1. 기본 개념

1-1. Kubernetes

Kubernetes는 애플리케이션 컨테이너의 배포, 스케일링, 오퍼레이팅을 자동화 해주는 오픈 소스 플랫폼

1-2. Cluster

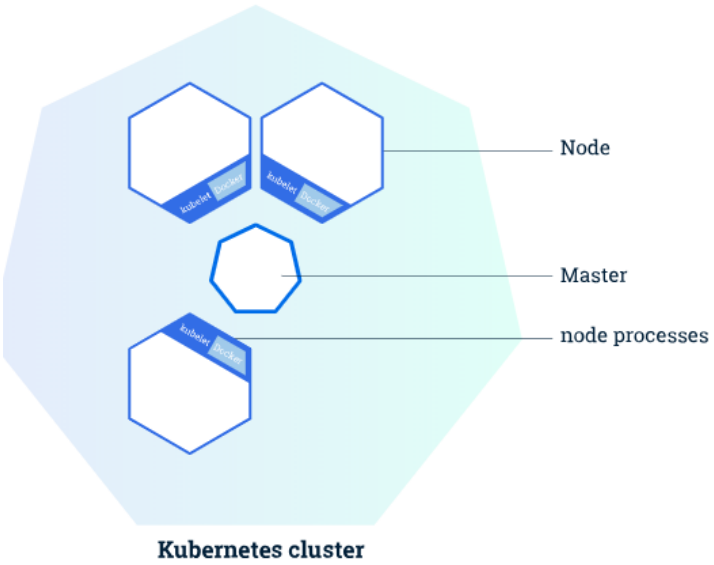
Kubernetes 내 추상 개념으로, 컨테이너 애플리케이션을 배포, 스케일링, 오퍼레이팅 할 수 있는 단위 환경. 마스터 노드와 일반 노드로 구성되어 있음

1) 마스터 노드

- Cluster 전체를 관리하는 주체
- 노드의 글로벌 이벤트를 감지하고 응답하는 등의 의사결정 수행

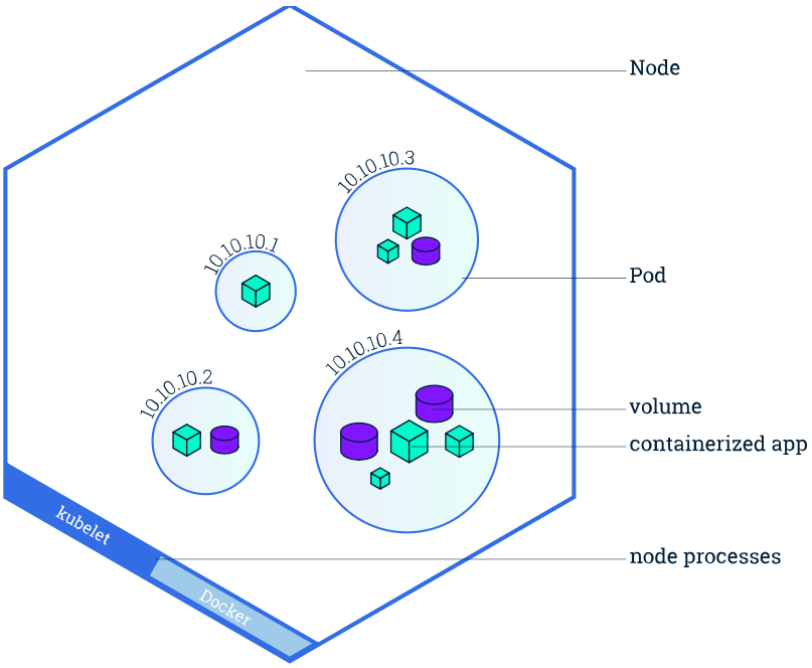
2) 일반 노드

- 가상 머신 또는 실제 머신을 의미
- Kubelet 에이전트를 통해 마스터 노드와 통신함
- 실제 컨테이너인 pod가 생성되는 곳



1-3. Pod object

kubernetes에서 관리되는 가장 작은 단위



1-4. Deployment object

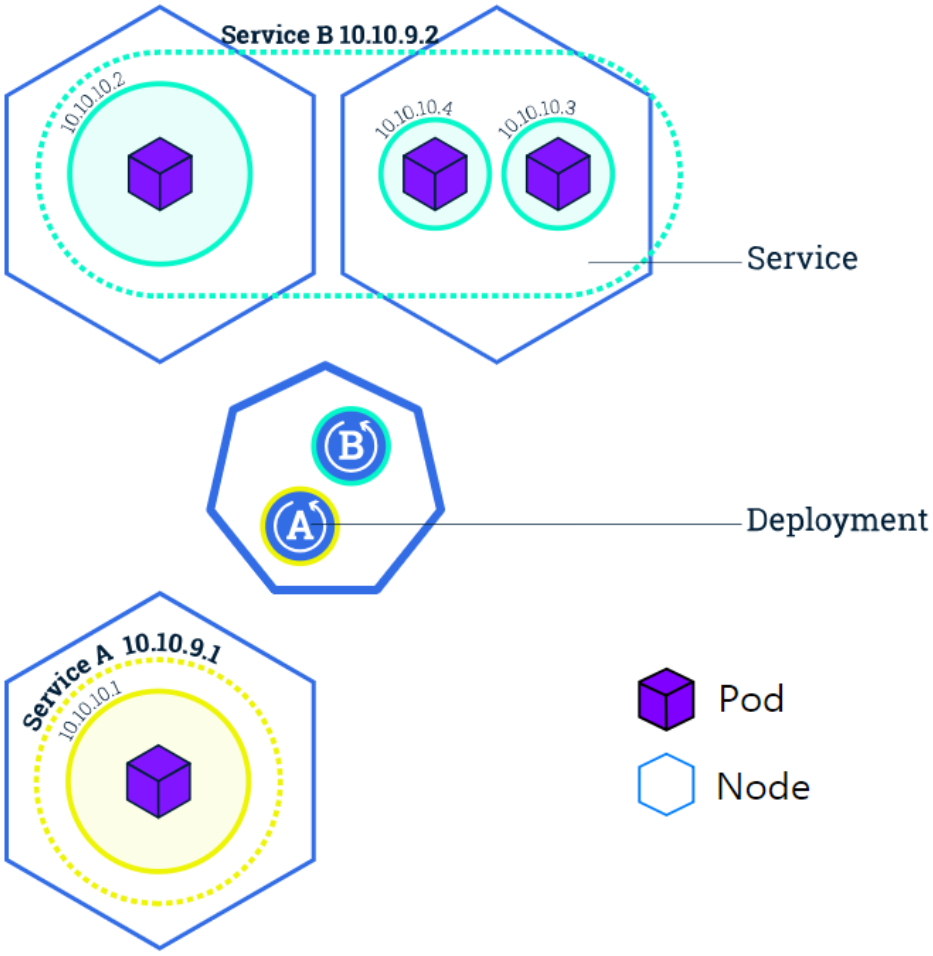
애플리케이션의 배포, 삭제, scale out 의 역할

Deployment를 생성하면 Deployment가 pod와 replicaSets를 함께 생성

Pod에 containerized app들이 포함되고, pod이 생성되면 애플리케이션이 배포되는 원리. ReplicaSets는 replica 개수를 모니터링하여 pod가 삭제되어 replica 개수가 맞지 않으면 지정된 replica 개수에 맞춰 pod를 생성함

1-5. Service object

Pod의 논리적 집합과 액세스 정책을 정의하는 추상화된 개념



2. 설치 및 설정

2-1. Docker

1) 설치 스크립트 실행

$ curl -fsSL https://get.docker.com/ | sudo sh

2) 현재 접속중인 사용자에게 권한주기

$ sudo usermod –aG docker $USER

3) 설치 버전 확인

$ sudo docker version

2-2. Kubernetes

- 동일한 설치 과정을 진행한 후, 마스터와 일반 노드의 개별 설정이 있음

2.1 동일 과정

1) Swap 비활성화

$ swapoff -a

2) 설치 명령어

$ sudo apt-get update && apt-get install -y apt-transport-https curl

$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -

$ cat <<EOF >/etc/apt/sources.list.d/kubernetes.list

$ deb http://apt.kubernetes.io/ kubernetes-xenial main

$ EOF

$ sudo apt-get update

$ sudo apt-get install kubelet kubeadm kubectl

3) Docker cgroup drive 확인

$ docker info | grep –i cgroup

4) /etc/system/system/kubelet.service.d/10-kubeadm.conf 수정

Environment="KUBELET_CGROUP_ARGS=--cgroup-driver=cgroupfs 추가

```
# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
Environment="KUBELET_CGROUP_ARGS=--cgroup-driver=cgroupfs"
```

5) Cgroup driver 변경

$ sed -i "s/cgroup-driver=systemd/cgroup-driver=cgroupfs/g"
/etc/systemd/system/kubelet.service.d/10-kubeadm.conf

2.2 마스터 노드

1) 마스터 초기화

$ kubeadm init

2) 클러스터 사용을 위한 명령어 실행
- root가 아닌 일반 유저로 실행

$ mkdir -p $HOME/.kube

$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

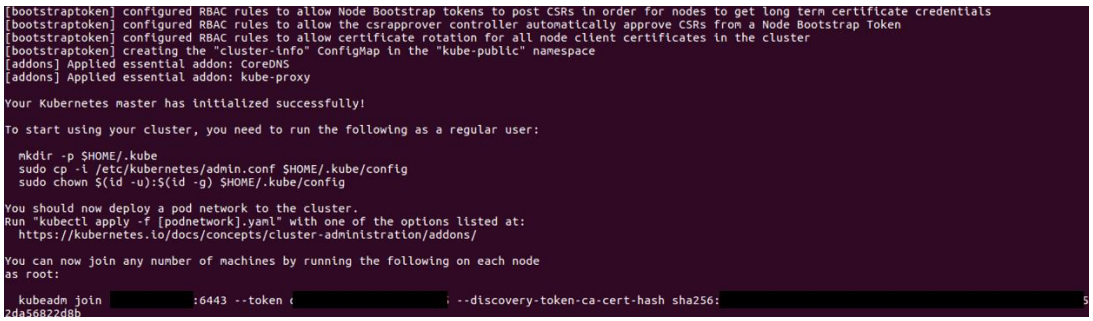
3) 네트워크 배포
- https://kubernetes.io/docs/concepts/cluster-administration/addons/ 에서 사용하고자 하는 네트워크를 선택
- 각 네트워크에서 제공하는 명령어 실행
- 본 설치에서는 'Weave Net' 으로 진행함

$ kubectl apply -f https://cloud.weave.works/k8s/net?k8s-
version=$(kubectl version | base64 | tr -d '₩n')

4) 조인 명령어 확인

- 마스터 초기화 과정이 끝나면 일반 노드가 마스터 노드에 조인하기 위한 명령어가 주어짐
- 해당 명령어는 일반 노드에서 실행

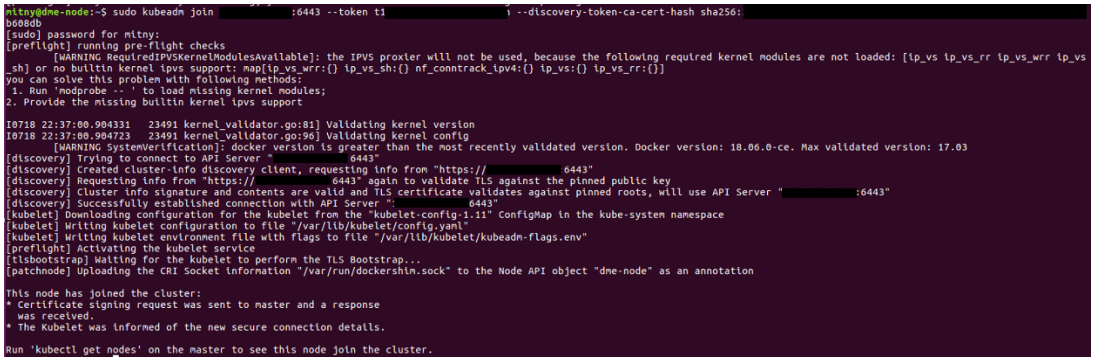
예시) kubeadm join 192.168.0.51:6443 --token value --discovery-token-ca-cert-hash sha256:value



2.3 일반 노드

1) 조인

- 설치와 설정이 완료된 일반 노드에서 마스터 초기화 시 출력된 조인 명령어 실행



2-3. Apache Thrift

3.1 설치

1) 필요한 라이브러리 설치

$ sudo apt-get install wget

$ sudo apt-get install ant

$ sudo apt-get install libboost-dev libboost-test-dev libboost-program-options-dev libboost-filesystem-dev libboost-thread-dev libevent-dev automake libtool flex bison pkg-config g++ libssl-dev

2) wget으로 Apache Thrift 설치 파일을 받아옴

$ wget http://apache.mirror.cdnetworks.com/thrift/0.11.0/thrift-0.11.0.tar.gz

```
mitny@ubuntu:~/Documents$ wget http://apache.mirror.cdnetworks.com/thrift/0.11.0
/thrift-0.11.0.tar.gz
--2018-07-04 01:39:34--  http://apache.mirror.cdnetworks.com/thrift/0.11.0/thrif
t-0.11.0.tar.gz
Resolving apache.mirror.cdnetworks.com (apache.mirror.cdnetworks.com)... 14.0.10
1.165
Connecting to apache.mirror.cdnetworks.com (apache.mirror.cdnetworks.com)|14.0.1
01.165|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3667154 (3.5M) [application/x-gzip]
Saving to: 'thrift-0.11.0.tar.gz'

thrift-0.11.0.tar.g 100%[===================>]   3.50M  3.94MB/s    in 0.9s

2018-07-04 01:39:35 (3.94 MB/s) - 'thrift-0.11.0.tar.gz' saved [3667154/3667154]
```

3) 압축 해제

$ tar –xvf thrift-0.11.0.tar.gz

4) make

$ cd thrift-0.11.0

$ ./bootstrap.sh

$ ./configure

$ sudo make

$ sudo make install

5) 설치 버전 확인

$ thrift –version

3. 쿠버네티스 명령어

No	명령어	기능
1	Kubectl version	쿠버네티스 버전 명시
2	kubeadm init	마스터 초기화
3	kubectl get {nodes,pods,services,deployments}	각 리소스들의 리스트 출력
4	kubectl get {node,pod,service,deployment} {name}	각 리소스 중 name에 대한 것만 출력
5	kubectl get {nodes,pods,services,deployments} --show-labels	각 리소스들의 리스트를 label과 함께 출력
6	kubectl describe {nodes,pods,services,deployments}	각 리소스들의 상태 출력
7	kubectl describe {node,pod,service,deployment} {name}	각 리소스 중 name에 대한 상태만 출력
8	kubectl describe {node,pod,service,deployment} -l {label=value}	value값의 label을 가진 리소스들의 상태 출력
9	Kubectl run {pod name} --image={image} --replicas={pod 개수} --port={port number}	Docker의 image 파일로 pod 생성
10	Kubectl expose deployment {name} –target-port={port number}	Service 생성
11	Kubectl exec {container ID} –ti bash	컨테이너 내부 접속
12	Kubectl delete {node,pod,service,delpoyment} {name}	각 리소스 중 name에 대해 삭제

4. Thrift 작성법과 빌드

4-1. 작성법

1) 타입

No	타입	설명
1	Bool	Boolean, one byte
2	Byte	Signed byte
3	i16	Signed 16-bit integer
4	i32	Signed 32-bit integer
5	i64	Signed 64-bit integer
6	Double	64-bit floating point value
7	String	String
8	Binary	Blob (byte array)
9	map<t1,t2>	Map from one type to another
10	list<t1>	Ordered list of one type
11	set<t1>	Set of unique elements of one type

2) 타입 정의

예시: typedef i32 int
typedef i64 long

3) .thrift 파일 작성 예시

namespace {작성 언어} {패키지명}
 	
service {서비스명} {
{타입} {메소드명}()
}
	Namespace py ping_pong
	
	Service ping_pong {
	String ping()
	}

4-2. 빌드

$ thrift --gen {작성 언어} {파일명}

예시: thrift --gen py ping_pong.thrift

5. Thrift 구현 방법

5-1. 서버

1.1 kuber.thrift

1) 코드

```
1   namespace cpp kuber
2
3   service DEManagement {
4           i32 add_dmEngine(1:string name,2:string fileName,3:i32 port),
5           i32 remove_dmEngine(1:string name),
6           i32 describe_dmEngine(1:string name),
7   }
```

2) 인터페이스 설명

① `i32 add_dmEngine(1:string name,2:string fileName,3:i32 port)`

pod를 생성하는 메소드.

생성할 pod의 이름과 사용할 이미지 파일의 파일명, 포트 번호를 받음

② `i32 remove_dmEngine(1:string name)`

deployment를 제거하는 메소드. 제거할 deployment의 이름을 받음

③ `i32 describe_dmEngine(1:string name)`

deployement의 상태를 조회하는 메소드.

조회하고자 하는 deployment의 이름을 받음

3) 빌드

$ thrift –gen cpp kuber.thrift

1.2 DEManagement_server.skeleton.cpp

1) 코드

① `int32_t add_dmEngine(const std::string& name, const std::string& fileName, const int32_t port)`

```
23      int32_t add_dmEngine(const std::string& name, const std::string& fileName, const int32_t port) {
24        // Your implementation goes here
25           char doCommand[500];
26           char command[500];
27           sprintf(doCommand,"docker import /home/kuber/DockerImages/%s %s:new",fileName.c_str(),name.c_str());
28           system(doCommand);
29           sprintf(command,"kubectl run %s --image=%s --replicas=1 --port=%d",name.c_str(),name.c_str(),port);
30           system(command);
31      }
```

Line 27: /home/kuber/DockerImages 내에 있는 파일로 도커 이미지 import

Line 28: 위 명령어 실행

Line 29: import 된 이미지와 매개변수 이름, 포트 번호로 pod 생성

Line 30: 위 명령어 실행

② `int32_t remove_dmEngine(const std::string& name)`

```
int32_t remove_dmEngine(const std::string& name) {
        char command[500];
        sprintf(command,"kubectl delete deployment %s",name.c_str());
        system(command);
}
```

③ `int32_t describe_dmEngine(const std::string& name)`

```
int32_t describe_dmEngine(const std::string& name) {
       char command[500];
       sprintf(command,"kubectl describe deployment %s",name.c_str());
       system(command);
}
```

2) 컴파일

$ g++ DEManagement_server.skeleton.cpp kuber_types.cpp DEManagement.cpp –o kuber –lthrift