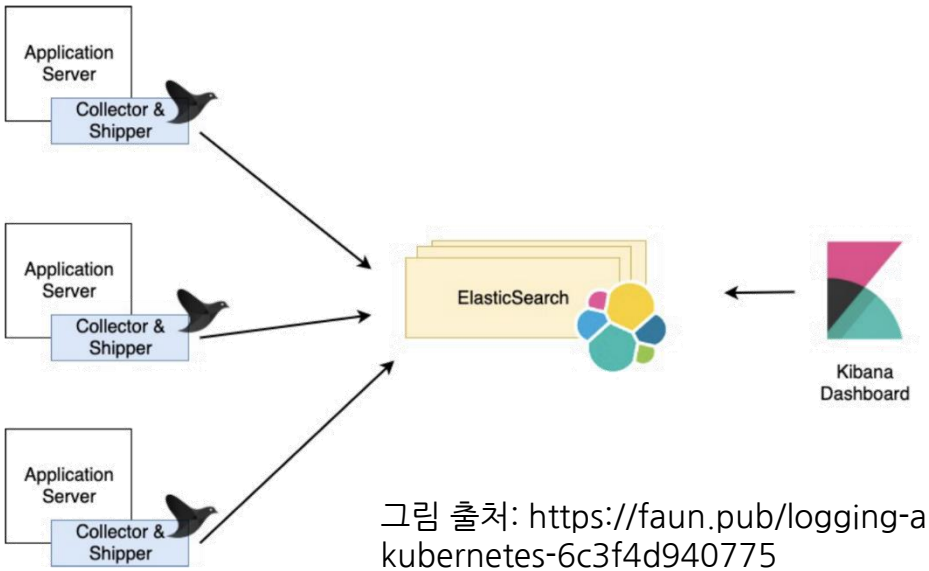


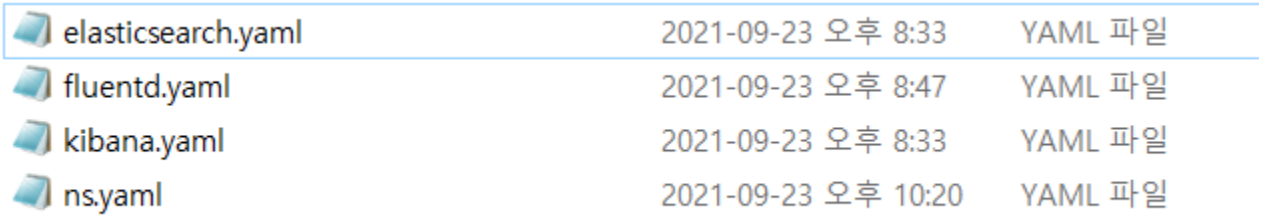
그림 출처: https://faun.pub/logging-architectures-efk-stack-and-kubernetes-6c3f4d940775

EFK를 활용한 k8s 로그 모니터링

EFK 설치 환경

- https://blog.naver.com/isc0304/221860255105
- https://blog.naver.com/isc0304/221879552183

- 예제 파일 다운로드



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

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


www.boanproject.com

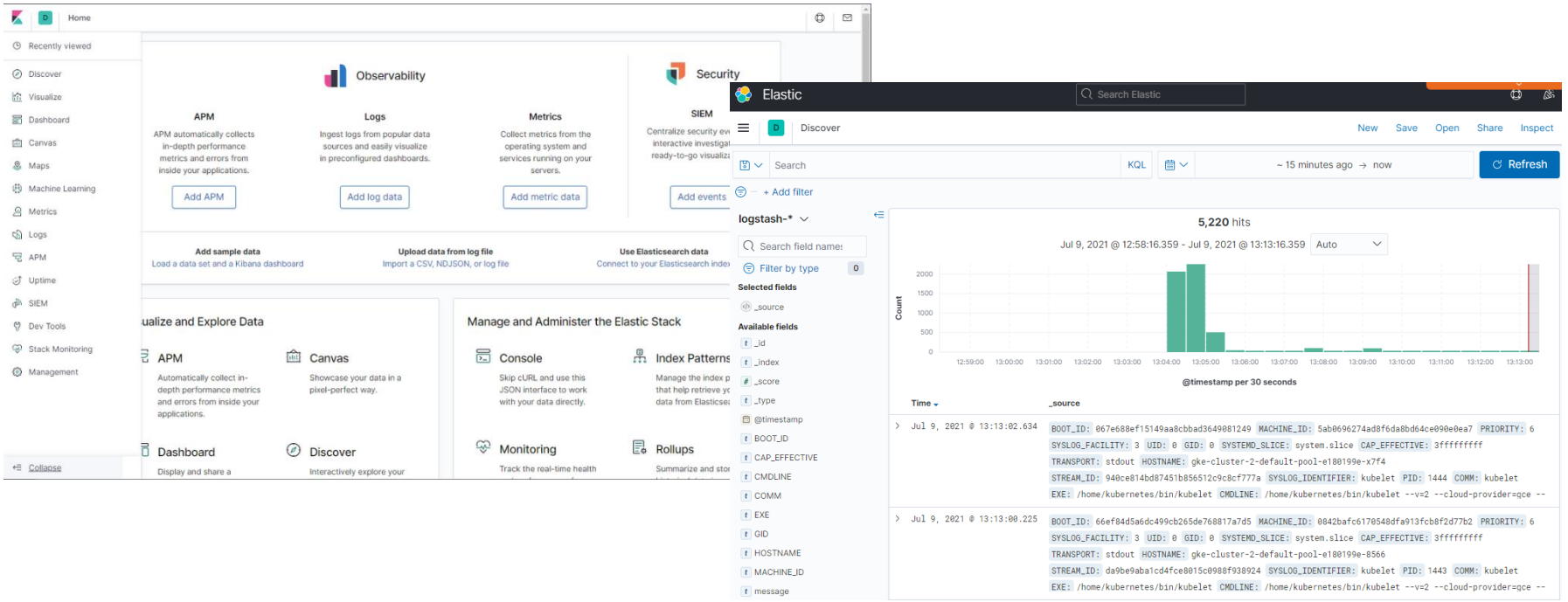
EFK를 활용한 k8s 로그 모니터링

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



www.boanproject.com

오토 스케일링 HPA 워크스루

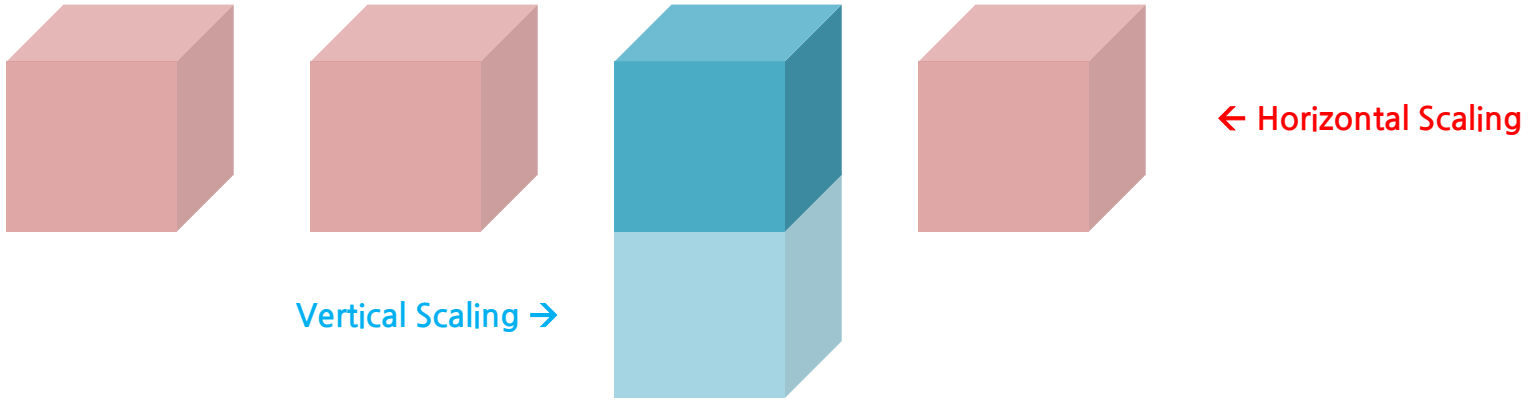
오토 스케일링 HPA 워크스루

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.CjwKCAiA-vLyBRBWEiwAzOkGVAWhHWKuFV0kIRfpVDRA5yaGh1wIUZOHQZ6Z9lZYS2lZtzlyC5lBbhoCT3AQAvD_BwE

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


www.boanproject.com

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
```

- HPA yaml을 작성하여 타겟 파드 지정

autoscaling.yaml

```yaml
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
```


www.boanproject.com

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
```

dockerfile

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

index.php

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


www.boanproject.com

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

/ # while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!...
```

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`
```


www.boanproject.com

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%

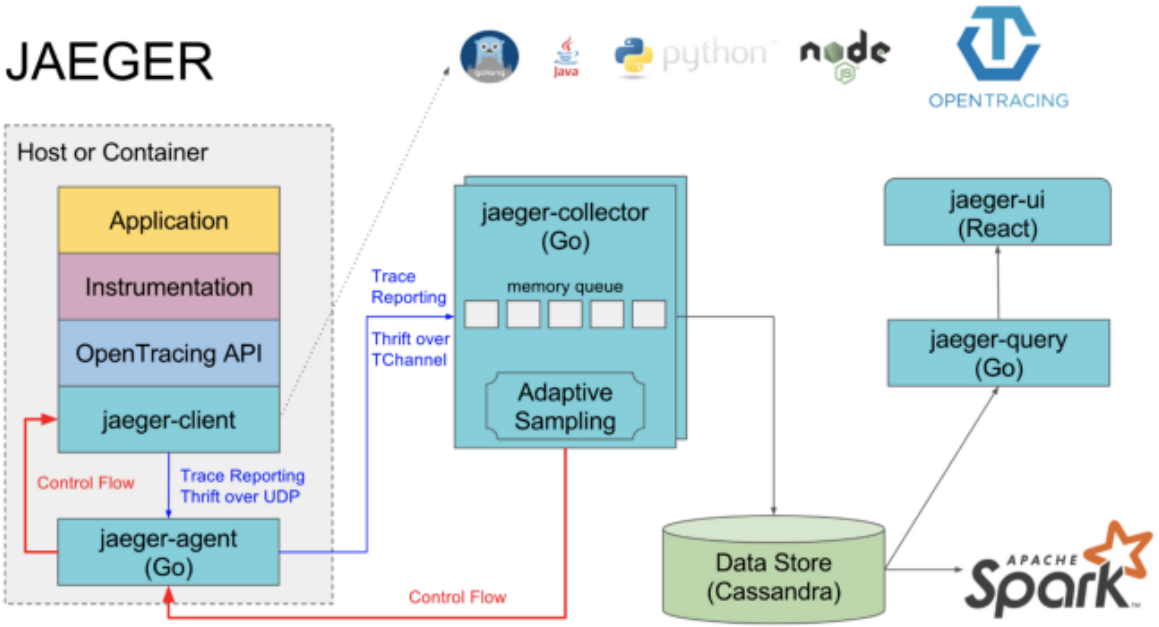
www.boanproject.com

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



www.boanproject.com

Jaeger 트레이싱 튜토리얼

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

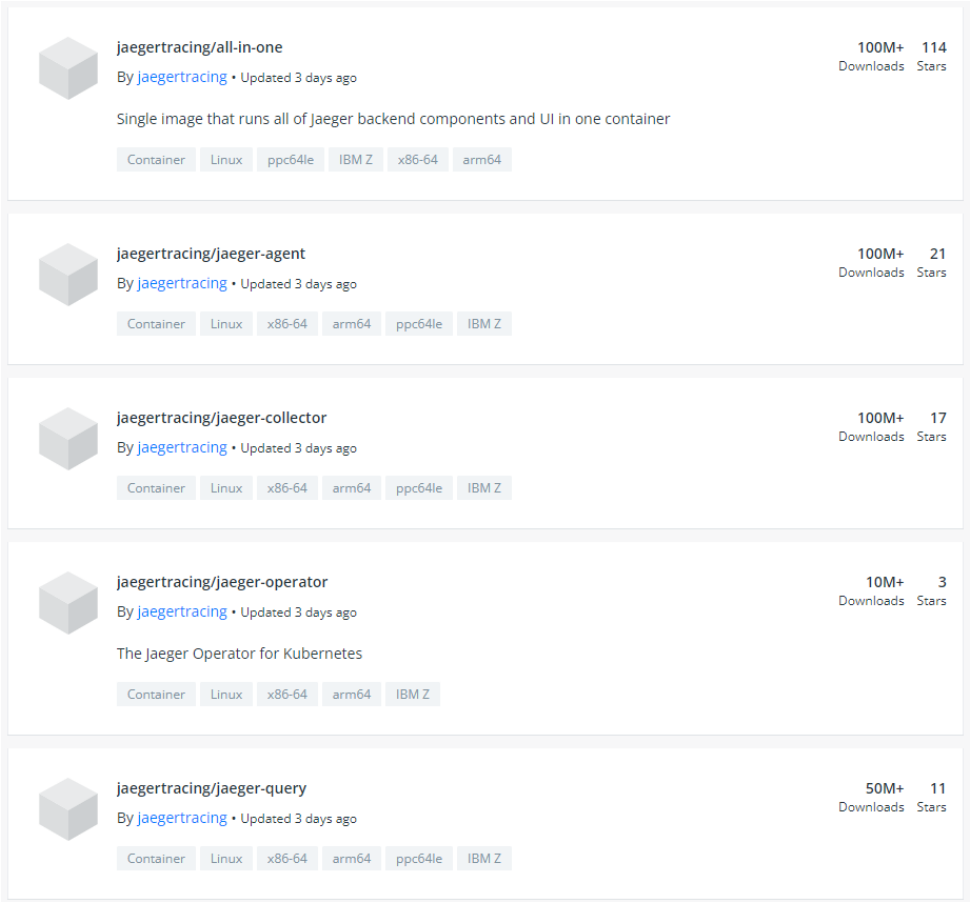
Jaeger 트레이싱 튜토리얼

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
```

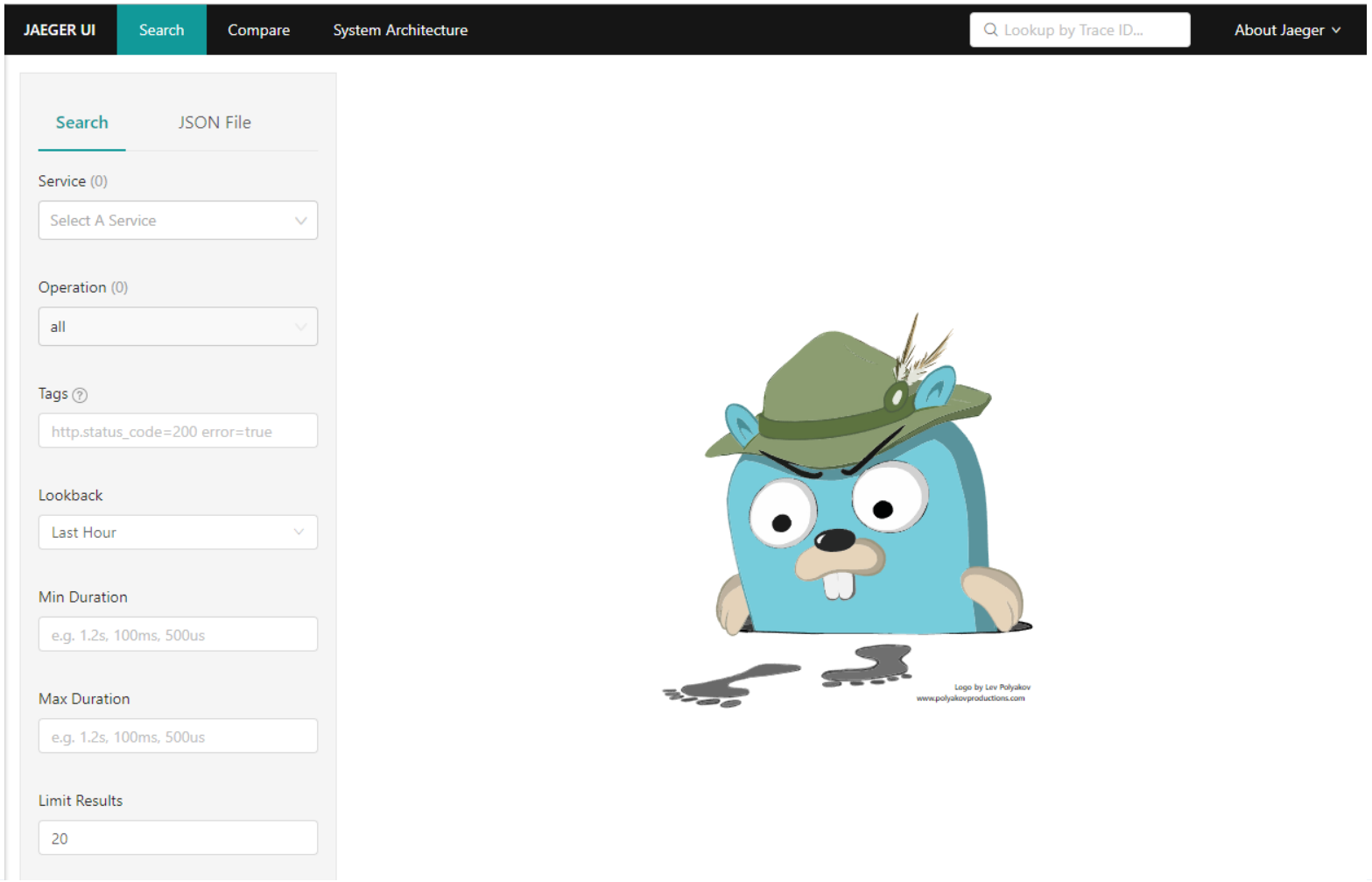


www.boanproject.com

Jaeger 트레이싱 튜토리얼

예거 시스템에 접속하기

- 16686 포트로 접속하기



www.boanproject.com

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

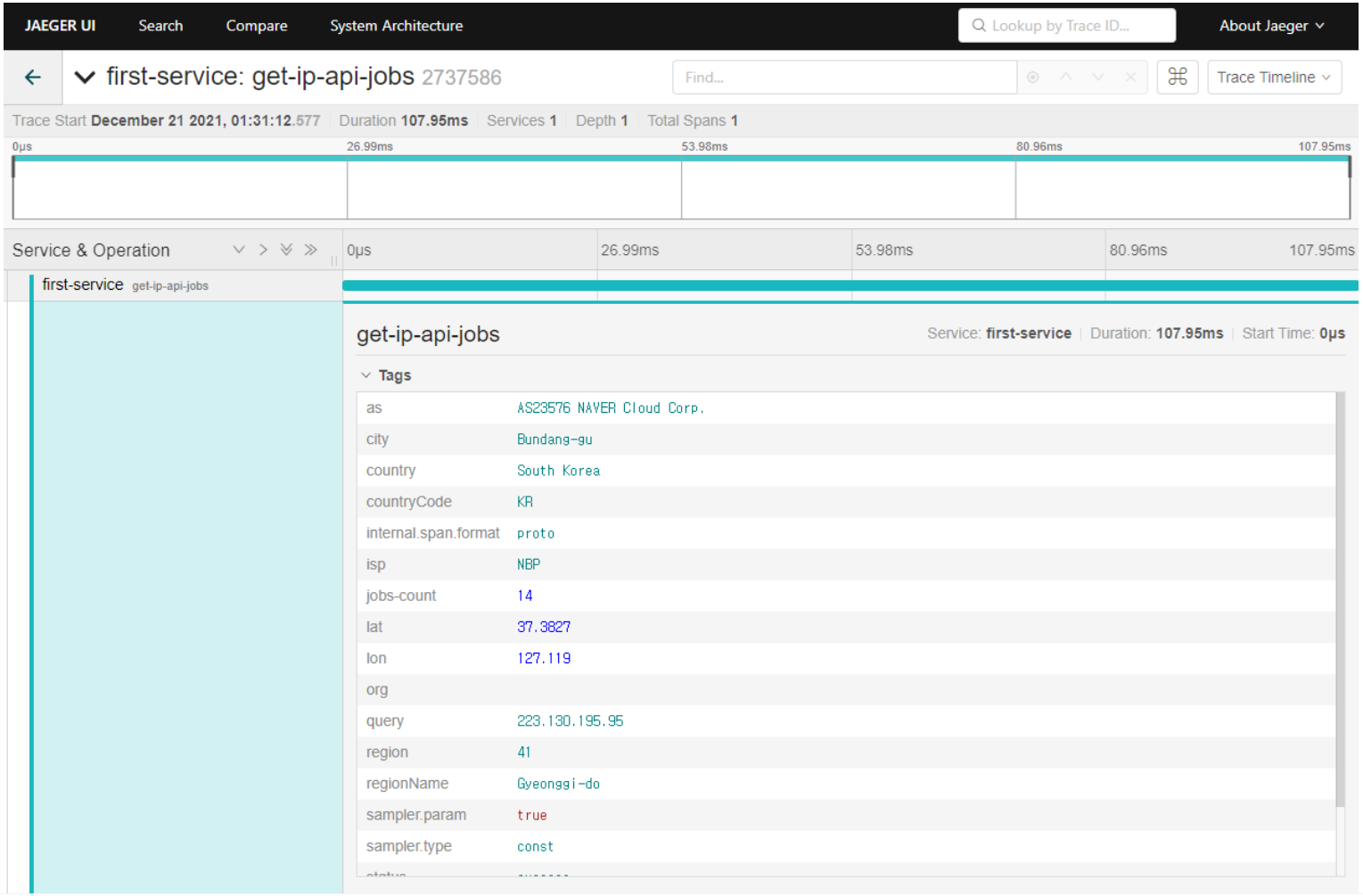
```
tracer = init_tracer('first-service')

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')
```


www.boanproject.com

Jaeger 트레이싱 튜토리얼

파이썬 프로그램에 트레이싱 예제

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



www.boanproject.com