쿠버네티스 학습

kubctl이란?

- 쿠버네티스에 내가 원하는 것을 요청할 때 쓰이는 명령어

kubectl 명령어 구조

- kubectl [command] [TYPE] [NAME] [flags]

command: 자원(object)에 실행할 명령(create,get,delete,edit 등)

TYPE: 자원의 타입(node, pod, service)

NAME: 자원의 이름(원하는대로 붙일 수 있음 소문자, 숫자, 대시문자 사용가능)

flags: 부가적으로 설정할 옵션 ( --help, -o options 등)

kubectl get pod webserver -o wide

kubectl commands

- kubtetl –-help

- kubectl command –help

- kubectl run <자원이름> <옵션>

- kubectl create -f obj.yaml

- kubectl apply -f obj,yaml

- kubectl get <자원이름> <객체이름>

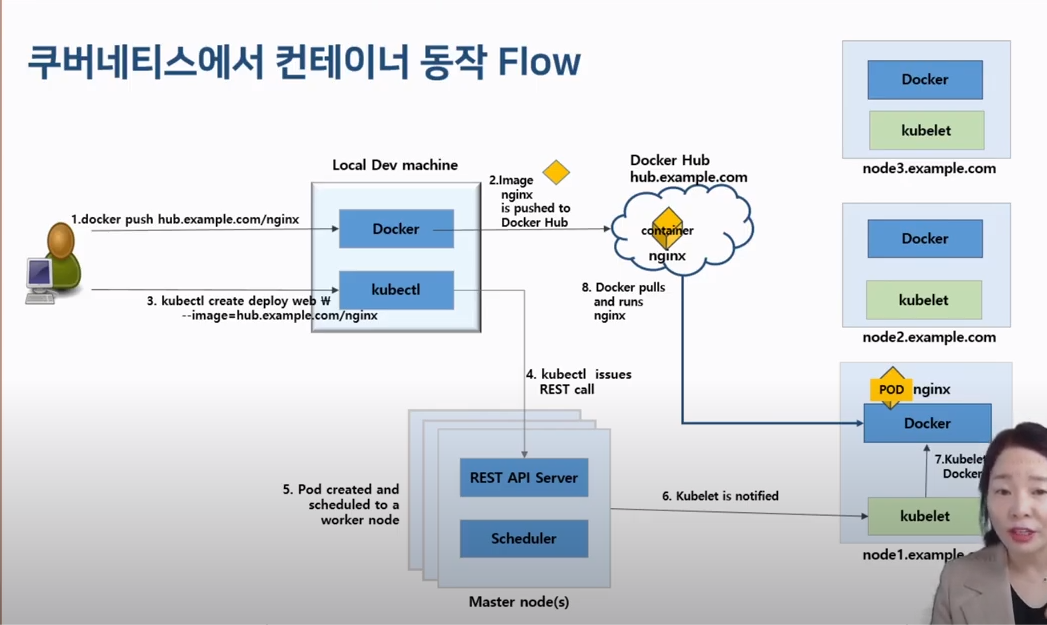
- kubectl edit <자원이름> <객체이름>

- kubectl describe <자원이름> <객체이름>

- kubectl delete pod main

쿠버네티스 아키텍처

1.Kubernetes 동작원리



컨테이너를 쿠버네티스 플랫폼에 올려서 사용하려는게 이 사용자의 목적이다. 먼저 컨테이너를 빌드 하겠죠. 예를들어 mainUI를 만들어 주는 컨테이너, login작업을 수행하는 컨테이너, 주문에 관련된 작업을 해주는 컨테이너 이런 컨테이너들을 쭉 만들어서 doker명령어로 저장로라고 불려지는 허브 어디에? 사내에 만들어져 있을 수 도 있고 또는 도커닷컴에서 만들어서 운영하는 퍼블릭 공간의 허브가 될 수 도 있다. 이렇게 허브가 있으면 만들어진 컨테이너를 도커 푸시 명령어를 이용해서 허브에 여러 컨테이너들을 저장해 놓는다. 이제 사용자는 쿠버네티스 명령어를 이용해서 쿠버네티스에게 이 컨테이너가 실행될 수 있도록 요청한다. 어떻게?? Yaml 파일이나 cli 명령어를 이용해서 가능 여기서는 cli명령어로 hube.example.com에 있는 nginx를 deploy로 실행해 주세요라고 kubectl명령어를 만들어서 마스터에게 보낸다. 이때 마스터라고 불리긴 하지만 control-plane이라고도 불린다. 마스터 시스템이 한대,두대, 여러대던지 어쨌든 마스터에게 보내면 api 서버가 존재해서 쿠버네티스 관련된 명령어 요청을 받아준다. 그런다음 이 api는 이 요청에 따라서 nginx라는 컨테이너를 node1,2,3 등 여러 대 중에 어디에다 배치하면 좋을지 scheduler에게 요청한다. 스케줄러는 현재 노드의 상태를 보고 어느 node가 가장 좋을지 선택한 다음 응답해 준다. 예를 들어 node1이 가장 좋을 때 이 api는 node` 시스템이 있는 kubelet에 요청한다. Node1의 kubelet아 너가 nginx하나를 실행해 줬으면 좋겠어 라고, 그러면 node1의 kubelet은 그 요청을 받아서 doker명령어로 바꿔서 도커 데몬에게 실제 컨테이너 실행 요청을 해주게 된다. 그러면 도커데몬은 정의가 되어 있는 hub.example.com에 nginx컨테이너가 있는지 검색한 다음 있으면 그것을 받아와서 컨테이너로 실행해 준다. 쿠버네티스는 이렇게 동작되는 컨테이너를 pod라는 단위로 관리하게 된다. 여기까지가 컨테이너를 빌드해서 저장하고 k8s 환경에서 실행 했을 때 동작되는 과정이다.

2. 쿠버네티스 컴포넌트

|  |  |
| --- | --- |
| 마스터 컴포넌트 | 워커 노드 컴포넌트 |
| etcd  - key-value 타입의 저장소 | kubelet  - 모든 노드에서 실행되는 k8s 에이전트  - 데몬 형태로 동작 |
| kube-apiserver  - k8s API를 사용하도록 요청을 받고 요청이 유효한지 검사 | kube-proxy  - k8s의 network 동작을 관리  - iptables rule을 구성 |
| kube-scheduler  - 파드를 실행할 노드 선택 | 컨테이너 런타임  - 컨테이너를 실행하는 엔진  - docker, containerd, runc |
| kube-contorller-manager  - 파드를 관찰하며 개수를 보장 |  |

3. 애드온 (출처: https://ikcoo.tistory.com/3 [이쿠의 슬기로운 개발생활])

|  |  |
| --- | --- |
| 네트워크 애드온 | CNI – weave,calico, flaneld, kube-route 등  컨테이너 간의 네트워킹을 제어할 수 있는 플러그인을 만들기 위한 표준  Kubernetes Cluster 내부는 Master Node에 의해 여러 컨테이너가 생성 삭제 복구를 반복하고 있음  그에 따라 각 컨테이너의 고정적이지 않고 재할당이 빈번함  이러한 특징을 해결하기 위해 Kubernetes Cluster는 가상 네트워크가 구성되어 있는데  기본적으로는 Worker Node의 kube-proxy 가 네트워크를 관리하지만  보다 효율적인 네트워크 환경을 구성하기 위해  다양한 네트워크 관련 Addon 이 제공됨 |
| DNS 애드온 | coreDNS  다른 Addon 은 필수적이지 않지만 DNS Addon 만큼은 필수적임.  실제로 Kubernetes Cluster 내에서 작동하는 DNS Server.  Kubernetes Service Object에게 DNS 레코드를 제공하는 역할을 수행함.  kubernetes 내부에서 실행된 컨테이너들은 자동으로 DNS 서버에 등록되어 자동 탐색이 가능해짐 |
| 대시보드 애드온 | kubernetes Cluster를 위한 Web 기반 UI.  일반적으로 kubernetse Cluster에 명령을 내릴 때 CLI에서 kubectl을 사용하는데  가시화하여 관리의 편의성을 제공하기 위해 대시보드를 제공함 |
| 컨테이너 애드온 | cAdvisor  어떤 시스템이던 가장 중요한 부분 중 하나는 바로 모니터링.  Kubernetes 환경에서도 모니터링을 제공하는 Addon이 존재함.  Cluster 내부에서 실행 중인 Container의 CPU, 메모리와 같은 리소스 데이터를  저장하고 볼 수 있는 방법을 제공하는 Addon. |
| 클러스터 로깅 | 컨테이너 로그, k8s 운영 로그들을 수집해서 중앙화  Kubernetes Cluster 내부에서 생성되는 모든 Log를 중앙화 하여 관리할 수 있음.  Master Node, Worker Node, Worker Node 내부의 Container 등 모든 로그를 중앙화 된  로그 수집 시스템에 모아서 볼 수 있도록 해주는 Addon.  클라우드 서비스를 이용 중이라면 각 클라우드 벤더에서 제공해주는 로깅 서비스를 사용하면 되지만  직접 kubernetes를 설치한 경우에는 Log 부분도 고려해야함  해당 addon 은 직접 kubernetes 를 설치한 경우에 사용하는 Addon.  ELK(ElasticSearch, Logstash, Kibana), EFK(ElasticSearch, Fluentd, Kibana),DataDog |

쿠버네티스 namespace

|  |
| --- |
| - Namespace는 쿠버네티스 API 종류 중 하나이다.  - 클러스터 하나를 여러 개의 논리적인 단위로 나눠서 사용  - 쿠버네티스 클러스터 하나를 여러 팀이나 사용자가 함께 공유  - 용도에 따라 실행해야 하는 앱을 구분할 때 사용 |
| namespace를 사용하면 좋은점  - 수많은 파드들이 존재하는데 필요한 것들을 namespace로 묶을 수 있어서 관리하기가 용이  - 여러 개의 버전을 namespace로 분류 가능 |
| namespace 생성  - CLI  $kubectl create namespace blue  $kubectl get namesapces  - Yaml  $kubectl create namespace green –dry-run -o yaml > green-ns.yaml  $vim green-ns.yaml  $kubectl create -f green-ns.yaml |
| namespace 관리  $kubectl get namespace  $kubectl delete namespace |
| namespace 생성 실습 내용  $kubectl get pod -n [namespace이름]  $kubectl create -f [오브젝트명]: 파일을 불러들여와 오브젝트를 생성한다.  $kubectl get pods -n kube-system: kube-system에서 동작중인 pod리스트를 볼 때 사용  $kubectl create namespace orange –dry-run -o yaml > orange-ns.yaml  $kubectl create -f orange-ns.yaml  --dry-run : 실제로 실행하지 않고 실행 가능 여부만 확인  -o yaml : 실행 결과를 yaml 파일로 출력  $kubectl create -f nginx.yaml -n blue : blue namepsace에 yaml파일 설정내용 |
| 사용할 네임스페이스 switch  기본으로 사용하는 namespace를 default가 아닌 다른 이름의 namespace로 switch  방법  1) namespace를 포함한 context 등록  $kubectl config --help  $kubectl config set-context NAME --cluster=Kubernetes …  $kubectl config view  2) 등록된 namespace로 context 변경  $kubectl config use-context NAME |
| 사용할 네임스페이스 switch 실습 내용  $kubectl config --help  $kubectl config view  $kubectl config set-context blue@kubernetes --cluster=kubernetes --user=kubernetes-admin –namespace=blue    $kubectl config current-context  $kubectl config use-context blue@kubernetes    $kubectl delete pods mypod -n default  $kubectl get pods -n default  $kubectl get pods -n blue  kubectl delete [오브젝트] : 오브젝트를 삭제  $kubectl delete namespaces blue |

yaml 템플릿

|  |
| --- |
| 사람이 쉽게 읽을 수 있는 데이터 직렬화 양식  기본문법  - 구조화된 데이터를 표현하기 위한 데이터 포맷  - python처럼 들여쓰기로 데이터 계층을 표기  - 들여쓰기를 할 때에는 Tab이 아닌 Space Bar를 사용  - 가독성이 좋아 설정 파일에 적합한 형식  - Scalar 문법 : ‘:’을 기준으로 key: value 를 설정  - 배열 문법: ‘-‘문자로 여러 개를 나열  - 공식 사이트: http:/yaml.org/  - Kubernetes yam example  $ cat nginx-pod.yaml |

API version

|  |  |
| --- | --- |
| alpha -> beta -> stable  kubernetes Object 정의 시 apiVersion이 필요  kubernetes가 update하는 API가 있으면 새로운 API가 생성됨 | |
| API Object 종류 및 버전 | |
| Deployment  Pod  ReplicaSet  ReplicationController  Service  PersistentVolume | apps/v1  v1  apps/v1  v1  v1  v1 |
| $kubectl explain pod 이용해서 version확인 가능 | |

Pod 학습내용

pod 개념 및 사용하기

livenessProbe를 사용한 self-healing Pod

init container

infra container(pause) 이해하기

static pod 만들기

Pod에 resource 할당하기

환경변수를 이용해 컨테이너에 데이터 전달하기

pod구성 패턴의 종류

**1. Pod 개념 및 사용하기**

|  |
| --- |
| Container 정리      container는 하나의 application 처럼 사용  쿠버네티스에서는 Pod라고하는 단위를 통해서 container를 동작시킴 |
| Pod란?  - 컨테이너를 표현하는 k8s API의 최소 단위    Pod에는 하나 또는 여러 개의 컨테이너가 포함될 수 있음 |
| pod 생성 하기 |
| 실습  $kubectl run web1 –image=nginx:1.14 --port=80    $cat pod-nginx.yaml  $kubectl create -f pod-nginx.yaml  $kubectl get pods  $kubectl get pods -o wide  $kubectl get pods web1  $kubectl get pods web1 -o yaml : 동작중인 web1 pod를 yaml파일로 보여주는 명령어  $ watch kubectl get pods -o wide  \*watch [명령어] : watch뒤의 명령어를 2초마다 한번씩 실행  $ curl 10.40.0.1    $kubectl get pods web1 -o json | grep -I podip : podip 내용 추출 |
| multi-container pod 생성하기 |
| 실습  $kubectl describe [오브젝트타입] [오브젝트명]  - 특정 또는 그룹의 오브젝트를 더욱 자세하게 보여주는 명령어  $kubectl dscribe pod web1  $kubectl create -f pod-multi.yaml    $kubectl describe pod multipod  $kubectl exec multipod -c nginx-container -it -- /bin/bash  $cd /usr/share/nginx/html/  $cat index.html  $echo “TEST web” > index.html  $exit  $curl 10.46.0.1  $kubectl exec multipod -c centos-container -it -- /bin/bash  $ps -ef  $curl localhost:80 🡪 동일한 ip를 가지고 web-server가 nginx를 송출하기 때문에 centos-container에도 웹접속됨  - multi-container pod에서 container들의 pod명과 ip는 동일  $kubectl logs multipod -c nginx-container  $kubectl logs [pod명] -c [container명] : [pod명]안의 [container명]의 로그를 출력  - 멀티 컨테이너는 container명까지 입력해야 하지만 싱글 컨테이너는 pod명까지만 입력하면됨 |
| Pod 동작 flow |
| Pod 관리하기  동작중인 파드 정보 보기  $kubectl get pods  $kubectl get pods -o wide  $kubectl describe pod webserver  동작중인 파드 수정  $kubectl edit pod webserver  동작중인 파드 삭제  $kubectl delete pod webserver  $kubectl delete pod --all |

|  |
| --- |
| Question & Answer  1. 현재 namespace에서 동작중인 Pod는 몇 개인가?  $kubectl get pods  2. 현재 시스템에서 동작중인 Pod 수는?  $kubectl get pods --all-namespaces  3. 컨테이너 nginx를 실행하는 nginx-pod라는 이름의 Pod를 생성하시오.  $kubectl run nginx-pod --image=nginx  4. 앞에서 생성한 Pod의 image정보를 확인하는 명령은 무엇인가?  $kubectl describe pod nginx-pod  5. 앞에서 생성한 nginx-pod는 어느 node에 배치되었나?  6. 앞에서 생성한 Pod에는 몇 개의 컨테이너가 포함되어 있나?  $kubectl get pods 의 Ready상태를 보면 알 수 있다. 또는  $kubectl describe pod [pod명]으로 일일이 확인  7. 앞에서 생성한 Pod의 현재 상태는 어떠한가?  $kubectl get pods 의 Ready상태를 보면 알 수 있다. 또는  $kubectl describe pod [pod명]으로 일일이 확인  8. 새 Pod의 컨테이너 상태는 어떻습니까?  describe 명령어로 확인 가능  9. ‘kubectl get pods’명령의 출력에서 READY 열은 무엇을 의미하나?  현재 준비완료인 것 / 전체 Pod 수  10. 생성한 pod를 삭제하시오.  $kubectl delete pod nginx-pod  11. 컨테이너 image ‘redis123’을 실행하는 pod ‘redis’를 redix.yaml 을 이용해서 생성하시오.  $kubectl run redis --image=redis123 –dry-run o yaml > redis.yaml  redis.yaml 수정  $kubectl create -f redis.yaml  12. 앞서 만든 redis pod의 image를 redis로 수정하여 동작시키시오.  $kubectl edit pod redis -> image를 redis로 수정 후 저장 |

**2. livenessProbe를 이용해 self-healing Pod(kubelet으로 컨테이너 진단하기)**

|  |  |
| --- | --- |
| Liveness Probe  Pod가 계속 실행할 수 있음을 보장  Pod의 spec에 정의  periodSeconds: health check 반복 실행 시간(초)  initialDelaySeconds: Pod 실행 후 delay할 시간(초)  timeoutSeconds: health check후 응답을 기다리는 시간(초) | |
| **Pod-definition** | **livenessProbe definition** |
| apiVersion: v1  kind: Pod  metadata:  name: nginx-pod  spec:  containers:  - name: nginx-container  image: nginx:1.14 | apiVersion: v1  kind: Pod  metadata:  name: nginx-pod  spec:  containers:  - name: nginx-container  image: nginx:1.14  livenessProbe:  httpGet:  path: /  port: 80  initialDelaySeconds: 15  periodSeconds: 20  timeoutSeconds: 1  successThreshold: 1  failureThreshold: 3 |
| livenessProbe 매커니즘  **httpGet** probe : 지정한 IP주소, port, path에 HTTP GET 요청을 보내, 해당 컨테이너가 응답하는지를 확인한다. 반환코드가 200이 아닌 값이 나오면 오류, 컨테이너를 다시 시작한다.  livenessProbe:  httpGet:  path: /  port: 80  **tcpSocket** probe: 지정된 포트에 TCP연결을 시도. 연결되지 않으면 컨테이너를 다시 시작한다.  livenessProbe:  tcpSocket:  port: 22  **exec** probe: exec 명령을 전달하고 명령의 종료코드가 0이 아니면 컨테이너를 다시 시작한다.  livenessProbe:  exec:  command:  - ls  - /data/file | |
|  | |
| [문제풀이]  apiVersion: v1  kind: Pod  metadata:  name: liveness-exam  spec:  containers:  - name: busybox-container  image: busybox  args:  - /bin/sh  - -c  - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600  livenessProbe:  exec:  command:  - ls  - /tmp/healthy  initialDelaySeconds: 10  failureThreshold: 2  periodSeconds: 5  successThreshold: 1  timeoutSeconds: 1 | |

**3. init container를 적용한 Pod**

|  |
| --- |
| 앱 컨테이너 실행 전에 미리 동작시킬 컨테이너  본 Container가 실행되기 전에 사전 작업이 필요할 경우 사용  초기화 컨테이너가 모두 실행된 후에 앱 컨테이너를 실행 |

**4. infra container(pause) 이해하기**

|  |
| --- |
| Pod의 환경을 만들어주는 컨테이너 |
| 실습  $kubectl run webserver –image=nginx:1.14 –port=80  $ssh node2 (새로운 shell생성)  $docker ps : 현재 동작중인 컨테이너 정보를 출력하는 명령어 (새로 생성된 shell)  $kubectl delete pod webserver  $docker ps (새로 생성된 shell) |

**5. static Pod 만들기**

|  |
| --- |
| API에게 요청을 보내지 않음  kubelet Daemon에 의해 동작되는 Pod  **static container**  API 서버 없이 특정 노드에 있는 kubelet 데몬에 의해 직접 관리  /etc/kubernetes/manifests/ 디렉토리에 k8s yaml 파일을 저장 시 적용됨  Static pod 디렉토리 구성  # vi /var/lib/kubelet/config.yaml  …  staticPodPath: /etc/kubernetes/manifests  디렉토리 수정시 kubelet 데몬 재실행  #systemctl restart kubelet |
| 실습  Master 접속  $kubectl delete pod –all  Node2 접속  $cat /var/lib/kubelet/config.yaml  $cd /etc/Kubernetes/manifests/  $cat > nginx.yaml  apiVersion: v1  kind: Pod  metadata:  name: nginx-pod  spec:  containers:  - name: nginx-container  image: nginx:1.14  ports:  - containerPort:80  protocol: TCP  $vi /var/lib/kubelet/config.yaml  staticPodPath 원하는 곳으로 수정 저장  $systemctl restart kubelet  staticPodPath에 yaml파일 생성  master manifests에 yaml 파일을 생성하면 master가 아닌 node들 중 하나에 static Pod가 배치됨  static Pod 정리:  API 도움 없이 kubelet Daemon으로 Pod를 실행시킬 수 있고 그 디렉토리는 미리 static Pod path로 정의되어 있고 그 정의된 곳에 Pod yaml를 넣어주면 동작한다. |

**6.Pod에 리소스(cpu, memory) 할당하기**

|  |
| --- |
| Pod Resource 요청 및 제한  Resource Requests  파드를 실행하기 위한 최소 리소스 양을 요청  Resource Limits  파드가 사용할 수 있는 최대 리소스 양을 제한  Memory limit을 초과해서 사용되는 파드는 종료(OOM Kill)되며 다시 스케줄링 된다. |
| 실습  Master 접속  $cat > nginx-pod-resources.yaml  apiVersion: v1  kind: Pod  metadata:  name: nginx-pod-resources  spec:  containers:  - name: nginx-container  image: nginx:1.14  ports:  - containerPort: 80  protocol: TCP  resources:  requests:  memory: 500Mi  cpu: 1  $kubectl create -f pod-nginx-resources.yaml  $kubectl describe pod nginx-pod-resources  $kubectl delete pod –all  $gedit pod-nginx-resources.yaml  Requests를 limits로 수정후 저장  $kubectl create -f pod-nginx-resources.yaml  $kubectl describe pod nginx-pod-resources  Limits만 설정하면 requests도 동일한 값으로 설정됨  $gedit pod-nginx-resources.yaml  Limits를 requests로 수정하고 cpu 2로 수정  $ kubectl create -f pod-nginx-resources.yaml  Status가 Pending상태가되고 node에 pod가 할당 되지 않음 왜냐하면 가상머신이 2core이기 때문에 2core cpu가 확보가 안되는 상황이기 때문이다. |

**7. Pod의 환경변수 설정하기**

|  |
| --- |
| 환경변수  Pod내의 컨테이너가 실행될 때 필요로 하는 변수  컨테이너 제작 시 미리 정의  NGINX Dockerfile의 예  ENV NGINX\_VERSION 1.19.2  ENV NJS\_VERSION 0.4.3  Pod 실행 시 미리 정의된 컨테이너 환경변수를 변경할 수 있다. |
| 실습  $kubectl delete pod –all  $ gedit pod-nginx-env.yaml  apiVersion: v1  kind: Pod  metadata:  name: nginx-pod-env  spec:  containers:  - name: nginx-container  image: nginx:1.14  ports:  - containerPort: 80  protocol: TCP  env:  - name: MYVAR  value: "testvalue"  $kubectl create -f pod-nginx-resources.yaml  $kubectl exec nginx-pod-env –it -- /bin/bash  $env |

**8. Pod 구성 패턴의 종류**

|  |
| --- |
|  |

Pod 운영 실습

|  |
| --- |
|  |
| 1. Node1 서버 접속  $cd /etc/kubernetes/manifests/  $gedit mydb.yaml  apiVersion: v1  kind: Pod  metadata:  name: mydb-node01  spec:  containers:  - name: mydb  image: redis  ports:  - containerPort: 80  protocol: TCP  2. master 접속  $kubectl create namespace product  myweb.yaml 내용  apiVersion: v1  kind: Pod  metadata:  name: myweb  spec:  containers:  - image: nginx:1.14  name: myweb  env:  - name: DB  value: mydb  resources:  requests:  memory: 500Mi  cpu: 200m  limits:  memory: 1Gi  cpu: 1  $kubectl create -f myweb.yaml -n product  $kubectl get pods -n product  $kubectl describe pod myweb -n product |

컨트롤러 학습내용

ReplicationController

ReplicaSet

Deployment

DaemonSet

StatefulSet

Job

CronJob

Controller란

|  |
| --- |
| Pod의 개수를 보장    Controller 종류 |

**1.ReplicationController**

|  |  |
| --- | --- |
| 요구하는 Pod의 개수를 보장하며 Pod 집합의 실행을 항상 안정적으로 유지 하는 것을 목표  요구하는 Pod의 개수가 부족하면 template를 이용해 Pod를 추가  요구하는 Pod 수 보다 많으면 최근에 생성된 Pod를 삭제  기본 구성  selector  replicas  template    ReplicationController 동작원리 | |
| ReplicationController definition | |
| **Pod-definition** | **ReplicationController-definition** |
| apiVersion: v1  kind: Pod  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image: nginx:1.14 | apiVersion: v1  kind: RelicationController  metadata:  name: rc-nginx  spec:  replicas: 3  selector:  app: webui  template:  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image: nginx:1.14 |
| 실습    $watch kubectl get pods -o wide  $kubectl create -f rc-nginx.yaml  $kubectl get replicationcontrollers  $kubectl get rc  $kubectl describe rc rc-nginx  $kubectl run redis –image=redis --dry-run  $kubectl run redis –image=redis --labels=app=webui --dry-run  $ kubectl run redis –image=redis --labels=app=webui --dry-run > redis.yaml  $vi redis.yaml 수정  apiVersion: v1  kind: Pod  metadata:  labels:  app: webui  name: redis  spec:  containers:  - image: redis  name: redis  $kubectl get pod --show-labels  $kubectl create -f redis.yaml 하면 동작 될까요 안될까요?  -> 만들어지자마자 중지되고 사라짐  $kubectl edit rc rc-nginx  replicas 3을 4로 바꾸고 저장  -> pod 1개 더 생성됨 scale out  $kubectl scale rc rc-nginx --replicas=2  -> scale down  $kubectl edit rc rc-nginx  image: nginx:1.14 를 nginx:1.15 수정 저장 nginx버전이 바뀐다 안바뀐다?  -> 변화없음 why? Controller는 selecotr만 보기 때문이다. 그런데  $kubectl describe pod rc-nginx-6kcmj 로 버전 정보 확인하면 nginx 1.14 이다  이때  $kubctl delete pod rc-nginx-6kcmj를 하면 pod하나가 삭제되고 새로 생성되는데 이때 생성된  Pod는 nginx: 1.15버전이다  서비스 중간에 버전을 바꿔주는 것을 롤링업데이트라고 한다. | |
| Question & Answer  1. 다음의 조건으로 ReplicationController를 사용하는 rc-lab.yaml 파일을 생성하고 동작시킵니다.  -labels(name: apache, app:main, rel:stable)를 가지는 httpd:2.2 버전의 Pod를 2개 운영합니다  - rc name : rc-mainui  - controller: httpd:2.2  - 현재 디렉토리에 rc-lab.yaml 파일이 생성되어야 하고, 애플리케이션 동작은 파일을 이용해 실행 합니다.  rc-lab.yaml  apiVersion: v1  kind: ReplicationController  metadata:  name: rc-mainui  spec:  replicas: 2  selector:  name: apache  app: main  rel: stable  template:  metadata:  name: http-pod  labels:  name: apache  app: main  rel: stable  spec:  containers:  - name: rc-mainui  image: httpd:2.2  2. 동작되는 http:2.2 버전의 컨테이너를 3개로 확장하는 명령을 적고 실행하세요  $kubectl edit rc rc-mainui  replicas 를 3으로 변경  또는  $kubectl scale rc rc-mainui --replicas=3 | |

**2. ReplicaSet**

|  |  |
| --- | --- |
| ReplicationController 와 같은 역할을 하는 컨트롤러  ReplicationController 보다 풍부한 selector  selector:  matchLabels:  component: redis  matchExpressions:  - {key: tier, operator: In, values: [cache]}  - {key: environment, operator: NotIn, values: [dev]}  matchExpressions 연산자  In: key와 values를 지정하여 key, value가 일치하는 Pod만 연결  NotIn: key는 일치하고 value는 일치하지 않는 Pod에 연결  Exists: key에 맞는 label의 pod를 연결  DoesNotExist: key와 다른 label의 pod를 연결 | |
| ReplicationController  spec:  replicas: 3  selector:  app: webui  version: “2.1”  temp..  ReplicaSet  spec:  replicas: 3  selector:  matchLabels:  app: webui  matchExpressions:  - {key: version, operator: In, values:[“2.1”,“2.2”]  temp.. | |
| **ReplicationController definition** | **ReplicaSet definition** |
| **apiVersion: v1**  **kind: ReplicationController**  metadata:  name: rc-nginx  spec:  replicas: 3  **selector:**  **app: webui**  template:  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image: nginx:1.14 | **apiVersion: apps/v1**  **kind: ReplicaSet**  metadata:  name: rc-nginx  spec:  replicas: 3  **selector:**  **matchLabels:**  **app: webui**  template:  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image: nginx:1.14 |
| ReplicaSet의 selector | |
| In | Exists |
| spec:  replicas: 3  **selector:**  **matchExpressions:**  **- {key: ver, operator: In, values: [“1.14”]}**  template: | spec:  replicas: 3  **selector:**  **matchExpressions:**  **- {key: ver, operator: Exists}**  template: |
| 실습  $kubectl delete rc rc-nginx  $cat rs-nginx.yaml  apiVersion: apps/v1  kind: ReplicaSet  metadata:  name: rs-nginx  spec:  replicas: 3  selector:  matchLabels:  app: webui  template:  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image: nginx:1.14  $kubectl create -f rs-nginx.yaml  $kubectl get pod --show-labels  $kubectl get replicaset  $kubectl get rs  $kubectl delete pod rs-nginx-jbmqh  -> 삭제됨과 동시에 하나 만들어짐  $kubectl scale rs rs-nginx --replicas=2  --replicas=[갯수]옵션 Replica의 개수를 수정하는 옵션  $kubectl delete rs rs-nginx --cascade=orphan  --casecade= orphan 옵션 연쇄 삭제 기능을 비활성화  $kubectl get rs  $kubectl get pods --show-labels  $cat rc-nginx.yaml 이때 rc-nginx.yaml을 실행하면 어떻게 될까요?  -> rs-nginx2개는 그대로 가지고 rc-nginx pod 한 개 생성 됨  $kubectl create -f rc-nginx.yaml | |
| rc.lab.yaml  apiVersion: apps/v1  kind: ReplicaSet  metadata:  name: rs-mainui  spec:  replicas: 2  selector:  matchLabels:  app: main  template:  labels:  name: apache  app: main  rel: stable  spec:  containers:  - image: httpd:2.2  name: httpd  $kubectl create -f rs-lab.yaml  kubectl scale rs rs-mainui --replicas=1 | |

**3. Deployment**

|  |
| --- |
| ReplicaSet을 제어해 주는 부모 역할  Deployment의 목적: Rolling Update  ReplicaSet을 컨트롤해서 Pod수를 조절  Rolling Update & Rolling Back    롤링 업데이트는 파드 인스턴스를 점진적으로 새로운 것으로 업데이트하여 디플로이먼트 업데이트가 서비스 중단 없이 이루어질 수 있도록 해준다. 새로운 파드는 가용한 자원을 보유한 노드로 스케줄될 것이다.      deploy-nginx.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  name: deploy-nginx  spec:  replicas: 3  selector:  matchLabels:  app: webui  template:  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image: nginx:1.14  $kubectl create -f deploy-nginx.yaml  $kubectl get pods -o wide  $kubectl get deploy,rs,pod  $kubectl delete pod deploy-nginx-6d4c4cc4b8-xdv5s  $kubectl get rs deploy-nginx-6d4c4cc4b8  $kubectl delete rs deploy-nginx-6d4c4cc4b8      kubectl set image deployment 롤링 업데이트 커맨드    $kubectl create -f deployment-exam1.yaml  $kubectl describe pod app-deploy-67bb978b99-m8smj  $kubectl rollout history deployment app-deploy  $kubectl delete -f deployment-exam1.yaml  $kubectl create -f deployment-exam1.yaml --record  --record 옵션 업데이트 과정을 history로 기록  $kubectl set image deploy app-deploy web=nginx:1.15 --record  $kubectl describe pod app-deploy-86bd8f9bb4-wcx5r  $kubectl set image deploy app-deploy web=nginx:1.16 --record  $kubectl rollout status deployment app-deploy  $kubectl set image deploy app-deploy web=nginx:1.17 --record  $kubectl rollout pause deployment app-deploy  kubectl rollout(pause / resume) 업데이트 일시정지 / 재시작  $kubectl set image deploy app-deploy web=nginx:1.18 --record  $kubectl rollout pause deployment app-deploy  $kubectl rollout resume deployment app-deploy  $kubectl rollout history deployment app-deploy    maxSurge  replicas=3 50%=1.5 => 3+2 = 5 5개까지 pod 유지  replicas=3 25%=0.75 => 3+1 = 4 4개까지 pod 유지  -> update속도를 이걸 통해서 custom가능  maxUnavailable  terminating되는 갯수를 조절  $kubectl rollout --help  $kubectl rollout undo deployment app-deploy --to-revision=3  kubectl rollout undo 업데이트 롤백 커맨드 (history 기준 한 단계 전)  --to-revision=[REVISION 값] 옵션 [REVISION 값]으로 롤백  $kubectl describe pod app-deploy-557d44d98f-qs96z  $kubectl rollout history deployment app-deploy  ->12456존재 3번자리가 붕떠서 6으로 이동한 것  $kubectl rollout undo deployment app-deploy  -> 1.18로 rollback  $kubectl rollout history deployment app-deploy  ->12467  $kubectl rollout undo deployment app-deploy  -> 1.16로 rollback  $kubectl rollout history deployment app-deploy  ->12478  $kubectl delete deployments.apps app-deploy  $kubectl apply -f deployment-exam2.yaml  더보기 apply vs create  $kubectl describe pod deploy-nginx-67bb978b99-svjg7  $vi deployment-exam2.yaml  annotation 1.15 image 1.15로 수정 후 저장  $kubectl apply -f deployment-exam2.yaml  -> 버전 업데이트 됨  $kubectl describe pod deploy-nginx-86bd8f9bb4-lz77s  $kubectl rollout history deployment deploy-nginx  $kubectl delete deployments.apps deploy-nginx    1. kubectl apply -f dep-lab.yaml  dep-lab.yaml 내용  apiVersion: apps/v1  kind: Deployment  metadata:  name: dep-mainui  annotations:  kubernetes.io/change-cause: version 2.2  spec:  progressDeadlineSeconds: 600  revisionHistoryLimit: 10  strategy:  rollingUpdate:  maxSurge: 25%  maxUnavilable: 25%  type: RollingUpdate  replicas: 2  selector:  matchLabels:  app: main  template:  metadata:  labels:  name: apache  app: main  rel: stable  spec:  containers:  - name: web  image: httpd:2.2  ports:  - containerPort: 80  2. kubectl apply -f dep-lab.yaml  dep-lab.yaml 내용 : 1번의 dep-lab.yaml 파일에서 annotation 과 image version 만 2.4 로 바꿔주면 됩니다.  3. 히스토리확인 : kubectl rollout history deployment dep-mainui  rollback : kubectl rollout undo deployment dep-mainui  4. kubectl describe pod dep-mainui-59877787b9-lrvfh  롤백 되어서 2.2로 확인됩니다. |

**4. DaemonSet**

|  |  |
| --- | --- |
| 전체 노드에서 Pod가 한 개씩 실행되도록 보장  로그 수집기, 모니터링 에이전트와 같은 프로그램 실행 시 적용  롤링 업데이트, 롤백도 지원 | |
| DaemonSet definition | |
| **ReplicaSet definition** | **DaemonSet definition** |
| **apiVersion: apps/v1**  **kind: ReplicaSet**  metadata:  name: rs-nginx  spec:  **replicas: 3**  selector:  matchLabels:  app: webui  template:  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image:nginx:1.14 | **apiVersion: apps/v1**  **kind: DaemonSet**  metadata:  name: daemonset-nginx  spec:    selector:  matchLabels:  app: webui  template:  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image: nginx:1.14 |
| $kubectl get nodes  $kubectl delete nodes node3.example.com  $kubectl get nodes  $cat daemonset-exam.yaml  apiVersion: apps/v1  kind: DaemonSet  metadata:  name: daemonset-nginx  spec:  selector:  matchLabels:  app: webui  template:  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image: nginx:1.14  $kubectl create -f daemonset-exam.yaml  $kubeadm token list 현재 보유한 token 출력  $kubeadm token delete 토큰이름  $kubeadm token create --ttl 1h  -> kubeadm token create --ttl [시간] [시간] 만큼의 유효기간을 가진 토큰 생성  node3 접속  $kubeadm reset  $kubeadmin join 10.100.0.104:6443 --token b1h06o.0cc9hlev4bbnplcg --discovery-token-ca-cert-hash sha256:649c9e5d5675e3bb0f0f017e541a9560cdc75e967233606963704a54bcc721af  마스터  $kubectl get nodes  $kubectl delete pod daemonset-nginx-jp48h  $kubectl get daemonsets.apps  $kubectl edit daemonsets.apps daemonset-nginx  -> nginx 버전 1.15로 수정 저장  $ kubectl describe pod daemonset-nginx-tzn5m  $ kubectl rollout undo daemonset daemonset-nginx | |

**5. Statefulset**

|  |  |
| --- | --- |
| Pod의 상태를 유지해주는 컨트롤러  Pod 이름  Pod의 볼륨(스토리지) | |
| $create -f rc-nginx.yaml  $kubectl delete pod rc-nginx-vvwgp | |
| StatefulSet definition | |
| ReplicaSet definition | StatefulSet definition |
| **apiVersion: apps/v1**  **kind: RelicaSet**  metadata:  name: rs-nginx  spec:  **replicas: 3**  selector:  matchLabels:  app: webui  template:  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image: nginx:1.14 | **apiVersion: apps/v1**  **kind: StatefulSet**  metadata:  name: sf-nginx  spec:  **replicas: 3**  **serverName: sf-nginx-service**  selector:  matchLabels:  app: webui  template:  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image: nginx:1.14 |
| $cat statefulset-exam.yaml  apiVersion: apps/v1  kind: StatefulSet  metadata:  name: sf-nginx  spec:  replicas: 3  serviceName: sf-service  # podManagementPolicy: OrderedReady  podManagementPolicy: Parallel  selector:  matchLabels:  app: webui  template:  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image: nginx:1.14  $kubectl create -f statefulset-exam.yaml  $kubectl delete pod sf-nginx-1  -> 이름을 그대로 보장하지만 위치는 보장못함  $kubectl scale statefulset sf-nginx --replicas=4  ->kubectl scale [object] [pod명] --replicas object의 repllicas수를 scale하는 명령어  $kubectl scale statefulset sf-nginx --replicas=2  $kubectl edit statefulsets.apps sf-nginx  -> nginx 1.15로 수정 저장 하면 롤링업데이트 됨  $kubectl describe pod sf-nginx-1  $kubectl rollout undo statefulset sf-nginx 롤백됨  $kubectl delete rc rc-nginx  $kubectl delete statefulsets.apps sf-nginx | |

**6, Job**

|  |
| --- |
| kubernetes는 Pod를 running 중인 상태로 유지  Batch 처리하는 Pod는 작업이 완료되면 종료됨  Batch 처리에 적합한 컨트롤러로 Pod의 성공적인 완료를 보장  비정상 종료 시 다시 실행  정상 종료 시 완료 |
| $kubectl run testpod --image=centos:7 --command sleep 5  $kubectl delete pod testpod |
| Job definition |
| **apiVersion: apps/vi**  **kind: Job**  metadata:  name: **job-example**  spec:  template:  spec:  containers:  - name: centos-container  image: centos:7  command: [“bash”]  args:  - “-c”  - “echo ‘Hello World’; sleep 50; echo ‘Bye’”  **restartPolicy: Never**      $kubectl create -f job-exam.yaml  $kubectl delete pod centos-job-qcc7h |

**7. CronJob**

|  |  |
| --- | --- |
| job컨트롤러로 실행할 Application Pod를 주기적으로 반복해서 실행  Linux의 cronjob의 스케쥴링 기능을 Job Controller에 추가한 API  다음과 같은 반복해서 실행하는 Job을 운영해야 할 때 사용  Data Backup  Send email  Cleaning tasks  Cronjob Schedule: “0 3 1 \* \*”  Minutes (from 0 to 59)  Hours (from 0 to 23)  Day of the month (from 1 to 31)  Month(from 1 to 12)  Day of the week (from 0 to 6)  분 시 일 월 요일 | |
| CronJob definition | |
| Job definition | CronJob definition |
| apiVersion: batch/v1  kind: Job  metadata:  name: centos-job  spec:  template:  spec:  containers:  - name: hello  image: busybox  command: [“bash”]  args:  - “-c”  - “echo ‘Hello’; sleep 5; echo ‘Bye’”  restartPolicy: Never | apiVersion: batch/v1  kind: CronJob  metadata:  name: cronjob-definition  spec:  schedule: “0 3 1 \* \*”  jobTemplate:  spec:  template:  spec:  containers:  - name: hello  image: busybox  args:  - /bin/sh  - -c  - date; echo Hello  restartPolicy: Never |
| $kubectl create -f cronjob-exam.yaml  $kubectl get cronjobs.batch -o yaml  $kubectl delete cronjobs.batch cronjob-exam | |

**서비스 학습내용**

서비스 개념

서비스 타입

서비스 사용하기

헤드리스 서비스

kube-proxy

**1. Kubernetes Service의 개념**

|  |  |
| --- | --- |
| 동일한 서비스를 제공하는 Pod 그룹의 단일 진입점을 제공 | |
| Service definition | |
| **Deployment-definition** | **Service-definition** |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: webui  spec:  replicas: 3  selector:  matchLabels:  app: webui  template:  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image: nginx:1.14 | apiVersion: v1  kind: Service  metadata:  name: webui-svc  spec:  clusterIP: 10.96.100.100  selector:  app: webui  ports:  - protocol: TCP  port: 80  targetPort: 80 |

**2. Kubernetes Service 타입**

|  |
| --- |
| 4가지 Type 지원  ClusterIP(default)  Pod 그룹의 단일 진입점(Virtual IP) 생성  NodePort  ClusterIP가 생성된 후  모든 Worker Node에 외부에서 접속가능 한 포트가 예약  LoadBalancer  클라우드 인프라스트럭처(AWS, Azure, GCP 등)나 오픈스택 클라우드에 적용  LoadBalancer를 자동으로 프로 비전하는 기능 지원  ExternalName  클러스터 안에서 외부에 접속 시 사용할 도메인을 등록해서 사용  클러스터 도메인이 실제 외부 도메인으로 치환되어 동작 |

**3. Kubernetes Service 사용하기**

|  |
| --- |
| ClusterIP  selector의 label가 동일한 파드들의 그룹으로 묶어 단일 진입점(Virtual\_IP)을 생성  클러스터 내부에서만 사용가능  type 생략시 default 값으로 10.96.0.0/12 범위에서 할당됨    deploy-nginx.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  name: webui  spec:  replicas: 3  selector:  matchLabels:  app: webui  template:  metadata:  name: nginx-pod  labels:  app: webui  spec:  containers:  - name: nginx-container  image: nginx:1.14  $kubectl create -f deploy-nginx.yaml  clusterip-nginx.yaml  apiVersion: v1  kind: Service  metadata:  name: clusterip-service  spec:  type: ClusterIP  clusterIP: 10.100.100.100  selector:  app: webui  ports:  - protocol: TCP  port: 80  targetPort: 80  $kubectl get service  $kubectl describe svc clusterip-service  $curl 10.100.100.100  $kubectl exec webui-6d4c4cc4b8-2xbt9 -it -- /bin/bash  $cd /usr/share/nginx/html/  $echo "webui #1" > index.html  $exit  나머지들도 반복  $kubectl describe svc clusterip-service  $curl 10.100.100.100  $kubectl scale deployment webui --replicas=5  $kubectl describe svc clusterip-service |
| NodePort  모든 노드를 대상으로 외부 접속 가능한 포트를 예약  Default NodePort 범위: 30000-32767  ClusterIP 를 생성 후 NodePort를 예약  즉, 외부에서 접속할 수 있도록 도움    $kubectl delete service --all  $kubectl create -f nodeport-nginx.yaml  $kubectl get svc  node1 접속  $netstat -napt | grep 30200 |
| LoadBalancer  Public 클라우드(AWS,Azure, GCP 등)에서 운영가능  LoadBalancer를 자동으로 구성 요청  NodePort를 예약 후 해당 nodeport로 외부 접근을 허용      $kubectl delete svc --all  $kubectl create -f loadbalancer-nginx.yaml  $kubectl delete svc --all |
| ExternalName  클러스터 내부에서 External(외부)의 도메인을 설정  DNS를 지원    $kubectl create -f external-name.yaml  $kubectl get svc  $kubectl run testpod -it --image=centos:7  $curl externalname-svc.default.svc.cluster.local  $exit  $kubectl delete deployments.apps webui  $kubectl delete service externalname-svc |

**4. Kubernetes 헤드리스 서비스**

|  |
| --- |
| ClusterIP가 없는 서비스로 단일 진입점이 필요 없을 때 사용  Service와 연결된 Pod의 endpont로 DNS 레코드가 생성됨  Pod의 DNS 주소: pod-ip-addr.namespace.pod.cluster.local    $kubectl create -f deploy-nginx.yaml  $kubectl create -f clusterip-nginx.yaml  $kubectl create -f headless-nginx.yaml  $kubectl get svc  $kubectl describe svc headless-service  $kubectl run testpod -it --image=centos:7 /bin/bash  $cat /etc/resolv.conf  $curl 10-44-0-3.default.pod.cluster.local  $exit |

**5. kube-proxy 란?**

|  |
| --- |
| Kubernetes Service의 backend 구현  endpoint 연결을 위한 iptables 구성  nodePort로의 접근과 Pod 연결을 구현(iptables 구성) |
| 헤드리스 서비스랑 이어서 진행  $kubectl get pods -all-namespaces  $kubectl get svc  node1접속  $iptables -t nat -S | grep 80 |
| kube-proxy mode  userspace  클라이언트의 서비스 요청을 iptables를 거쳐 kube-proxy가 받아서 연결  Kubernetes 초기버전에 잠깐 사용  iptables  default Kubernetes network mode  kube-proxy는 service API 요청 시 iptables rule 이 생성  클라이언트 연결은 kube-proxy가 받아서 iptables 룰을 통해 연결  IPVS  리눅스 커널이 지원하는 L4 로드밸런싱 기술을 이용  별도의 ipvs 지원 모듈을 설정한 후 적용가능  지원 알고리즘: rr(round-robin), lc(least connection), dh(destination hashing), sh(source hashing), sed(shortest expected delay) nq(never queue) |

**인그레스**

인그레스의 이해

인그레스 컨트롤러 설치

실습: 인그레스를 이용해 웹서비스 운영

**1.Kubernetes Ingress**

|  |
| --- |
| HTTP나 HTTPS를 통해 클러스터 내부의 서비스를 외부로 노출  기능  Service에 외부 URL을 제공  트래픽을 로드밸런싱  SSL 인증서 처리  Virtual hosting을 지정 |

**2. Ingress Controller 설치**

|  |
| --- |
| $wget https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.48.1/deploy/static/provider/baremetal/deploy.yaml  $kubectl create -f deploy.yaml  $kubectl get pod -n ingress-nginx  $kubectl get svc -n ingress-nginx |

**3. Ingress를 이용한 웹서비스 운영**

|  |
| --- |
| $kubectl delete -f deploy.yaml  $gedit deploy.yaml  Nodeport 설정 쪽에  다음과 같이 수정  $kubectl create -f deploy.yaml  $kubectl get namespaces  $kubectl get all -n ingress-nginx    Default namespace 치환 : kubectl config  $kubectl config --help  $kubectl config view  $kubectl config set-context ingress-admin@kubernetes --cluster=Kubernetes user=Kubernetes-admin --namesapce=ingress-nginx  $kubectl config view  $kubectl config use-context ingress-admin@kubernetes  $kubectl config current-context  $kubectl get all  $kubectl apply -f marvel-home.yaml -f pay.yaml |

레이블과 애너테이션

레이블이란

워커 노드에 레이블 설정

레이블과 애너테이션

레이블을 이용한 카나리 배포

**1.Label**

|  |  |
| --- | --- |
| node를 포함하여 pod, deployment 등 모든 리소스에 할당  리소스의 특성을 분류하고, Selector를 이용해서 선택  Key-value 한쌍으로 적용 | |
| Label Template | |
| pod template | pod + label template |
| apiVersion: v1  kind: Pod  metadata:  name: appjs-pod  spec:  containers:  - name: appjs-container  image: smlinux/appjs  ports:  - containerPort: 8080 | apiVersion: v1  kind: Pod  metadata:  name: appjs-pod  labels:  app: web  version: “1.15”  spec:  containers:  - name: appjs-container  image: smlinux/appjs  ports:  - containerPort: 8080 |
| $kubectl get nodes  $kubectl run cmdpod --image=nginx:1.14 --port=80  $cat > pod1.yaml  apiVersion: v1  kind: Pod  metadata:  name: lable-pod-demo  spec:  containers:  - name: nginx  image: nginx:1.14  ports:  - containerPort: 80  $cp pod1.yaml pod2.yaml  pod2.yaml 내용  apiVersion: v1  kind: Pod  metadata:  name: pod-demo  labels:  name: mainui  rel: stable  spec:  containers:  - name: nginx  image: nginx:1.14  ports:  - containerPort: 80  $kubectl create -f pod1.yaml -f pod2.yaml  $kubectl get pods  $kubectl get pods --show-labels  $kubectl get pods -l name=mainui  kubectl get pods -l name=[label명] label 명이 같은 pod 를 출력(selector)  $kubectl get pods --selector name=mainui  $kubectl delete pods --selector name=mainui  $kubectl label pod pod-demo name=test  $kubectl get pods --show-labels  $kubectl label pod pod-demo name=login --overwrite  --overwrite 속성 이미 존재하는 label 위에 덮어쓰기  $kubectl get pods --show-labels  $kubectl label pod cmdpod name=order rel=beta  kye=value 형식에 맞춰 label 복수 할당 이미 있는 label 이라면 --overwirte 필수  $kubectl label pod cmdpod run-  label 명 뒤에 -(대시문자) label 삭제  $kubectl get pods --show-labels  $kubectl label pod pod-demo rel=stable  $kubectl get pods --show-labels  $kubectl get pods --selector rel=stable  $kubectl get pods --selector rel=beta  $kubectl delete pods --selector rel=beta | |

**2. 워커 노드에 Label 설정**

|  |
| --- |
| Node Label  Worker Node의 특성을 Label로 설정  kubectl label nodes <노드이름> <레이블 키>=<레이블 값>  노드를 선택해서 파드를 배치할 수 있다.      $kubectl get nodes  $kubectl get nodes --show-labels  $kubectl label nodes node1.example.com gpu=true disk=ssd  $kubectl label nodes node2.example.com gpu=true  $kubectl label nodes node3.example.com disk=ssd  $kubectl get nodes --show-labels  $kubectl get nodes -L disk,gpu  pod1.yaml  apiVersion: v1  kind: Pod  metadata:  name: pod-demo  spec:  containers:  - name: nginx  image: nginx:1.14  ports:  - containerPort: 80  $cp pod1.yaml nodeselector.yaml  $vi nodeselector.yaml  apiVersion: v1  kind: Pod  metadata:  name: pod-nodeselector  spec:  nodeSelector:  gpu: “true”  disk: ssd  containers:  - name: nginx  image: nginx:1.14  ports:  $kubectl create -f nodeselector.yaml  $kubectl get pods -o wide  $kubectl delete pod pod-nodeselector |

**3. 레이블과 애너테이션**

|  |
| --- |
| **Annotation**  Label과 동일하게 key-value를 통해 리소스의 특성을 기록  kubernetes에게 특정 정보 전달할 용도로 사용  예를들어 Deployment의 rolling update 정보 기록  annotation:  Kubernetes.io/change-cause: version 1.15  관리를 위해 필요한 정보를 기록할 용도로 사용  릴리즈, 로깅, 모니터링에 필요한 정보들을 기록  annotation:  builder: “seongmi Lee ([seongmi.lee@gmail.com](mailto:seongmi.lee@gmail.com))”  buildDate: “20210502”  imageRegistry: <https://hub.docker.com/>    annotation.yaml  apiVersion: v1  kind: Pod  metadata:  name: pod-annotation  annotations:  builder: “seongmi Lee (seongmi.lee@gmail.com)”  buildDate: “20210502”  imageRegistry: https://hub.docker.com/  spec:  containers:  - name: nginx  image: nginx:1.14  ports:  - containerPort: 80  $kubectl create -f annotation.yaml  $kubectl get pods  $kubectl describe pod pod-annotation |

**4. 레이블을 이용한 카나리 배포**

|  |
| --- |
| **Canary Deployment**  포드를 배포(업데이트)하는 방법  블루 그린 업데이트  카나리 업데이트  롤링 업데이트  Canary 배포  기존 버전을 유지한 채로 일부 버전만 신규 버전으로 올려서  신규 버전에 버그나 이상은 없는지 확인    $kubectl delete pod pod-annotation    $kubectl create -f mainui-stable.yaml  $kubectl get pods  $kubectl get pods --show-labels  port 80으로 변경  $kubectl create -f mainui-service.yaml  $kubectl get svc  $kubectl describe service mainui-svc  $curl 10.109.196.210    $kubectl create -f mainui-canary.yaml  $kubectl describe service mainui-svc  $kubectl get deployments.apps  $kubectl scale deployment mainui-canary --replicase=2  $kubectl delete deployments.apps mainui-canary |

**ConfigMap**

ConfigMap 생성

ConfigMap의 일부분을 적용하기

ConfigMap 전체를 적용하기

ConfigMap을 볼륨으로 적용하기

**1. ConfigMap 생성**

|  |
| --- |
| ConfigMap: 컨테이너 구성 정보를 한곳에 모아서 관리      $cat > text.file  This is a file |
| file은 단 1MiB를 초과할 수 없다.  $kubectl create configmap ttabae-config --from-literal=INTERVAL=2 --from-literal=OPTION=boy --from-file=config.dir/  kubectl create configmap [이름] configmap 생성하기  $kubectl describe get configmaps ttabae-config  $kubectl describe configmaps ttabae-config  $kubectl edit configmaps ttabae-config |

**2. ConfigMap의 일부분을 적용하기**

|  |
| --- |
| configMapRef:  name:ttabae-config로 수정 |