aws ec2 run-instances --image-id ami-08ebab39574d1e681 --key-name LaptopKey --security-group-ids sg-08944ea9f10aa4121 --instance-type t2.micro --placement AvailabilityZone=us-east-1b --subnet-id subnet-0ac37305c1dc6efc5 --tag-specifications ResourceType=instance,Tags=[{Key=Name,Value=DockerSwarm-}] --count 3

for host in '3.83.78.48' '3.80.9.65' '3.218.244.146' '3.210.200.182'

do

ssh -i laptopkey.pem ubuntu@$host sudo docker swarm join --token SWMTKN-1-2swjqgcmzywmkqqhaj9rmrarxnsld92zmtl6yd356h5ivy6yci-erqe3fgi80giui4u1alk8ld4w 10.40.1.178:2377

Done

<https://www.baeldung.com/ops/docker-reduce-build-context>

docker node ls

docker node promote node

docker node demote node

docker swarm leave

docker swarm leave --force

docker service rm nginxtest

docker service ps

docker service ps --format 'table'

docker service create --name nginx --replicas 3 nginx

1. Create Swarm Cluster & Swarm basic commands. RAFT DB.

2. Understand overlay driver & Create overlay network.

<https://docs.docker.com/v17.09/engine/swarm/networking/#firewall-considerations>

3. Understand services and Create a service.

4. Updating & Scaling Swarm service.

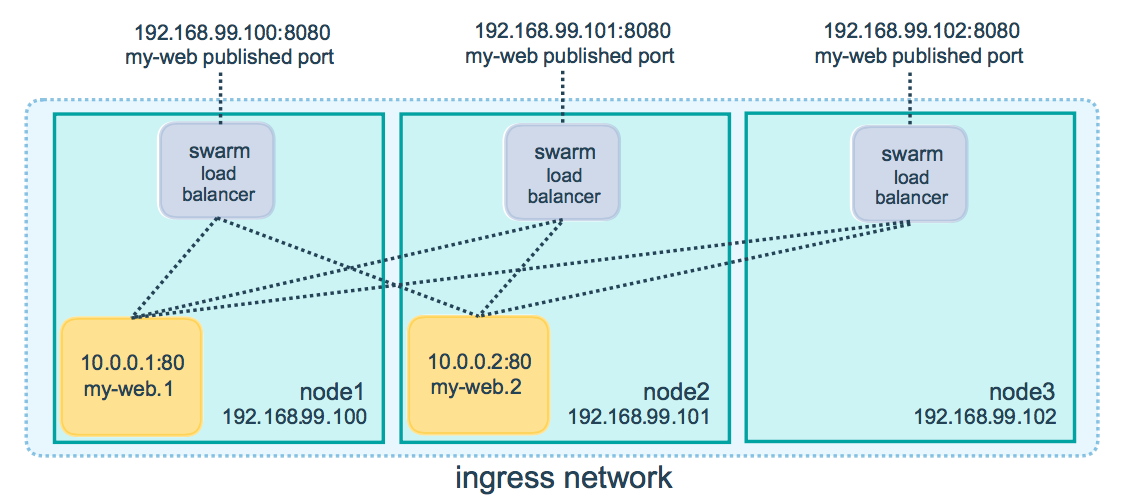
5. Routing Mesh Swarm visualizer.

<https://github.com/dockersamples/docker-swarm-visualizer>

docker run -it -d -p 8080:8080 -v \

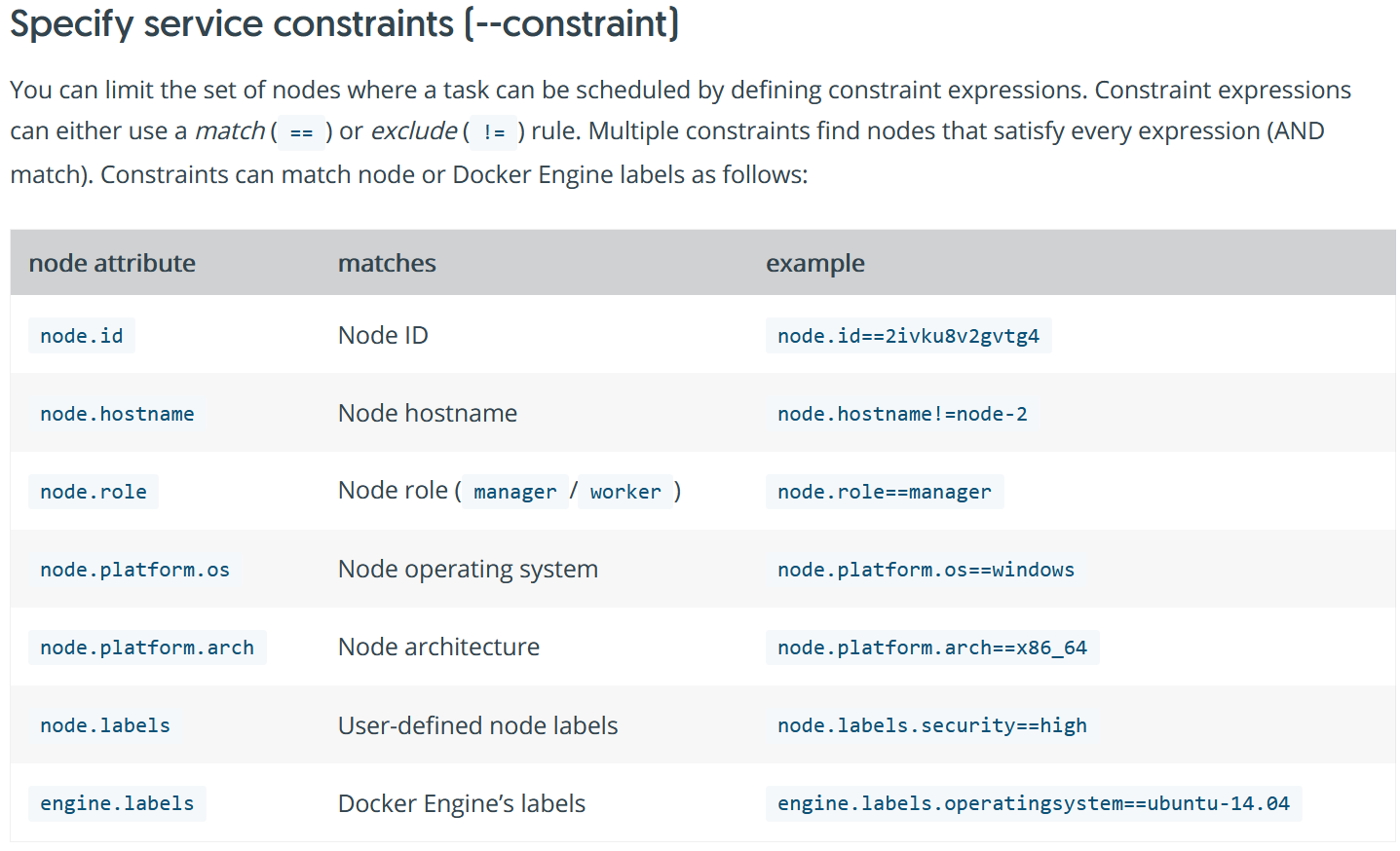
/var/run/docker.sock:/var/run/docker.sock \

dockersamples/visualizer



6. Container Placement.

* Service Constraint: [**https://docs.docker.com/engine/reference/commandline/service\_create/**](https://docs.docker.com/engine/reference/commandline/service_create/)

****

* docker service create --constraint=node.role==manager nginx
* docker service create --constraint=node.role!=worker nginx
* docker service create --name TEST --constraint 'node.role == manager' ...
* docker service create --name TEST --constraint 'node.id == w1rxedpb1mwkwbg97tb45x2dd' ...
* docker service create --name TEST --constraint 'node.hostname != docker01' ...
* node.<label>=custom label
* node.role=inbuult label
* docker node update --label-add=DEV=YES ip-10-40-1-212
* docker node update --label-add=PROD=YES ip-10-40-1-241
* docker service create --name DEVSERVICE --constraint=node.labels.DEV==YES --replicas 4 sreeharshav/rollingupdate:v5
* docker service create --name PRODSERVICE --constraint=node.labels.PROD==YES --replicas 4 sreeharshav/rollingupdate:v1

**Rolling Updates vs Blue-Green vs Canary**

[**https://opensource.com/article/17/5/colorful-deployments**](https://opensource.com/article/17/5/colorful-deployments)

7. Swarm Rolling Updates.

docker service create \

--replicas 10 \

--publish 8000:80 \

--name nginx \

--update-parallelism 1 \

--update-delay 10s \

sreeharshav/rollingupdate:v5

docker service inspect --pretty nginx

docker service update --image sreeharshav/testcontainer:v1 nginx

docker service inspect --pretty nginx

docker service update nginx

docker service ps nginx

Rolling Updates -> Perform Roll back

docker service create --name Rollback -p 8500:80 --replicas 4 sreeharshav/rollingupdate:v1

docker service update --image sreeharshav/rollingupdate:v5 Rollback

docker service update --rollback Rollback

By default Docker service brings up new container upon availability and marks it as healthy irrespective of the application status.So, HTTP requests from the client will be forwarded to new container before application came up, which then returns an error . To Solve this we need embed health check in the Dockerfile or provide health-cmd command during service update.

<https://pavankjadda.medium.com/docker-swarm-continuous-deployment-using-native-service-health-checks-5ce5b461c96c>

**Health Check in Dockerfile:**

FROM sreeharshav/rollingupdate:v5

RUN apt update && apt install -y curl jq wget

HEALTHCHECK --start-period=60s --interval=30s --timeout=5s CMD curl -f http://localhost | grep Color || exit 1

CMD ["nginx", "-g", "daemon off;"]

docker build -t sreeharshav/rollingupdate:v6 .

**Health Check while doing service update:**

docker service update --update-parallelism=5 --update-delay=30s --update-failure-action=rollback --update-order=start-first --health-cmd="curl -f http://localhost | grep Color || exit 1" --health-start-period=60s --health-interval=5s --health-timeout=30s --image sreeharshav/testcontainer:v1 rollingupdate

**Swarm Blue-Green:**

**Method 1:**

1. Deploy Two Services with 5 Replicas each using two different images
2. Service1 -> sreeharshav/rollingupdate:v5 Port-8000
3. Service2 -> sreeharshav/testcontainer:v1 Port-9000
4. Deploy AWS TG with name service1 and port 8000 and add all swarm machines to it.
5. Deploy ALB and update DNS Record and check the website is reachable. Make sure ALB has an SSL Certificate which u can get it from AWS ACM.
6. Now we need to flip the traffic from Service1 to Service2 as part of Blue-Green Deployment.
7. Create a new Listener ALB with Port HTTPS on 443 and route traffic to Servcie2.
8. Check the website connectivite from the browser.
9. Edit the service1 listener which is TCP80 and redirect to port TCP 443.
10. Check the website is reachable on service2 and all old users also redirected to the new service.

**Method 2:**  BLUE-GREEN-USING-TRAEFIK

docker service create \

--name blue \

--label traefik.port=80 \

--network traefik-net \

--label traefik.frontend.rule=Host:blue.awstelugu.xyz \

--replicas 3 sreeharshav/rollingupdate:v1

docker service create \

--name green \

--label traefik.port=80 \

--network traefik-net \

--label traefik.frontend.rule=Host:green.awstelugu.xyz \

--replicas 3 sreeharshav/testcontainer:v1

--ROLL-OUT--

docker service update --label-add 'traefik.frontend.rule=Host:blue.awstelugu.xyz' green

docker service update --label-rm 'traefik.frontend.rule=Host:green.awstelugu.xyz' green

docker service update --label-rm 'traefik.frontend.rule=Host:blue.awstelugu.xyz' blue

docker service update --label-add 'traefik.frontend.rule=Host:green.awstelugu.xyz' blue

--ROLL-BACK---

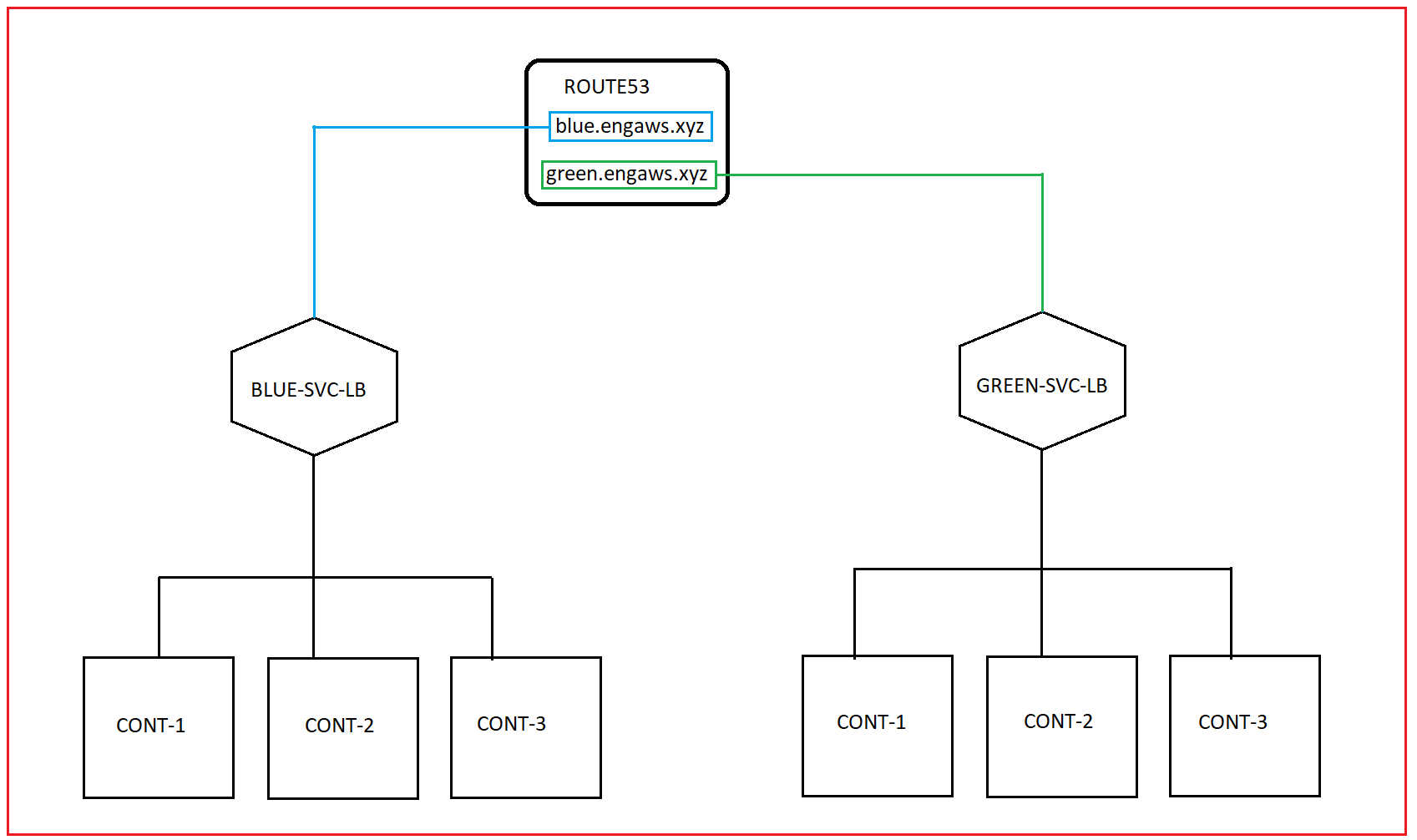
docker service update --label-rm 'traefik.frontend.rule=Host:blue.awstelugu.xyz' green

docker service update --label-add 'traefik.frontend.rule=Host:green.awstelugu.xyz' green

docker service update --label-add 'traefik.frontend.rule=Host:blue.awstelugu.xyz' blue

docker service update --label-rm 'traefik.frontend.rule=Host:green.awstelugu.xyz' blue

**Method 3:**

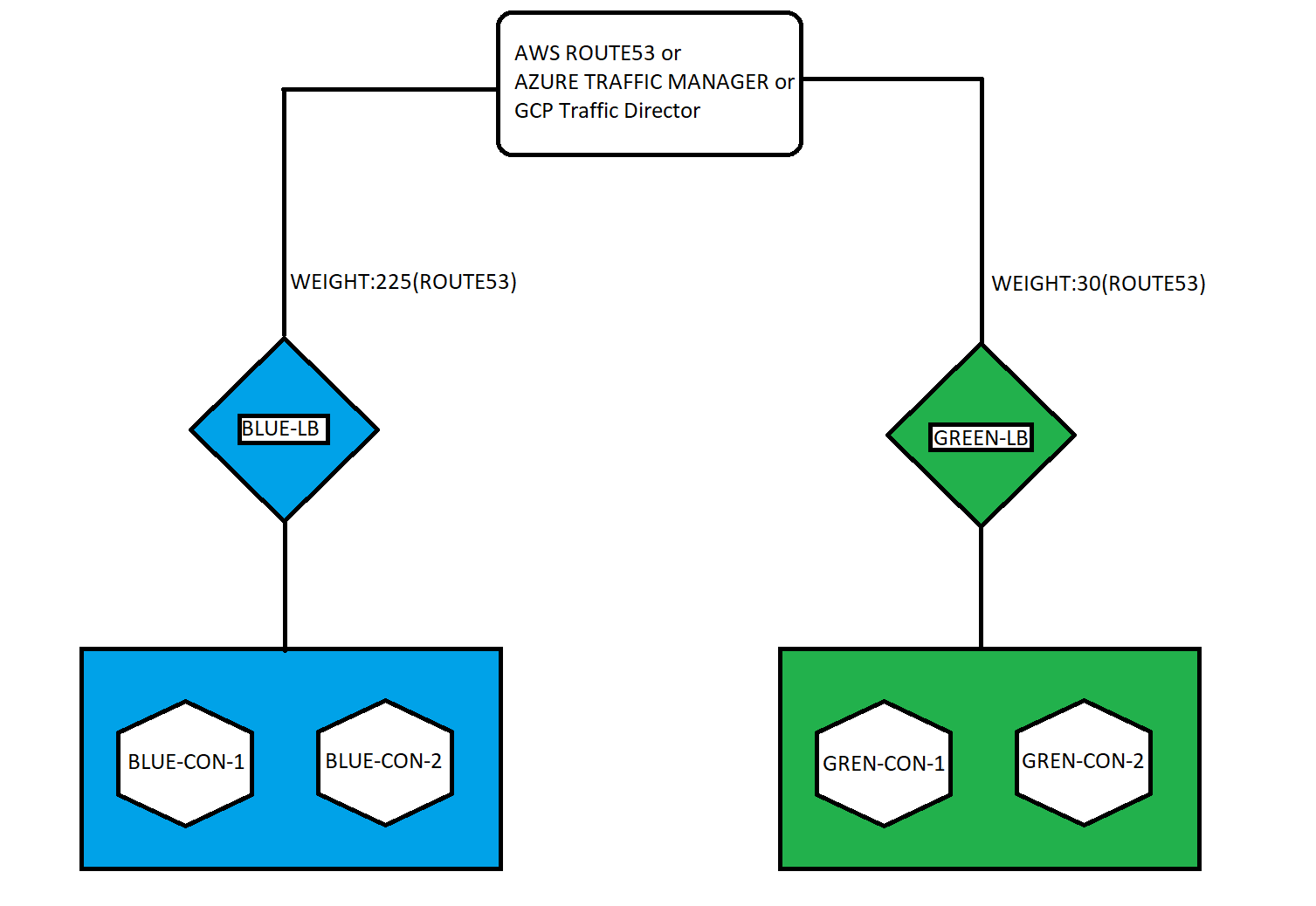


When ready change the DNS record of the blue.engaws.xyz towards GREEN-SVC-LB

--------------------------------------------------------------------------------------------------------------------------------------------

**CANARY DEPLOYMENT:**

Deploy the environment as similar to BLUE-GREEN and use the Route53 weights for sending major chunk of traffic to blue and minor chunk to green service. Check the application is working normally when the traffic is diverted to the GREEN. If everything is working well, then disable the traffic to BLUE by making the weight 0 and assigning 255 weight to green.



docker run -it -d -p 8080:8080 -v /var/run/docker.sock:/var/run/docker.sock dockersamples/visualizer

Default Labels:

docker node update --label-add mgmt=yes node1

docker node update --label-rm mgmt node1

docker node update --label-add PRODUCTION node5

docker node update --label-rm PRODUCTION node5

docker node update --label-add dev=yes node4

docker node update --label-add dev=yes node5

docker service create --name prodtest --constraint node.labels.prod==yes --replicas 2 sreeharshav/rollingupdate:v3

docker service create --name ptest --constraint node.labels.dev==yes --replicas 4 sreeharshav/rollingupdate:v3

OR

docker node update --label-add ssd=yes node2

docker node update --label-add ssd=yes node3

docker node update --label-add hdd=yes node4

docker node update --label-add hdd=yes node5

docker service create --name SSD-APP --constraint node.labels.ssd==yes --publish 9000:80 --replicas 2 sreeharshav/rollingupdate:v1

docker service create --name HDD-APP --constraint node.labels.hdd==yes --publish 6000:80 --replicas 2 sreeharshav/rollingupdate:v1

docker service create --name HDD-APP --constraint node.labels.hdd==yes --publish 6000:80 --replicas 6 sreeharshav/rollingupdate:v1

docker service create --name TESTING1 --constraint=node.role==manager -p 5000:80 --replicas 3 nginx

docker service create --name TESTING2 -p 5000:80 --constraint=node.role!=manager --replicas 4 sreeharshav/rollingupdate:v3

docker node update --label-add dev=true node2

docker node update --label-add dev=true node3

docker node update --label-add prod=true node4

docker node update --label-add prod=true node5

docker service create --name DEVNGINX --publish 7000:80 --constraint=node.labels.dev==true --replicas 4 nginx

docker service create --name PRODNGINX --publish 5000:80 --constraint=node.labels.prod==true --replicas 4 nginx

docker inspect

docker stats

docker logging drivers

**Replicas vs Global Mode:**

Deploy below repo to install prometheus and understand global mode.

git clone<https://github.com/opvizordz/docker-swarm-monitor.git>

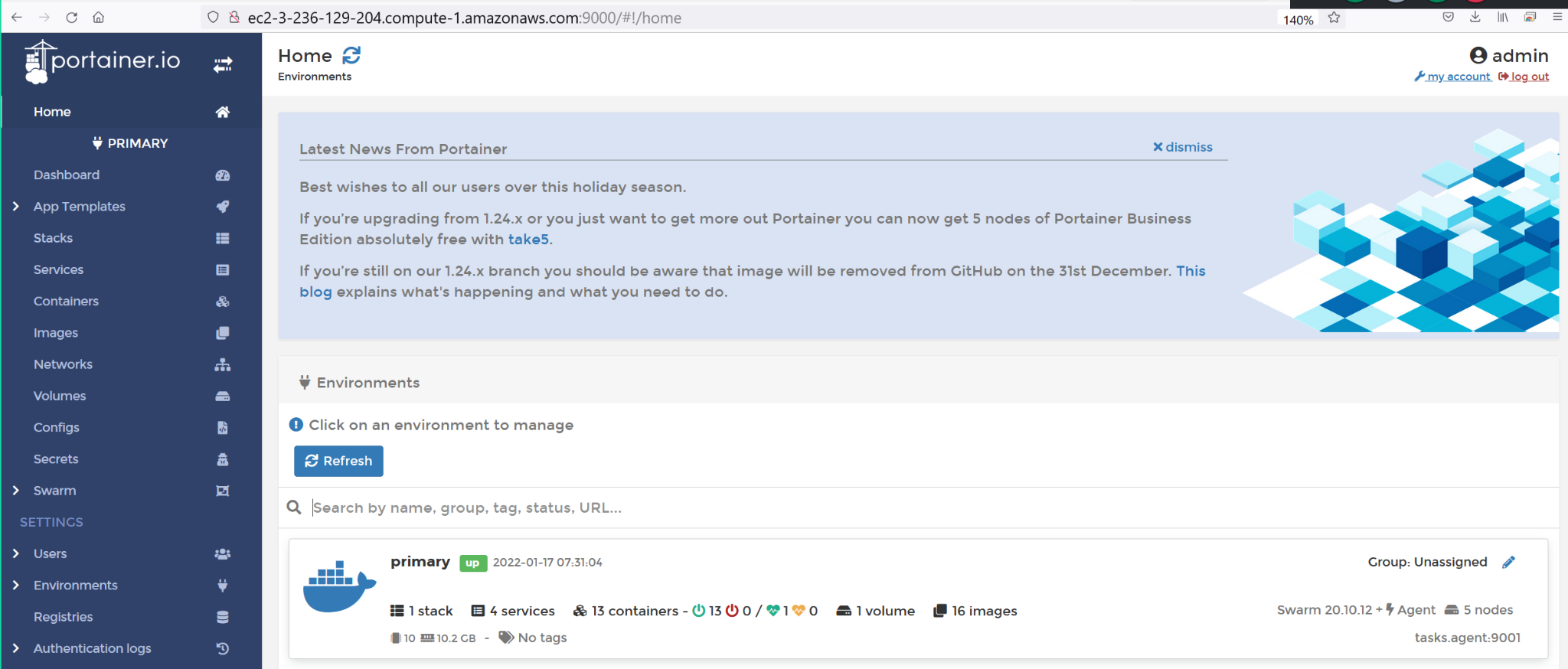
docker stack deploy -c docker-compose.stack.yml monitoring

We can test the global and replicated modes with portainer GUI application.

curl -L https://downloads.portainer.io/portainer-agent-stack.yml \

-o portainer-agent-stack.yml

docker stack deploy -c portainer-agent-stack.yml portainer



## **Swarm Scheduling Strategies**

Docker swarm uses the different scheduling strategies where to distribute the load

a.Spread

b.binpack

c.random

## Spread :

a.This is the default Strategies in docker swarm cluster. b.In this strategy docker swarm distributes the load evenly in all available Worker Nodes. c.If we have three nodes swarm cluster ,Docker swarm distribute the one containers in each node.

## binpack :

a.In this strategy docker swarm distribute the load on the node which is most packed with many container until that node can not run any containers.

## random :

a.In this strategy docker swarm distribute the load randomly on the different nodes.

We can choose the strategy by specifying the --strategy flag while swarm creation.

**Docker Stack Deploy:**

<https://github.com/dockersamples/example-voting-app>

**Docker Traefik Steps:**

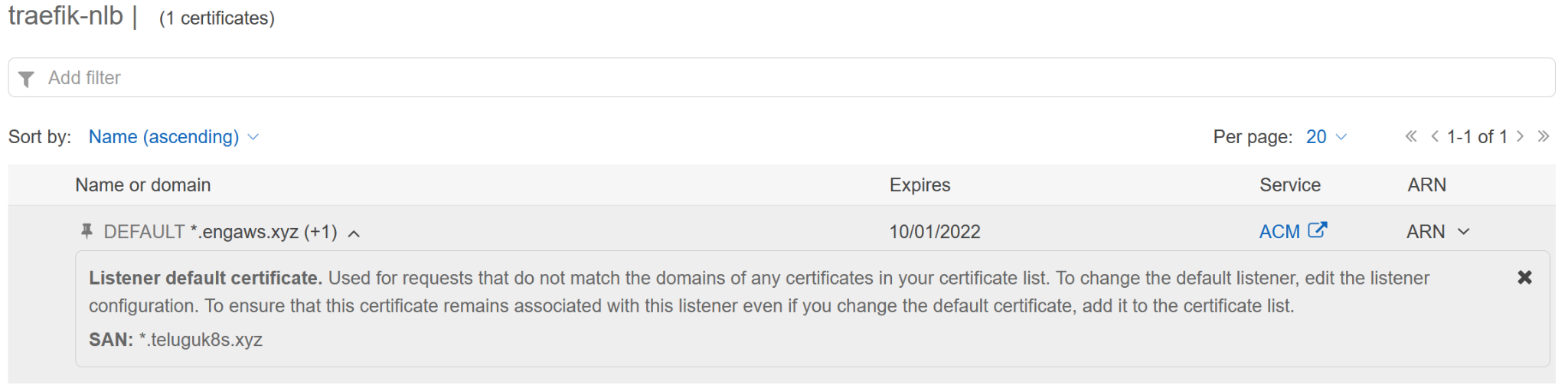
<https://blog.programster.org/using-traefik-with-docker-swarm-for-deploying-web-applications>

For Traefik 2.0 :

<https://blog.creekorful.com/2019/10/how-to-install-traefik-2-docker-swarm/>

~~We can use same traffic traefik controller for multiple domains but when using SSL Certificate, AWS NLB cannot have SSL on TCP 443 multiple domains. Even AWS ALB dont accept multiple SSL and due to that we cannot use multiple domains on SSL for TRAFIK. HTTP dont have any issue.~~

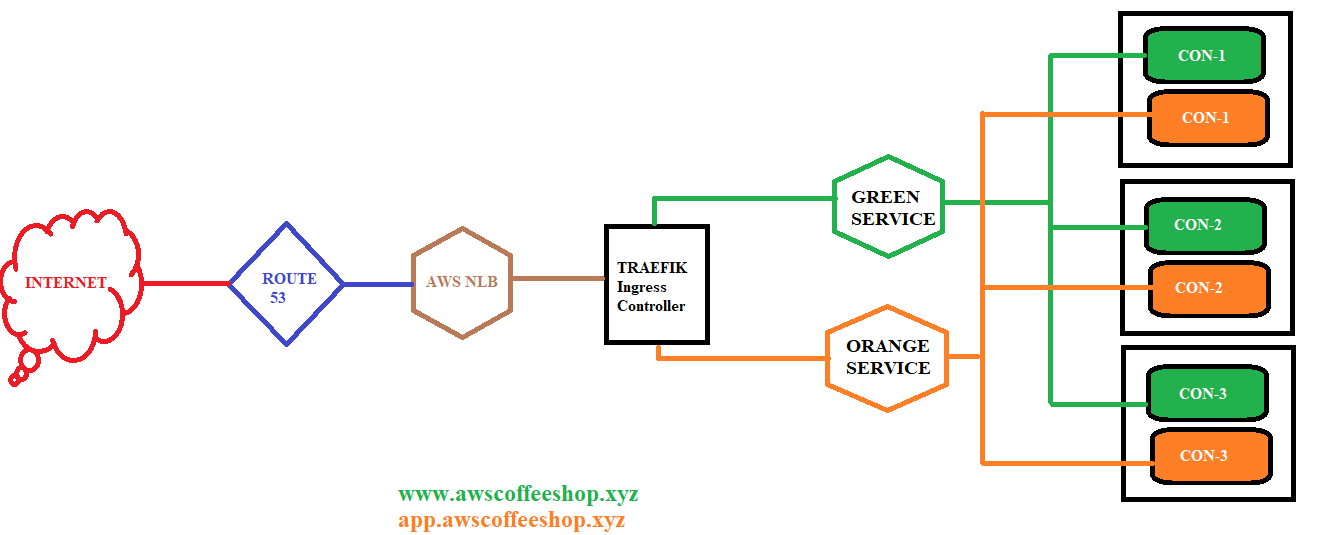
We can use multiple domain names with SSL with AWS NLB by generating certificate with multiple domain names as below.

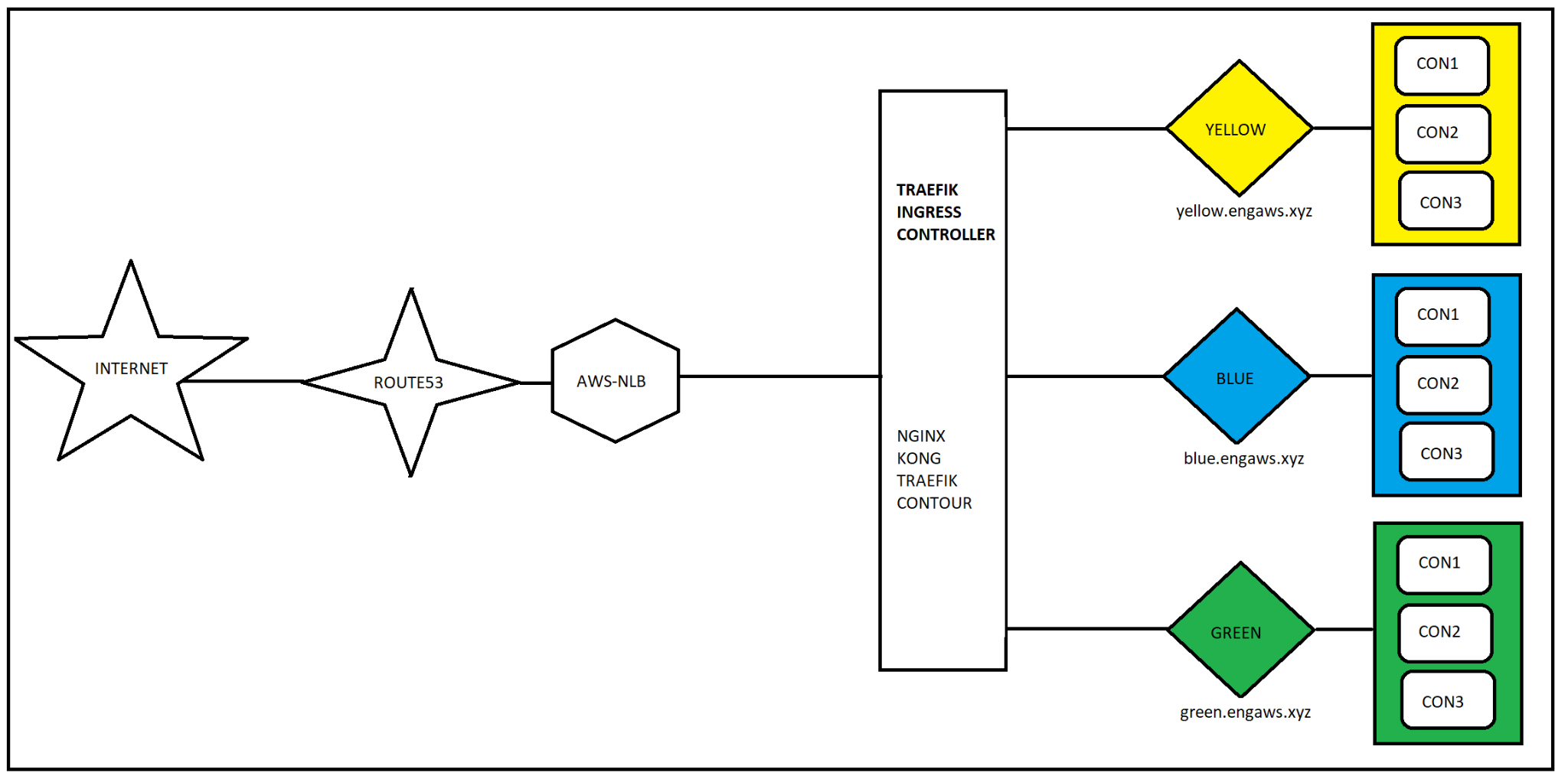


The above will generate a single certificate but the additional domain name will be a Subject Alternate Name(SAN) as shown above.

If we need to assign multiple certificates, then ALB supports.

<https://aws.amazon.com/blogs/aws/new-application-load-balancer-sni/>





**DNS Records alias to NLB:**

web1.sreetrainings.xyz

web2.sreetrainings.xyz

web3.sreetrainings.xyz

**Traefik Service Creation:**

docker network create --driver=overlay traefik-ne**t**

**TRAEFIK 1.6:**

docker service create \

--name traefik16 \

--constraint=node.role==manager \

--publish 80:80 \

--publish 9080:8080 \

--mount type=bind,source=/var/run/docker.sock,target=/var/run/docker.sock \

--network traefik-net \

traefik:v1.6 \

--docker \

--docker.swarmmode \

--docker.domain=traefik \

--docker.watch \

--web

docker service create \

--name green \

--label traefik.port=80 \

--network traefik-net \

--label traefik.frontend.rule=Host:[www.awscoffeeshop.xyz](http://www.awscoffeeshop.xyz) \

sreeharshav/rollingupdate:v5

docker service create \

--name orange \

--label traefik.port=80 \

--network traefik-net \

--label traefik.frontend.rule=Host:app.awscoffeeshop.xyz \

sreeharshav/testcontainer:v1

docker service create \

--name blue \

--label traefik.port=80 \

--network traefik-net \

--label traefik.frontend.rule=Host:[www.engaws.](http://www.engaws.xyz.com)xyz \

--replicas 3 \

sreeharshav/rollingupdate:v1

docker service create \

--name vishwadev \

--label traefik.port=80 \

--network traefik-net \

--label traefik.frontend.rule=Host:dev.vishwadrona.com\

sreeharshav/rollingupdate:v3

TRAEFIK 2.0 Deployment:

In V 2 we have:

* ***router*** that replaced *fronted*
* ***service*** that replaced *backend*
* ***middleware*** that replaced *rules*

NODE AVAILABILITY:

================

docker node update --availability pause node5 - Dont accept new tasks , runs existing.

docker node update --availability active node5

docker node update --availability drain node5 - Reschedule tasks

Resource Requirements:

====================

Limits: max

Reservations: min

docker service create --reserve-memory 300M --reserve-cpu 1 --name MEMCPUTEST1 --replicas 3 --publish 4000:80 sreeharshav/rollingupdate:v3

docker service create --name LIMITTEST --limit-cpu .25 --limit-memory 100M --replicas 3 --publish 3000:80 sreeharshav/rollingupdate:v3

docker service create --name LIMITTEST --limit-cpu .25 --limit-memory 100M --replicas 3 --publish 3000:80 sreeharshav/stress:256m

docker service create --name LIMITTEST --limit-cpu .25 --limit-memory 300M --replicas 3 --publish 3000:80 sreeharshav/stress:256m

docker service update LIMITTEST --limit-memory 10M --limit-cpu .10

docker service update LIMITTEST --limit-memory 0 --limit-cpu 0

docker service rm DEVNGINX LIMITTEST MEMCPUTEST1 PRODNGINX

docker service create --name CPULARGE2 --reserve-cpu 80 sreeharshav/rollingupdate:v3

version: "3.8"

services:

vote:

image: sreeharshav/stress:256m

deploy:

mode: replicated

replicas: 3

resources:

limits:

cpus: '0.50'

memory: 300M

reservations:

cpus: '0.25'

memory: 100M

ports:

- "5000:80"

networks:

- front-tier

- back-tier

networks:

front-tier:

back-tier:

STRESS Dockerfile:

================

FROM debian:latest

RUN apt-get update && apt-get install -y stress \

--no-install-recommends && rm -r /var/lib/apt/lists/\*

CMD ["stress", "--verbose", "--vm", "1", "--vm-bytes", "256M"]

Container Healthchecks:

====================

Dockerfile for health-checks:

FROM sreeharshav/rollingupdate:v4

RUN apt update && apt install -y curl

HEALTHCHECK --interval=5s --timeout=10s --start-period=5s --retries=1 CMD curl --fail http://localhost || exit 1

CMD ["nginx", "-g", "daemon off;"]

docker build -t <yourdockerhubaccount>/healthcheck:latest

docker service create --name healthcheck -p 8100:80 --replicas 3 sreeharshav/healthcheck:v5

Docker container healthchecks

<https://blog.sixeyed.com/docker-healthchecks-why-not-to-use-curl-or-iwr/>

HEALTHCHECK CMD curl --fail http://localhost:3000/ || exit 1

docker inspect --format='{{json .State.Health}}' your-container-name

docker service logs NGINX

docker service logs wza88dx6v4pr

docker service logs wza88dx6v4pr --no-task-ids

docker service logs --raw --no-trunc wza88dx6v4pr

docker service logs --raw --no-task-ids --no-trunc wza88dx6v4pr

docker service logs --tail 10 --follow --raw --no-trunc NGINX

==========================================================================

https://docs.docker.com/engine/reference/commandline/events/

=====================================================================

**DOCKER CONFIG:**

docker config create nginxindex1 index.html

docker config create nginxconf1 default.conf

Nginx config in ubuntu is /etc/nginx/conf.d/nginx.conf

Nginx default config in ubuntu location /etc/nginx/conf.d/default.conf

docker service create --name nginx1 --config src=nginxindex1,target=/usr/share/nginx/html/index.html --publish 8000:80 sreeharshav/rollingupdate:v5

docker service create --name nginx1 --config src=nginxindex1,target=/usr/share/nginx/html/index.html --config src=nginxconf1,target=/etc/nginx/conf.d/default.conf --publish 9100:80 --publish 9000:88 sreeharshav/rollingupdate:v5

docker service update --config-rm config1 --config-add src=config2,target=/usr/share/nginx/html/index.html nginx1

docker service update nginx1 --config-rm nginxindex1 --config-add src=nginxindex2,target=/usr/share/nginx/html/index.html

We can also mount certificates as well with cofnig. Create a file with a sample certificate as shown below with filename as hello.pem

-----BEGIN CERTIFICATE-----

MIIESTCCAzGgAwIBAgITBntQXCplJ7wevi2i0ZmY7bibLDANBgkqhkiG9w0BAQsF

CAYGZ4EMAQIBMA0GCSqGSIb3DQEBCwUAA4IBAQAfsaEKwn17DjAbi/Die0etn+PE

gfY/I6s8NLWkxGAOUfW2o+vVowNARRVjaIGdrhAfeWHkZI6q2pI0x/IJYmymmcWa

ZaW/2R7DvQDtxCkFkVaxUeHvENm6IyqVhf6Q5oN12kDSrJozzx7I7tHjhBK7V5Xo

-----END CERTIFICATE-----

docker config create hellocrt hello.pem

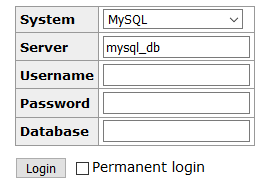
docker service create --name nginx3 --config src=hellocrt,target=/etc/ssl/certs/hello.crt --publish 9100:80 sreeharshav/rollingupdate:v5

Docker Swarm Secrets:

===================

<https://blog.ruanbekker.com/blog/2017/11/23/use-docker-secrets-with-mysql-on-docker-swarm/>

**Use the file docker\_secrets\_mysql.yml for testing the secrets. Access adminer service on port 8080 to login to MySQL.**

****

openssl rand -base64 12 | docker secret create db\_root\_password -

openssl rand -base64 12 | docker secret create db\_dba\_password -

docker service create --name nginx01 --secret db\_root\_password --secret db\_dba\_password --publish 8000:80 sreeharshav/rollingupdate:v5

docker network create mysql\_private --driver overlay

openssl rand -base64 12 | docker secret create mysql\_root\_password -

openssl rand -base64 12 | docker secret create mysql\_password -

docker service create \

--name mysql \

--replicas 1 \

--network mysql\_private \

--mount type=volume,source=mydata,destination=/var/lib/mysql \

--secret source=mysql\_root\_password,target=mysql\_root\_password \

--secret source=mysql\_password,target=mysql\_password \

-e MYSQL\_ROOT\_PASSWORD\_FILE="/run/secrets/mysql\_root\_password" \

-e MYSQL\_PASSWORD\_FILE="/run/secrets/mysql\_password" \

-e MYSQL\_USER="wordpress" \

-e MYSQL\_DATABASE="wordpress" \

mysql:latest

It is not possible to directly access docker secrets as ENV variables. We need to declare the ENV variables by reading the password from the /run/secrets. In kubernetes its possible to direcvtly mount the passwords as ENV variables.

Jenkins Docker Swarm Deploy:

=========================

nano /lib/systemd/system/docker.service

ExecStart=/usr/bin/dockerd -H unix:// -H tcp://0.0.0.0:2375

systemctl daemon-reload

systemctl restart docker

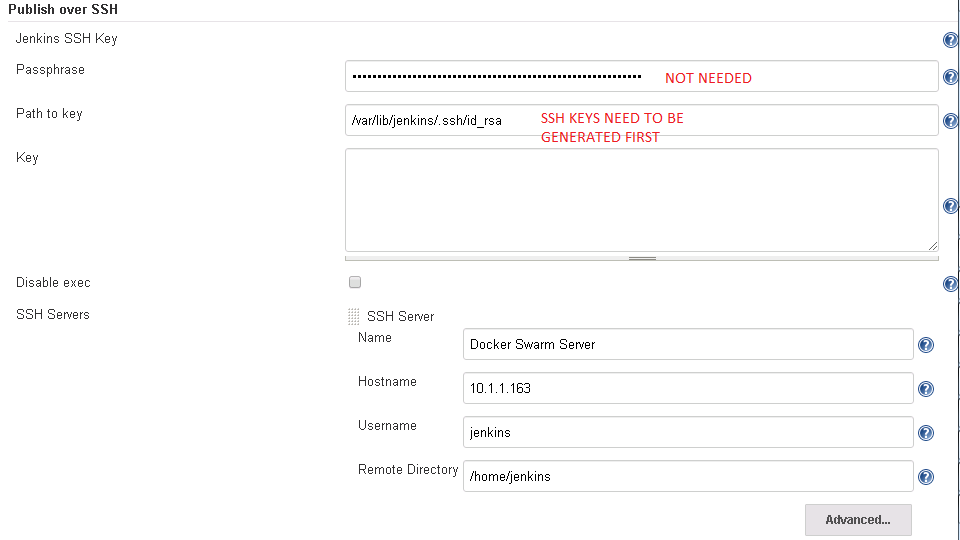
sudo nohup docker daemon -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock &

sudo usermod -a -G root jenkins

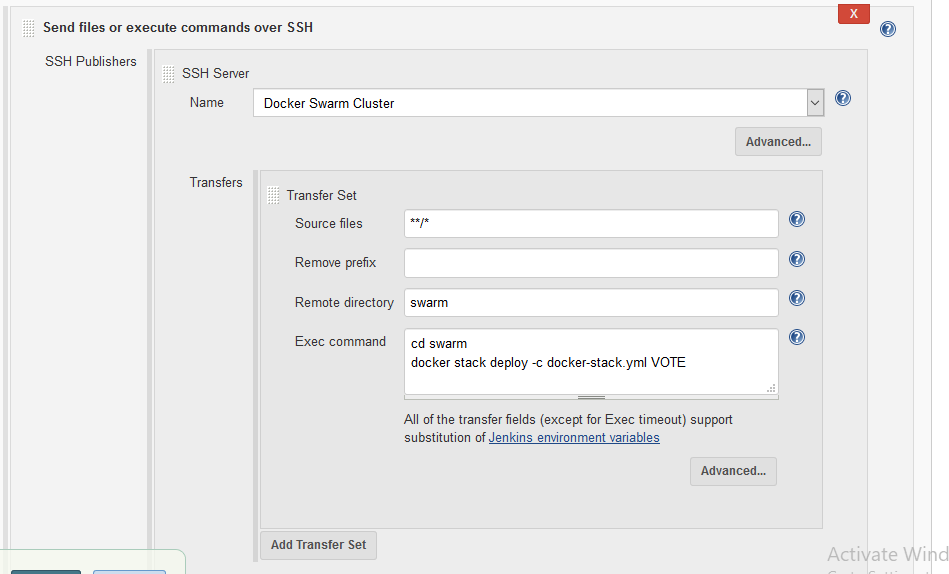
usermod -a -G docker jenkins

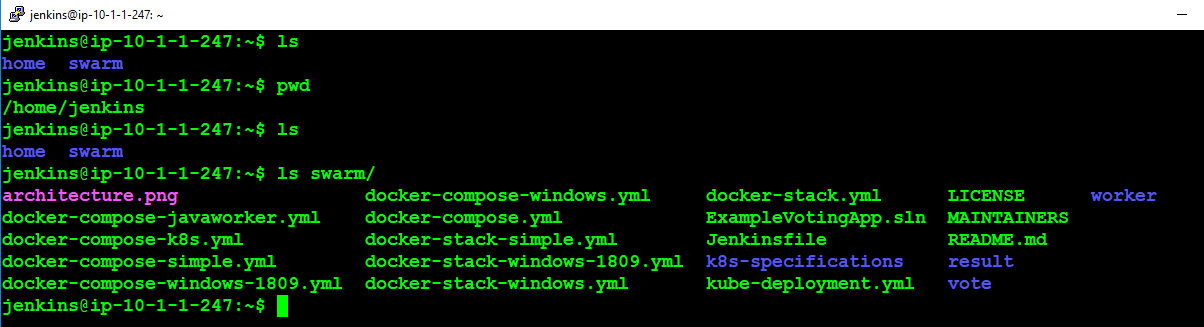
Following is valid if you want to run docker-compose and build images from jenkins in docker host.If you are planning to run a stack deploy, then there is not neeed to build any images. So, we can run docker -H tcp://10.1.1.156:2375 stack deploy -c compose-file

<http://www.littlebigextra.com/automate-service-deployment-docker-swarm-using-jenkins/>



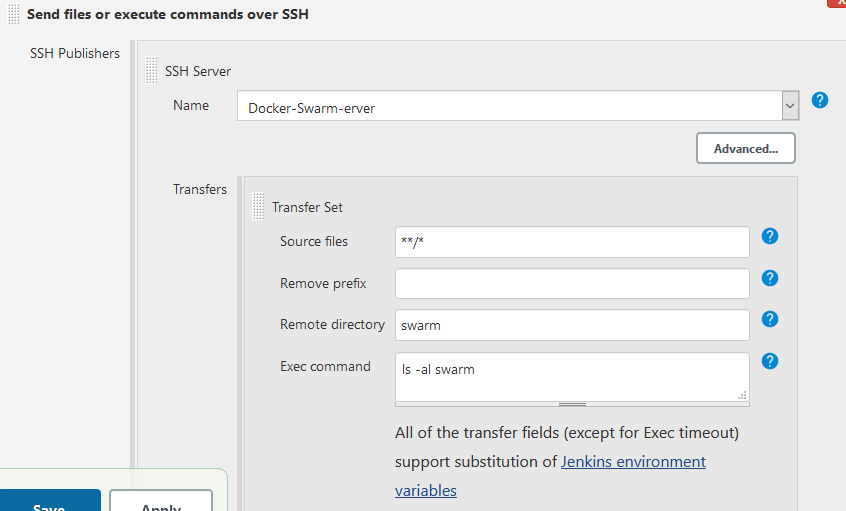
OLD-PIC:





NEW-PIC:

Now Create a new Jenkins Job with the following settings in the PIC:



<https://linuxize.com/post/how-to-install-jenkins-on-ubuntu-18-04/>

nano /lib/systemd/system/docker.service

ExecStart=/usr/bin/dockerd -H unix:// -H tcp://0.0.0.0:2375

systemctl daemon-reload

systemctl restart docker

sudo nohup docker daemon -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock &

sudo usermod -a -G root jenkins

usermod -a -G docker jenkins

systemctl daemon-reload

systemctl restart docker

service jenkins restart #Do restart jenkins from browser using /restart

===========DOCKER-COMPOSE=====================

<https://docs.docker.com/compose/wordpress/>

version: '3.3'

services:

db:

image: mysql:5.7

volumes:

- db\_data:/var/lib/mysql

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: somewordpress

MYSQL\_DATABASE: wordpress

MYSQL\_USER: wordpress

MYSQL\_PASSWORD: wordpress

wordpress:

depends\_on:

- db

image: wordpress:latest

ports:

- "8000:80"

restart: always

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: wordpress

WORDPRESS\_DB\_PASSWORD: wordpress

WORDPRESS\_DB\_NAME: wordpress

volumes:

db\_data: {}

=========================

pipeline {

environment {

registry = "sreeharshav/devopsb16"

registryCredential = 'dockerhub\_id'

dockerImage = ''

}

agent any

stages {

stage('Cloning our Git') {

steps {

git 'https://github.com/mavrick202/dockertest1.git'

}

}

stage('Building our image') {

steps {

script {

dockerImage = docker.build registry + ":v$BUILD\_NUMBER"

}

}

}

stage('Push Image To DockerHUB') {

steps {

script {

docker.withRegistry('', registryCredential) {

dockerImage.push()

}

}

}

}

stage('Cleaning up') {

steps {

**sh** "docker rmi $registry:v$BUILD\_NUMBER"

}

}

stage('Deploying to Docker Swarm') {

steps {

**sh** "docker -H tcp://10.40.1.172:2375 service rm testing1 || true"

**sh** "docker -H tcp://10.40.1.172:2375 service create --name testing1 -p 8100:80 $registry:v$BUILD\_NUMBER"

}

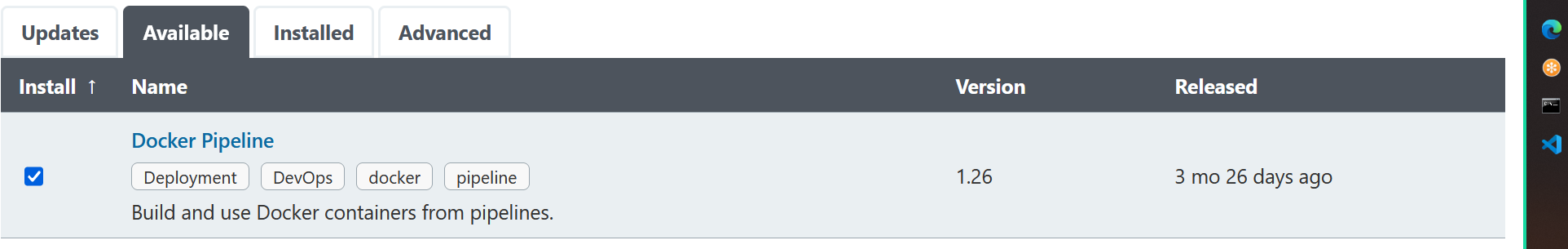
}

}

}

===============================================================

Install below plugin prior to running the following pipeline



pipeline {

environment {

registry = 'sreeharshav/devopsb17'

registryCredential = 'dockerhub\_id'

dockerSwarmManager = '10.40.1.26:2375'

dockerhost = '10.40.1.26'

dockerImage = ''

}

agent any

stages {

stage('Cloning our Git') {

steps {

git 'https://github.com/mavrick202/dockertest1.git'

}

}

stage('Building our image') {

steps {

script {

dockerImage = docker.build registry + ":v$BUILD\_NUMBER"

}

}

}

stage('Push Image To DockerHUB') {

steps {

script {

docker.withRegistry( '', registryCredential ) {

dockerImage.push()

}

}

}

}

stage('Cleaning up') {

steps {

sh "docker rmi $registry:v$BUILD\_NUMBER"

}

}

stage('Deploying to Docker Swarm') {

steps {

sh "docker -H tcp://$dockerSwarmManager service rm testing1 || true"

sh "docker -H tcp://$dockerSwarmManager service create --name testing1 -p 8100:80 $registry:v$BUILD\_NUMBER"

}

}

stage('Verifying The Deployment') {

steps {

sh 'curl http://$dockerhost:8100 || exit 1'

}

}

}

}

Used Docker Contexts and multi-branch build - Jan 20th 2022:

pipeline {

environment {

registry = 'sreeharshav/devopsb22prod'

registryCredential = 'dockerhub\_id'

devcontext = 'dev-swarm'

prodcontext = 'prod-swarm'

devnode = '10.40.1.237'

prodnode = '10.40.1.149'

dockerImage = ''

}

agent any

stages {

stage('Building Prod Docker Branch') {

when {

expression {

return env.BRANCH\_NAME != 'DevOpsB22-Prod'

}

}

steps {

script {

dockerImage = docker.build registry + ":v$BUILD\_NUMBER"

}

}

}

stage('Push Image To DockerHUB') {

steps {

script {

docker.withRegistry( '', registryCredential ) {

dockerImage.push()

}

}

}

}

stage('Cleaning up') {

steps {

sh "docker rmi $registry:v$BUILD\_NUMBER"

}

}

stage('Deploying to Prod Docker Swarm') {

steps {

sh "docker context use $prodcontext"

sh 'docker service rm testing1 || true'

sh "docker service create --name testing1 -p 8100:80 $registry:v$BUILD\_NUMBER"

}

}

stage('Verifying The Deployment') {

steps {

sh 'curl http://$prodnode:8100 || exit 1'

}

}

stage('Change Context To Default') {

steps {

sh 'docker context use default'

}

}

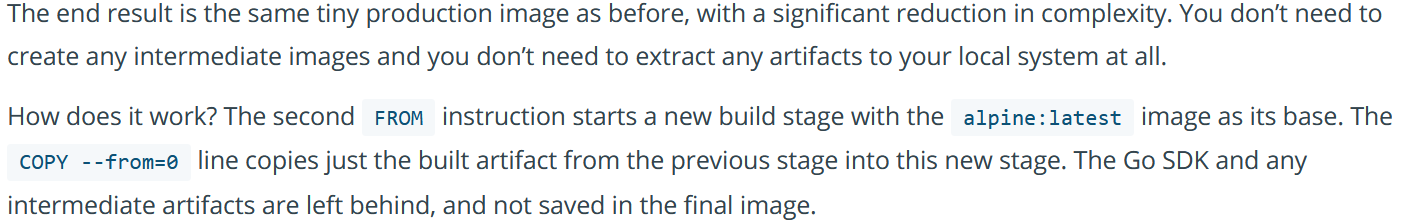
}

}

Docker Multi Stage Builds:

<https://docs.docker.com/develop/develop-images/multistage-build/>





Distroless Images:

<https://betterprogramming.pub/how-to-harden-your-containers-with-distroless-docker-images-c2abd7c71fdb>

<https://medium.com/@luke_perry_dev/dockerizing-with-distroless-f3b84ae10f3a>

**Docker Contexts:**

Before using the contexts make sure you have made the following changes in the Prod and Dev Swam Managers.

nano /lib/systemd/system/docker.service

ExecStart=/usr/bin/dockerd -H unix:// -H tcp://0.0.0.0:2375

systemctl daemon-reload

systemctl restart docker

docker context create dev-swarm \

--default-stack-orchestrator=swarm \

--docker host=tcp://10.40.1.237:2375

docker context create prod-swarm \

--default-stack-orchestrator=swarm \

--docker host=tcp://10.40.1.149:2375

docker context ls

docker context use dev-swarm

docker service create --name nginx001 -p 8000:80 sreeharshav/rollingupdate:v5

Check *docker service ls* in dev-swarm cluster