클라우드와 정보보안(1)

- <김영욱 교장선생님>
- 2022년 11월 3일(목요일)

서버 우분투 환경에서 도커를 설치하는 방법

1) apt-get을 사용해서 우분투를 업데이트하고 업그레이드 시켜준다.

- sudo apt-get update
- sudo apt-get upgrade

2) 우분투 환경에서 필요한 패키지를 apt-get install을 사용해서 설치해준다.

- sudo apt-get install \ apt-transport-https \ ca-certificates \ curl \ gnupg \ lsb-release

3) 도커 파일을 다운로드 받아준다.

- curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

4) 도커 파일을 다운로드 받아준다.

- echo \ "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \ $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

5) 도커 파일을 다운 받은 후 최신버선으로 업데이트

- sudo apt-get update
- sudo apt-get install docker-ce docker-ce-cli containerd.io

6) 도커잘 잘 설치되었는지 hello-world이미지 파일을 실행시켜준다.

- sudo docker run hello-world

7) ps명령어를 통해 어떤게 실행 중인지 본다.

- ps <- 어떠한 프로그래밍 실행중인지 보는 것이다.

8) ps -al 명령어를 사용하면 더 많은 정보를 볼 수 있다.

- ps -al <- -al 하게 되면, 더 많은 정보를 볼 수 있다.

9) 도커의 ps확인

- docker ps

10) docker ps의 명령어가 안되서 sudo 추가

- sudo docker ps

11) 도커에게 관리자 권한을 부여한다. 그 후 exit후 서버를 재시작 한다.

- sudo usermod -a -G docker $USER

12) restart를 하였지만, 적용이 되지 않아서 exit함.

- sudo service docker restart <- 하지만 이걸하게 되면 적용이 되지 않아서 exit를 하고 다시 로그인 해준다.

13) 도커에게 관리자 권한을 부여한 것을 적용하기 위해 exit

- exit

14) 도커 실행 중인것 파악

- docker ps <- 도커가 진행중인게 무엇인지 본다. 그리고 -a를 하게 되면 이전 실행되었던 것을 볼 수 있다.

15) --help를 통해 pull에 대한 정보를 찾아 볼 수 있다.

- docker pull --help <- tag는 버전을 말한다.

16) 도커의 pull을 사용하여 ubuntu:18.04가지고 온다.

- docker pull ubuntu:18.04 <- 18.04이미지를 가지고 온다.

17) 도커의 이미지 개수를 파악

- docker images <- 이미지가 몇개인지 확인을 해준다.

18) --help를 통해 run에 대한 정보를 찾아 본다.

- docker run --help <- help를 치게 되면, 실행하는 것에 대한 것의 도움말을 볼 수 있다.

19) run(특정 이미지를 실행시킨다.)

- docker run <- run은 특정 이미지를 실행 시킬 수 있다.
- docker run -it --name demo1 ubuntu:18.04 /bin/bash <- 이걸 실행시키면 컨테이너 안에 들어간다.(데모1)
- docker run -it --name demo2 ubuntu:18.04 /bin/bash <- 데모2
- docker run -it -d --name demo3 ubuntu:18.04 /bin/bash <- -d 옵션을 주면 daemon이다. 이렇게 하면 계속해서 실행중에 있다. (docker ps를 하게 되면 demo3가 계속 실행중인 상태인 것을 알 수 있다.)

20) exec(만들어진 데모를 실행시킨다.)

- docker exec -it demo3 /bin/bash <- 만들어진 demo3를 실행시킨다. 빈 밑에 있는 쉘을 실행시킨다.

21) logs(어떤 일이 일어났는지 파악)

- docker logs <- 어떤 일이 있었는지 쭉 보는 것이다.
- docker logs --help
- docker run --name demo4 -d busybox sh -c "while true;do $(echo date);sleep 1; done" <- busybox는 도커를 테스트 할 때 많이 쓰이는 것이다. echo는 출력을 해준다 날짜를 1초간 쉬었다가 계속 실행시켜준다.
- docker ps
- docker logs demo4 <- demo4의 값을 계속해서 찍고 있는 것을 볼 수 있다.
- docker logs demo4 -f <- -f를 하면 계속 지켜 볼 수 있다.
- docker stop demo4 <- demo4가 정지가 된다. 메모리에는 컨테이너는 그대로 남아있고 중지만 시킨거다.
- docker ps -a

22) rm과 rmi를 사용하여 삭제

- docker rm demo4 # demo4 rm을하게 되면 메모리 안에서만 지워졌지 이미지가 지워진 것은 아니다.
- docker images # <- images가 어떤게 있는지 확인 할 수 있다.
- docker rmi ubuntu:18.04 # <- 이렇게 하면 ubuntu의 이미지를 삭제 시킬 수 있다.
- docker images # <- 도커의 이미지가 삭제된 것을 확인 할 수 있다.

---

vi 에디터

---

vi hello.txt <- vi를 이용해서 파일을 만들어 준다.

<vi 명령어 모음>

<입력 모드>

i <- i는 INSERT로 바뀌게 되면서 편집 모드로 들어가게 된다.

<명령 모드>

ESC 키는 명령모드에 진입하게 된다.

- :w <- 작성한 것을 저장해준다.
- :q <- 작성중인 파일에서 나갈때 사용을 해준다.
- :wq
- :q! <- 작성한 것을 저장하지 않고 강제로 빠져 나간다.

- cat hello.txt <- cat을 하게 되면, vi로 열지 않고 내용을 확인할 수 있다.

- touch world.txt <- 빈파일을 만들어준다.
- cat world.txt <- 내용일 없는 파일을 만들었기 때문에 아무런 내용이 나오지 않는다.
- rm hello.txt
- rm world.txt

도커의 파일을 만들어준다.

- cd $HOME <- 홈 디렉토리로 이동이 가능하다.
- mkdir docker-practice <- 여기에서 도커파일을 만들어서 우리만의 이미지를 만든다.
- touch Dockerfile <- touch는 빈 파일을 만들어주는 역할을 한다.
- vi Dockerfile <- vi를 사용해서 Dockerfile을 수정해본다.

Dockerfile안의 내용

- FROM ubuntu:18.04
- RUN apt-get update <- 최신의 상태로 만들어 준다.
- CMD ["echo", "Welcome to Microsoft AI School"] <- 도커 컨테이너가 실행될 때 마다 실행되는 명령어이다. <- echo는 print문하고 똑같다.
- ESc누른 후 :wq <- 저장 후 빠져 나간다.
- cat Dockerfile
- docker build --help
- docker build -t my-image:v1.0.0 . <- t는 태그를 지정할 수 있다. 도커 파일로 이미지를 생성한다. 그리고 뒤에 .을 찍어서 현재 디렉토리에서 하라는 것을 만들어 준다.

이렇게 하면 도커 파일 이미지를 생성한 것을 볼 수 있다.

- docker images <- 만들어낸 이미지는 docker images에 있는 것을 볼 수 있다. 원래 있던 ubuntu 63.1MB에 추가를 해서 106MB가 만들어 진 것을 볼 수 있다.
- docker images | grep my-image <- |와 grep을 사용해서 my-image가 있는 것을 찾아 줘 하는 것이다.
- ls -al
- ls -al | grep Dockerfile <- ls -al 에 추가적으로 파이프 | 이것과 grep를 사용해서 Dockerfile의 위치를 알 수 있다.

- Docker Registry <- 를 해서 로컬에 도커 레지스트리(이것도 하나의 레지스트리로 되어있다.)를 만든다.

- docker run -d -p 5000:5000 --name registry registry <- port는 5000번 포트로 접근할 수 있다.
- docker ps <- 5000번 포트로 실행되고 있는 것을 볼 수 있다.

만들었던 도커 이미지를 도커 레지스티리에 저장시켜준다.

- docker tag my-image:v1.0.0 <- 이렇게 하면 현재 내 토커에서만 동작하게 된다.

하지만

- docker tag my-image:v1.0.0 localhost:5000/my-image:v1.0.0 <- 이렇게 해주게 되면 소속을 다르게 해줄 수 있다.
- docker images <- localhost:5000/my-image 소속이 다른애가 만들어진다.
- docker push localhost:5000/my-image:v1.0.0 <- 로컬에 있는 레지스트리에다가 저장을 할 수 있다. # 하지만 localhost로 되어 있어서 다른 사람들은 못가지고 가지만 주소를 만들어 주게 되면 다른 사람들 끼리 서로 주고 받고 할 수 있게 된다.

---

curl <- 간단한 웹통신을 요청 할 수 있다.

- curl -X GET http://localhost:5000/v2/_catalog <- catalog를 하게 되면 이 안에 있는 목록을 돌려주게 된다.
- curl -X GET http://localhost:5000/v2/my-image/tags/list <- 테그의 리스트까지 확인을 해줄 수 있다. # v2는 지금 사용하고 있는 레포지터리의 버전이다.

- docker hub로 접속한다.
- docker login
- Username: 아이디를 입력해준다.
- Password: 패스워드를 입력해준다.
- docker tag my-image:v1.0.0 changwoo03/my-image:v1.0.0 <- 자신의 도커 허브이메일을 tag로 지정해준다.
- docker push changwoo03/my-image:v1.0.0 <- 도커 허브로 날라가는 것을 볼 수 있다.
- docker pull changwoo03/my-image:v1.0.0 <- 도커 허브에 있는 것도 가지고 올 수 있다.
- docker run -d changwoo03/my-image:v1.0.0 <- 이렇게 하면 이미지가 실행된다.

쿠버네티스에 대해 다루어 본다.

- 우분투 서버 에서 실행을 해본다.
- curl -L0 curl -LO https://storage.googleapis.com/minikube/releases/v1.22.0/minikube-linux-amd64 <- 이 명령을 치고 미니쿠버네티스를 설치한다.

- sudo install minikube-linux-amd64 /usr/local/bin/minikube <- 여기 경로에 설치를 한다.
- minikube --help <- 설치가 잘되었는지 --help를 쳐서 확인해 준다.
- minikube version <- 버전을 확인해 준다.

미니쿠버네티스를 설치를 하였다.

미니쿠버네티스를 실행해줄 쿠베 씨티엘을 설치해준다.

- curl -LO https://dl.k8s.io/release/v1.22.1/bin/linux/amd64/kubectl
- sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
- kubectl --help
- kubectl version
- minikube start --driver=docker <- 메모리가 최소 2기가 필요하다.
- minikube status <- 상태를 확인해 볼 수 있다.
- kubectl get pod <- defalut에 아무것도 없다고 나온다.
- kubectl get pod -n kube-system <- 쿠베의 시스템 파일이 나오게 된다.

컨테이너를 관리하기 위한 쿠버네티스도 컨테이너로 되어있다.

- minikube delete <- minikube를 삭제해준다.

Pod를 생성하고 하는 방법에 대해서 해본다.

- 셋팅은 YAML 파일을 만들어서 해본다.
- 알아들을 수 있게 해주는게 YAML파일이다.
- vi pod.yaml

---

pod.yaml파일에 대한 내용

```
apiVersion: v1
kind: Pod
metadata:
   name: counter
spec:
    containers:
    - name: count
      image: busybox
      args: [/bin/sh, -c, 'i=0;while true;do echo "$i: $(date)"; sleep 1;done']
```

- kubectl apply -f pod.yaml <- 파일형태로 있는 YAML파일로 실행을 하겠다는 것이다.
- kubectl get pod <- Pod가 보이게 된다.
- kubectl get pod -A <- -A 은 모든 Pod시스템이 다 보이게 된다.
- kubectl describe pod counter <- 전체의 실행과정을 볼 수 있다.(자세한 정보를 볼 수 있다.)
- kubectl get pod -o wide <- 파드 목록을 자세하게 출력을 해준다.
- kubectl get pod -w <- 특정 pod를 계속 보고 있어야 할 때 사용해준다.(모니터링 할때 사용해준다. 그럼 계속 변화를 트레킹 해준다.)

pod의 로그값을 확인해 본다.

- kubectl logs counter <- logs들을 찍어주고, pod의 이름이 counter였다.
- kubectl logs counter -f <- pod에서 나오는 것을 계속 모니터링 할 수 있다.
- kubectl exec -it counter sh <- -it는 인터렉티브 모드이다. / sh는 쉘명령이라서 쉘로 들어가겠다는 것이다. 그리고 여기에서 ps를 치게 되면 pod안에서 ps가 실행되는 것을 알 수 있다.

- kubectl delete pod counter <- 첫 번째는 삭제를 해준다.
- kubectl delete pod -f pod.yaml <- 두 번째는 YAML파일을 사용해서 삭제를 해준다.

- vi Deployment.yaml <- Deployment.yaml 파일을 만든다.

---

nginx는 웹서버이다.

Deployment.yaml안의 내용

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3 # replicas는 복제본이다. replicas를 3개 하게되면, 똑같은 pod가 3개가 생성이 된다.
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
```

```
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

- kubectl apply -f Deployment.yaml <- kubectl을 사용해서 yaml을 실행해준다.
- kubectl get deployment
- kubectl get pod <- 리플리카에 의해서 3개의 pod가 실행된다.

각 pod 안에는 컨테이너가 하나씩 되어있다. 쿠버네티스가 자동으로 pod 안에 컨테이너를 넣는다는 것을 볼 수 있다.

- kubectl delete pod nginx-deployment-66b6c48dd5-6dxk4 <- pod를 강제로 삭제했을 때 다시 생성이 되는지 본다. 그럼 auto healing으로 다시 생성이 되는 것을 알 수 있다.

스케일링을 강제로 해줄 수 있다.

- kubectl scale deployment/nginx-deployment --replicas=5 <- Deployment의 메타데이터를 이용하는 거다. 그럼 5개로 바뀌어 있는 것을 볼 수 있다.
- kubectl get pod <- 그럼 5개로 만들어져 있는 것을 볼 수 있다.

scale을 사용해서 --replicas의 개수를 늘릴 수도 있고 줄일 수도 있다.

pod보다는 replicas가 더 큰 개념이다. 중요!!!

- kubectl delete deployment nginx-deployment <- 삭제를 시켜준다.
- kubectl apply -f Deployment.yaml <- 만들어 준다.
- kubectl get pod -o wide
- curl -X GET 172.17.0.4 -vvv <- 접근이 안되겠지만, curl을 사용해서 IP Adress에 접근을 해본다.
- ping yahoo.com <- ping이 되는지 해보았다.

서비스를 만들어 두었다.

서비스는 내부에 있는 pod들을 외부로 뽑아내주는 역할을 한다.

- vi Service.yaml

Service.yaml에 대한 내용

그럼 80번 포트로 연결시켜준다.

```
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    run: my-nginx
spec:
  type: NodePort
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: nginx
```

- kubectl apply -f Service.yaml
- kubectl get service
- curl -X GET 10.108.50.235:80

END