**Create docker container on ec2 instance**

1. Create an ec2 instance from the aws console.
2. Log into ec2 instance using Xshell
3. Install and start docker on the ec2 instance

Update the installed packages and package cache on your instance.

$yum update -y

Install the most recent Docker Engine package.  
 $yum install docker

Start the Docker service.  
 $service docker start

(Optional) On Amazon Linux 2, to ensure that the Docker daemon starts after each system reboot, run the following command:  
 $sudo systemctl enable docker

Add the ec2-user to the docker group so you can execute Docker commands without using sudo.

$usermod -a -G docker ec2-user

1. Pull a docker image

(In here I have used ubuntu image)

$docker pull ubuntu

1. Create container from that image

$docker create -it -- name <container-name> -p 8080:80 ubuntu

(You need to expose port 80 and 8080 in aws security group which used to create the instance)

1. Start the container

$docker start <container-name>

1. Log into the container

$docker exec -it <container-name> /bin/sh

references : ***https://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html***

# 

# **Deploy flask app with Nginx**

(Now you should be in the container)

1. First of all, you should update your local package index and then install the packages.

# update your local packages

$apt-get update

# install dependencies

$apt-get install python3-pip python3-dev nginx

1. Install Flask and Gunicorn

$pip install gunicorn flask

1. Install requests

$pip install requests

1. Create a folder to put the scripts (in here I have created a folder named lseg)

$mkdir lseg

1. Goto the lseg folder

$cd lseg

1. Create **app.py** and **time\_zone.py** files inside the lseg folder
2. Create the wsgi entry point

Next, we’ll create a file that will serve as the entry point for our application. This will tell our Gunicorn server how to interact with the application.

Create **wsgi.py** file

1. Testing the gunicorn’s ability to serve the project

gunicorn --bind 0.0.0.0:80 wsgi:app

Visit http//<ec2-ip>:8000

You will see results in browser

1. Create a systemd unit file

systemd unit file will allow Ubuntu’s init system to automatically start Gunicorn and serve our Flask application whenever the server boots.

Create the unit file **app.service** within the /etc/systemd/system directory

1. Start and enable gunicorn service

systemctl start app

systemctl enable app

After these steps A new file app.sock will be created in the project directory automatically.

Our folder structure:

lseg

|\_\_\_\_ app.py

|\_\_\_\_ time\_zone.py

|\_\_\_\_ wsgi.py

|\_\_\_\_ app.sock

1. Configuring nginx

Gunicorn application server is now up and running and it waits for requests on the socket file in the project directory. We need to configure *Nginx* to pass web requests to that socket by making some small additions to its configuration file.

We’ll need to tell NGINX about our app and how to serve it.

create a new server block configuration file in Nginx’s sites-available directory named app

Location : /etc/nginx/sites-available/**app**

1. Enable nginx server block

Link the file to the sites-enabled directory to enable the Nginx server block we’ve just created.

ln -s /etc/nginx/sites-available/app /etc/nginx/sites-enabled

* Test syntax errors by typing : $nginx -t
* Restart the nginx server

$systemctl restart nginx

Now go to your server’s domain name or IP address in your web browser , your application is running.

http://<Ip>:8080

references: <https://faun.pub/deploy-flask-app-with-nginx-using-gunicorn-7fda4f50066a>

**Install aws cli in the docker container**

1. Install curl in docker container

$pip install curl

1. Download aws cli zip file

$curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

1. Install unzip

$pip install unzip

1. Unzip the aws cli zip file

$unzip awscliv2.zip

1. Install the cli

$./aws/install

**Create an image from the container and push it to the amazon ECR**

1. Create image from the container

Command : docker commit c10ff3d190c9 flask-app:version01

→C10ff3d190c9 : container id

→Flask-app : image name

→Version01 : tag

1. Create ECR repository from aws console

references: **https://docs.aws.amazon.com/AmazonECR/latest/userguide/repository-create.html**

My repository name : flask-app

1. Push the image to ECR

##### Retrieve an authentication token and authenticate your Docker client to your registry

Command : $aws ecr get-login-password --region region | docker login --username AWS --password-stdin aws\_account\_id.dkr.ecr.region.amazonaws.com

My command : $aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 257997452906.dkr.ecr.us-east-1.amazonaws.com

##### tag your image so you can push the image to this repository

Command : $docker tag flask-app:latest 257997452906.dkr.ecr.us-east-1.amazonaws.com/flask-app:latest

##### Run the following command to push this image to your newly created AWS repository

Command : $docker push 257997452906.dkr.ecr.us-east-1.amazonaws.com/flask-app:latest

* Pull image from aws repository

Command : $docker pull aws\_account\_id.dkr.ecr.region.amazonaws.com/hello-world:latest

docker pull 257997452906.dkr.ecr.us-east-1.amazonaws.com/flask-app:latest

**Create multiple containers from the image**

1. Initialize swarm

Command : $docker swarm init

1. Create container

Command : $docker service create --name flask-app --replicas 3 --publish published=8080,target=80 lseg3 tail -f /dev/null

Flask-app : container name

Lseg3 : image name

Stop the service : $docker service rm flask-app (please use when you need to stop the service)

**Create mail service**

1. Create a topic in amazon sns service https://console.aws.amazon.com/sns/v3/home?region=us-east-1#/homepage
2. Then create a subscription using Create Subscription
3. Then create **mail.py** file in lseg folder in docker container
4. Update the app.py file

You may need to install boto3 in docker container

$pip install boto3

**Configure logging mechanism**

1. Create a log group in aws cloudwatch service

https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#

1. Then create a log stream
2. Update the app.py script in docker container

When you update any file in docker container please restart app service and nginx.

$systemctl restart app

$systemctl restart nginx

**\*\*\*\*Check the results**

Please type the following url in your browser then you can see the results of the running application.

**http://34.201.65.107:8083/**

All the scripts and files are available in the **github repository**.