도커 명령 실행 -> 도커 데몬(dockerd) -> containerd

도커 데몬과 containerd와 착각하는게 도커데몬이 작업을 하는 것이 아니라 containerd가 작업을 진행한다.

containerd : 컨테이너들이 정상적으로 작동할 수 있도록 호스트os에서 시스템 정보를 제공한다.
: runC와 연계하여 컨테이너들의 관리 운영을 관장한다.
-> runC는 커널의 Cgroup, namespace 등에 접근하여 자원을 추상화하고 이를 통해 컨테이너를 생성하는 역할을 수행한다.

----docker network 개념
docker network ls
driver (bridge, host, null)

bridge : 호스트에서 가상의 스위치(브릿지)를 생성하고 이 스위치에 컨테이너가 연결되도록 사용하는 방법으로 DOCKER를 설치하면 기본적으로 생성되는 docker0가 해당된다.
해당 bridge는 nat의 기능을 포함하므로 컨테이너는 외부와 통신할 때 docker0의 스위치를 통하여 호스트의 실제 NIC 주소로 NAT 된 뒤, 외부와 통신한다.

host : 컨테이너의 네트워크 포트를 호스트(진짜 pc)의 NIC와 동일하게 사용

null : 네트워크 사용하지 않음

overlay : 전체 클러스터 환경에 동일한 네트워크 대역을 제공하고 해당 네트워크에 연결된 컨테이너들을 물리적, 논리적 위치에 상관없이 사설 주소를 통하여 통신할 수 있게 된다.
overlay를 구성하게 되면 자동으로 외부와 연결을 위한 독립적인 브릿지가 별도로 생성되고 해당 브리지는 로드밸런싱의 기능을 갖는다.

도커 스웜(stack)은 클러스터 환경을 통해 일관적인 인프라 환경 및 컨테이너를 제공할 수 있어야 한다. 이를 위해 yaml파일을 작성하고 이 yaml팡ㄹ의 내용을 compose가 읽고 이를 각 worker에게 전달할 수 있어야 한다.

docker-compose 설치 (마스터에서만)
curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose

docker-compose --version

클러스터 구성을 위해서는 클러스터에 join하기 위한 token이 필요하다.
해당 토큰은 manager에서 발행하고 worker들이 이 토큰을 이용하여 join을 시도하면 manager는 토큰의 유효성을 검사한 뒤, join 허용한다.
토큰 발행시에는 기본적으로 manager, worker용 토큰이 발행된다.

실습
docker swarm init --advertise-addr 211.183.3.100

docker swarm join --token
SWMTKN-1-3u5pzhh73oaonpyfxquy177uer874moekeywdqdo9ri7nqqism-5isqgzq6oi2v0rpufftnokabp
211.183.3.100:2377

각 노드에
docker swarm join --token

SWMTKN-1-3u5pzhh73oaonpyfxquy177uer874moekeywdqdo9ri7nqqism-5isqgzq6oi2v0rpufftnokabp
211.183.3.100:2377
적용

docker inspect master --format {{.Spec.Role}}

docker node inspect node1 --pretty <-----node가 앞에 붙어야 pretty옵션이 적용된다.

클러스터에서 빠져나가고 싶다면
docker swarm leave

manager에서는 down된 노드를 삭제하고자 한다면 docker node rm

==========
예제1)
nginx를 배포한다.
- 컨테이너의 배포는 worker에만 배포된다.
- 각 컨테이너의 호스트의 80번 포트와 매핑되어 서비스 된다.
- 이미지는 manager에서 docker login을 하고 인증정보를 worker들에게 전달하여 각노드에서 해당 정보를 통해 이미지를 다운 받고 컨테이너를 배포할 수 있어야한다.

docker service create --name mynginx --with-registry-auth -p 80:80 --replicas 2 --constraint 'node.role != manager' nginx

docker service scale mynginx=3

docker service rm mynginx

==========

docker hub 접속해서 리포지토리 공인으로 하나 만듬 testweb으로

master에서
mkdir testweb
cd testweb
mkdir blue green

curl http://www.keduit.com > blue/index.html
curl https://www.naver.com > green/index.html

도커 이미지만들기
testweb:blue
testweb:green

두개의 이미지를 docker-hub에 등록하세요
아래의 조건들을 만족해야한다.

1. 각 이미지의 base이미지는 nginx 사용.
2. 이미지 등록되었다면 먼저 docker service create를 통해 blue 버전을 3개 배포하기!!

vi Dockerfile

```
FROM nginx:latest
COPY blue/index.html /usr/share/nginx/html
CMD ["nginx", "-g", "daemon off;"]

docker build -t mhkim1560/testweb:blue .
docker push mhkim1560/testweb:blue

docker service create --name blue --with-registry-auth -p 80:80 --mode global --constraint 'node.role != manager' qkrtjswo03/testweb:blue

vi Dockerfile
FROM nginx:latest
COPY green/index.html /usr/share/nginx/html
CMD ["nginx", "-g", "daemon off;"]

docker build -t qkrtjswo03/testweb:green .
docker push qkrtjswo03/testweb:green
```

CI/CD 에서는 특정 이벤트가 발생한 것을 확인하고 변경 사항을 특정 환경에 즉시 반영하도록 할 수 있다.
예를 들어 github 에서의 webhook은 저장소에 코드가 push되고 새로운 commit 번호가 확인되면 해당 코드를 웹서버에 즉시 반영할 수 있다.
dockerhub에서는 저장소에 새로운 이미지가 등록되면 해당 이미지를 docker swarm, k8s등의 환경에 즉시 반영하여 새로운 이미지로 update 시킬 수 있다.
(rolling update의 자동화)

```
기존에 서비스한 것들 새로운 이미지로 업데이트
docker service update --image mhkim1560/testweb:green blue
```

docker stack을 이용한 wordpress 배포
compose의 yaml을 이용하여 swarm에서 배포하는 방식
- 장점 : 재사용 가능

```
wq.yml
==========================================================================
mkdir wordpress
cd wordpress

version: '3.6'

services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
    deploy:
      placement:
        constraints:
```

```
          - node.role == worker

      relicas: 3
    restart: always

  wordpress:
    image: wordpress:latest
    ports:
      - "8001:80"
    depends_on:
      - db
    environment:
      WORDPRESS_DB_HOST: db              #기본포트로 3306이라 들어가지고 dns 서버가 있어서 db로 써도
됨.
      WORDPRESS_DB_PASSWORD: wordpress
    deploy:
      placement:
        constraints:
          - node.role == worker

      replicas: 3
    restart: always
```

모든 노드에서 docker volume ls를 해보면 wordpress가 배포된 node1~3까지는 새로운 볼륨이 생성된 것을 확인할 수 있따. 이는 db의 /var/lib/mysql을 연결한 볼륨이다.

로컬 볼륨을 각 컨테이너 사용한다면 데이터의 일관성을 유지할 수 없다. 각 컨테이너는 각 호스트에 볼륨을 사용하므로 나중에 동일한 볼륨으로 접근한다라는
보장을 할 수가 없게 된다. 따라서 외부에 디스크를 두고 연결하는 방법을 사용해야한다. 이를 영구 볼륨 (persistent volume) 이라 한다.
볼륨 접근은 컨테이너가 직접 접근한느 방식이 아니라 호스트에서 접근하는 방식의 proxy방식을 한다. 예를 들어 각 컨테이너에서 nfs로 연결된 볼륨을 사용하고자 하면
node에 nfs 클라이언트를 설치해두면 컨테이너가 볼륨을 요청할때 호스트에서 볼륨을 끌어다 연결시켜주게 된다.

실습 : 2

```
version: '3.6'

services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
    deploy:
      placement:
        constraints:
```

```
          - node.role == worker
      replicas: 3
    restart: always

  wordpress:
    image: wordpress:latest
    ports:
      - "8001:80"
    depends_on:
      - db
        #environment:
        #WORDPRESS_DB_HOST: db:3306
        #WORDPRESS_DB_PASSWORD: wordpress
    deploy:
      placement:
        constraints:
          - node.role == worker
      replicas: 3
    restart: always

volumes:
  db_data:
```

다하고 실습 다지우기!!!!!!!!!!

모든 노드에 sudo swapoff -a 하기 일시적인거라 매번해주기 싫으니까 /etc/fstab에서 해주기
#/swapfile none swap sw 0 0 <---- 현재 이렇게 유지해야됨.

!!k8s 설치
1.
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 -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl

systemctl daemon-reload
systemctl restart kubelet

안될 때 aptclear 하기

*apt사용시 문제가 발생할 경우 아래 실행한 뒤 진행
sudo rm /var/lib/apt/lists/lock
sudo rm /var/cache/apt/archives/lock
sudo rm /var/lib/dpkg/lock*
sudo dpkg --configure -a
sudo apt update

2. 각 마스터, 노드의 /etc/docker/daemon.json을 아래와 같이 입력

```
{
                        "exec-opts": ["native.cgroupdriver=systemd"]
}
systemctl restart docker
```

3-1

```
kubeadm init --apiserver-advertise-address 211.183.3.100
```

3-2

```
  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

3-3

```
export KUBECONFIG=/etc/kubernetes/admin.conf
```

root로 접속한 node에서

```
kubeadm join 211.183.3.100:6443 --token jy8ddr.nq4nqbe8xuwmut3v ₩
        --discovery-token-ca-cert-hash
sha256:6f4e2e94d9391020691c932737a10d3682db3ee00aa4f8529a819dad2831ed6a
```

잘안되면 전체 호스트서버에 kubeadm

```
kubectl get node
```

```
root@master:~# kubectl get node
NAME      STATUS      ROLES                  AGE      VERSION
master    NotReady    control-plane,master   3m51s    v1.23.5
node1     NotReady    <none>                 96s      v1.23.5
node2     NotReady    <none>                 94s      v1.23.5
node3     NotReady    <none>                 93s      v1.23.5
```

오버레이를 위한 애드온 중 calico를 아래와 같은 방법으로 설치

```
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```

kubectl get node 로 status 확인

안정적으로 k8s 클러스터가 운영되지 않는다면, 전체 서버에 kubeadm reset
위의 과정 다시 진행

```
kubectl get pod -n kube-system
```

애드온 종류 weave net이 잘됨.
calico

```
kubectl delete -f https://docs.projectcalico.org/manifests/calico.yaml
```

Weave Net

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

```
root@master:~# kubectl get node
NAME      STATUS    ROLES                  AGE    VERSION
```

```
master   Ready    control-plane,master   25m   v1.23.5
node1    Ready    <none>                 23m   v1.23.5
node2    Ready    <none>                 23m   v1.23.5
node3    Ready    <none>                 23m   v1.23.5
```

export KUBECONFIG=/etc/kubernetes/admin.conf <---- 껐다 켰을때 안되면 이거 다시 적용.

두가지 확인하기!!!!!!!

```
root@master:~# kubectl api-resources | grep pod
pods                                  po          v1                                     true         Pod
podtemplates                                      v1                                     true         PodTemplate
horizontalpodautoscalers              hpa         autoscaling/v2                         true
HorizontalPodAutoscaler
poddisruptionbudgets                  pdb         policy/v1                              true
PodDisruptionBudget
podsecuritypolicies                   psp         policy/v1beta1                         false        PodSecurityPolicy
```

```
root@master:~# kubectl api-resources | grep Deployment
deployments                           deploy      apps/v1                                true         Deployment
```