Docker Swarm 사용하기

1. Swarm node 구성 계획
2. Swarm 구성하기
3. Stack / Service 실행
4. Docker private registry 구축하기
5. Deploy를 위한 Compose file 작성하기

1. Swarm node 구성 계획

> 테스트용 Swarm node 구성

- book server (manager node) / 192.168.101.30
- samsung notebook (worker node) / 192.168.100.46
- msi notebook (worker node) / 192.168.100.47

> 프로덕트용 Swarm node 구성

- app server (manager node) / 192.168.101.80
- api server (worker node) / 192.168.101.70
- web server (worker node) / 192.168.101.100

> 아래 두가지의 Error 메시지를 보지 않기 위해 해야 할 일

```
"swarm node is missing network attachments, ip addresses may be exhausted."
"failed to allocate gateway (10.0.1.1): Address already in use."
```

1. Hostname 설정

```
$ sudo hostnamectl status
```

```
$ sudo hostnamectl set-hostname ITS-APISERVER.ITSROOM.COM
$ sudo hostnamectl set-hostname ITS-APPSERVER.ITSROOM.COM
$ sudo hostnamectl set-hostname ITS-WEBSERVER.ITSROOM.COM
```

```
$ sudo reboot
```

2. Docker Swarm Routing Mesh(Ingress network) 설정

1. Check port open

- Port 7946 TCP/UDP for container network discovery.

- Port 4789 UDP for the container ingress network.

```
# netstat -tnl
# netstat -unl
# firewall-cmd --zone=public --list-all
```

```
# firewall-cmd --zone=public --permanent --add-port=7946/tcp
# firewall-cmd --zone=public --permanent --add-port=7946/udp
# firewall-cmd --zone=public --permanent --add-port=4789/udp
# firewall-cmd --reload
# firewall-cmd --zone=public --list-all
```

참조 블로그 https://subicura.com/2017/02/25/container-orchestration-with-docker-swarm.html

2. Swarm 구성하기

> Swarm init (Swarm 구축): manager node에서 실행

```
- input
[root@ITS-WEBSERVER its]# docker swarm init --advertise-addr 192.168.101.80
```

```
- output(worker node에서 실행할 join-token 명령어)
Swarm initialized: current node (j047o4z6w71zwolmvaitbkb6e) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-2v3sdrfzc6mkr310wqseoq1cevwkxupqldozhhzxa8jd4d8mmy-
6c27bitjryr39a7wzr6j8ik1o \
    192.168.101.80:2377

- (manager node추가를 위한 join-token 명령어)
To add a manager to this swarm, run 'docker swarm join-token manager' and follow
the instructions.
```

> node list 확인 - node 구축이 잘 되었는지 확인

```
- input
[root@ITS-WEBSERVER its]# docker node ls

- output
ID                            HOSTNAME                STATUS         ENGINE
VERSION
nhyjv7tmnvzwe92fszb238dtm *   localhost.localdomain   Ready          19.03.8
```

manager node 에서 방화벽 설정

```
- input
[root@ITS-WEBSERVER its]# firewall-cmd --zone=public --permanent --add-
port=2377/tcp
[root@ITS-WEBSERVER its]# firewall-cmd --reload
[root@ITS-WEBSERVER its]# firewall-cmd --zone=public --list-all
```

worker node에서 join command 실행하기

```
- input
[root@ITS-WEBSERVER its]# docker swarm join --token SWMTKN-1-
1nkn5ygrdfrcjoowntchb8djsh1ufski1g4805y9b8s1odgp3r-1mcdbhcg831w57bvv25u9wdoy
192.168.101.30:2377
```

```
- output
This node joined a swarm as a worker.
```

- 방화벽 설정하지 않을 때 발생하는 Error

```
- output
-> error

Error response from daemon: rpc error: code = Unavailable desc = all SubConns are
in TransientFailure, latest connection error: connection error: desc = "transport:
Error while dialing dial tcp 192.168.101.30:2377: connect: no route to host"
```

manager node에서 node list 확인

```
- input
[root@ITS-WEBSERVER its]# docker node ls
```

```
- output

ID                            HOSTNAME                          STATUS  AVAILABILITY
MANAGER STATUS
063cxgv750ue99jcnxl1xljxx     its-apiserver.itsroom.com         Ready   Active
84yzzxr8fqyvi22jh9u4dlfsh     its-uyeg-webserver.itsroom.com    Ready   Active
w197c4yqd6kyyriv3p552upm8 *   ITS-WEBSERVER.ITSROOM.COM         Ready   Active
```

```
Leader
```

- 80번 서버의 hostname이 이미 webserver로 설정되어 있어 100번 서버의 hostname을 it-uyeg-webserver 로 설정함.

> tip: manager node에서 join 명령어 검색하는 방법

```
[root@ITS-WEBSERVER its]# docker swarm join-token manager
[root@ITS-WEBSERVER its]# docker swarm join-token worker
```

- Docker는 고 가용성을 구현하기 위해 클러스터 당 3 개 또는 5 개의 관리자 노드를 권장합니다. 스웜 모드 관리자 노드는 Raft를 사용하여 데이터를 공유하므로 홀수의 관리자가 있어야합니다. 관리자 노드의 절반 이상이 사용 가능한 한 웜은 계속 작동 할 수 있습니다.

> tip: Node에서 제거

- Worker node에서 Leave Swarm 명령어 실행

```
[root@ITS-WEBSERVER its]# docker swarm leave

Node left the swarm.
```

- Manager node에서 Remove Node 명령어 실행

```
[root@ITS-WEBSERVER its]# docker node rm node-2
```

3. Stack / Service 실행

> Source download

- git clone API_Server

1. Compose up 명령어를 통해 image 생성하기

```
[root@ITS-WEBSERVER its]# docker-compose up -d
```

2. Deploy API Server

- Nginx
- Gateway (bm4server)
- StandAlone (bm3server)

```
- input
[root@bookserver API_Server]# docker stack deploy --compose-file docker-
compose.yml api_server
```

```
- output
Ignoring unsupported options: build, restart

Ignoring deprecated options:

container_name: Setting the container name is not supported.

Creating network api_server_backend
Creating service api_server_nginx
Creating service api_server_gateway
Creating service api_server_standalone
```

- Test -Stack deploy MariaDB

```
[root@bookserver mariadb-docker]# docker stack deploy -c docker-compose.yml
mariadb1
Creating service mariadb1_mariadb
```

4. Docker private registry 구축하기

> "No such image: ~" 와 같은 Error 발생하는 것을 해결하기 위한 private registry 생성

1. Docker registry 설치
   <참조: https://novemberde.github.io/2017/04/09/Docker_Registry_0.html>

> VOLUME 권한을 위한 명령어

```
chmod a+rw /var/run/docker.sock
chmod 777 /home/its/data/docker (저장위치)
```

> registry 이미지를 가져오기 (안해도 상관 없음)

```
$ docker pull registry
```

> registry를 실행하기

```
$ docker run -d \
 -p 5000:5000 \
```

```
 --restart=always \
 --privileged \
 --name docker-registry \
 -e REGISTRY_STORAGE_DELETE_ENABLED=true \
 -v /home/its/data:/var/lib/registry \
 registry:2
```

- docker registry에서 image 삭제를 위한 환경설정 명령어

```
 -e REGISTRY_STORAGE_DELETE_ENABLED=true
```

- docker registry 의 volume 설정을 위한 명령어

```
 -v /home/its/data:/var/lib/registry
```

- Registry 에 image push 하기 pull하기

> hello-world 이미지가 없으니 docker hub에서 pull하자.

```
$ docker pull hello-world
```

> localhost/hello-world 이미지를 만들어보자.

```
$ docker tag hello-world localhost:5000/hello-world
```

> 이미지 push하기

```
$ docker push localhost:5000/hello-world
```

> 이미지 확인하기

```
$ curl -X GET http://localhost:5000/v2/_catalog
# 출력 {"repositories":["hello-world"]}
```

> 태그 정보 확인하기

```
$ curl -X GET http://localhost:5000/v2/hello-world/tags/list
# 출력 {"name":"hello-world","tags":["latest"]}
```

<참조: https://waspro.tistory.com/532>

> "server gave HTTP response to HTTPS client" 라는 메시지가 출력되는 경우 다음과 같이 daemon.json 파일을 수정

```
$ vi /etc/docker/daemon.json
{
    "insecure-registries": ["192.168.101.70:5000"]
}
```

> 재실행

```
$ systemctl daemon-reload
$ systemctl restart docker
```

> docker registry로 부터 image pull 하기

```
$ docker pull 192.168.101.70:5000/test:latest
latest: Pulling from 192.168.101.70:5000/test
68ced04f60ab: Already exists
c4039fd85dcc: Already exists
c16ce02d3d61: Already exists
7469e3cf55fe: Pull complete
Digest: sha256:335079834819530f9746329187a4c26a011ea3f74d4b607106e3d2fbd339d273
Status: Downloaded newer image for 192.168.101.70:5000/test:latest
```

> Docker registry에서 image 삭제하기

<참조: https://stackoverflow.com/questions/25436742/how-to-delete-images-from-a-private-docker-registry>

- to delete an image, you should run the registry container with REGISTRY_STORAGE_DELETE_ENABLED=true parameter.
- I set "delete: enabled" value to true in /etc/docker/registry/config.yml file. For this configuration no need to set REGISTRY_STORAGE_DELETE_ENABLED variable

> digest 확인을 위한 명령어

```
[root@its-apiserver its]# curl -k -I -H Accept:\*
http://localhost:5000/v2/test/manifests/latest
HTTP/1.1 200 OK
Content-Length: 12076
Content-Type: application/vnd.docker.distribution.manifest.v1+prettyjws
```

```
Docker-Content-Digest:
sha256:e034004b3d20f3a95d0dabd48aece8b5db1eaa2bc6d9b02163e5434c7013eb0a
Docker-Distribution-Api-Version: registry/2.0
Etag: "sha256:e034004b3d20f3a95d0dabd48aece8b5db1eaa2bc6d9b02163e5434c7013eb0a"
X-Content-Type-Options: nosniff
Date: Mon, 06 Apr 2020 08:36:04 GMT
```

Delete 명령어 테스트(에러 메시지 발생)

```
[root@its-apiserver its]# curl -X DELETE
http://localhost:5000/v2/test/manifests/latest
{"errors":[{"code":"DIGEST_INVALID","message":"provided digest did not match
uploaded content"}]}
```

```
[root@its-apiserver its]# curl -X DELETE
http://localhost:5000/v2/test/manifests/sha256:e034004b3d20f3a95d0dabd48aece8b5db1
eaa2bc6d9b02163e5434c7013eb0a
{"errors":[{"code":"MANIFEST_UNKNOWN","message":"manifest unknown"}]}
```

docker registry를 위한 각종 명령어

```
01. registry 내부의 repository 정보 조회

$ curl -X GET <REGISTRY URL:포트>/v2/_catalog



02. repository 에 대하여 tag 정보 조회

$ curl -X GET <REGISTRY URL:포트>/v2/<REPOSITORY 이름>/tags/list



03. content digest 조회

$ curl -v --silent -H "Accept:
application/vnd.docker.distribution.manifest.v2+json" -X GET http://<REGISTRY URL:
포트>/v2/<REPOSITORY 이름>/manifests/<TAG 이름> 2>&1 | grep Docker-Content-Digest |
awk '{print ($3)}'



04. manifest 삭제

$ curl -v --silent -H "Accept:
application/vnd.docker.distribution.manifest.v2+json" -X DELETE http://<REGISTRY
URL:포트>/v2/<REPOSITORY 이름>/manifests/<DIGEST 정보>
```

```
05. GC(Garbage Collection)

$ docker exec -it docker-registry registry garbage-collect
/etc/docker/registry/config.yml



출처: https://trylhc.tistory.com/entry/Docker-registry-삭제-2 [오늘처럼..?]
```

5. Deploy를 위한 Compose file 작성하기

1. api server yml파일 비교

- 기존 compose 파일

```
version: '3'
services:
  nginx:
      container_name: nginx

      build:
        context: ./nginx
        dockerfile: ./Dockerfile

      ports:
       - "8083:8083"

      restart: always

      networks:
      - backend

      depends_on:
        - gateway
        - standalone

  gateway:
      container_name: bm4server

      build:
        context: ./gatewayAPI
        dockerfile: ./Dockerfile

      ports:
       - "9210:9210"

      restart: always
```

```
      networks:
      - backend

  standalone:
      container_name: bm3server

      build:
        context: ./standAloneAPI
        dockerfile: ./Dockerfile

      ports:
       - "9211:9211"

      restart: always

      networks:
      - backend

networks: # 가장 기본적인 bridge 네트워크
  backend:
    driver: bridge
```

- deploy 용 yml 파일

```
version: '3'
services:
  nginx2:
      container_name: nginx_swarm
      image: api_server_nginx_swarm
      build:
        context: ./nginx
        dockerfile: ./Dockerfile

      ports:
       - "8083:8083"

      restart: always

      volumes:
       - /var/run/docker.sock:/var/run/docker.sock

      networks:
        - frontend
        - backend

      depends_on:
        - gateway
        - standalone

  gateway:
      container_name: bm4server
```

```
        image: api_server_gateway
        build:
          context: ./gatewayAPI
          dockerfile: ./Dockerfile

        ports:
         - "9210:9210"

        restart: always

        networks:
        - backend

    standalone:
        container_name: bm3server
        image: api_server_standalone
        build:
          context: ./standAloneAPI
          dockerfile: ./Dockerfile

        ports:
         - "9211:9211"

        restart: always

        networks:
        - backend

networks:
    backend:
    frontend:
```