LoadBalancer

노드의 특정 포트가 아닌 LoadBalancer를 통해서 외부 traffic전달

로드밸런서의 port: 서비스 포트

apiVersion:v1

kind:Service

metadata:

name: mynginx

spec:

type: LoadBalancer

ports:

- port: 80

protocol: TCP

targetPort: 80

nodePort: 30080

selector:

run: mynginx

ExternalName

외부 도메인이름 alias

externalName: 외부주소

외부에 있는 서비스를 참조하기 위해 사용

apiVersion: v1

kind: Service

metadata:

name: my-google-svc

spec:

type: ExternalName

externalName: google.com

도커가 NAT모델 사용함

도커는 컨테이너간의 통신에 각각 포트를 통해서 통신

쿠버네티스는 서로간의 통신 IP를 통해 통신, 자유로움

NAT - 도커

노드 IP를 이용 특정포트와 컨테이너의 특정 포트를 연결하여 NAT통신 사용

A B POD 끼리 통신시에 직접 통신하는 것이 아니라 NODE IP의 포트 레퍼런싱을 통해 연결 외부 컨테이너끼리 통신또한 그러함ㅁ

라우팅은은 각각 IP를 할당 노드의 네트워크인터페이스를 통해 바로 라우팅가능

NAT는 특정 IP와 포트의 맵핑 테이블이 필요하고 그 테이블을 통해 통신

라우팅은 각 POD들이 하나의 서버처럼 직접 IP를 할당받아 통신

네트워크 인터페이스와 도커간 브릿지로 연결 트래픽 전달

브릿지는 각각 컨테이너안의 버추얼 네트워크 인터페이스 생성 밑 IP 생성후 전달

실제로는 하나의 Pod에 각 컨테이너들이 하나의 버추얼 네티워크 인터페이스를 가지기 때문에 각각의 컨테이너들을 포트로 구분지어 localhost로 서로서로 통신함.

노드를 벗어난 노드간의 pod 레퍼런싱 통신

레플리카셋

레플리카셋이라는 리소스가 파드의 레플리카를 관리함

우리는 여러개의 팟을 둠. 특정파드가 망가져도 다른 팟으로 연결 팟들은 동일한 스펙의 팟

팟이 죽으면 서비스가 중단되기 때문에 레플리카셋을 둠. 복제본을 둠. 레플리카셋이 팟을 관리함. 레플리카셋은 라벨링 시스템으로 팟들을 관리함. 라벨셀렉트를 통하여 관리.

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: mynginx-rs

spec:

replicas: 1 ==> 몇개의 레플리카셋을 둘것인가

selector: ==> 어떤 라벨을 가진 팟들로 관리할 것인가.

matchLabels:

app: mynginx-rs

template: 팟의 앱정의들

metadata:

labels:

app: mynginx-rs

spec:

containers:

- name: nginx

image: nginx

kubectl get replicaset 으로 확인 가능 (rs)도 가능

디플로이먼트 리소스 - 배포를 책임진다 !

배포하는 과정을 책임짐. 버전의 업그레이트 전략 히스토리 롤백 롤아웃기능을 제공

디플로이먼트가 레플리카셋 만들고 팟을 만듬.

recreate : 기본에꺼 삭제 새로 만듬

RollingUpdate : 조금씩 새로만들면서 삭제

max unaba

apiVersion: apps/v1

kind: Deployment

metadata:

name: mynginx-deploy

spec:

replicas: 5

strategy:

type: RollingUpdate

rollingUpdate:

maxUnavailabe: 25% ==> 최대 몇개의 파드가 종료되도 되는지 설정

maxSurge: 25% ===> 최대 몇개의 파드가 초과되도 되는지

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.7.0

deployment 확인 kubectl get deployment

kubectl get pod mynginx-deploy- ~~ -o yaml | grep image

kubectl rollout status deployment mnginx-deploy ==> 상태확인

디플로이 배포

kubectl apply -f deployment.yaml

이미지 변경

kubectl set image deployment mynginx-deploy nginx=nginx:1.9.1 --record

--record 붙이면 히스토리에 저장

업데이트 상태 확인

kubectl rollout status deployment mynginx-deploy

업데이트 히스토리 확인

kubectl rollout history deployment mynginx-deploy

롤백 기능

kubectl rollout undo deployment mynginx-deploy

scaling 기능

kubectl scale deplyment mynginx-deploy --replicas=5 --record

kubectl describe deployment mynginx-deploy

스캐일 업 스캐일 다운 정보도 볼수 있음.

이미지 변경 ! kubectl set image deployment mynginx-deploy nginx=nginx:1.9.1 --record

1.7.9 ==> 1.9.1

kubectl rollout history deployment mynginx-deploy

히스토리 까지 보기

kubectl set image deployment mynginx-deploy nginx=nginx:1.9.21 --record

1.2.21 은 잘못된 버전

이미지 풀 에러가 난다. 실제로 존재하지 않는 것이기 때문. 세개만 에러가 발생하고 나머지가 유지된 이유는 maxunabalable을 했기 때문에 기존 팟들이 유지되고 있다.

kubectl rollout undo deployment mynginx-deploy

예전버전으로 돌려주는 기능

kubectl rollout undo deployment mynginx-deploy --to-revision=1

하면 특정 구역으로 돌아갈수 있다.

데몬셋 - 모든 노드에 자동으로 실행됨.

사용자가 데몬셋을 생성하면 자동으로 노드의 개수만큼 동일한 팟이 실행되게 됨.

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: myapp

spec:

selector:

matchLabels:

name: myapp

template:

metadata:

labels:

name: myapp

spec:

containers:

- name: log

image: nginx

모든 노드에서 실행되는 Pod

< 사용처 >

로깅

리소스 사용량 수집

서버 설정값 일괄 적용

kubectl get daemonset 하면 전부 확인

Desired ==> 우리의 노드

Job

1회성 배치작업

데이터 수집

데이터 분석

기계학습

apiVersion: batch/v1

kind: Job

metadata:

name: myfirstjob

spec:

template::

spec:

containers:

- name: print-date

image: busybox

args: [”date”]

restartPolicy: OnFailure ==> 종료가되었을때 설정하는 옵션

backoffLimit: 2 ==> 몇번 리스타트 할것인가.

RestartPolicy

- Never ==> 어쩌다가 종료되면 끝

- OnFailure ==> 실패시 리스타트

- Always ==> 디플로이나 레플리카 셋 할때 팟이 죽거나 할때 관리를 위해 줌.

kubectl get job

kubectl get job myfirstjob -o yaml

지우면 팟도 사라짐

하다가 안되면 자동으로 팟도 사라짐

CronJob

주기성 배치작업

데이터 수집

데이터 분석

기계학습

apiVersion: batch/v1beta1

kind: CronJob

metadata:

name: hello

spec:

schedule “\* \* \* \* \*”

jobTemplate:

spec:

template:

spec:

containers:

- name: print-date

image: busybox

args: [”date”]

restartPolicy: OnFailure

kubectl get cronjob

간혹 실제 잡이 실행 안될수도 있다.

Secret 리소스 - 민감정보를 저장

암호화 데이터 저장

apiVersion: v1

kind: Secret

metadata:

name: mysecret

type: Opaque ==> 사용자의 크리덴셜정보를 저장하는 타입

data:

username: YWRtaW4=

password: MWYyZDF1MmU2N2Rm

echo -ne YWRtaW4= | base64 -d 하면 admin이 출력

똑같이 해독 가능.

etcd 데이터베이스에 저장되어있고 사용시에는 복호화 되어 사용된다.

각각 base64로 암호화 되어있다.

ConfigureMap 처럼 암호화해야할 정보를 저장

apiVersion: v1

kind: Secret

metadata:

name: default-token-n6zps

type: kubernetes.io/service-account-token

data:

ca.crt: LS0tLS1CRUdJTiBDRVJUS

namespace: ZGVmYXVsdA==

token: ZXlKaGJHY21PaUpTVXpJMU5pSXNJb

다른 쿠버네티스에 팟을 생성, 삭제 할때는 적절한 권한이 있어야하는데 팟의 토큰을 기준으로 정확히는 서비스 어카운트를 통하여 권한을 관리하며 서비스 어카운트를 통해 구분함.

활용법

apiVersion: v1

kind: Pod

metadata:

name: cat-env

spec:

containers:

- name: cat-env

image: k8s.gcr.io/busybox

command: [ “printenv” ]

args: [ “USER” ]

env:

- name: USER

valueFrom:

secretKeyRef:

name: mysecret

key: username

kubectl get secret 으로 확인 가능

로그로 팟 확인해보면 admin이 찍혀있음