7주차

쿠버네티스

다수의 컨테이너를 자동으로 운영하기 위한 오케스트레이션 도구

등장 배경

마이크로 서비스 아키텍처의 세분화된 서비스 + 컨테이너의 확장성 → 컨테이너들을 쉽게 관리, 배포할 수 있는 Docker

서비스 규모가 커질 수록 관리해야하는 컨테이너 수가 급격히 증가 → 컨테이너 관리를 자동화해줄 툴 필요

모놀리식 아키텍처

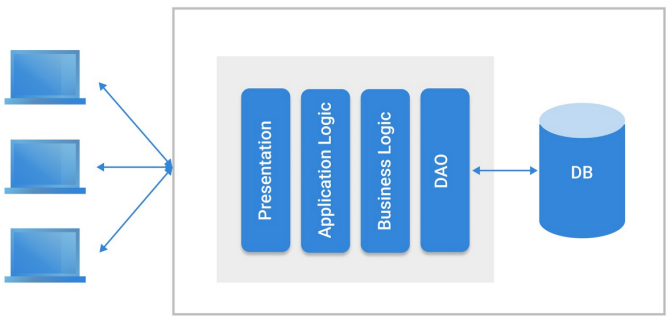
소프트웨어의 모든 구성요소가 한 프로젝트에 통합되어있는 형태

- 부분의 장애가 전체의 장애로 확대될 수 있음
- 규모가 커질 수록 수정이 어렵고 장애의 영향을 파악하기도 어려움 → 주로 소규모 프로젝트에 이용
- 배포시간 오래걸림

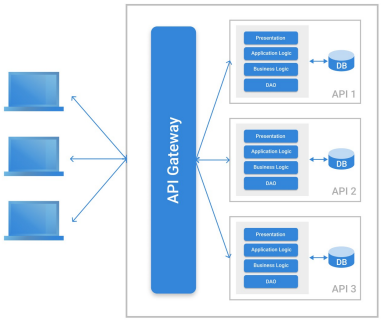
마이크로서비스(MSA) 아키텍처

하나의 큰 애플리케이션을 여러 개의 작은 애플리케이션으로 쪼개어 작고, 독립적으로 배포 가능하도록 만든 형태

- 각각의 서비스가 모듈화 되어있음
- API를 통해서만 상호작용 가능
- 서비스별로 독립적인 배포 가능



모놀리식 아키텍처*(Monolithic Architecture)*



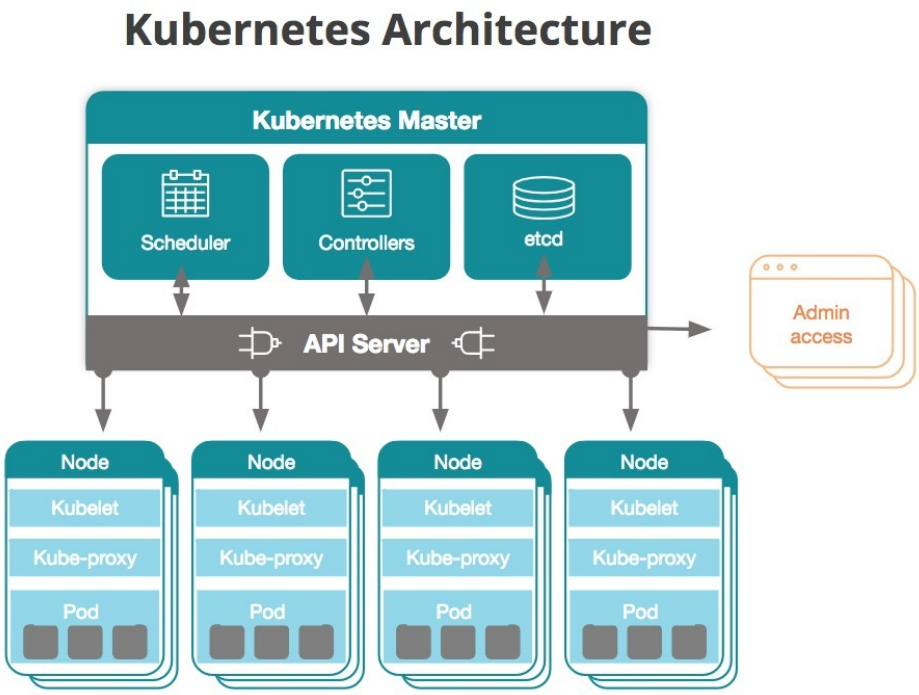
마이크로서비스 아키텍처*(MSA)*

기능

- **상태관리** : 상태를 선언하고 선언한 상태를 계속해서 유지시켜준다. 예를 들어 노드가 죽거나 컨테이너 응답이 없을 경우 자동으로 새로운 컨테이너를 띄우거나, 자동으로 죽여주는 등 **선언한 처음 그대로의 상태**를 계속해서 유지시켜줌
- **스케줄링** : 어떤 서버에 컨테이너를 띄울까를 고민하지 않아도 쿠버네티스가 조건에 맞는 노드를 찾아서 컨테이너를 배치

- **클러스터** : 가상 네트워크를 통해 통신하기 때문에 하나의 서버에 있는 것처럼 관리할 수 있음
- **서비스 디스커버리** : 서로 다른 서비스를 쉽게 찾고 통신할 수 있음

아키텍처



마스터 노드

쿠버네티스 클러스터 전체를 관리하는 노드
크게 API서버, 스케줄러, 컨트롤 매니저, etcd로 구성

- **API 서버** - 쿠버네티스의 모든 기능들을 REST API로 제공하고 그에 대한 명령을 처리
- **etcd** - 쿠버네티스 클러스트의 데이터 베이스 역할이 되는 서버, 설정값이나 클러스터의 상태 저장
- **스케줄러** - Pod, 서비스 등 각 리소스들을 적절한 노드에 할당하는 역할
- **컨트롤러 매니저** - 컨트롤러(레플리카 컨트롤러, 서비스 컨트롤러, 볼륨 컨트롤러, 노드 컨트롤러..등)를 생성하고 이를 각 노드에 배포하고 관리하는 역할

워커 노드

마스터에 의해 명령을 받고 실제 워크로드를 생성하여 서비스하는 컴포넌트
노드에는 kubelet, kube-proxy, 컨테이너 런타임이 배포됨

- **kubelet** - 노드에 배포되는 에이전트, 마스터의 API서버와 통신하면서 노드가 수행해야 할 명령을 받아서 수행하고, 반대로 노드의 상태등을 마스터로 전달
- **kube-proxy** - 노드로 들어오는 네트워크 트래픽을 적절한 컨테이너로 라우팅, 로드밸런싱, 노드와 마스터 간의 네트워크 통신을 관리
- **컨테이너 런타임 엔진** - Pod를 통해서 배포된 컨테이너들을 구동시키는 엔진

오브젝트 종류

▼ **파드 (Pod)** - 쿠버네티스에서 실행되는 가장 작은 배포 단위. 독립적인 공간과 고유한 IP를 가짐, 여러개의 컨테이너가 하나의 Pod에 속할 수 있음

```
apiVersion: v1 # 스크립트를 실행하기 위한 쿠버네티스 API 버전
kind: Pod # 리소스의 종류 정의
metadata: # 리소스의 라벨, 이름 등을 지정
  name: nginx
spec: # 각 컴포넌트에 대한 상세 설명. 어떤 오브젝트 종류인지에 따라 다른 내용
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

▼ **네임스페이스 (Namespace)** - 쿠버네티스 클러스터에서 사용되는 리소소들을 구분해서 관리하는 그룹

```
# test-namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: test
spec:
  limits:
- default:
      cpu: 1
  defaultRequest:
      cpu: 0.5
  type: Container
```

▼ **볼륨 (Volume)** - 파드가 사라지더라도 저장/보존이 가능하며 파드에서 사용할 수 있는 디렉터리를 제공한다.

```
apiVersion: v1
kind: Pod
metadata:
  name: test-ebs
  app.kubernetes.io/name: MyApp
spec:
  containers:
  - image: registry.k8s.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-ebs
      name: test-volume
  volumes:
  - name: test-volume
    # This AWS EBS volume must already exist.
    awsElasticBlockStore:
      volumeID: "<volume id>"
      fsType: ext4
```

▼ **서비스 (Service)** - 파드는 유동적이기 때문에 접속 정보가 고정되지 않으므로, 파드 접속을 안정적으로 유지하기 위한 기능이다.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app.kubernetes.io/name: MyApp
```

```
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

▼ **레플리카셋 (ReplicaSet)** - 여러개의 Pod을 관리. 몇 개의 Pod을 관리할 지 결정하고 지정한 개수만큼 항상 실행될 수 있도록 관리(실행되는 pod 개수에 대한 가용성 보증)

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

▼ **디플로이먼트 (Deployment)** - . 내부적으로 ReplicaSet을 이용하여 Pod을 업데이트하고 이력을 관리하여 롤백하거나 특정 버전으로 돌아가는 등 배포 버전을 관리.

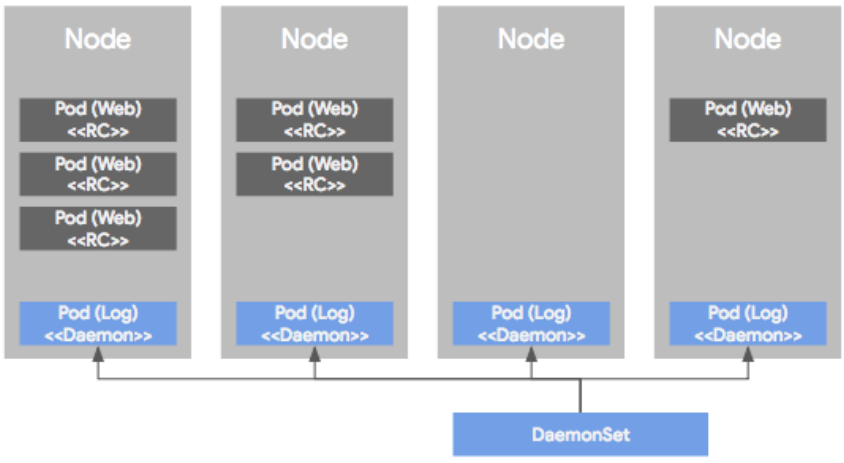
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

▼ **스테이트풀셋 (StatefulSet)** - 애플리케이션 상태 저장 및 관리, pod 집합의 디플로이먼트와 스케일링을 관리하며, pod들의 순서 및 고유성을 보장

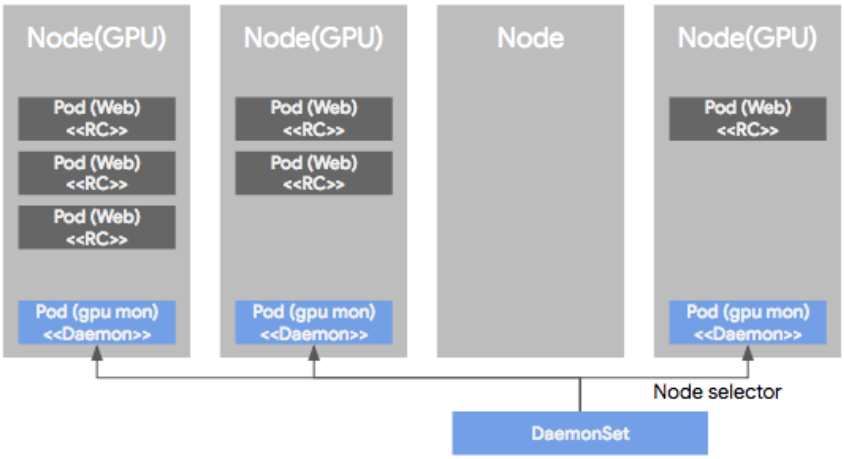
```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
```

```
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.metadata.labels
  serviceName: "nginx"
  replicas: 3 # by default is 1
  minReadySeconds: 10 # by default is 0
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.selector.matchLabels
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: registry.k8s.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

▼ **데몬셋 (DaemonSet)** - 클러스터 전체에 pod를 띄울 때 사용하는 컨트롤러, 특정 노드 혹은 모든 노드에 pod를 균등하게 배포, 관리



모든 노드에 대해



특정 노드 select

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
      # these tolerations are to have the daemonset runnable on control plane nodes
      # remove them if your control plane nodes should not run pods
      - key: node-role.kubernetes.io/control-plane
        operator: Exists
        effect: NoSchedule
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
        image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

▼ **컨피그맵 (ConfigMap)**

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

▼ **시크릿(Secret)** - 시크릿은 암호, 토큰 또는 키와 같은 소량의 중요한 데이터를 포함하는 오브젝트

```
apiVersion: v1
kind: Secret
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: { ... }
  creationTimestamp: 2020-01-22T18:41:56Z
  name: mysecret
  namespace: default
  resourceVersion: "164619"
  uid: cfee02d6-c137-11e5-8d73-42010af00002
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

▼ **CR(custom resource)** - 사용자 정의 오브젝트

아래 `mycr.yaml` 파일을 이용해 객체를 생성하고자 할 때,
쿠버네티스는 kind:Hello 인 객체를 알 수 없음

```
# mycr.yaml
apiVersion: "extension.example.com/v1"
kind: Hello
metadata:
  name: hello-sample
size: 3
```

따라서 커스텀 리소스(kind:Hello)를 쿠버네티스 etcd에 등록하기 위해서는 CRD를 이용해야함

아래와 같이 `mycrd.yaml` 파일을 먼저 작성

```
# mycrd.yaml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: hellos.extension.example.com
spec:
  group: extension.example.com
  version: v1
  scope: Namespaced
  names:
```

```
    plural: hellos
    singular: hello
    kind: Hello
```

```
# crd 적용
$ kubectl apply -f mycrd.yaml
customresourcedefinition.apiextensions.k8s.io/hellos.extension.example.com created

# crd 확인
$ kubectl get crd
NAME                          CREATED AT
hellos.extension.example.com   2021-07-19T12:40:23Z

# crd 상세 설명 확인
$ kubectl explain hello
KIND:     Hello
VERSION:  extension.example.com/v1

DESCRIPTION:
     <empty>
```

이제 cr(kind:Hello) 생성 가능

```
# cr 생성
$ kubectl apply -f mycr.yaml
hello.extension.example.com/hello-sample created

# 생성된 cr 확인
$ kubectl get hello
NAME           AGE
hello-sample   2m54s
```

- **Helm** - 쿠버네티스용 패키지 매니지먼트 도구 (리눅스의 apt, node.js의 npm…)

▼ **Helm chart** - helm의 패키지 포맷, app을 실행시키기위한 모든 리소스가 정의되어 있음

install

kubectl을 사용할 수 있는 노드로 이동하여 설치

```
$ curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 > get_helm.sh
$ chmod 700 get_helm.sh
$ ./get_helm.sh
```

버전 확인

```
$ helm version
```

helm chart repository를 추가

```
$ helm repo add stable https://kubernetes-charts.storage.googleapis.com/
```

chart list 출력

```
$ helm search repo stable
```

chart update

```
$ helm repo update
```

참고 https://kubernetes.io/docs/concepts/

참고 https://bcho.tistory.com/1256

실습

1. minikube 설치하기
2. efk 설치하기

Minikube란?

마스터 노드의 일부 기능과 개발 및 배포를 위한 단일 워커 노드를 제공해 간단한 쿠버네티스 플랫폼 환경을 로컬 환경에서 만들어볼 수 있게 도와주는 도구

Minikube 설치

Mac

참고 링크

```
curl -Lo minikube https://github.com/kubernetes/minikube/releases/download/v1.25.1/minikube-darwin-arm64 \
  && chmod +x minikube
sudo install minikube /usr/local/bin/minikube
```

- homebrew로 설치할 수도 있는데, m1은 최신인 1.26에서 에러가 나서 1.25로 설치해야함

```
minikube version
# minikube version: v1.25.1
```

```
minikube start --kubernetes-version=v1.23.1 --driver=docker
```

- 시간 좀 걸림

```
minikube kubectl get nodes
# kubectl 자동으로 깔림
```

```
kubectl version --client
# 어쩌구 저쩌구 Kustomize Version: v4.5.4
```

```
kubectl get nodes
---
NAME       STATUS   ROLES                  AGE     VERSION
minikube   Ready    control-plane,master   2m54s   v1.23.1
# 반드시 STATUS가 Ready 상태여야 함
```

minikube 준비 끝!

Windows(wsl)

참고 링크

```
kvewsl
```

```
curl -LO https://storage.googleapis.com/minikube/releases/v1.25.1/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

```
minikube version
# minikube version: v1.25.1
```

```
minikube start --kubernetes-version=v1.23.1 --driver=docker
```

- 시간 좀 걸림

```
minikube kubectl get nodes
# kubectl 자동으로 깔림
```

```
kubectl version --client
# 어쩌구 저쩌구 Kustomize Version: v4.5.4
```

```
kubectl get nodes
---
NAME       STATUS   ROLES           AGE     VERSION
minikube   Ready    control-plane   9m45s   v1.24.3
# 반드시 STATUS가 Ready 상태여야 함
```

minikube 준비 끝!

KubectI이란?

쿠버네티스 클러스터에 명령을 내리는 CLI(command line interface)

```
kubectl [COMMAND] [TYPE] [NAME] [FLAGS]
```

- **COMMAND** : create, get, describe와 같이 하나 혹은 여러개의 리소스에 대한 operation 종류 선언
- **TYPE** : 리소스 타입에 대해 선언
- **NAME** : 특정 리소스의 이름 선언
- **FLAGS** : 추가적인 옵션 선언

kubectl create/apply

```
// example-service.yaml 파일이름의 서비스를 생성합니다.
$ kubectl create -f example-service.yaml

// example-controller.yaml 파일이름의 RC를 생성합니다.
$ kubectl create -f example-controller.yaml
```

kubectl get

```
// pod list를 출력
$ kubectl get pods

// pod list(+ 추가적인 정보 node 이름 등)를 출력
$ kubectl get pods -o wide

// 특정 <rc-name>의 정보를 출력
$ kubectl get replicationcontroller <rc-name>

// 모든 rc, service들 정보를 출력
$ kubectl get rc,services

// 모든 ds(daemon sets)에 대한 정보를 출력(uninitialized ds도 포함)
$ kubectl get ds --include-uninitialized

// 특정 node(server01)에 배포된 pod 정보를 출력
$ kubectl get pods --field-selector=spec.nodeName=server01
```

kubectl describe

```
// 특정 <node-name>의 node 정보 출력
$ kubectl describe nodes <node-name>

// 특정 <pod-name>의 pod 정보 출력
$ kubectl describe pods/<pod-name>

// 특정 <rc-name>의 rc가 제어하는 pod들 정보 출력
$ kubectl describe pods <rc-name>

// 모든 pod 정보 출력(uninitialized pod은 제외)
$ kubectl describe pods --include-uninitialized=false
```

kubectl delete

```
// pod.yaml로 선언된 pod들을 제거
$ kubectl delete -f pod.yaml

// 특정 <label-name>이 정의된 pod, service들 제거
$ kubectl delete pods,services -l name=<label-name>

// 특정 <label-name>이 정의된 pod, service들 제거(uninitialized pod, service 포함)
$ kubectl delete pods,services -l name=<label-name> --include-uninitialized

// 모든 pod
$ kubectl delete pods --all
```

kubectl exec

```
// 특정 <pod-name>을 가진 pod의 첫번째 container에 'date' 라는 명령어 호출
$ kubectl exec <pod-name> date

// 특정 <pod-name>을 가진 pod의 특정 <container-name>이라는 이름의 container에 'date' 라는 명령어 호출
$ kubectl exec <pod-name> -c <container-name> date

// 특정 <pod-name>을 가진 pod의 첫번째 container에 bash shell실행
$ kubectl exec -ti <pod-name> /bin/bash
```

kubectl logs

```
// 특정 <pod-name> 이름을 가진 pod의 로그 조회
$ kubectl logs <pod-name>

// 특정 <pod-name> 이름을 가진 pod의 로그 tail -f 조회
$ kubectl logs -f <pod-name>
```

`Kubectl Refernce` https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands

EFK 란?

> ElasticSearch + Fluentd (fluent bit) + Kibana

마이크로 서비스는 각각의 애플리케이션을 띄워서 로그 또한 각각 관리하는 것이기 때문에 서비스간에 이어지는 트랜젝션의 흐름을 파악하기 어렵다. **EFK**는 마이크로 서비스에서 로그를 중앙 집중적으로 저장하고 분석하기 위해 사용하는 로깅 솔루션이다.

Elasticsearch : 로그 저장소, 검색 엔진

실시간 로그를 저장하여 필요한 내용을 검색하는 검색엔진

Fluentd : 로그 적재기

로그 수집하고 저장소에 저장하는 로그 적재기
서로 다른 애플리케이션에서 로그를 수집하고 트래픽을 조정해 로그저장소에 로그를 수집힌다.

Kibana : 대시 보드

수집한 로그를 시각화하는 대시보드

EFK 설치 (with Minikube)

1. source(server)

image

미리 관련 이미지를 dockerhub에 빌드해놓았음. 자세한 건 깃허브 참고

install

편한 디렉토리로 이동, 오늘 사용할 디렉토리 생성

```
cd {{ 디렉토리 }}
mkdir {{ 디렉토리명 }}
```

source yaml 파일 생성vi

```
vi source.yaml
# 입력 후 저장
---
apiVersion: v1
kind: Service
metadata:
  name: efk-source
  namespace: source
spec:
  selector:
    app: efk-source
  ports:
  - name: http
    protocol: TCP
    port: 8088
    targetPort: 8080
  type: LoadBalancer

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: efk-source
  namespace: source
spec:
  selector:
    matchLabels:
      app: efk-source
  replicas: 2
  template:
    metadata:
      labels:
        app: efk-source
    spec:
      containers:
      - name: efk-source
        image: ryann3/boaz19-efk-source:0.2
        ports:
        - containerPort: 8080
```

- 여기서 퀴즈! 같은 오브젝트를 나타내는 efk-source끼리 묶어보기!

  ▼ 정답

```
---
apiVersion: v1
kind: Service
metadata:
  name: efk-source
  namespace: real
spec:
  selector:
    app: efk-source
  ports:
  - name: http
    protocol: TCP
    port: 8088
    targetPort: 8080
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: efk-source
  namespace: logging
spec:
  selector:
    matchLabels:
      app: efk-source
  replicas: 2
  template:
    metadata:
      labels:
        app: efk-source
    spec:
      containers:
      - name: efk-source
        image: ryann3/boaz19-efk-source
        ports:
        - containerPort: 8080
```

```
kubectl create namespace source # 위에서 사용한 ns 생성 필요
kubectl apply -f source.yaml
kubectl get namespace
```

```
kubectl get all -n source
# 반드시 pod 두 개 모두 Ready 1/1이고, Status가 Running이어야함
# watch -d -n 1 'kubectl get all -n source' # 이런 명령어로도 볼 수 있음
---
NAME                                READY   STATUS    RESTARTS   AGE
pod/efk-source-7954668fdd-d7v96   1/1     Running   0          2m16s
pod/efk-source-7954668fdd-zv8tn   1/1     Running   0          2m16s
```

```
NAME                 TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/efk-source   LoadBalancer   10.101.39.71   <pending>     8088:30184/TCP   2m16s

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/efk-source    2/2     2            2           2m16s

NAME                                     DESIRED   CURRENT   READY   AGE
replicaset.apps/efk-source-7954668fdd    2         2         2       2m16s
```

port-forward

```
kubectl port-forward svc/efk-source -n source 8088:8088 &
```

```
curl http://127.0.0.1:8088/?message=hello
# Handling connection for 8088
# {"message":"hello"}
```

2. elasticsearch+fluentd+kibana

```
kubectl create ns logging
```

yaml 생성

```
vi elasticsearch.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: elasticsearch
  namespace: logging
  labels:
    app: elasticsearch
spec:
  replicas: 1
  selector:
    matchLabels:
      app: elasticsearch
  template:
    metadata:
      labels:
        app: elasticsearch
    spec:
      containers:
      - name: elasticsearch
        image: elastic/elasticsearch:6.4.0
        env:
        - name: discovery.type
          value: single-node
        ports:
        - containerPort: 9200
        - containerPort: 9300
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: elasticsearch
  name: elasticsearch-svc
  namespace: logging
```

```
spec:
  ports:
  - name: elasticsearch-rest
    nodePort: 30920
    port: 9200
    protocol: TCP
    targetPort: 9200
  - name: elasticsearch-nodecom
    nodePort: 30930
    port: 9300
    protocol: TCP
    targetPort: 9300
  selector:
    app: elasticsearch
  type: NodePort
```

```
vi kibana.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kibana
  namespace: logging
  labels:
    app: kibana
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kibana
  template:
    metadata:
      labels:
        app: kibana
    spec:
      containers:
      - name: kibana
        image: elastic/kibana:6.4.0
        env:
        - name: SERVER_NAME
          value: kibana.kubenetes.example.com
        - name: ELASTICSEARCH_URL
          value: http://elasticsearch-svc:9200
        ports:
        - containerPort: 5601
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: kibana
  name: kibana-svc
  namespace: logging
spec:
  ports:
  - nodePort: 30561
    port: 5601
    protocol: TCP
    targetPort: 5601
  selector:
    app: kibana
  type: NodePort
```

```
vi fluentd.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
  name: fluentd
  namespace: logging

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: fluentd
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - namespaces
  verbs:
  - get
  - list
  - watch

---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: fluentd
roleRef:
  kind: ClusterRole
  name: fluentd
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: fluentd
  namespace: logging
---
apiVersion: apps/v1
kind: DaemonSet # 모든 노드에 배포되어야하므로 데몬셋!
metadata:
  name: fluentd
  namespace: logging
  labels:
    k8s-app: fluentd-logging
    version: v1
spec:
  selector:
    matchLabels:
      k8s-app: fluentd-logging
      version: v1
  template:
    metadata:
      labels:
        k8s-app: fluentd-logging
        version: v1
    spec:
      serviceAccount: fluentd
      serviceAccountName: fluentd
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd
        image: fluent/fluentd-kubernetes-daemonset:v1-debian-elasticsearch
        env:
          - name:  FLUENT_ELASTICSEARCH_HOST
            value: "elasticsearch-svc"
          - name:  FLUENT_ELASTICSEARCH_PORT
            value: "9200"
          - name: FLUENT_ELASTICSEARCH_SCHEME
            value: "http"
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
```

```
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

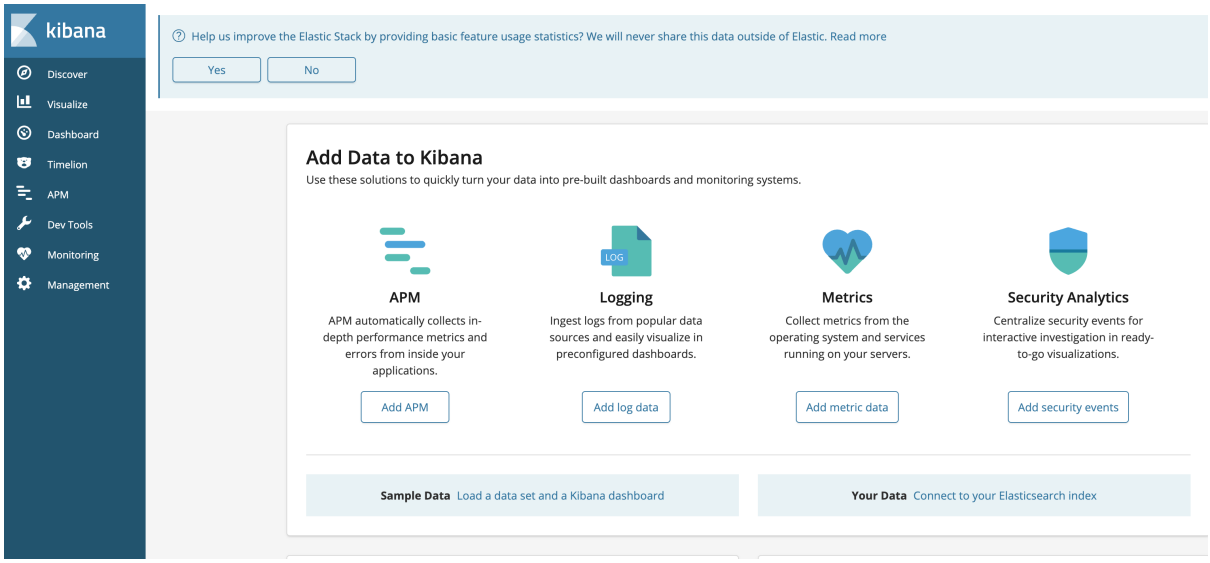
```
kubectl apply -f elasticsearch.yaml
kubectl apply -f kibana.yaml
kubectl apply -f fluentd.yaml
```

kibana 확인

포트포워딩

```
kubectl port-forward svc/kibana-svc -n logging 5601:5601 &
```

http://localhost:5601에 접근 시 다음과 같은 UI 확인



에러가 나면

1) 다시 해보거나

2) 터미널(cmd)를 두 개 띄워서 하나는 로그 확인, 하나는 포트 포워딩 해보기

```
# 로그 확인
watch -d -n 1 'kubectl logs {{ kibana pod name }} -n logging'
```

전체 확인

터미널로 확인

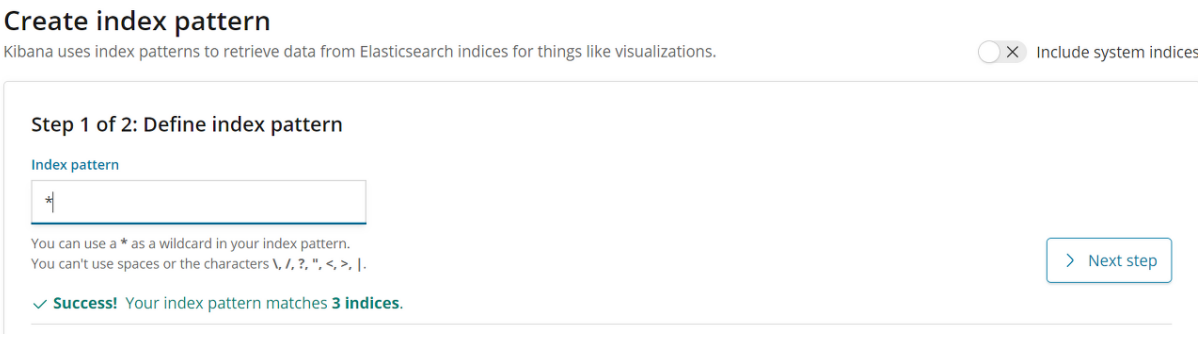
- 또 퀴즈: 어떤 명령어를 써야 logging namespace의 모든 리소스 상태를 확인할 수 있을까?

  ▼ 정답

```
kubectl get all -n logging
---
NAME                                    READY   STATUS    RESTARTS   AGE
pod/elasticsearch-6b79d9746d-lbg4w      1/1     Running   0          20m
pod/fluentd-g49ps                       1/1     Running   0          5m17s
pod/kibana-9f875ff88-dwns8              1/1     Running   0          16m

NAME                          TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)                         AGE
service/elasticsearch-svc     NodePort   10.99.175.84    <none>        9200:30920/TCP,9300:30930/TCP   20m
service/kibana-svc            NodePort   10.110.40.129   <none>        5601:30561/TCP                  16m

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/fluentd      1         1         1       1            1           <none>          5m17s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/elasticsearch       1/1     1            1           20m
deployment.apps/kibana              1/1     1            1           16m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/elasticsearch-6b79d9746d       1         1         1       20m
replicaset.apps/kibana-9f875ff88               1         1         1       16m
```

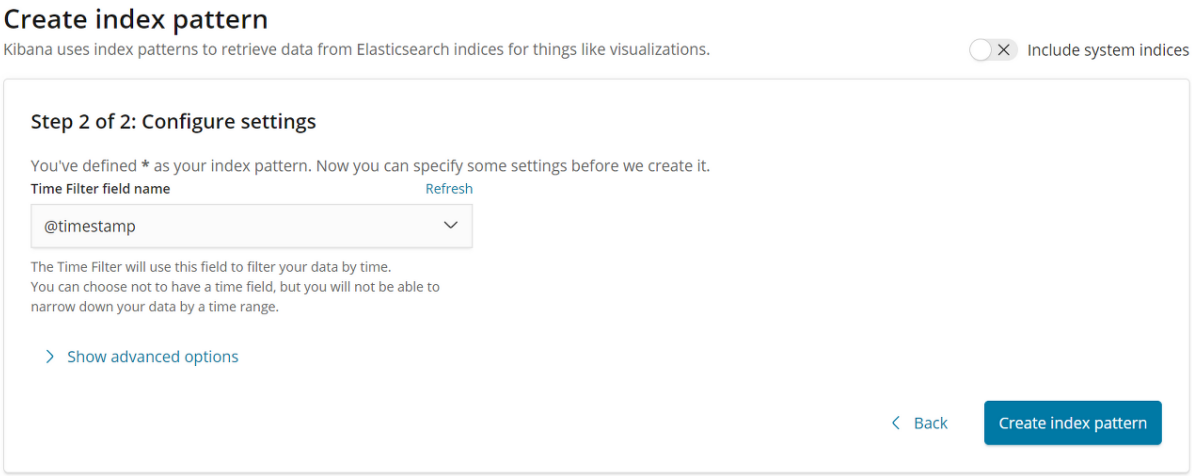
웹으로 확인

1. Kibana에 접근(http://localhost:5601)
2. 왼쪽 네비게이션에서 Discover 선택
3. Index pattern에 “*” 입력



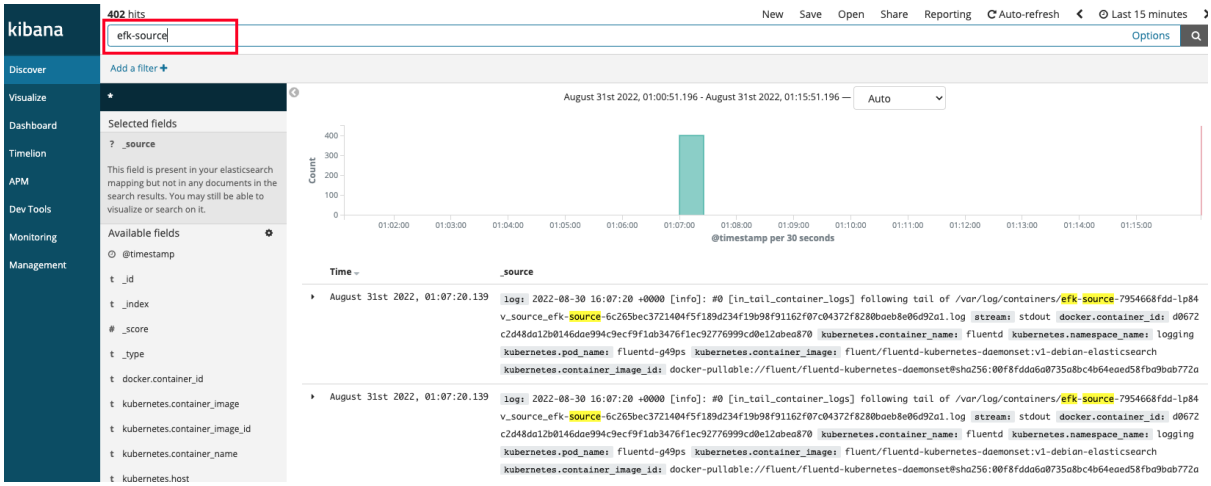
- 만약 제대로 연결이 되어있지 않다면 여기서부터 아무것도 안 뜬다

4. Time Filter field name으로 “@timestamp” 선택



5. 다시 Discover로 돌아오면 모든 로그에 접근할 수 있다.

- 배포한 svc의 이름(efk-source)을 검색하면 하이라이트 되는 것을 볼 수 있다.



마무리

리소스를 굉장히 많이 잡아먹는 애증의 kubernetes... 지우고 정리하는 것이 마음이 편합니다.

```
kubectl delete -f elasticsearch.yaml
kubectl delete -f kibana.yaml
kubectl delete -f fluentd.yaml
kubectl delete -f source.yaml
kubectl delete ns logging, source
```

```
minikube delete
```

전부 삭제해버립니다.. 클러스터까지~