

## LDF Document - [ Installation of Docker CE on CentOS ]

\*\*\*\*\*

1. Core Concepts and Components of Docker
2. Building Docker images
3. Container Registries
4. Run Container
5. Dockerfile
6. Container Networking
7. Docker-volume
8. Docker-compose
9. Introduction to Podman daemonless Containers

### Video Course Steps

1. Docker VS Virtualization
2. Docker image
- 3.

```
$ dnf config-manager
```

```
--add-repo=https://download.docker.com/linux/centos/docker-ce.repo
```

```
$ dnf install docker-ce docker-ce-cli containerd.io
```

```
$ systemctl status docker
```

```
$ systemctl enable docker
```

```
$ docker run hello-world
```

### [ Managing Docker images and Container ]

\*\*\*\*\*

```
$ docker images
```

```
$ docker pull <image_name>
```

```
$ docker run -it --name <container_name> <image_name> bash
```

```
$ docker exec <container_id> /usr/bin/cat /var/www/html/index.html
```

```
$ docker exec <container_name> /usr/bin/cat /var/www/html/index.html
```

```
$ docker exec -it mongos1 bash -c "echo 'sh.status()' | mongo"
$ docker run -d -p 8080:80 apache2 -name web_server
$ docker exec <container_name> mkdir test_dir
(creating directory in container)
```

```
$ docker ps
$ docker ps -a
$ docker container ls
```

```
$ docker container stop
$ docker container start
$ docker kill <containerID>
$ docker pause <containerID>
$ docker unpause <containerID>
$ docker top
$ docker status
```

```
$ docker container rm <container_ID or containerName>
$ docker rmi <image_nameOR imageID>
$ docker prune ( to delete all images )
```

Note : We can use a short form of container hash ID instead of using a long hash ID.

Example :

```
$ docker container stop a42fsd1246hh6i93451g13
$ docker container stop a42f
```

## [Running Dockerfile ]

\*\*\*\*\*

```
$ docker build -t repo/image_name:tag <dockerfilePath>
$ docker build -t repo/image_name:tag .
$ docker build -t repo/image_name:tag -f /dockerfilePath ( Note: If
Dockerfile Name is diff)
( OR )
$ docker build -t image_name:tag <dockerfilePath>
```

FROM Ubuntu

LABEL maintainer="kyawswar"

ENV TZ=Asia/Yangon

```
RUN In -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ >  
/etc/timezone
```

```
RUN apt-get update \  
    && apt-get install -y apache2
```

```
COPY html/* /var/www/html/
```

```
EXPOSE 80
```

```
CMD ["/usr/sbin/apachectl", "-D", "FOREGROUND"]
```

## Understanding of CMD and ENTRYPOINT in Dockerfile

The jobs of both are the same but CMD can be overridden with some command line. Here is an example of CMD.

```

Dockerfile .../hello-world-python x Dockerfile .../hello-world-nodejs Dockerfile .../hello-world-java JS index.js launch.py
hello-world > hello-world-python > Dockerfile
1 FROM python:alpine3.10
2 WORKDIR /app
3 COPY requirements.txt /app/requirements.txt
4 RUN pip install -r requirements.txt
5 EXPOSE 5000
6 COPY . /app
7 CMD python ./launch.py
8
9 #COPY requirements.txt /app/requirements.txt
10 #ENTRYPOINT ["python", "./launch.py"]
11

```

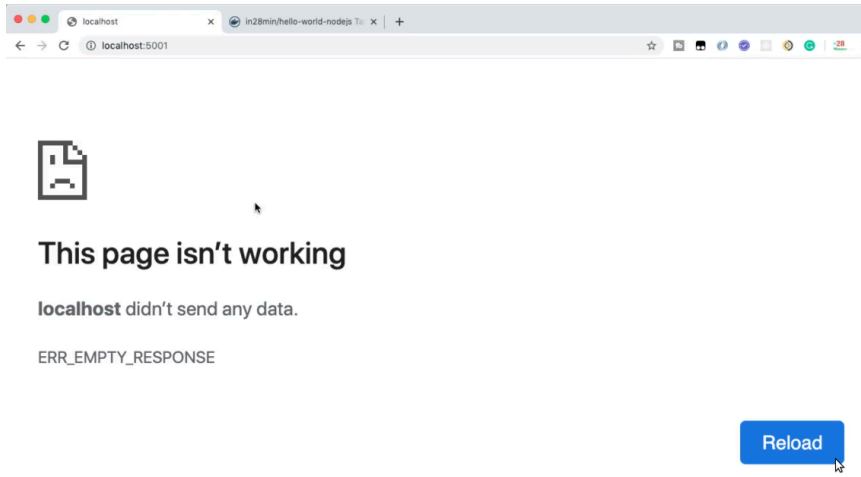
We have Dockerfile to integrate a python application with CMD for launching our application file ( launch.py ). Whenever we run this file, the upcoming container will execute the CMD stage. But on the other hand, we can override with some execution as shown in here.

```

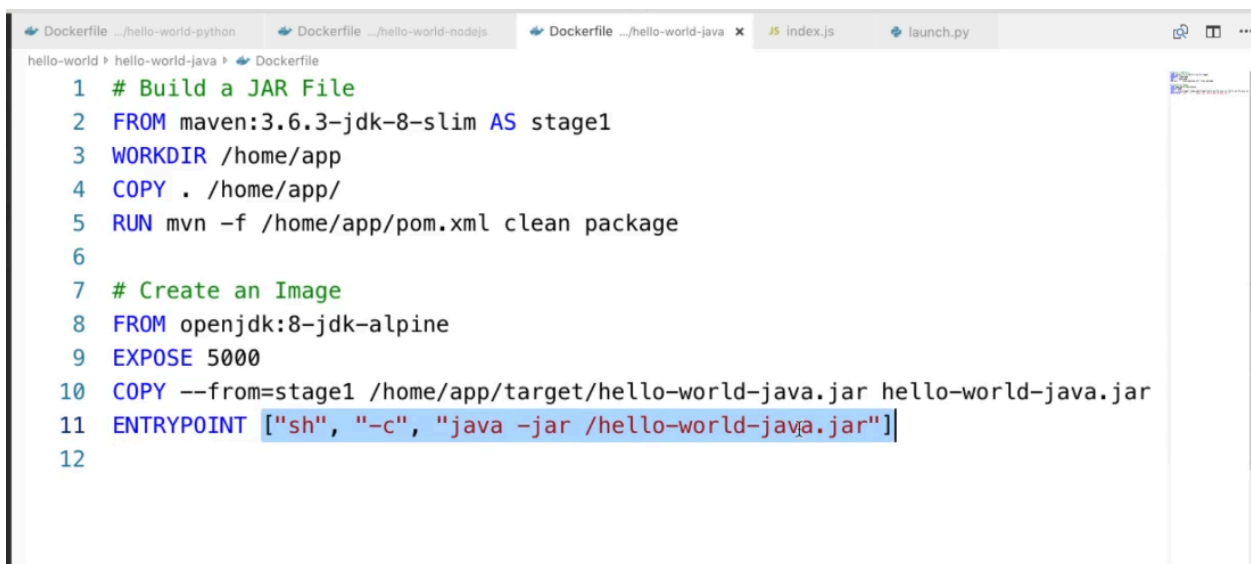
rangaraokaranam$ docker run -d -p 5001:5000 in28min/hello-world-nodejs:0.0.3.RELEASE
ping google.com
197d225b05762efca1e0e470dd69bedfff05fa0d8f46048c49a31029e4044630
rangaraokaranam$ docker logs -f 197d225b05762efca1e0e470dd69bedfff05fa0d8f46048c49a31029e4044630
PING google.com (216.58.197.78): 56 data bytes
64 bytes from 216.58.197.78: seq=0 ttl=37 time=17.658 ms
64 bytes from 216.58.197.78: seq=1 ttl=37 time=17.173 ms
64 bytes from 216.58.197.78: seq=2 ttl=37 time=18.177 ms
64 bytes from 216.58.197.78: seq=3 ttl=37 time=18.411 ms
64 bytes from 216.58.197.78: seq=4 ttl=37 time=16.249 ms
64 bytes from 216.58.197.78: seq=5 ttl=37 time=17.478 ms
64 bytes from 216.58.197.78: seq=6 ttl=37 time=18.878 ms
64 bytes from 216.58.197.78: seq=7 ttl=37 time=16.762 ms
64 bytes from 216.58.197.78: seq=8 ttl=37 time=18.569 ms
64 bytes from 216.58.197.78: seq=9 ttl=37 time=16.317 ms
64 bytes from 216.58.197.78: seq=10 ttl=37 time=17.924 ms
64 bytes from 216.58.197.78: seq=11 ttl=37 time=17.095 ms
64 bytes from 216.58.197.78: seq=12 ttl=37 time=18.532 ms
64 bytes from 216.58.197.78: seq=13 ttl=37 time=18.766 ms
64 bytes from 216.58.197.78: seq=14 ttl=37 time=17.213 ms
64 bytes from 216.58.197.78: seq=15 ttl=37 time=17.351 ms

```

We pass the CMD with `ping google.com`. It works but our predefined python application would not be executed.

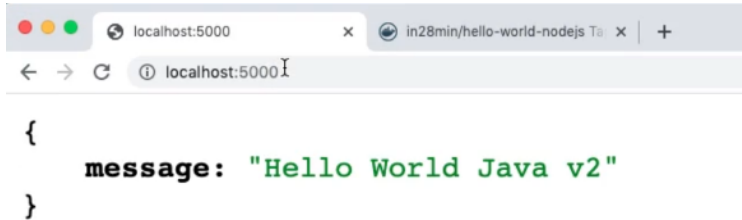


The ENTRYPOINT cannot be passed by like this CMD. The ENTRYPOINT is static. As shown in the figure.



Although we passed with CMD to java application ENTRYPOINT. It didn't work. Because it's ENTRYPOINT behavior.

```
rangaraokaranam$ docker run -d -p 5000:5000 in28min/hello-world-java:0.0.2.RELEASE p
ing google.com
b2a9e6ac6d921ea45ea514c787ec0419a2b062b5636a7adaba1ae40fd054a2c0
rangaraokaranam$
```



Ping command cannot pass the ENTRYPOINT. So We can see our java application is up and running.

```
rangaraokaranam$ docker run -d -p --entrypoint 5000:5000 in28min/hello-world-java:0.0.2.RELEASE
```

The important thing about entrypoint is that can be pass by only using `--entrypoint` option as shown in the above photo.

## [ Docker System ]

\*\*\*\*\*

```
$ docker system
$ docker system df
$ docker system events
$ docker system info
$ docker system prune
$ docker top
$ docker stats << Show all containers >>
$ docker stats < container ID >
```

## { Limit Resources for the container }

\*\*\*\*\*

```
$ docker run --name oursql -m 512m -e MYSQL_ROOT_PASSWORD=secret -d
mysql ( to run container with 512 MB of RAM )
```

```
$ docker run --name oursql -m 512m --cpu-quota=50000 -e
MYSQL_ROOT_PASSWORD=secret -d mysql
( to run container with 50% of CPU ) {100000 = 100% of CPU}
```

## [ Managing DockerNetwork ]

\*\*\*\*\*

```
$ docker network ls
$ docker network inspect < network_name >
$ docker network create -d < network_type > < network_name >
$ docker network connect < network_name> < container_name or ID >
$ docker network disconnect < network_name> < container_name or ID >
$ docker network rm < network_name1 > <network_name2 > etc..
$ docker network prune -f
```

### {Practice of Docker Link Between Microservices}

As a default docker network is bridge. That why if we want to communicate with two containers (running different micro-services), we can use `--link` option to establish those services.

Example: Currency Exchange JAVA Application

We already have two containers running. They are

1. Currency Exchange
2. Currency Conversion

```
$ docker run -p -d 8000:8000 --name=currency-exchange
in28min/currency-exchange:0.0.1-RELEASE
```

<http://localhost:8000/currency-exchange/from/USD/to/INR>

```
$ docker run -d -p 8100:8100 --env
CURRENCY_EXCHANGE_SERVICE_HOST=http://currency-exchange
--name=currency-conversion --link currency-exchange
in28min/currency-conversion:0.0.1-RELEASE
```


<http://localhost:8100/currency-conversion/from/USD/to/INR>

When these microservices are launched in the default bridge network, they cannot talk to each other. We have to make a link between them.

```
rangaraokaranam$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
16ead8b13b48        bridge             bridge              local
e287a8868c75        host               host                local
c2d6d5379272        none               null                local
rangaraokaranam$ docker network create currency-network
d147c6fe32b597ca8db1e7b9d2f7fccd6a1a49a37a46cd318d31a2d62f58a84d
```

As shown in the above photo, we have different docker networks. The Default bridge, host (can use only in Linux that attached to Host Network ) and none (no connection between containers).

So, We created a custom network defined as currency-network to establish two microservices.



```
/in28Minutes/git/devops-master-class/projects/hello-world/hello-world-python — AWS, Azure, Microservices and Full Stack Courses with Java and Spring Boot — -bash
...world/hello-world-python — AWS, Azure, Microservices and Full Stack Courses with Java and Spring Boot    /devops-master-class — AWS, Azure, Microservices and Full Stack Courses with Java and Spring Boot
rangaraokaranam$ docker run -d -p 8000:8000 --name=currency-exchange --network=currency-network in28min/currency-exchange:0.0.1-RELEASE
cdf82118a38e6b8562ac4e61b198140a40c3972bb26afca69a97f8a5a9c210f1
rangaraokaranam$ docker run -d -p 8100:8100 --env CURRENCY_EXCHANGE_SERVICE_HOST=http://currency-exchange --name=currency-conversion --network=currency-network in28min/currency-conversion:0.0.1-RELEASE
04755f7f8c614b3c51ab8b0ce006e3934d77e5b57143ed05cfda1af18f187f78
rangaraokaranam$
```

After that, we don't need to set a link between them. See the above picture.



### [ Save Container as images at Local ]

\*\*\*\*\*

```
$ docker container ls
```

```
$ docker commit <containerID> <kyawswartun/image_name>
```

```
$ docker images ( will show results images and saving is done )
```

### [ Push to Docker Hub ]

\*\*\*\*\*

```
$ docker tag <kyawswartun/image_name:tag>
```

```
<kyawswartun/docker_hub_img_name:tag>
```

```
$ docker push <kyawswartun/docker_hub_img_name:tag>
```

```
Example : $ docker push <kyawswartun/docker_hub_img_name:v1>
```

### [ Step for PHPMyadmin Server with MySQL ]

\*\*\*\*\*

```
$ docker run --name oursql -e MYSQL_ROOT_PASSWORD=secret -d mysql
```

```
$ docker run --name phpmyadmin --link oursql:db -p 8080:80 -d  
phpmyadmin
```

### [ Docker Volume ]

\*\*\*\*\*

```
$ docker volume ls
```

```
$ docker volume create <volume_name>
```

```
$ docker volume inspect <volume_name>
```

```
$ docker volume rm <volume_name>
```

```
$ docker run -it -v source:target <image_name>
```

```
$ docker run -it -v volume01:/mnt centos
```

```
$ docker inspect <volume_name>
```

```
$ docker run -i -v /data1/Downloads:/Downloads:z ubuntu bash
```

Note :: **z option** indicates that the bind mount content is shared among multiple containers.

**Z option** also indicate too but their contents are private and unshared

## [ Docker Compose ]

\*\*\*\*\*

To Run Multiple containers.....

In docker-compose.yml,

```
version: '3'

services:
  db:
    image: mysql:5.7
    container_name: db
    environment:
      MYSQL_ROOT_PASSWORD: my_secret_password
      MYSQL_DATABASE: app_db
      MYSQL_USER: db_user
      MYSQL_PASSWORD: db_user_pass
    ports:
      - "6033:3306"
    volumes:
      - dbdata:/var/lib/mysql
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: pma
    links:
      - db
    environment:
      PMA_HOST: db
      PMA_PORT: 3306
      PMA_ARBITRARY: 1
    restart: always
    ports:
      - 8081:80
volumes:
  dbdata:
```

**Note:** Automatically connect to default network as present location of docker-compose.yml.  
(eg: if docker-compose.yml exists under myapp directory, connect to myapp\_default network.)  
(eg: if you connect to a predefined custom network, need to write the following photo in docker-compose.yml.)

```
networks:
  default:
    external:
      name: service-tier
```

```
$ docker-compose up -d
$ docker exec -it <container_name>
(enter to container)
$ docker-compose events
$ docker-compose config (To validate compose yml file)
```

```
rangaraokaranam$ docker-compose config
ERROR: The Compose file './docker-compose.yml' is invalid because:
Unsupported config option for services.currency-exchange: 'imag'
rangaraokaranam$
```

```
$ docker-compose ps
$ docker-compose top
$ docker-compose pause
$ docker-compose unpause
$ docker-compose stop
$ docker-compose kill
```

## Example: Currency Exchange JAVA Application compose file

```
! deployment.yaml  docker-compose.yml x
projects > microservices > docker-compose.yml
1  version: '3.7'
2  services:
3    currency-exchange:
4      image: in28min/currency-exchange:0.0.1-RELEASE
5      ports:
6        - "8000:8000"
7      restart: always
8      networks:
9        - currency-compose-network
10
11    currency-conversion:
12      image: in28min/currency-conversion:0.0.1-RELEASE
13      ports:
14        - "8100:8100"
15      restart: always
16      environment:
17        CURRENCY_EXCHANGE_SERVICE_HOST: http://currency-exchange
18      depends_on:
19        - currency-exchange
20      networks:
21        - currency-compose-network
22
23  # Networks to be created to facilitate communication between containers
24  networks:
25    currency-compose-network:
```

Docker Network type တွေကို ရှာဖွေရန်

bridge ipvlan macvlan overlay

## Bridge Network

\*\*\*\*\*

- Hardware or Software device that divides a network into segments.
- Work on Data Link layer (layer 2).
- Inspect incoming frame (traffic) and decide where to forward or filter it.
- (depending on the destination MAC address which is written into every data frame)
- Three types of bridge network:
  - **Transparent Bridge**  
If the table is empty and has no MAC address, the bridge will send the frame to every node.
  - **Simple Bridge**
  - **Multit-port Bridge**

## Ipvlan

\*\*\*\*\*

L1 : distinct MAC

L2 : same MAC (share MAC)

L3 : distinct IP

## Macvlan

\*\*\*\*\*

## Comparing ipvlan to macvlan

\*\*\*\*\*

**Ipvlan** : should used in case where some switches restrict the maximum number of mac address per physical port. (due to port security configuration)

**Macvlan** : should used in case where common dhcp server is used (dhcp server would need unique mac address)

## Overlay

\*\*\*\*\*

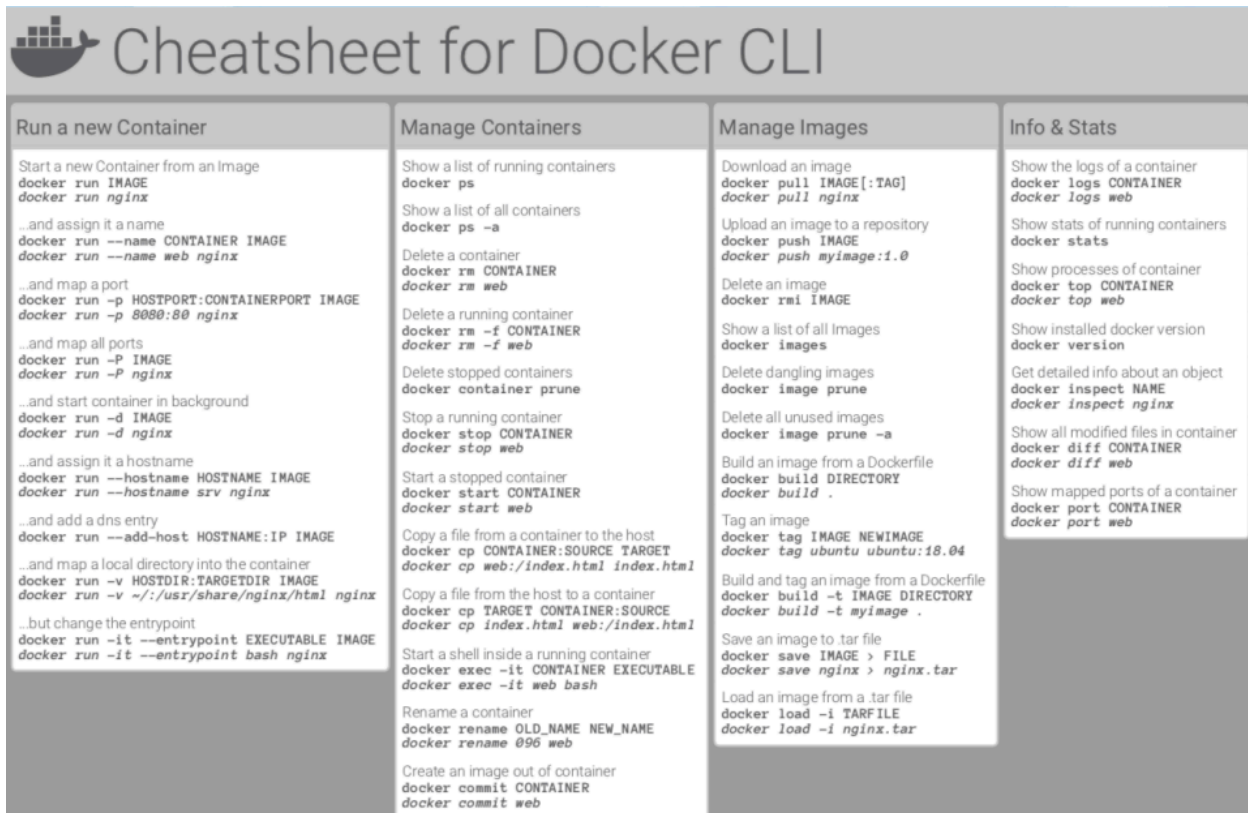
**Virtual network nodes and logical links (built on top of an existing network)**

**Aim** : To enable a new service or function without having to reconfigure the entire network design.

**Include virtual private network, peer to peer network, voice over IP services like Skype.**

**Distributed systems (peer to peer, client server app) are overlay networks.**

**(nodes run on top of the internet.)**



```
kubeadm join 192.168.1.11:6443 --token tjaj7r.jbflhxugdeitcbhm \
--discovery-token-ca-cert-hash
sha256:a495a077b9b0560b5ef58501ab7904352786a0808d602bdc537ec919e71228d4
```

eyJhbGciOiJSUzI1NiIsImtpZCI6IlE2TIB3bG8wdzg4em91U3UtQIRZZ04zQnI0cEVtWGNmMXNjWHhRUmdZNTQifQ.eYJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcY5pbpy9zZXJ2aWNiYWVudC9uYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFiY291bnQvc2VjcmV0Lm5hbWUuIiwiZG1pb10b2tlbi16Z25nYilslmt1YmVybmV0ZXMuaW8vc2VydmljZWFiY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUuIiwiZG1pb1slmt1YmVybmV0ZXMuaW8vc2VydmljZWFiY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImRmZTZgyOGJlTFIMmltNGY5Yi04YjA3LWE0M2FIMmQ1MGRjOSIsInN1Yil6lnN5c3RlbTpzZXJ2aWNiYWVudDprdWJlcm5ldGVzLWRhc2hib2FyZDphZG1pb1J9.RepepMSuzpl\_25vmq-CDWOWApefdD7G-dNFadAedp-P6jdAZC2BS0WR8fPpZHJZ7T2lsTkuKqX65AqrstdxD0Na0Yh1nHO8\_3owtU1DNS1pxWdr5sGduChHLE-5M8CzMbzYz1YWoqHdd5-dHBGtPv9DNQBvDympVRNwpgH25WglQEJ0F83SNOOnVYP24NdqKQigH0UmloO-67BAQ2t4Edn-8u-H4gkGZut394ucDnd9ZA9jNFUG3WhsCGHE\_xmibk79WIYi\_T70VB\_3A5WghISVRxzhAOK1Udh2PSiJQgAFX9wo2PEoflxYxyvwlSzu7behggQrOqOUtLiJrLAUFN\_w

## Admin Dashboard Tokens

## Testing Deployments Example

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rss-site
  labels:
    app: web
spec:
  replicas: 2
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: front-end
          image: nginx
          ports:
            - containerPort: 80
        - name: rss-reader
          image: nickchase/rss-php-nginx:v1
          ports:
            - containerPort: 88
```

```
kubeadm init --apiserver-advertise-address=192.168.203.154
--pod-network-cidr=10.244.0.0/16
```

**CGroups error >> /etc/docker/daemon.json ( fixed )**

```
sed -i "s/cgroupDriver: systemd/cgroupDriver: cgroupfs/g" /var/lib/kubelet/config.yaml
systemctl daemon-reload
systemctl restart kubelet
```

