AWS 쿠버네티스 적용기

목차

Bastion 서버 생성하기
    Bastion Host란
Eksctl 환경 설정
    kubectl 설치
    eksctl 설치
    aws cli 설치
    aws cli configuration 설정
    eksctl을 사용하여 reverse 클러스터를 위한 pair-key 생성
eksctl 클러스터 만들기
    eksctl 옵션

Bastion 서버 생성하기

AWS에 EKS 생성을 위해 `eksctl` 을 사용하려 한다.

Bastion Host란

퍼블릭 네트워크에서 프라이빗 네트워크에 대한 액세스를 제공하기 위한 목적을 가진 서버

Amazon VPC (Virtual Private Cloud) 의 Public Subnet 에 있는 Amazon EC2 인스턴스에서 실행됨

리눅스 인스턴스는 공개적으로 액세스할 수 없는 Subnet에 있으며,

Bastion Host를 실행하는 기본 EC2 인스턴스에 연결된 Security Group에 연결된 Security Group에서

Eksctl 환경 설정

```
eksctl create cluster
```

해당 커맨드의 경우 내 기본 지역을 토대로 m5.large 노드 2개를 포함하는 노드그룹을 자동으로 만들어준다.
클러스터가 만들어지고 나면 그것에 맞는 configuration이 kubeconfig file에 추가된다.
kubeconfig 파일은 ~/.kube/config에 있다.

eksctl cli를 실행시키는 환경은 laptop 보다는 aws 에 bastion 서버를 만들어서 그곳에서 실행하는 것을 추천한다. laptop에 이런 저런 환경을 옮겨가면서 작업하다보면 실수가 나오며 bastion 서버를 이용하면 언제든 내가 원할 때 들어가서 작업을 할 수 있다.

bastion 서버의 경우 인스턴스 타입은 t3.small 이어도 충분하다. 그리고 적절한 security group을 만들어서 서버로 들어간다.

kubectl 설치

kubectl을 linux에 맞게 다운로드 하고 권한 설정을 바꾼다.

```
curl -o kubectl https://s3.us-west-2.amazonaws.com/amazon-eks/1.23.7/2022-06-29/bin/linux/amd64/kubectl

chmod +x ./kubectl

// 바이너리를 PATH의 폴더에 복사한다. kubectl 버전이 이미 설치된 경우 $HOME/bin/kubectl을 생성하고 $HOME/bin이 $PATH로 시작하도록 해야 한다.
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin

// 셸 초기화 파일에 $HOME/bin 경로를 추가하면 셸을 열 때 구성한다.
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc

// kubectl을 설치한 후 다음 명령어를 사용하여 체크한다.
kubectl version --short --client
```

eksctl 설치

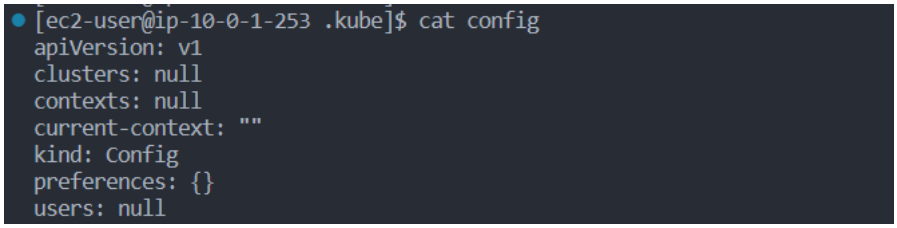
eksctl을 설치한다.

```
// eksctl 설치
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /

// 다운로드 받은 파일을 /usr/local/bin에 옮긴다.
sudo mv /tmp/eksctl /usr/local/bin

// 버전 확인
eksctl version
```

eksctl 설치 시, `~/.kube/config` 파일이 생성되었음을 확인할 수 있다.



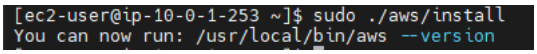
aws cli 설치

최신 버전의 AWS CLI 설치 또는 업데이트

이 주제에서는 지원되는 운영 체제에서 AWS Command Line Interface(AWS CLI)의 최신 릴리스를 설치하거나 업데이트하는 방법을 설명합니다. AWS CLI의 최신 릴리스에 대한 자세한 내용은 GitHub에서 AWS CLI 버전 2 변경 로그 를 참조하세요. AWS CLI의 이전 릴리스를 설치하려면 AWS CLI 버전 2의 이전 릴리스 설치 단원을 참조하세요.

https://docs.aws.amazon.com/ko_kr/cli/latest/userguide/getting-started-install.html

```
// aws cli 설치
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```



이제 `aws --version` 을 통해 설치된 것을 확인할 수 있다.

aws cli configuration 설정

```
$ aws configure
AWS Access Key ID [None]: ~~~
AWS Secret Access Key [None]: ~~~
Default region name [None]: ap-northeast-2
Default output format [None]: json
```

eksctl을 사용하여 reverse 클러스터를 위한 pair-key 생성

먼저 pair-key 를 생성해준다.

`ssh-keygen` 명령어를 사용하여 pair-key를 생성하고, 이것을

`aws ec2 import-key-pair --key-name "reverse-key" --public-key-material fileb://~/.ssh/id_rsa.pub` 명령어로 사용중인 ec2 리전에 업로드 한다.

eksctl 클러스터 만들기

```
eksctl create cluster --version 1.23 --name eks-demo --vpc-nat-mode HighlyAvailable --node-private-networking --region ap-northeast-2
```

```
# Reverse ClusterConfig object:

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: reverse-cluster
  region: ap-northeast-2

iam:
  withOIDC: true

managedNodeGroups:
  - name: reverse-ng-private
    instanceType: m5.large
    desiredCapacity: 2
    minSize: 2
    maxSize: 4
    volumeSize: 20
    volumeType: gp2
    ssh:
      allow: true
      publicKeyPath: ~/.ssh/id_rsa.pub
    labels: { role: worker }
    tags:
      nodegroup-role: worker
    privateNetworking: true

    iam:
      withAddonPolicies:
        imageBuilder: true
        autoScaler: true
        appMesh: true
        albIngress: true
```

위의 코드에서 metadata를 통해 클러스터 이름과 리전을 정해줄수가 있고,
이미 내가 만들어놓은 vpc가 있고 그것을 사용하고 싶다면 vpc 옵션을 넣고 subnet에서 public 과 private 영역을 넣어서 만들어줄수 있다.
nodeGroup은 실제 워커 노드와 그리고 public subnet에 들어갈 노드로 구분지어줄 수 있으며 인스턴스 타입과 라벨 그리고 iam 설정을 통해 만들어줄 수 있다.

그 후 eksctl create cluster -f cluster.yaml 을 통해서 적용해주면 된다.

노드에서 시스템 자원들은 kubelet 구성에 따라 할당될 수 있다. 이걸 추천하는 데 그 이유는 kubelet에 대한 리소스가 부족하면 kubelet은 pods들을 제거할 수가 없게되고 node는 결국 `NotReady` 상태가 되기 때문이다. 따라서 kubelet에 자원을 할당하기 위해 우리는 kubeletExtraConfig 필드를 포함한다.
예를 들어 위에서 명시한 cpu, memory, 일시적인 스토리지를 OS daemon에 부여한다. 그리고 메모리가 200Mi 보다 적어지거나 root filesystem의 10% 이하가 되면 더 이상 pod를 생성하지 않는다.

eksctl 옵션

`--version` : 사용할 쿠버네티스 버전

`--name` : 클러스터 이름

`--vpc-nat-mode` : kubernetes의 모든 outbound는 nat gateway를 통해 나가게 되는데 default option은 single이라 하나만 생성된다. 개발 환경에서는 상관없지만 운영에서는 각 subnet 마다 하나씩 만드는 HighAvailable 옵션을 사용한다.

`--node-private-networking` : 해당 옵션이 없으면 node group이 public subnet에 만들어진다. 보안을 위해 private subnet에 만들어지기 위해 이 옵션을 사용한다.

`--region` : 우리는 서울 리전을 사용할 것이므로 ap-northeast-2를 사용하자.

`--node-type` : 노드로 사용될 인스턴스의 타입을 지정한다. m5.large를 생성한다.

`--nodes` : 생성 될 노드의 갯수

`--with-oidc` : oidc를 적용

`--ssh-access` : ssh 허용

`--ssh-public-key` : default로 ~/.ssh/id_rsa.pub가 사용된다. 만약 내가 정한 이름으로 키 파일을 구성하고 싶다면 이 옵션을 사용한다.

`--managed` : 노드 그룹을 관리한다.

잠깐 딴 얘기를 해보자면,

```
ubuntu@ip-10-0-12-197:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

아무 컨테이너도 안보인다. 그 이유는 1.24 부터는 dockerd 가 아닌 containerd 로 동작하기 때문

따라서 containerd 를 볼 수 있는 crictl 이라는 패키지가 필요합니다

- 쿠버네티스에서는 cri표준을 준수하는 컨테이너 런타임을 통합하여 관리하기 위해 crictl이란 명령어 툴을 제공하고 있습니다
- crictl은 docker,ctr과 달리 cri를 준수하는 모든 컨테이너 런타임에 사용이 가능합니다
- 일반적인 패키지 매니저를 통해 kubeadm kubelet등을 설치했다면 자동으로 설치 되어 집니다

```
vi /etc/crictl.yaml
runtime-endpoint: unix:///run/containerd/containerd.sock
image-endpoint: unix:///run/containerd/containerd.sock
```

```
sudo crictl ps
sudo crictl pods
```

```
ubuntu@ip-10-0-12-197:~$ sudo crictl ps
CONTAINER           IMAGE               CREATED             STATE               NAME                      ATTEMPT             POD ID              POD
5d39c4008b949       d3377ffb7177c       9 hours ago         Running             kube-apiserver            5                   67e6e298d3ed1       kube-apiserver-ip-10-0-12-19
7
9195bab20cdcf       5d725196c1f47       10 hours ago        Running             kube-scheduler            2                   9a4d02bb4d159       kube-scheduler-ip-10-0-12-19
7
13c544dd76825       34cdf99b1bb3b       10 hours ago        Running             kube-controller-manager   2                   e466647c9ef23       kube-controller-manager-ip-1
0-0-12-197
770c5c5c3af5e       a4ca41631cc7a       4 weeks ago         Running             coredns                   0                   e6d7ed809c3ed       coredns-6d4b75cb6d-pnp2b
9bd94d748a6da       a4ca41631cc7a       4 weeks ago         Running             coredns                   0                   1620232b30420       coredns-6d4b75cb6d-hhqzk
6ed180c6c47c4       df29c0a4002c0       4 weeks ago         Running             weave                     1                   13e0b0599132c       weave-net-746qv
4c896ccd07a42       7f92d556d4ffe       4 weeks ago         Running             weave-npc                 0                   13e0b0599132c       weave-net-746qv
9c73f3f977e87       a634548d10b03       4 weeks ago         Running             kube-proxy                0                   aa50d402ccd84       kube-proxy-7bnbh
4ae674a2e547c       aebe758cef4cd       4 weeks ago         Running             etcd                      0                   ca4be5a105eb3       etcd-ip-10-0-12-197
```