https://testdriven.io/blog/deploying-django-to-ec2-with-docker-and-gitlab/ deploying dockerized Django app on EC2

Thursday, October 3, 2019 11:33 PM

Automation with Scripting for boto 3 tutoral
https://boto3.amazonaws.com/v1/documentation/api/latest/guide/sqs.html boto 3 tutorial
From <https://www.youtube.com/watch?v=9occfhrM4gg&list=PL2qzCKTbjutJ1zZFYNImrHNbzs6XgIHbI&index=3>
https://www.bmc.com/blogs/microservices-architecture/ microservies
https://gist.github.com/bradtraversy/cfa565b879ff1458dba08f423cb01d71 **django deployment on Ubuntu sever and digital ocean**
https://testdriven.io/blog/storing-django-static-and-media-files-on-amazon-s3/ user uploded static media file to S3 directly
https://simpleisbetterthancomplex.com/tutorial/2017/08/01/how-to-setup-amazon-s3-in-a-django-project.html uploading user uploaded static file into S3 directly.
https://mherman.org/blog/dockerizing-a-react-app/ dockerized React applications. https://dragonprogrammer.com/dockerized-django-api-angular-tutorial/ django & angular

What is EC2 and their security group ?
EC2 is elastic compute to create virtual machine on cloud, we can say EC2 is virtual machine server whenever you need to build any server we have to use EC2. security group allow which service you want to use in virtual machine. You can enable or disable your service thorough security group. Security group like HTTP, HTTPS, SSH.

Where we can use pem key and PPK key ?
To connect the Machine . If you use windows then **pem key(public key)** is enough. If you want to connect Linux machine then we use putty. Putty not support pem key so you have to convert into **PPK(private key)** by putty generator .then we can connect with Linux machine.

How many ways billing happening(charging cost) in S3?
Base on they charge
1. **Storage size:** how much data you store
2. **Transfer rate:** how much data you store
3. **Get and put request :** how many request are put and getting

How to transfer data directly between EC2 to S3 ?
Trough set up IAM role (s3 access)
And User credentials

What is the option to create communication between two different network ?
If I want to connect one EC2 instance to another EC2 instance with different network
- By using **VPC peering** we can connect with one network to another network.

Example of command to create OS backup on EC2 .
Aws ec2 create-image -- innstat -id <your Instant id > -- name " OS BKP"-- description "any discription you can write "

example of terminate the instance
Aws ec2 terminate_ instance - instance type <instance id>

What is the IOPS value for 20GB in provisioned volume type?
1000 IOPS (ex 1 gb = 50 iops)
For **Magnetic volume** there is no IOPS

Maximum How many bucket can create per region?
We can create maximum **100 Buckets** per region

If you want to deploy windows instance in AWS , which security services has to be enabled ?
RDP security services need to enable for windows instance deploy in AWS
If you deploy Linux machine we should do **SSH** security services enabled

What is the minimum subnets size you can have in VPC
30

Type of load balancer are available in AWS
3 types **Classic , Network and application** load balancer

What is the token (key) using for key (linux)
RSA

How buffer work as AWS
used to make the system more robust and manage traffic by synchronizing different components.The component processes the requests in an imbalanced way.Using buffer, the components work at the same speed for faster services and will also be balanced.

From <https://www.onlineinterviewquestions.com/aws-interview-questions/#accordionEx2>

What is auto scaling?

AWS Auto Scaling monitors your applications and automatically adjusts capacity to maintain steady, predictable performance at the lowest possible cost. Using **AWS Auto Scaling**, you can setup **scaling** for multiple resources across multiple services in minutes.

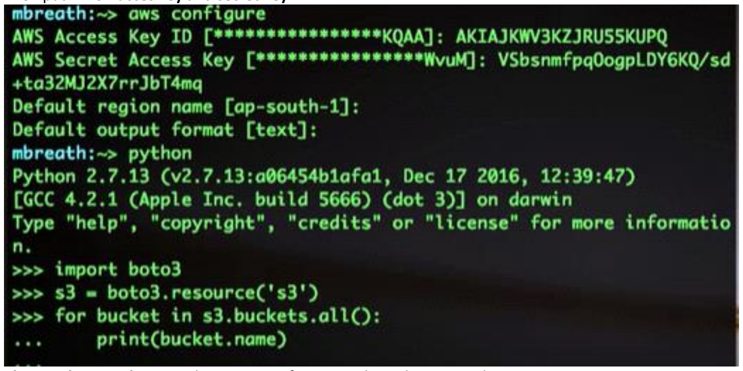
From <https://www.google.com/search?q=auto+scaling+in+aws&oq=auto&aqs=chrome.0.69i59j69i57j0l2j69i60j69i61.1419j0j7&sourceid=chrome&ie=UTF-8>

how to secure data caring in the cloud?

there is **no leakage with the security key** from various storerooms in the cloud, we can rest assured that the data in the cloud is secured.
Another option available is **segregation of the information** from the information of additional companies and then encrypting them by means of approved methods.

AWS Command line interface

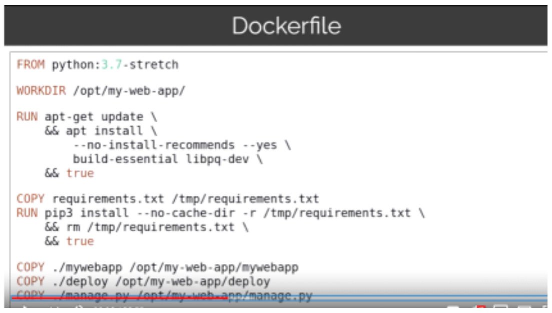
Pip install **boto 3**
Pip install awscli
Aws configure
before that you have to go **amazon web service at IAM** then create user then you will get access key
Then put **AWS Aacces key and secret key**



```
mbreath:~> aws configure
AWS Access Key ID [****************KQAA]: AKIAJKWV3KZJRU5SKUPQ
AWS Secret Access Key [****************WvuM]: VSbsnmfpqOogpLDY6KQ/sd
+ta32MJ2X7rrJbT4mq
Default region name [ap-south-1]:
Default output format [text]:
mbreath:~> python
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 12:39:47)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more informatio
n.
>>> import boto3
>>> s3 = boto3.resource('s3')
>>> for bucket in s3.buckets.all():
...     print(bucket.name)
```

Then we have to do S3 Bucket CORS Configuration through command prompt
Then go for Clint method for download file from S3

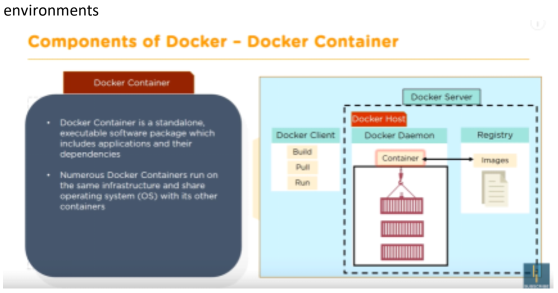
You can also download the file through resource method from s3 **resource method is faster then client method**



Dockerfile

```
FROM python:3.7-stretch

WORKDIR /opt/my-web-app/

RUN apt-get update \
    && apt install \
        --no-install-recommends --yes \
        build-essential libpq-dev \
    && true

COPY requirements.txt /tmp/requirements.txt
RUN pip3 install --no-cache-dir -r /tmp/requirements.txt \
    && rm /tmp/requirements.txt \
    && true

COPY ./mywebapp /opt/my-web-app/mywebapp
COPY ./deploy /opt/my-web-app/deploy
COPY ./manage.py /opt/my-web-app/manage.py
```

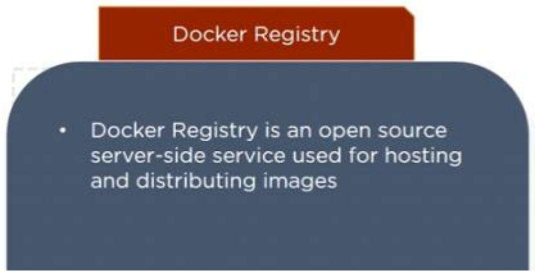
What will happen when we create Docker file
Ans : once we create Docker file that image store into Docker hub or Registry so that Docker hub allow other people to access the same structure of the Docker environments



Components of Docker – Docker Container

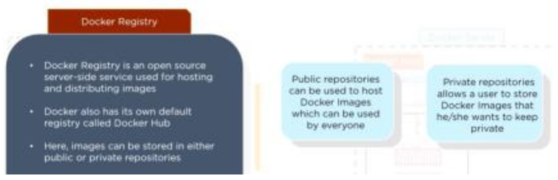
Docker Container

- Docker Container is a standalone, executable software package which includes applications and their dependencies
- Numerous Docker Containers run on the same infrastructure and share operating system (OS) with its other containers



Docker Registry

- Docker Registry is an open source server-side service used for hosting and distributing images



Docker Registry

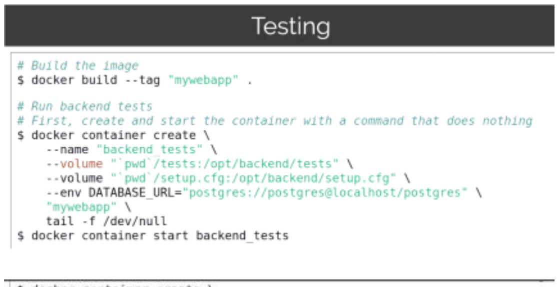
- Docker Registry is an open source server-side service used for hosting and distributing images
- Docker also has its own default registry called Docker Hub
- Here, images can be stored in either public or private repositories

Public repositories can be used to host Docker Images which can be used by everyone

Private repositories allows a user to store Docker Images that he/she wants to keep private

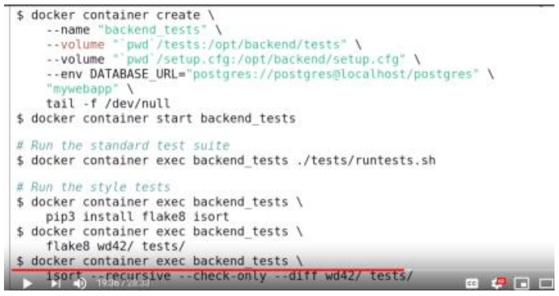
Docker Registry

In order to build a

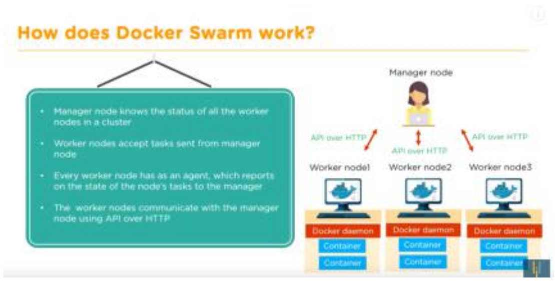


Testing

```
# Build the image
$ docker build --tag "mywebapp" .

# Run backend tests
# First, create and start the container with a command that does nothing
$ docker container create \
    --name "backend_tests" \
    --volume "`pwd`/tests:/opt/backend/tests" \
    --volume "`pwd`/setup.cfg:/opt/backend/setup.cfg" \
    --env DATABASE_URL="postgres://postgres@localhost/postgres" \
    "mywebapp" \
    tail -f /dev/null
$ docker container start backend_tests
```

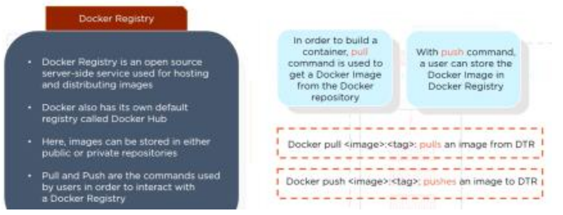


```
$ docker container create \
    --name "backend_tests" \
    --volume "`pwd`/tests:/opt/backend/tests" \
    --volume "`pwd`/setup.cfg:/opt/backend/setup.cfg" \
    --env DATABASE_URL="postgres://postgres@localhost/postgres" \
    "mywebapp" \
    tail -f /dev/null
$ docker container start backend_tests

# Run the standard test suite
$ docker container exec backend_tests ./tests/runtests.sh

# Run the style tests
$ docker container exec backend_tests \
    pip3 install flake8 isort
$ docker container exec backend_tests \
    flake8 wd42/ tests/
$ docker container exec backend_tests \
    isort --recursive --check-only --diff wd42/ tests/
```



How does Docker Swarm work?

- Manager node knows the status of all the worker nodes in a cluster
- Worker nodes accept tasks sent from manager node
- Every worker node has as an agent, which reports on the state of the node's tasks to the manager
- The worker nodes communicate with the manager node using API over HTTP


Docker Registry

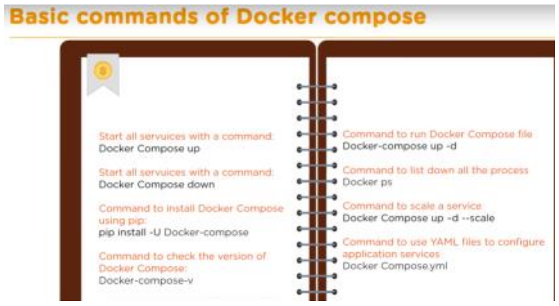
- Docker Registry is an open source server-side service used for hosting and distributing images
- Docker also has its own default registry called Docker Hub
- Here, images can be stored in either public or private repositories
- Pull and Push are the commands used by users in order to interact with a Docker Registry

In order to build a container, pull command is used to get a Docker Image from the Docker repository

With push command, a user can store the Docker Image in Docker Registry

Docker pull <image>:<tag>: pulls an image from DTR

Docker push <image>:<tag>: pushes an image to DTR


Basic commands of Docker compose

Start all servuices with a command:
Docker Compose up

Start all servuices with a command:
Docker Compose down

Command to install Docker Compose using pip:
pip install -U Docker-compose

Command to check the version of Docker Compose:
Docker-compose-v

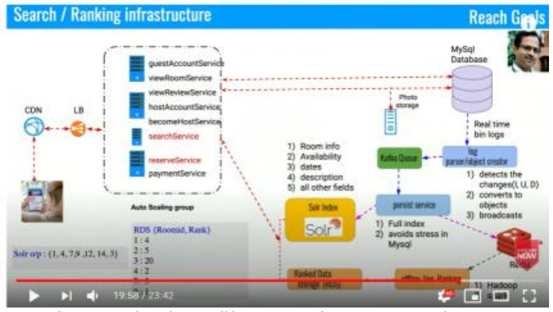
Command to run Docker Compose file
Docker-compose up -d

Command to list down all the process
Docker ps

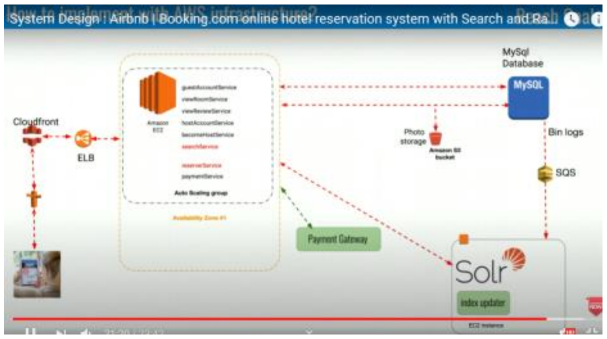
Command to scale a service
Docker Compose up -d --scale

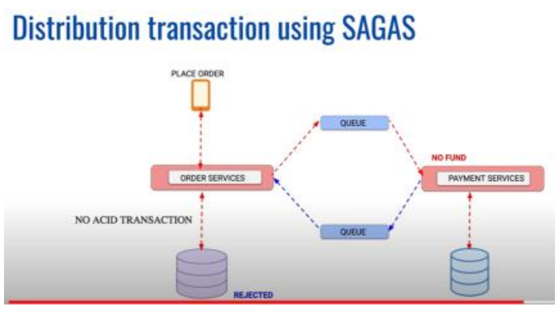
Command to use YAML files to configure application services
Docker Compose.yml

System design soa architecture and search and rank services

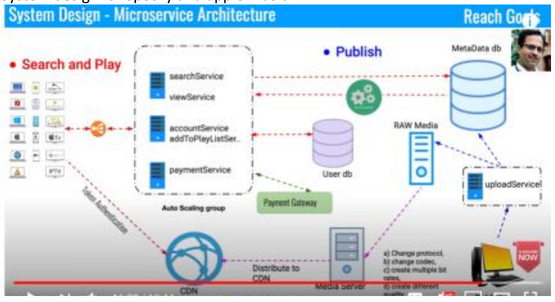


In SOA the auto scaling there will be many Docker container inside EC2 instance
And there will be different kind of micro serves runing inside Docker container.
AWS architectures

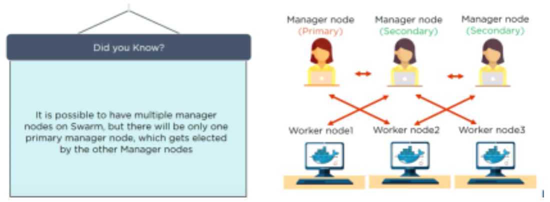




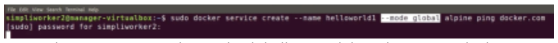
Distribution transaction
System design for Spotify and apple music



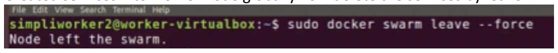
Creating multiple manager

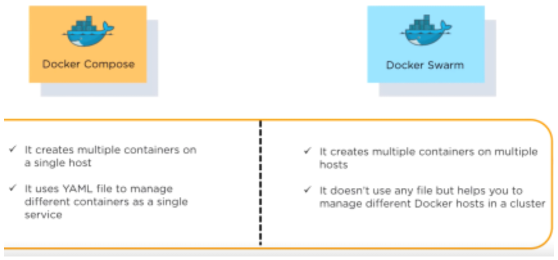


It is possible to have multiple manager nodes on Swarm, but there will be only one primary manager node, which gets elected by the other Manager nodes


```
simpliworker2@manager-virtualbox:~$ sudo docker service create --name helloworld1 --mode global alpine ping docker.com
[sudo] password for simpliworker2:
```

Created services into worker node globally now delete the services by leave


```
simpliworker2@worker-virtualbox:~$ sudo docker swarm leave --force
Node left the swarm.
```


Docker Compose

Docker Swarm

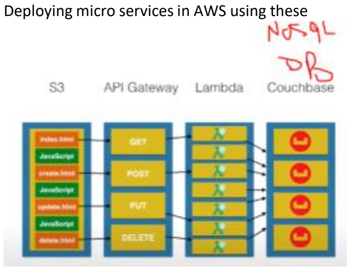
✓ It creates multiple containers on a single host

✓ It uses YAML file to manage different containers as a single service

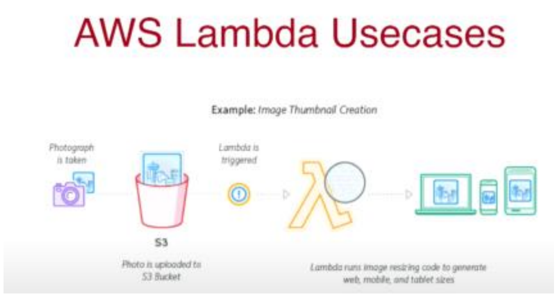
✓ It creates multiple containers on multiple hosts

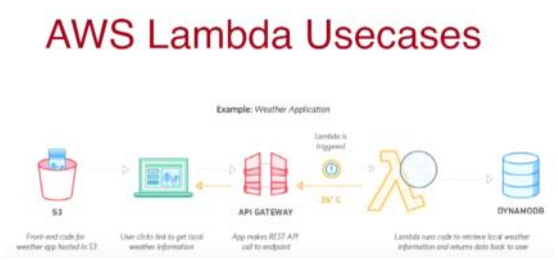
✓ It doesn't use any file but helps you to manage different Docker hosts in a cluster

Deploying micro services in AWS using these

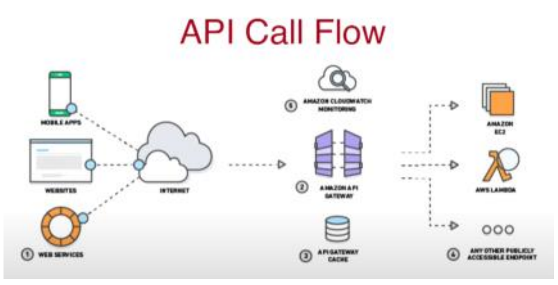


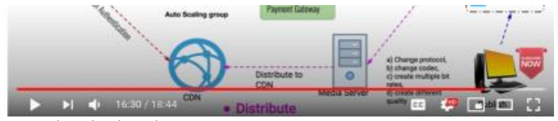
Aws usecase


AWS Lambda Usecases
Example: Image Thumbnail Creation
Photograph is taken
S3
Photo is uploaded to S3 Bucket
Lambda is triggered
Lambda runs image resizing code to generate web, mobile, and tablet sizes

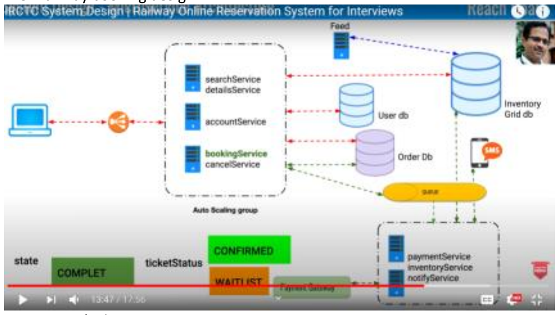

AWS Lambda Usecases
Example: Weather Application
S3
Front-end code for weather app hosted in S3
User clicks link to get local weather information
API GATEWAY
App makes REST API call to endpoint
Lambda is triggered
35° C
DYNAMODB
Lambda runs code to retrieve local weather information and returns data back to user

Api call flow

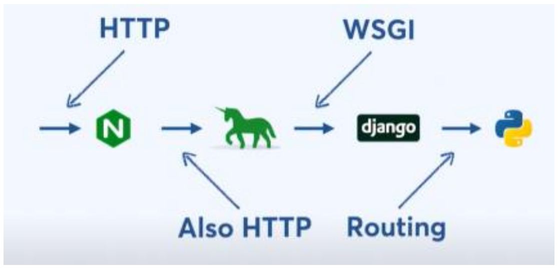




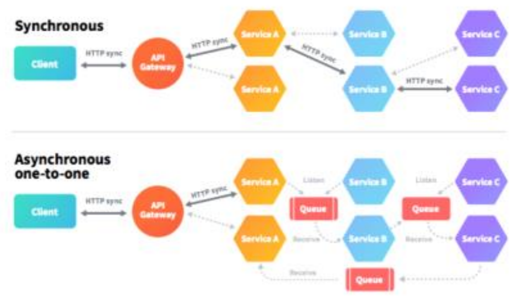
IRCT railway booking design



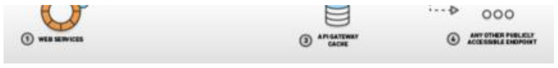
Using SAGA'S design pattern

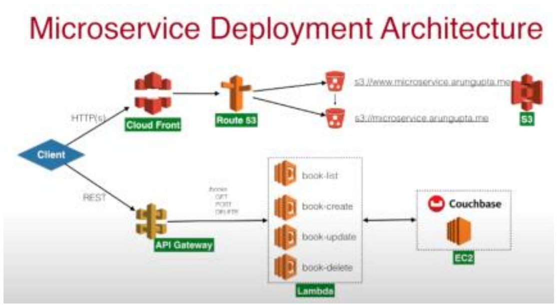


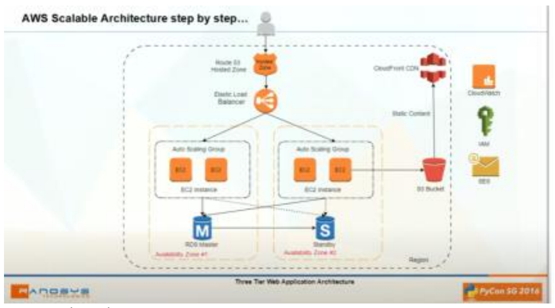
Communication between microservices



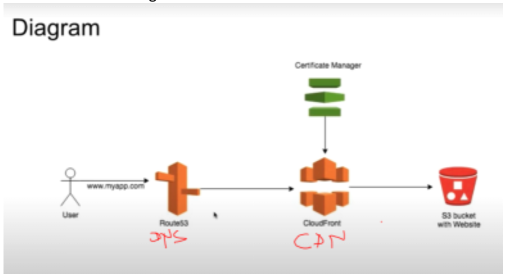
How server know the right request ?
 they can know using
JWT, Authentication server , cookies, IP address ,
 https://www.slatools.com/ **know about connectivity downside of your application.**
Uptime is the amount of time that a service is available and operational. The counterpart is downtime - the amount of time that a service is unavailable.



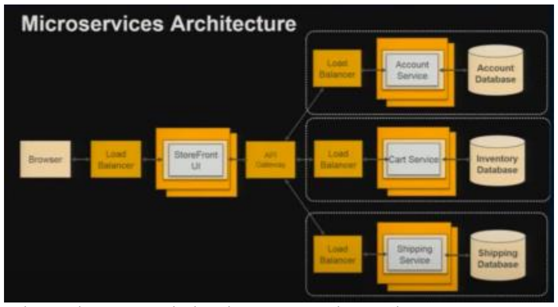




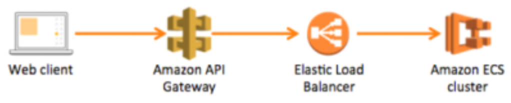
Static Website hosting on aws s3

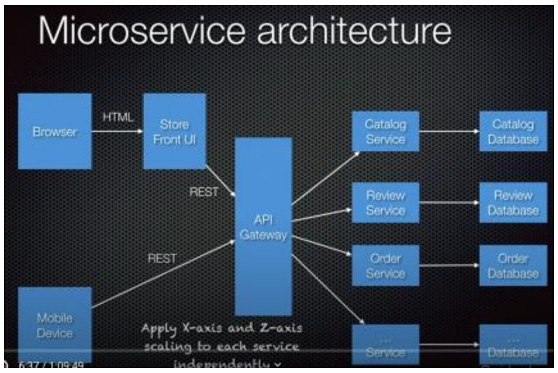


First we have to merge asw server name with domain name then We put aws given enpoint like into in Cloudfront then we can configure SSL, aws certificate manager in distribution network (couldfront) then we have to create record set in rout53 to link rout53 to cloud front https://www.youtube.com/watch?v=D6qB7MEFOe0
Micro services

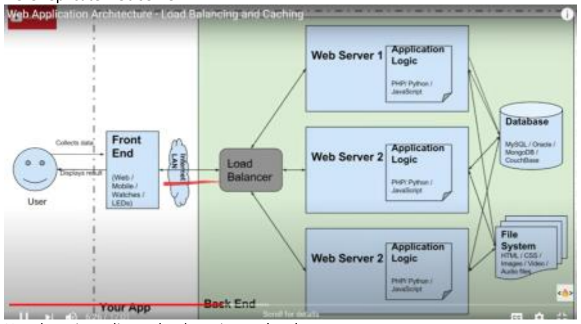


Dokoraised Django app deployed in AWS using these tools

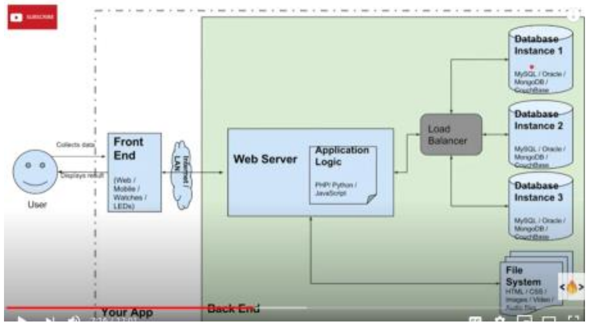




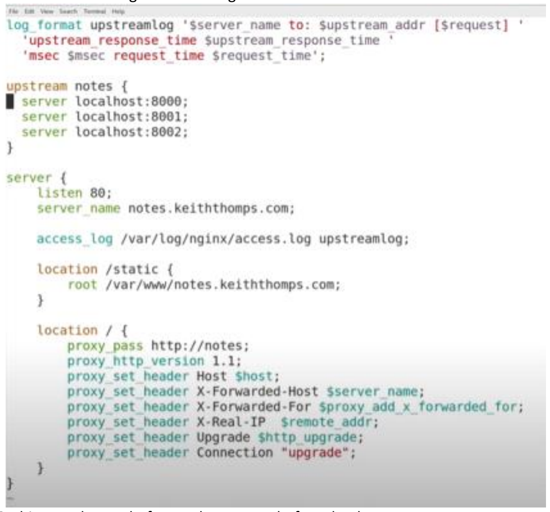
Load balancer algorithum : 1. round robin algorithum 2. ip hashinng
Here replicate web server



Now here is replicate database instead web server

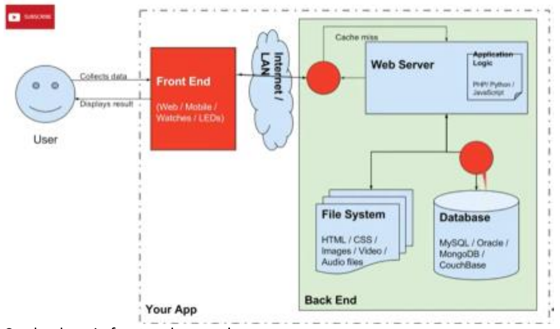


Load balancer configure in linux Nginx

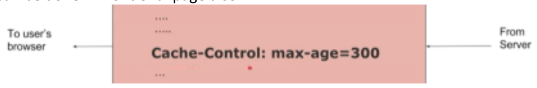


```
log_format upstreamlog '$server_name to: $upstream_addr [$request] '
  'upstream_response_time $upstream_response_time '
  'msec $msec request_time $request_time';

upstream notes {
  server localhost:8000;
  server localhost:8001;
  server localhost:8002;
}

server {
    listen 80;
    server_name notes.keiththomps.com;

    access_log /var/log/nginx/access.log upstreamlog;

    location /static {
        root /var/www/notes.keiththomps.com;
    }

    location / {
        proxy_pass http://notes;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-Host $server_name;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Real-IP  $remote_addr;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}
```

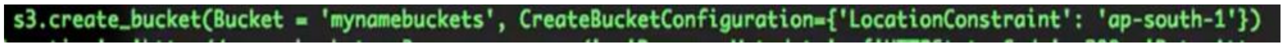
Cashing can be use before web server or before database



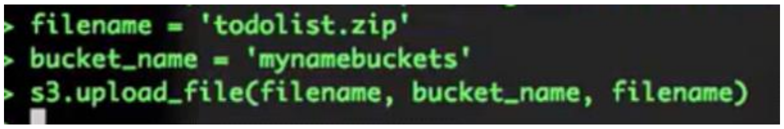
Can be done in front end page also



Creating bucket in s3 and setting up region (place where/which city you want to create it)
 uploading file into S3 bucket

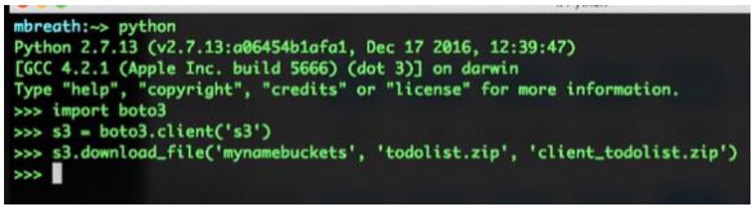


```
s3.create_bucket(Bucket = 'mynamebuckets', CreateBucketConfiguration={'LocationConstraint': 'ap-south-1'})
```

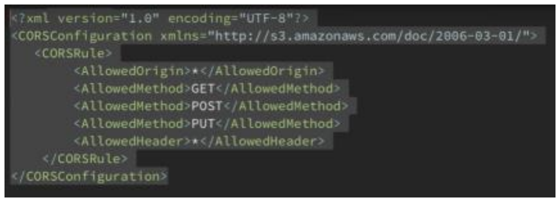


```
> filename = 'todolist.zip'
> bucket_name = 'mynamebuckets'
> s3.upload_file(filename, bucket_name, filename)
```

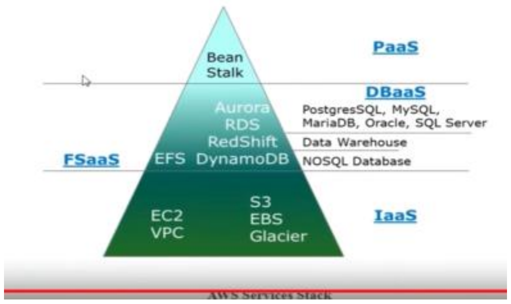
CORS configuration file for access all permission for world need to paste in s3 permition sections
https://www.youtube.com/watch?v=kt3ZtW9MXhw also need IAM setup for extra layer security
https://django-storages.readthedocs.io/en/latest/backends/amazon-S3.html django -storage for store to s3

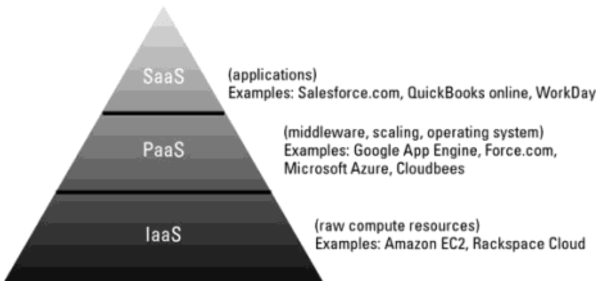


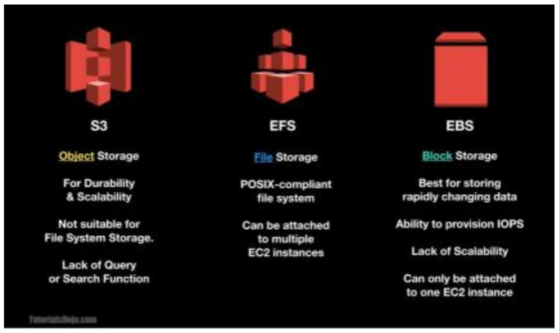
```
mbreath:~> python
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 12:39:47)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto3
>>> s3 = boto3.client('s3')
>>> s3.download_file('mynamebuckets', 'todolist.zip', 'client_todolist.zip')
>>> 
```



```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
   <CORSRule>
        <AllowedOrigin>*</AllowedOrigin>
        <AllowedMethod>GET</AllowedMethod>
        <AllowedMethod>POST</AllowedMethod>
        <AllowedMethod>PUT</AllowedMethod>
        <AllowedHeader>*</AllowedHeader>
    </CORSRule>
</CORSConfiguration>
```




SaaS
(applications)
Examples: Salesforce.com, QuickBooks online, WorkDay
PaaS
(middleware, scaling, operating system)
Examples: Google App Engine, Force.com, Microsoft Azure, Cloudbees
IaaS
(raw compute resources)
Examples: Amazon EC2, Rackspace Cloud


S3
Object Storage
For Durability & Scalability
Not suitable for File System Storage.
Lack of Query or Search Function
EFS
File Storage
POSIX-compliant file system
Can be attached to multiple EC2 instances
EBS
Block Storage
Best for storing rapidly changing data
Ability to provision IOPS
Lack of Scalability
Can only be attached to one EC2 instance
TutorialsDojo.com

S Storage Service Usage Patterns (S3 vs EFS vs EBS)

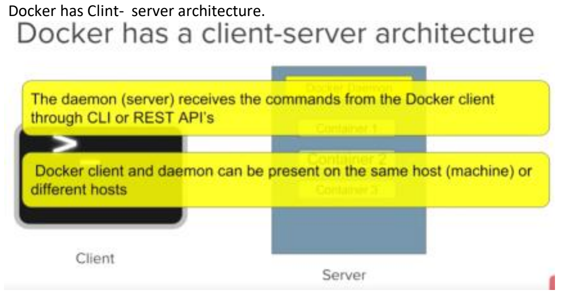
https://pythonise.com/series/learning-flask/building-a-flask-app-with-docker-compose for flask and docker

[Automation with Scripting](https://www.youtube.com/watch?v=9occfhrM4gg&list=PL2qzCKTbjutJ1zZFYNImrHNbzs6XgIHbI&index=3) for boto 3 tutoral

From <https://www.youtube.com/watch?v=9occfhrM4gg&list=PL2qzCKTbjutJ1zZFYNImrHNbzs6XgIHbI&index=3>

Docker tutorial

Docker has Clint- server architecture.



Docker has a client-server architecture

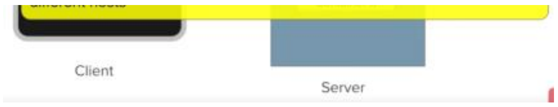
The daemon (server) receives the commands from the Docker client through CLI or REST API's

Docker client and daemon can be present on the same host (machine) or different hosts

ou can visit - https://get.docker.com/
or more installation related help

 to get any releted command go on that link
You can also go in down link

To install docker from binaries

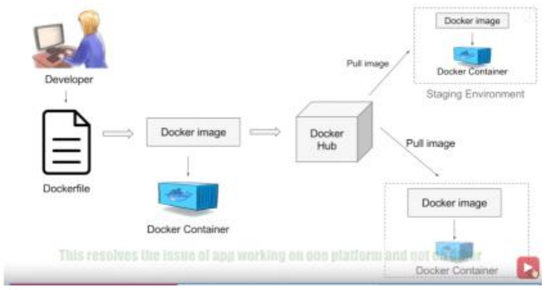


[Automation Step by Step - Raghav Pal](https://www.youtube.com/watch?v=YCrRy7pBzdc)
136K subscribers

You can also search on google **Docker manual**

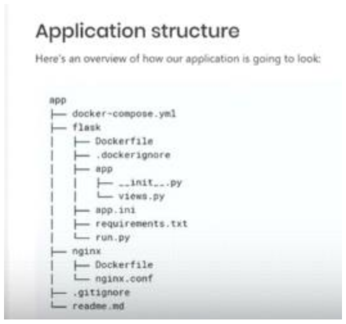
From <https://www.youtube.com/watch?v=YCrRy7pBzdc>

How Docker work



Difference between virtualization and container

In container we don't need separate OS for application there is one container who handle all dependency and all required things to run application



Application structure

Here's an overview of how our application is going to look:

```
app
├── docker-compose.yml
├── flask
|   ├── Dockerfile
|   ├── .dockerignore
|   ├── app
|   |   ├── __init__.py
|   |   └── views.py
|   ├── app.ini
|   ├── requirements.txt
|   └── run.py
├── nginx
|   ├── Dockerfile
|   └── nginx.conf
├── .gitignore
└── readme.md
```

By this command you will get container for develop your applications

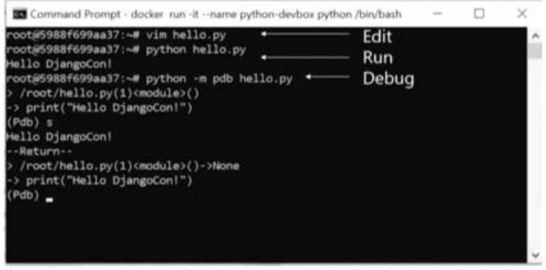
```
c:\demo>docker run -it --name python-devbox python bash
root@dbaf62199e97:/# ls
```

To get app update ad install text editor use this command

```
root@dbaf62199e97:/# apt-get update && apt-get install vim
```

```
root@dbaf62199e97:~# mkdir app
root@dbaf62199e97:~# cd app
root@dbaf62199e97:~/app# ls
root@dbaf62199e97:~/app# vim hello.py
root@dbaf62199e97:~/app# python hello.py
Hello DjangoCon!
```

You can debug your code by command line also by using python debuger like pbd


Command Prompt - docker run -it --name python-devbox python /bin/bash
Edit
Run
Debug

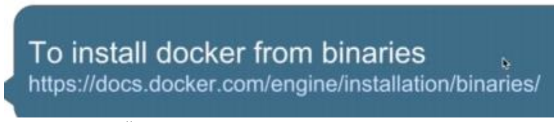
```
root@5988f699aa37:~# vim hello.py
root@5988f699aa37:~# python hello.py
Hello DjangoCon!
root@5988f699aa37:~# python -m pdb hello.py
> /root/hello.py(1)<module>()
-> print("Hello DjangoCon!")
(Pdb) s
Hello DjangoCon!
--Return--
> /root/hello.py(1)<module>()->None
-> print("Hello DjangoCon!")
(Pdb)
```

Containerized Debian development environment!

Have to remember all these docker commands
Limited to only command line developer tools (git, pdb, vim)
Code will disappear as soon as the container stops
Can't access web sites hosted in the container

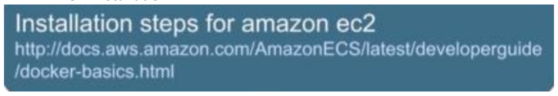
You need to put **extension** inside visual studio code

to get any releted command go on that link
You can also go in down link



To install docker from binaries
https://docs.docker.com/engine/installation/binaries/

In AMAZON installation

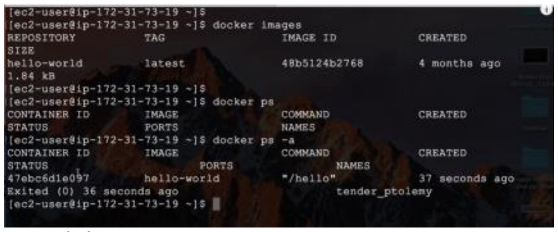


Installation steps for amazon ec2
http://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html

For installation try to get basic docker command from google
Sudo yum install -y docker
Sudo services docker start for start docker
Docker info for see all the installation and running application in system

To open usermod

```
[ec2-user@ip-172-31-73-19 ~]$
[ec2-user@ip-172-31-73-19 ~]$ sudo usermod -a -G docker ec2-user
[ec2-user@ip-172-31-73-19 ~]$
```



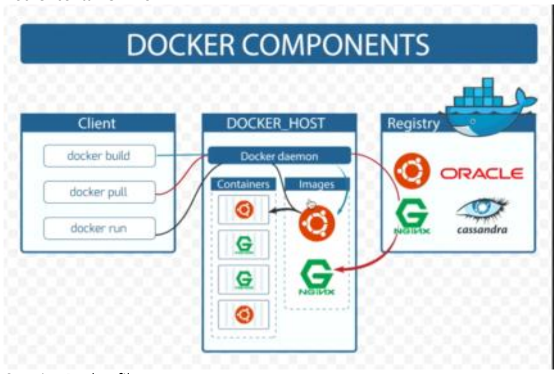
```
[ec2-user@ip-172-31-73-19 ~]$
[ec2-user@ip-172-31-73-19 ~]$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
hello-world         latest              48b5124b2768        4 months ago
1.84 kB
[ec2-user@ip-172-31-73-19 ~]$
[ec2-user@ip-172-31-73-19 ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
[ec2-user@ip-172-31-73-19 ~]$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                 PORTS               NAMES
47ebc6d1e097        hello-world         "/hello"            37 seconds ago
Exited (0) 36 seconds ago                  tender_ptolemy
[ec2-user@ip-172-31-73-19 ~]$
```

to stop docker

```
[ec2-user@ip-172-31-73-19 ~]$
[ec2-user@ip-172-31-73-19 ~]$ sudo service docker stop
Stopping docker:                                           [  OK  ]
[ec2-user@ip-172-31-73-19 ~]$
```

to uninstall
Sudo yum remove docker docker will uninstall from your system

Docker container info



Creating Docker file

```
Last login: Wed Jul  4 13:46:05 on ttys001
Raghavs-MacBook-Pro:~ raghav$ cd /Users/raghav/Desktop/
Raghavs-MacBook-Pro:Desktop raghav$
Raghavs-MacBook-Pro:Desktop raghav$
Raghavs-MacBook-Pro:Desktop raghav$ mkdir DockerFiles
Raghavs-MacBook-Pro:Desktop raghav$ cd DockerFiles/
Raghavs-MacBook-Pro:DockerFiles raghav$ touch Dockerfile
Raghavs-MacBook-Pro:DockerFiles raghav$
Raghavs-MacBook-Pro:DockerFiles raghav$ vim Dockerfile
```

Then edit file write inside your requirement cat dockerfile is just for see the file info don't write inside dockerfile

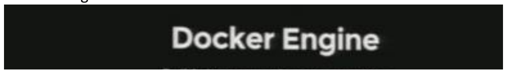
```
Raghavs-MacBook-Pro:DockerFiles raghav$ cat Dockerfile
# getting base image ubuntu
FROM ubuntu

MAINTAINER raghav pal <automation.devops@gmail.com>

RUN apt-get update

CMD ["echo", "Hello World...! from my first docker image"]
Raghavs-MacBook-Pro:DockerFiles raghav$
```

To build

```
Raghavs-MacBook-Pro:DockerFiles raghav$ docker build -t myimage1:1.0 .
```

```
docker build -t imageName:tagName "location of Dockerfile"
```

Docker build take the dockkerfile and make actual docker image then **docker run** take the docker image and make container and run it



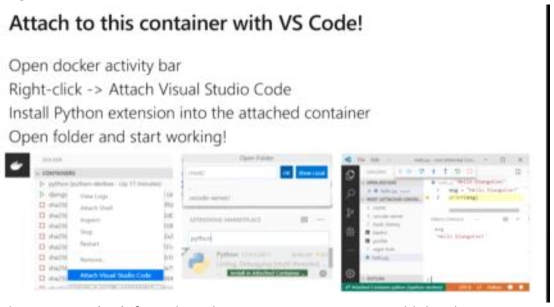
Docker Engine

Code will disappear as soon as the container stops
Can't access web sites hosted in the container

You need to put **extension** inside visual studio code

Python (lighting , debuging, multithereting)
Docker -8.2 (adds syntex hilighting)
Remote - Containers

Then



Attach to this container with VS Code!

Open docker activity bar
Right-click -> Attach Visual Studio Code
Install Python extension into the attached container
Open folder and start working!

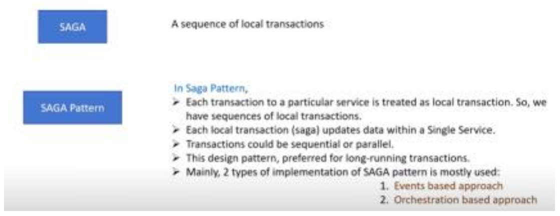
Then press **ctrl +shift +P t**hen choose remote-container : add development container configuration file option

Add API code by following the Django REST Framework tutorial:
https://www.django-rest-framework.org/tutorial/quickstart/

Then press **ctrl +shift +P t**hen choose **python: configuration test** option for better test result and action also you can add **format documents also remote-vontsiner reopen locally**

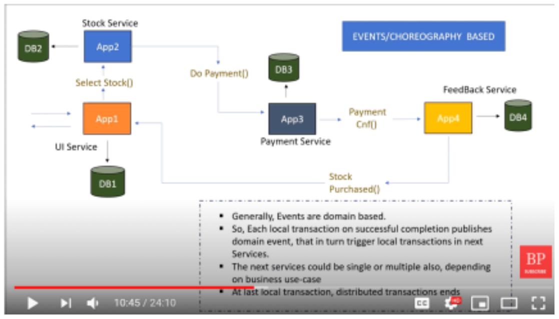
Now adding React font-end here

Saga's Design pattern

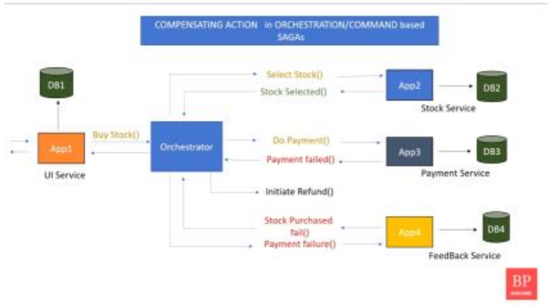


SAGA — A sequence of local transactions

SAGA Pattern — In Saga Pattern,
- Each transaction to a particular service is treated as local transaction. So, we have sequences of local transactions.
- Each local transaction (saga) updates data within a Single Service.
- Transactions could be sequential or parallel.
- This design pattern, preferred for long-running transactions.
- Mainly, 2 types of implementation of SAGA pattern is mostly used:
  1. Events based approach
  2. Orchestration based approach



- Generally, Events are domain based.
- So, Each local transaction on successful completion publishes domain event, that in turn trigger local transactions in next Services.
- The next services could be single or multiple also, depending on business use-case
- At last local transaction, distributed transactions ends

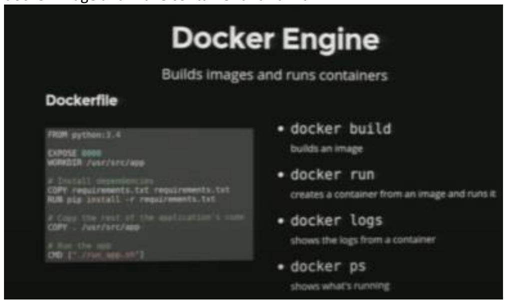


Mixin design pattern is all about multiple inheritance

Detour: What is a Mixin?

- **Mixin is a class used to add properties and**

Docker build take the dockkerfile and make actual docker image then **docker run** take the docker image and make container and run it

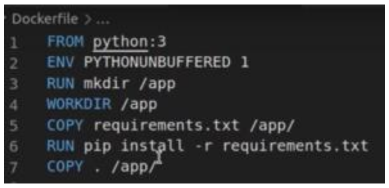


Docker Engine

Builds images and runs containers

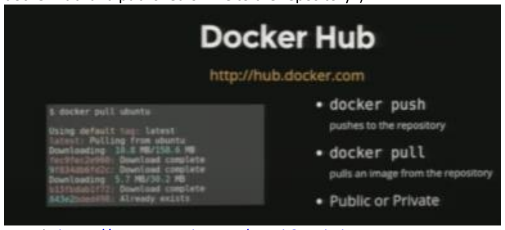
Dockerfile

- docker build
  builds an image
- docker run
  creates a container from an image and runs it
- docker logs
  shows the logs from a container
- docker ps
  shows what's running



```
Dockerfile > ...
FROM python:3
ENV PYTHONUNBUFFERED 1
RUN mkdir /app
WORKDIR /app
COPY requirements.txt /app/
RUN pip install -r requirements.txt
COPY . /app/
```

TO stop docker container : **docker stop container name**
After create docker image and container we need to make repository into docker hub the push your docker container to hub repository (basically copy your docker container app to docker hub and published online to the repository)



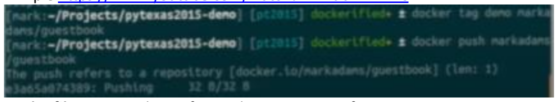
Docker Hub

http://hub.docker.com

- docker push
  pushes to the repository
- docker pull
  pulls an image from the repository
- Public or Private

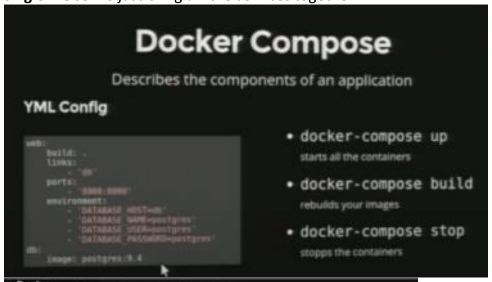
```
$ docker pull ubuntu
Using default tag: latest
latest: Pulling from ubuntu
Downloading 18.8 MB/158.6 MB
...: Download complete
...: Download complete
Downloading 5.7 MB/30.2 MB
...: Download complete
...: Already exists
```

Exmple https://www.youtube.com/watch?v=VhabrYF1nms



```
[mark:~/Projects/pytexas2015-demo] [pt2015] dockerified+ ± docker tag demo markadams/guestbook
[mark:~/Projects/pytexas2015-demo] [pt2015] dockerified+ ± docker push markadams/guestbook
The push refers to a repository [docker.io/markadams/guestbook] (len: 1)
e3a65a074389: Pushing      32 B/32 B
```

Dockerfile : its mostly use for single container configurations.
Docker-compose: **its wired bunch of different type of container and configuration into one single file** so we just bring all the services together.



Docker Compose

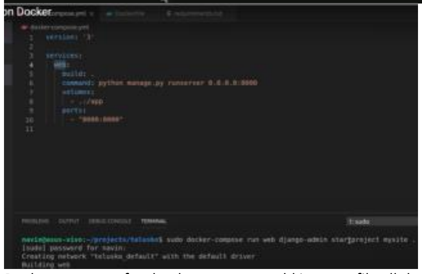
Describes the components of an application

YML Config

- docker-compose up
  starts all the containers
- docker-compose build
  rebuilds your images
- docker-compose stop
  stopps the containers

```
web:
  build: .
  links:
    - "db"
  ports:
    - "8000:8000"
  environment:
    - 'DATABASE_HOST=db'
    - 'DATABASE_NAME=postgres'
    - 'DATABASE_USER=postgres'
    - 'DATABASE_PASSWORD=postgres'
db:
  image: postgres:9.4
```

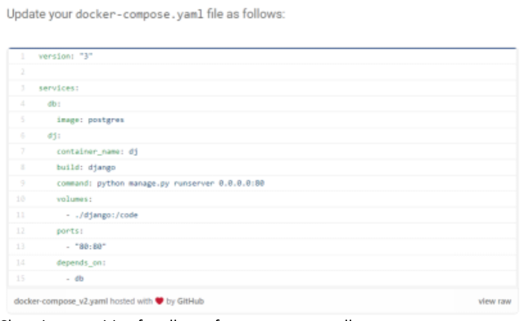
on Docker



```
version: '3'

services:
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/app
    ports:
      - "8000:8000"
```

```
navin@asus-vivo:~/projects/telusko$ sudo docker-compose run web django-admin startproject mysite .
[sudo] password for navin:
Creating network "telusko_default" with the default driver
Building web
```

Docker-compose for database we can add in same file all the database and all services

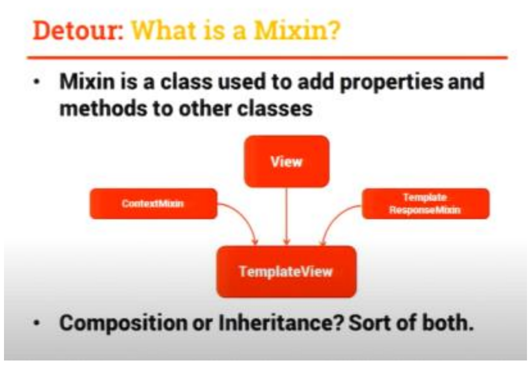


Update your docker-compose.yaml file as follows:

```
version: "3"

services:
  db:
    image: postgres
  dj:
    container_name: dj
    build: django
    command: python manage.py runserver 0.0.0.0:80
    volumes:
      - ./django:/code
    ports:
      - "80:80"
    depends_on:
      - db
```

docker-compose_v2.yaml hosted with ❤ by GitHub   view raw

Changing permition for all user from root user to all user



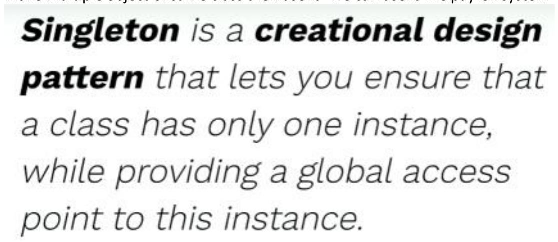
Detour: What is a Mixin?

- Mixin is a class used to add properties and methods to other classes

- Composition or Inheritance? Sort of both.

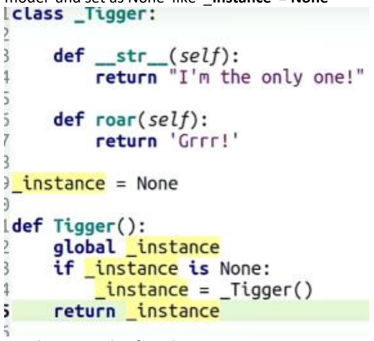
Make mixin class at beginning and base class in end
MVC is architecture pattern
REST is **architecture style**

Singleton Design pattern . Only One instance for one particular class if you don't need to make multiple object of same class then use it we can use it like payroll system



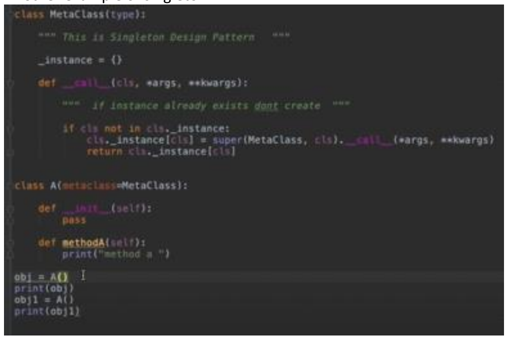
Singleton *is a* ***creational design pattern*** *that lets you ensure that a class has only one instance, while providing a global access point to this instance.*

To make class as singleton class
1st use _private classname like class _classname then create global variable for that model and set as None like **_Instance = None**



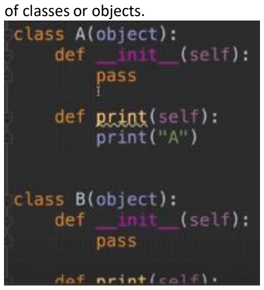
```python
class _Tigger:

    def __str__(self):
        return "I'm the only one!"

    def roar(self):
        return 'Grrr!'

_instance = None

def Tigger():
    global _instance
    if _instance is None:
        _instance = _Tigger()
    return _instance
```

Another example of singleton

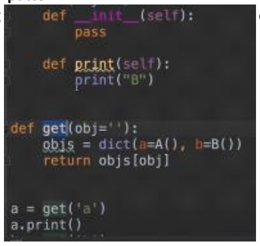


```python
class MetaClass(type):

    """ This is Singleton Design Pattern  """

    _instance = {}

    def __call__(cls, *args, **kwargs):

        """  if instance already exists dont create  """

        if cls not in cls._instance:
            cls._instance[cls] = super(MetaClass, cls).__call__(*args, **kwargs)
            return cls._instance[cls]


class A(metaclass=MetaClass):

    def __init__(self):
        pass

    def methodA(self):
        print("method a ")

obj = A()
print(obj)
obj1 = A()
print(obj1)
```

Factory design pattern :if you don't know how many class or object going to be create in future then use it you can do inheritance also but you have to change lot of things so better use this .
Suppose there is 10 class and you want to call all these class then you have to create object one by one or each class and that's not good way so we use design patter to solve this so you can use facade or **factory design pattern**
creational design pattern which help hiding creation of classes or objects.



```python
class A(object):
    def __init__(self):
        pass

    def print(self):
        print("A")


class B(object):
    def __init__(self):
        pass

    def print(self):
```

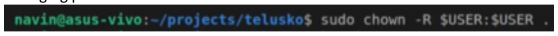


```python
    def __init__(self):
        pass

    def print(self):
        print("B")


def get(obj=''):
    objs = dict(a=A(), b=B())
    return objs[obj]

a = get('a')
a.print()
```

```yaml
      - "80:80"
    depends_on:
      - db
```

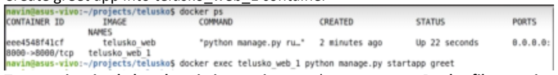
docker-compose_v2.yaml hosted with ❤ by GitHub view raw

Changing permition for all user from root user to all user



```
navin@asus-vivo:~/projects/telusko$ sudo chown -R $USER:$USER .
```

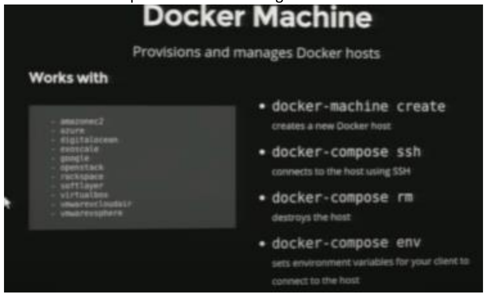
Create greet app into telusko_web_1 container



```
navin@asus-vivo:~/projects/telusko$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
                NAMES
eee4548f41cf        telusko_web         "python manage.py ru…"   2 minutes ago       Up 22 seconds       0.0.0.0:
8000->8000/tcp   telusko_web_1
navin@asus-vivo:~/projects/telusko$ docker exec telusko_web_1 python manage.py startapp greet
```

To containerized already existing project we have to **create Dockerfile, requirments.txt, docker-compose into project folder** not in app folder .

Docker Machine : provisions and manage the Docker hosts



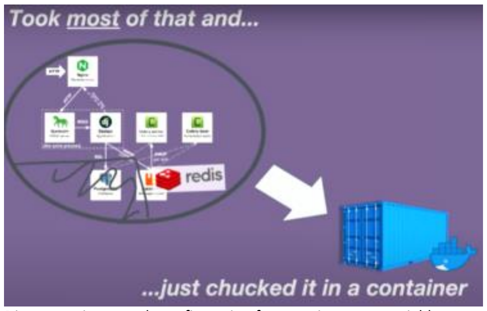
Docker Machine

Provisions and manages Docker hosts

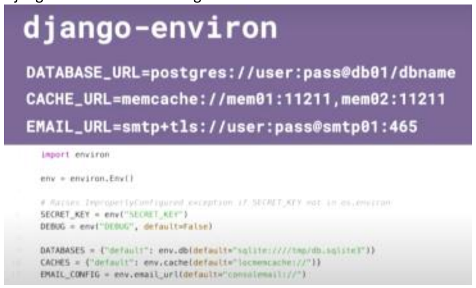
Works with

```
- amazonec2
- azure
- digitalocean
- exoscale
- google
- openstack
- rackspace
- softlayer
- virtualbox
- vmwarevcloudair
- vmwarevsphere
```

- `docker-machine create` creates a new Docker host
- `docker-compose ssh` connects to the host using SSH
- `docker-compose rm` destroys the host
- `docker-compose env` sets environment variables for your client to connect to the host



Django-environ: reads configuration form environment variable

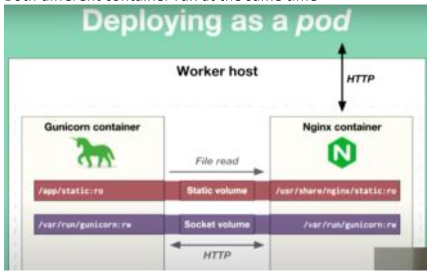


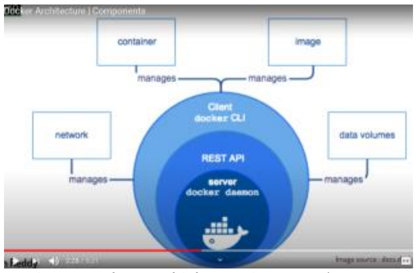
django-environ

```
DATABASE_URL=postgres://user:pass@db01/dbname
CACHE_URL=memcache://mem01:11211,mem02:11211
EMAIL_URL=smtp+tls://user:pass@smtp01:465
```

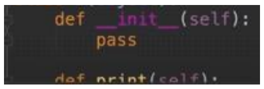
```python
import environ

env = environ.Env()

# Raises ImproperlyConfigured exception if SECRET_KEY not in os.environ
SECRET_KEY = env("SECRET_KEY")
DEBUG = env("DEBUG", default=False)

DATABASES = {"default": env.db(default="sqlite:////tmp/db.sqlite3")}
CACHES = {"default": env.cache(default="locmemcache://")}
EMAIL_CONFIG = env.email_url(default="consolemail://")
```

https://hub.docker.com/r/praekeltfoundation/django-bootstrap/dockerfile bootstrap
Both different container run at the same time



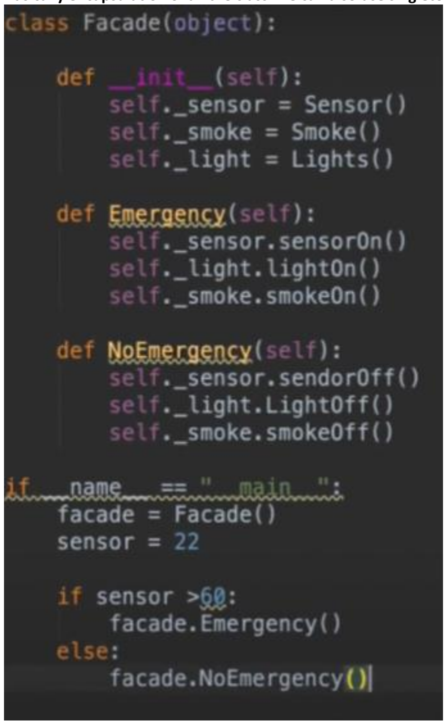


One image can have multiple container it can be create multiple container in one image
Once we delete container we will also lose data that's why we use **data volumes** to keep safe data . To store data into container we can use Storage driver **overlay2 (file system)**

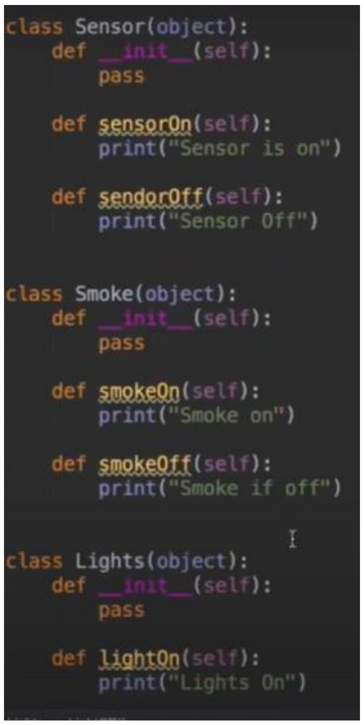


```python
    def __init__(self):
        pass

    def print(self):
```

Structural design pattern:
Facade Design Pattern : Facade Design Pattern. It hides the complexities of the system and provides an interface to the client from where the client can access the system.
is a **Structural** Design Pattern So, As the name suggests, it means the face of the building. The people walking past the road can only see this glass face of the building. They do not know anything about it, the wiring, the pipes and other complexities. It hides all the complexities of the building and displays a friendly face.
Basically encapsulation of all the class we can also use singleton design in façade design



```python
class Facade(object):

    def __init__(self):
        self._sensor = Sensor()
        self._smoke = Smoke()
        self._light = Lights()

    def Emergency(self):
        self._sensor.sensorOn()
        self._light.lightOn()
        self._smoke.smokeOn()

    def NoEmergency(self):
        self._sensor.sendorOff()
        self._light.LightOff()
        self._smoke.smokeOff()

if __name__ == "__main__":
    facade = Facade()
    sensor = 22

    if sensor >60:
        facade.Emergency()
    else:
        facade.NoEmergency()
```



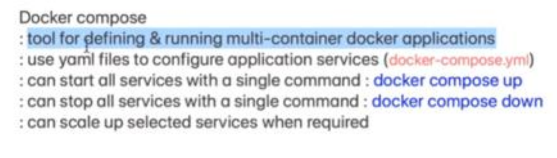
```python
class Sensor(object):
    def __init__(self):
        pass

    def sensorOn(self):
        print("Sensor is on")

    def sendorOff(self):
        print("Sensor Off")


class Smoke(object):
    def __init__(self):
        pass

    def smokeOn(self):
        print("Smoke on")

    def smokeOff(self):
        print("Smoke if off")


class Lights(object):
    def __init__(self):
        pass

    def lightOn(self):
        print("Lights On")
```

Proxy design pattern :

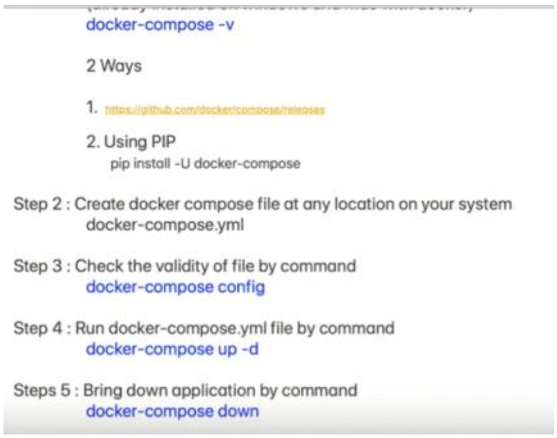
Adapter design pattern: structural design pattern
deal with assembling objects and classes into larger structures, while keeping those structures flexible and efficient.

safe data . To store data into container we can use Storage driver **overlay2 (file system)**

If you want to create **microservices** kind of application then you need compose



Docker compose
: tool for defining & running multi-container docker applications
: use yaml files to configure application services (docker-compose.yml)
: can start all services with a single command : docker compose up
: can stop all services with a single command : docker compose down
: can scale up selected services when required



docker-compose -v

2 Ways

1. https://github.com/docker/compose/releases

2. Using PIP
pip install -U docker-compose

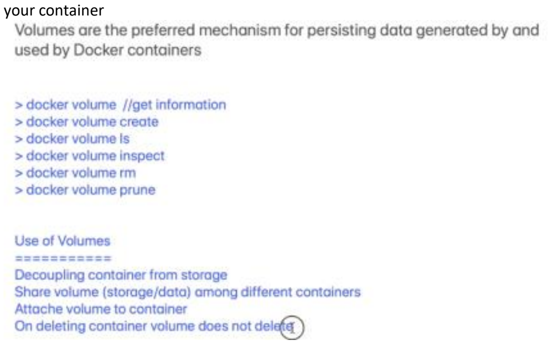
Step 2 : Create docker compose file at any location on your system
docker-compose.yml

Step 3 : Check the validity of file by command
docker-compose config

Step 4 : Run docker-compose.yml file by command
docker-compose up -d

Steps 5 : Bring down application by command
docker-compose down

For data store of container we need volumes data will be inside volumes even after delete your container

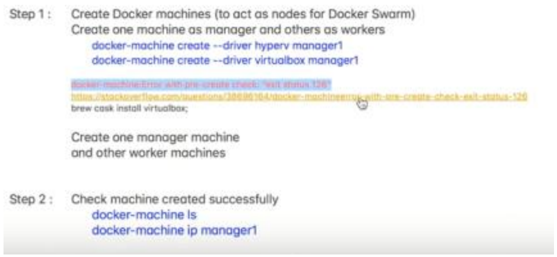


Volumes are the preferred mechanism for persisting data generated by and used by Docker containers

```
> docker volume  //get information
> docker volume create
> docker volume ls
> docker volume inspect
> docker volume rm
> docker volume prune
```

Use of Volumes
===========
Decoupling container from storage
Share volume (storage/data) among different containers
Attache volume to container
On deleting container volume does not delete

Swarm in Docker : swarm is just like manager its control and maintain all the Docker machine through one single machine .

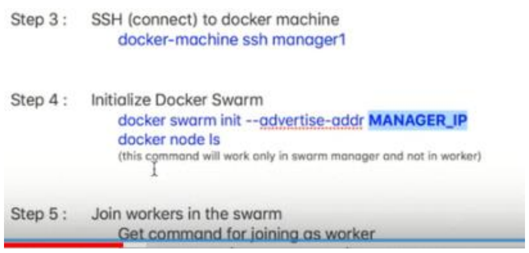


Step 1 : Create Docker machines (to act as nodes for Docker Swarm)
Create one machine as manager and others as workers
docker-machine create --driver hyperv manager1
docker-machine create --driver virtualbox manager1

docker-machine:Error with pre-create check: "exit status 126"
https://stackoverflow.com/questions/38696164/docker-machineerror-with-pre-create-check-exit-status-126
brew cask install virtualbox;

Create one manager machine
and other worker machines

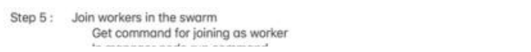
Step 2 : Check machine created successfully
docker-machine ls
docker-machine ip manager1



Step 3 : SSH (connect) to docker machine
docker-machine ssh manager1

Step 4 : Initialize Docker Swarm
docker swarm init --advertise-addr MANAGER_IP
docker node ls
(this command will work only in swarm manager and not in worker)

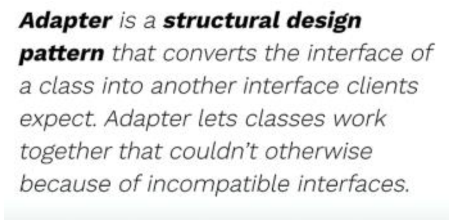
Step 5 : Join workers in the swarm
Get command for joining as worker



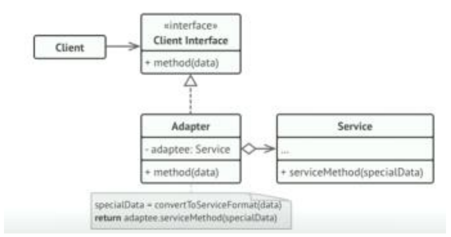
Step 5 : Join workers in the swarm
Get command for joining as worker

Proxy design pattern :

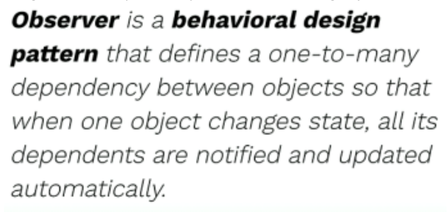
Adapter design pattern: structural design pattern
deal with assembling objects and classes into larger structures, while keeping those structures flexible and efficient.



Adapter *is a* ***structural design pattern*** *that converts the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.*

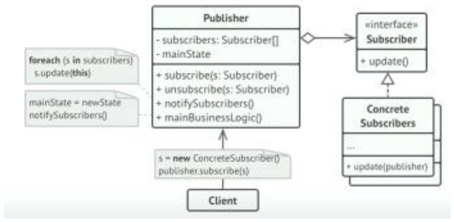

Client
«interface»
Client Interface
+ method(data)
Adapter
- adaptee: Service
+ method(data)
Service
...
+ serviceMethod(specialData)
specialData = convertToServiceFormat(data)
return adaptee.serviceMethod(specialData)

behavioral design pattern
Observer design pattern is behavioral design pattern deal with algorithms in general, and assignment of responsibility between interacting objects.

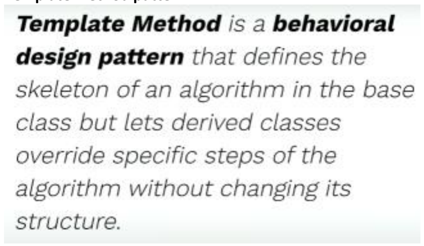


Observer *is a* ***behavioral design pattern*** *that defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.*

Mostly best use for subscription type of work

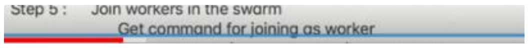


Template method pattern :

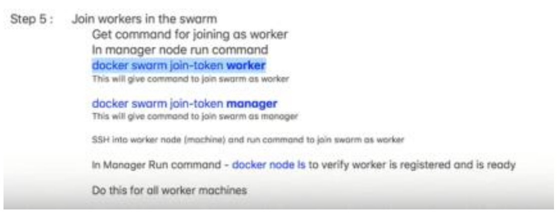


Template Method *is a* ***behavioral design pattern*** *that defines the skeleton of an algorithm in the base class but lets derived classes override specific steps of the algorithm without changing its structure.*

Mostly in template method create abstract class after importing ABC modules then drive class implement of abstract class and override base class
https://www.youtube.com/watch?v=o1FZ_Bd4DSM



Step 5 : Join workers in the swarm
Get command for joining as worker



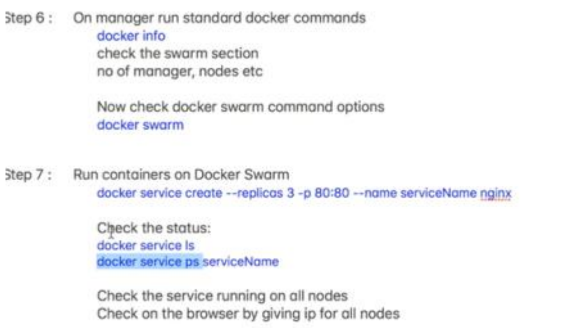
Step 5 : Join workers in the swarm
Get command for joining as worker
In manager node run command
docker swarm join-token **worker**
This will give command to join swarm as worker

docker swarm join-token **manager**
This will give command to join swarm as manager

SSH into worker node (machine) and run command to join swarm as worker

In Manager Run command - docker node ls to verify worker is registered and is ready

Do this for all worker machines



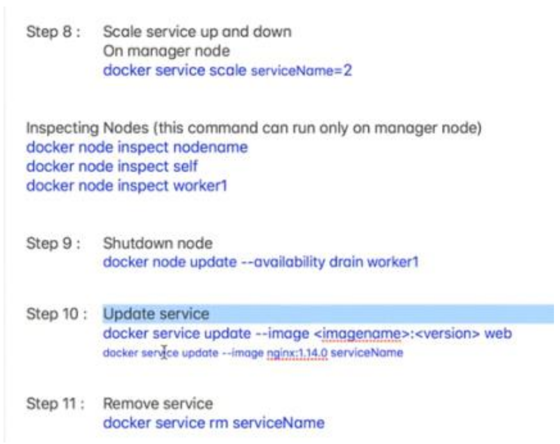
Step 6 : On manager run standard docker commands
docker info
check the swarm section
no of manager, nodes etc

Now check docker swarm command options
docker swarm

Step 7 : Run containers on Docker Swarm
docker service create --replicas 3 -p 80:80 --name serviceName nginx

Check the status:
docker service ls
docker service ps serviceName

Check the service running on all nodes
Check on the browser by giving ip for all nodes



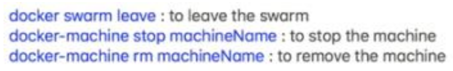
Step 8 : Scale service up and down
On manager node
docker service scale serviceName=2

Inspecting Nodes (this command can run only on manager node)
docker node inspect nodename
docker node inspect self
docker node inspect worker1

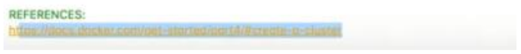
Step 9 : Shutdown node
docker node update --availability drain worker1

Step 10 : Update service
docker service update --image <imagename>:<version> web
docker service update --image nginx:1.14.0 serviceName

Step 11 : Remove service
docker service rm serviceName



docker swarm leave : to leave the swarm
docker-machine stop machineName : to stop the machine
docker-machine rm machineName : to remove the machine



REFERENCES:
https://docs.docker.com/get-started/part4/#create-a-cluster

Docker-compose.yml

```
version: '3.8'
services:
  web:
    image: 999999999.dkr.ecr.eu-central-1.amazonaws.com/ec2-web:latest
    command: /bin/bash ./docker-entrypoint.sh
    environment:
      DEBUG: 'False'
    secrets:
      - ec2.supersecret
    deploy:
      replicas: 1
    logging:
      driver: awslogs
      options:
        awslogs-group: /projects/ec2
        awslogs-region: eu-central-1
        awslogs-stream: app
    volumes:
      - static_volume:/src/staticfiles
  nginx:
    image: 99999999.dkr.ecr.eu-central-1.amazonaws.com/ec2-nginx:latest
    deploy:
      replicas: 1
    logging:
      driver: awslogs
      options:
        awslogs-group: /projects/ec2
        awslogs-region: eu-central-1
        awslogs-stream: nginx
    volumes:
      - static_volume:/src/staticfiles:ro
    ports:
      - 8000:80
    depends_on:
      - web
volumes:
  static_volume:
secrets:
```

```
ec2.supersecret:
  external: true
```

From <https://gonzalo123.com/2020/07/06/deploying-django-application-to-aws-ec2-instance-with-docker/>

```
version: '3.6'
services:
  db:
    image: postgres:10.4
    volumes:
      - postgres_data:/var/lib/postgresql/data/
  cache:
    image: redis:4.0.10
    volumes:
      - redis_data:/data
  web:
    build: .
    image: dockerdjangoexample
    command: bash -c "gunicorn demosite.wsgi:application -b 0.0.0.0:8000"
    volumes:
      - .:/code
    depends_on:
      - db
      - cache
  nginx:
    image: nginx:1.15.2-alpine
    ports:
      - "8000:8000"
    volumes:
      - ./docker-config/nginx:/etc/nginx/conf.d
    depends_on:
      - web
volumes:
  postgres_data:
  redis_data:
```

From <https://stackoverflow.com/questions/51855075/django-microservices-within-docker/51855227>