GEN LINKS

Saturday, July 2, 2022 5:04 PM

Flow charts

1. Draw.io
2. https://whimsical.com/flowcharts

Reading

- https://neptune.ai/blog/ml-model-testing-teams-share-how-they-test-models
  https://ploomber.io/blog/ml-testing-i/
- https://ploomber.io/blog/ml-testing-ii/
- https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning
- https://dagshub.com/blog/ci-cd-for-machine-learning-test-and-and-deploy-your-ml-model-with-github-actions/And
- https://particle1331.github.io/inefficient-networks/notebooks/deployment/cicd-pipelines.html

DECIDING MLOPS ARCHITETCURE

- https://mymlops.com/
- https://ml-ops.org/content/state-of-mlops
- https://ml-ops.org/content/mlops-stack-canvas

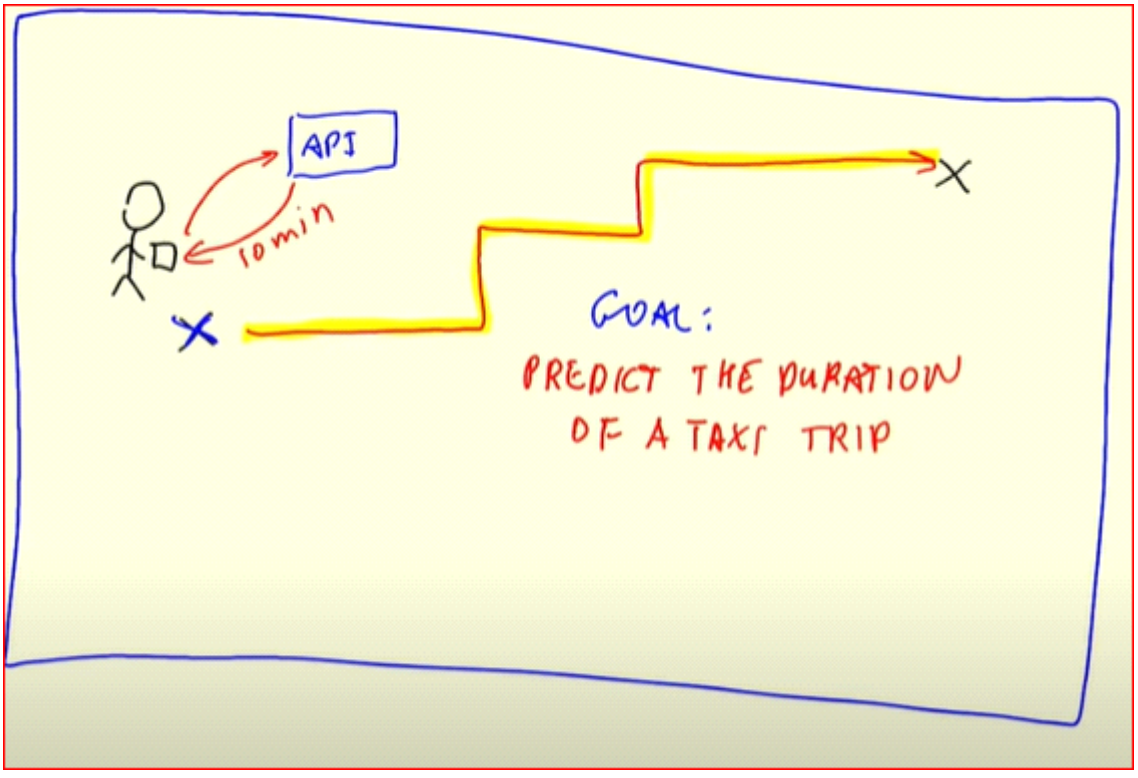
MLOPS ZoomCamp

Wednesday, May 18, 2022 1:05 AM

Case Preparation: Predicting Duration of Taxi Trip

Phases of ML Lifecycle

1. **Design**: Whether this problem is an ML problem
2. **Training**: Train a model which will be used for new data
3. **Operate**: Deploy the model to a webservice/API. Customer can interact using the API. This step also involves model monitoring - making sure the model doesn't degrade



Environment

Wednesday, May 18, 2022 1:10 AM

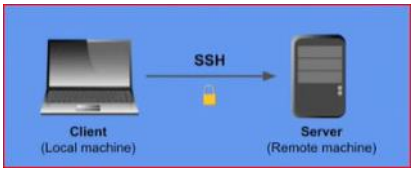
We want to use Linux so we'll setup a Linux/Ubuntu machine on AWS cloud.
We need a Linux Machine but if we don't have it, we'll set one up on the AWS cloud and SSH into it using git bash from windows machine.

We're using here
SSH
Docker
AWS Cloud
Port Forwarding
Docker-Compose
Python

SSH

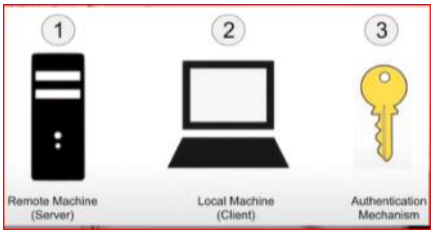
School Of Basics | What is SSH | How SSH works

SSH - acronym for secure shell is a protocol that allows two machines to communicate securely **on a network**.



SSH is secure because the information exchange happens using encryption

Components:



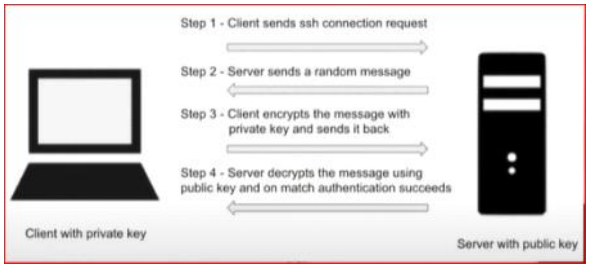
Server can be anything: windows system, Linux, Mac
- If it's mac : allow remote connection on that system
- Linux/windows : install open-ssh server
- Android/Ios: there are some apps

This is needed so that server can be accessed/connected remotely

Client can be anything: windows system, Linux, Mac
- Linux/Mac : no need to install anything, we can use our terminal directly as ssh client
- Windows : get any ssh client or use **git bash**
- Android/Ios : there are some apps

Authentication
- Keys : Public and Private Key
  Run this command from client machine
  Ssh-keygen -t rsa

  This will create a public and private key on the client system
  Copy public key to remote system



- Passwords
  To connect ssh to a remote system:
  Ssh user @ip *(eg: ssh root@165.32.10.10)*
  We'll be asked for the password and then we can access the remote system

Port Forwarding

Lastly, you need to activate the Port Forwarding to be able to access your web-based applications on the cloud, e.g. jupyter noteook running on the aws cloud

AWS

Why can't I use the free services (12 months that come with t2.micro EC2)?

I think the free tier EC2 instance is the t2.micro instance. Different instance types have different resources (CPUs and Memory among others). The t2.micro instance has 1 vCPU and 1 GB of memory (this is lower than a typical computer, which typically have 4 CPUs and 16 GB of memory). Because the resources for the t2.micro instance are small, they are probably not adequate to handle the data for this course. For more info on instances:
https://aws.amazon.com/ec2/instance-types/

General Suggestions:

A couple of suggestions:

1. Make sure when you stop an EC2 instance that it actually stops (there's a meme about it somewhere). There are green circles (running), orange (stopping), and red (stopped). Always refresh the page to make sure you see the red circle and status of stopped.
2. Even when an EC2 instance is stopped, there are WILL be other charges that are incurred (e.g. if you uploaded data to the EC2 instance, this data has to be stored somewhere, usually an EBS volume and this storage incurs a cost).
3. You can set up billing alerts. (I've never done this, so no advice on how to do this) - Samuel explained this to me
4. The small bill on EC2 instance that is shut down and not terminated is because of the 'hard-disk' (aka storage) that stores your data. Terminating an instance also deletes that data/drive.

Should I terminate or stop the EC2 instance every time I'm done with work on a single day?

Stop, instead of terminate. When we terminate, the data gets deleted and we have to setup the environment again. When we step, yes we do pay for the extra data storage charges but they are not as much.

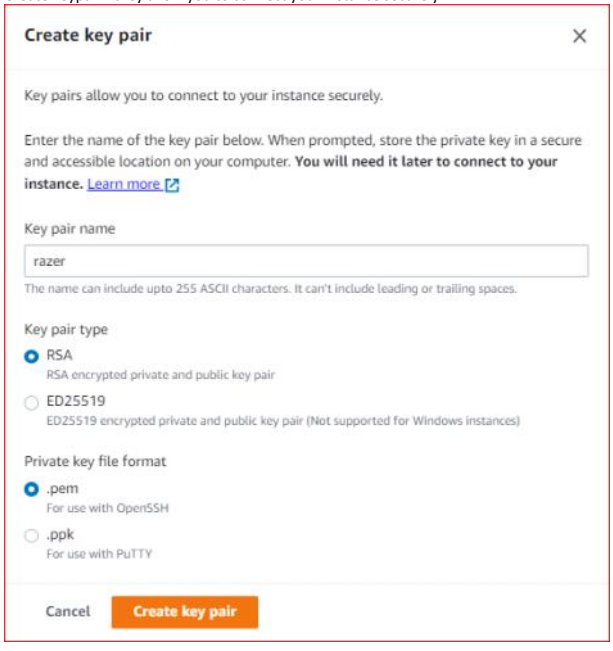
Track Pricing:
https://aws.amazon.com/ec2/pricing/on-demand/

Turn on billing alerts: https://us-east-1.console.aws.amazon.com/billing/home?region=us-east-1#/preferences

Create a cost Budget

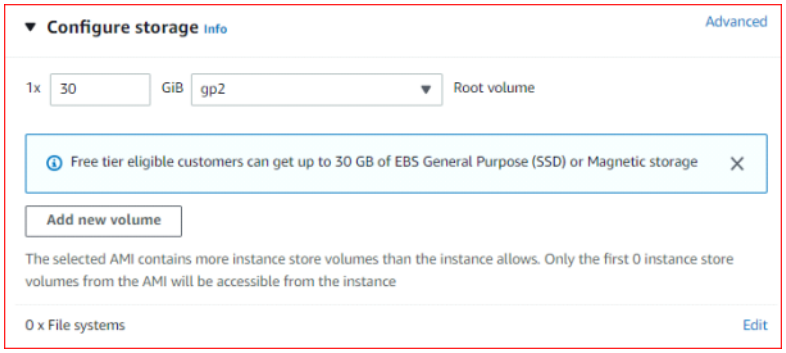
Actual Steps

1. Create EC2 instance - virtual server on cloud - on AWS Cloud. Choose Ubuntu OS
   Choose **t2.large** as the free tier t2.micro wouldn't suffice
   Create keypair - they allow you to connect your instance securely



   We're going to put this in ssh

   Choose 30GB of storage. Default was 8. We're going to use Docker Images and what not.



Step 2: Configure your email and name

```
akshit73@Akshit MINGW64 /
$ git config --global user.name "Akshit Miglani"

akshit73@Akshit MINGW64 /
$ git config --global user.email "akshit.miglani09@gmail.com"
```

Step 3: Generate SSH key and put razer.pem in .ssh folder - [Git for Everybody: Creating and adding your SSH Key (Windows, Mac and Linux)]

```
akshit73@Akshit MINGW64 /
$ ssh-keygen -o
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/akshit73/.ssh/id_rsa):
Created directory '/c/Users/akshit73/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/akshit73/.ssh/id_rsa.
Your public key has been saved in /c/Users/akshit73/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:k/APsrntQE7Ao6aPzgAZyICJIR5gpxQeQplUXPMLEO8 akshit73@Akshit
The key's randomart image is:
+---[RSA 3072]----+
|%@*o.+ E         |
|#o=o  *          |
|o=  +o.o         |
| o . ooo..       |
|o o   +.S        |
|.o   + + +       |
|o     =   .      |
|oo     +         |
|oo.   ..o        |
+----[SHA256]-----+

akshit73@Akshit MINGW64 /
$ cat .ssh/id_rsa.pub
cat: .ssh/id_rsa.pub: No such file or directory

akshit73@Akshit MINGW64 /
$ cat ^Csh/id_rsa.pub

akshit73@Akshit MINGW64 /
$ cat /c/Users/akshit73/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDgIZrqPokMFq0LqFibRMIRdmlo3yP24bxLeJrB2uZqLsYhYMhaXUnbJxPjnGxCwchuhjAgO899AfqseKdsL3H23nwH1bYnoHquiZsAPo54yzuZadUNl9/cfwli9Y6rz+hJaUqAFHwkOmw5Jhta2U5KBCwM
764w9yCqcxlXN5Cbhkw01FI9ZpIrcy7NOJE9YTAaAsfoFul8O6EoWQv3MDDQHvtjUI81IUKsvahWON2tJhugxOwPeSvK9dD0NiPeqyLDkAOqajcZkcwtsH6acNiPf81RmcOAOVX71qHDTm1dcDE1iRUIeC2CziZskQeXL3G2rpqrD4J9RCLzdhAQJGLYS/Ux
Fp2AXCHO/kadgr0KNkiSI9UF962ARD3U/HZBV/qTZw2Ni1+YG5S2ml+ZF+VCOavj44fln9J3bp6lBIxzCjkTEwxHPksKnKvl1RSTP5OoCTYJFRfXYOVFsFFkdnY0UzdgFaADsnzrmOd2DOOWM1JdMELeJxeOSGrAkpY140= akshit73@Akshit
```

Why is there a dot in **.ssh? DOT** is for a hidden folder.
.pub means public key which is shown in the last command. That's how we get the public key

Access ssh folder and see the keys in it

```
akshit73@Akshit MINGW64 /
$ cd  /c/Users/akshit73/

akshit73@Akshit MINGW64 ~
$ cd .ssh/

akshit73@Akshit MINGW64 ~/.ssh
$ ls
id_rsa  id_rsa.pub
```

Connect to ubuntu server(EC2) using downloaded razer.pem key

cd location of razer.pem
ssh -i razer.pem ubuntu@35.175.123.97

OR
PREFERRED
Move razer.pem file to .ssh folder by simply **copy pasting razer.pem to .ssh folder on local computer**
ssh -i .ssh/razer.pem ubuntu@35.175.123.97

ssh -i file.pem username@ip-address
ssh: Command to use SSH protocol
-i: Flag that specifies an alternate identification file to use for public key authentication.
username: Username that uses your instance
ip-address: IP address given to your instance

Why did we type ubuntu as user name?

Here are the user names listed: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/TroubleshootingInstancesConnecting.html#TroubleshootingInstancesConnectingPuTTY

Now when we type this command we get connected to the Ubuntu server on the AWS. But we don't want to type this command everytime
So we will edit the SSH file to save this information - remember to change the **public IP** in this command when we stop and start the instance

nano .ssh/config
Edit Mode:

```
Host mlops-zoomcamp
    HostName 3.235.109.228
    User ubuntu
    IdentityFile c:/Users/akshit73/.ssh/razer.pem
    StrictHostKeyChecking no
```

CTRL + O: Save
Write: **.ssh/config**
CTRL + X: Exit

Now we can simply type **ssh mlops-zoomcamp** instead of ssh -i... command and it will connect to remote server
(Remember to change the IP when you stop and start EC2 again)

There is nothing on this linux computer
We need to install Python
How to install anaconda on Linux: Get **installer** link from anaconda website for linux: https://repo.anaconda.com/archive/Anaconda3-2022.05-Linux-x86_64.sh

Command1: **wget** https://repo.anaconda.com/archive/Anaconda3-2022.05-Linux-x86_64.sh
Command2: bash Anaconda3-2022.05-Linux-x86_64.sh

Now when we do check our remote

```
(base) ubuntu@ip-172-31-12-129:~$ ls
Anaconda3-2022.05-Linux-x86_64.sh  anaconda3
(base) ubuntu@ip-172-31-12-129:~$ which python
/home/ubuntu/anaconda3/bin/python
```

It shows python install and defaults to anaconda3 (so we now have all the basic packages covered)

Check .bashrc file (**less .bashrc**) to see what was changed(anaconda was added here) **LEARN**

==========Python Installed==========

Now we will clone the repo
If you want to push from the remote machine : USE SSH
If you only want to pull: USE HTTPS

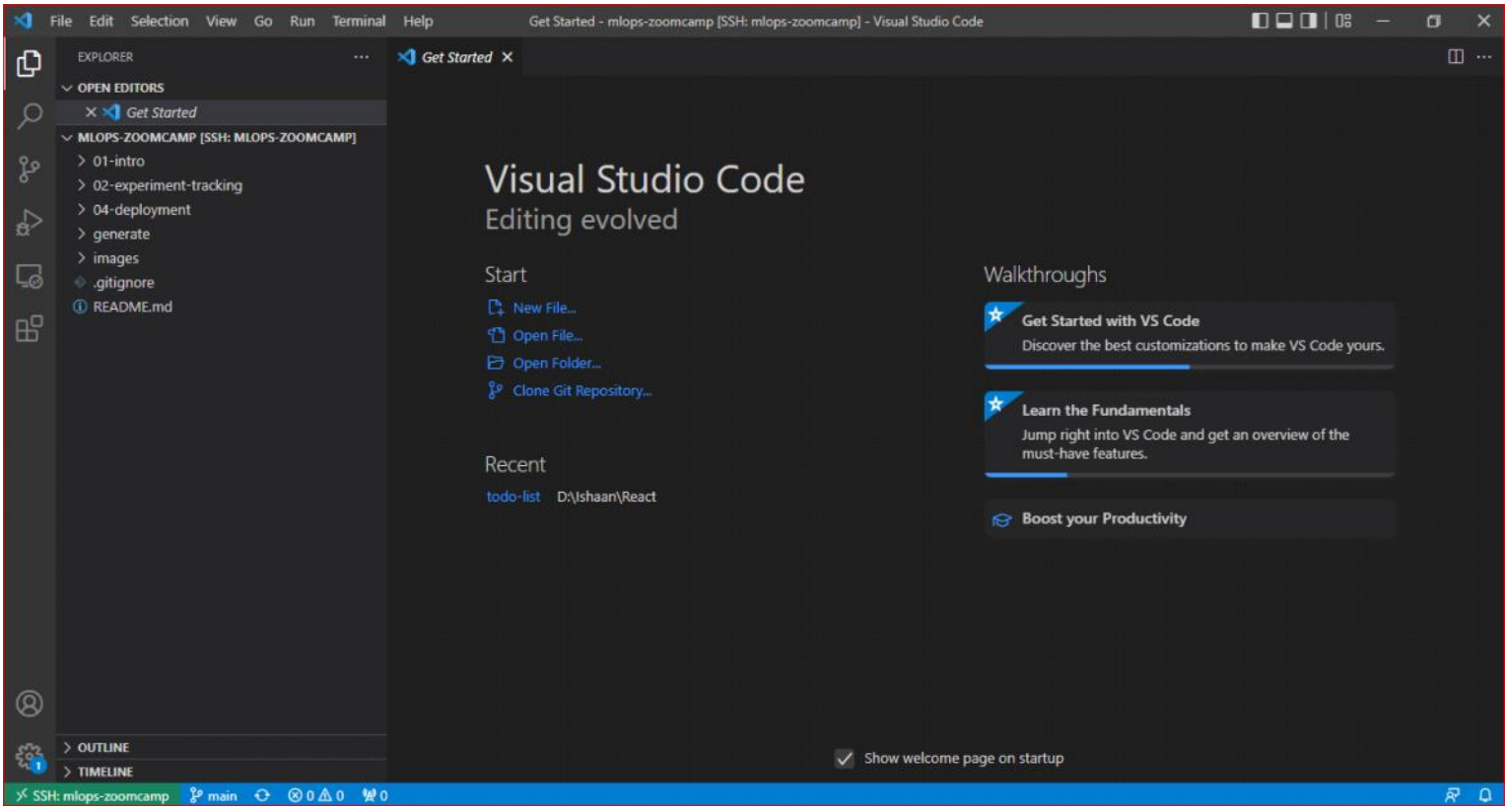
Git clone HTTPSLINK

==========How to get access to this EC2 computer using visual studio code==========

Install extension remote-ssh
We will see this icon on the bottom left : This is for "Open a remote window"

Click on "connect to host"
It will show the hosts we have in ssh/config - remember we added mlops-zoomcamp in that file

Click on "Open Folder" then open the mlops-zoomcamp folder that we cloned in the previous step



======How to start jupyter notebook ==========

- Create a new folder on ubuntu git bash desktop: mkdir notebooks
- cd notebooks
- jupyter notebook

So basically we have started a jupyter notebook on a remote system/host. We can use that host address to access it from a browser of our client/local machine

BUT WE WON'T BE ABLE TO ACCESS IT FROM THE LOCAL COMPUTER BECAUSE WE NEED TO CONNECT THE LOCAL COMPUTER TO THE REMOTE MACHINE

This is where Port Forwarding comes into play: Connect remote port of remote machine to local port of the local machine

We will do that using VSCODE as it's easier this way(We can also do this using ssh config file but it's easier on vscode)

In VS CODE

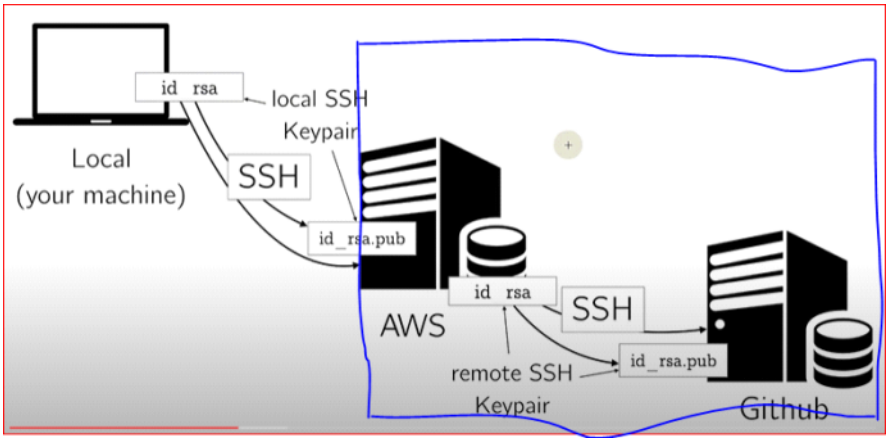
- Got to PORTS(next to Terminal - open by CTRL + ~). Click forward a port.
- Type **8888** this is the port where jupyter is running (we are forwarding it to **localhost:8888**)

Now run this on the local machine browser localhost:8888

Connecting Github from AWS EC2

Creat SSH key in Ubuntu to access Github via SSH

https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent



SSH1 : We already connected to AWS EC2 from our local machine using **.pem** file and public IPV4 key.
SSH2 : Once we're connected to Ubuntu server, we need to SSH again from AWS EC2 to our Github

1. Create key on AWS instance: **ssh-keygen**
   Default location will be /home/ubuntu/.ssh/**id_rsa**.
2. Display content of new key: cat ~/.ssh/**id_rsa.pub**
3. Add this key to Github to establish the connection Github->Settings->SSH Keys->Add SSH Key
4. Test connection: ssh git@github.com
5. Change username and configuration on ubuntu system: git config --global --edit

Committing to Github from Windows without SSH

After August 13th, Github UN and PWD doesn't work
We need to do this:
https://stackoverflow.com/questions/68775869/support-for-password-authentication-was-removed-please-use-a-personal-access-to

Linux Commands

Sunday, May 22, 2022 1:02 AM

- Paste: Shift Insert
- Copy: CTRL insert
- Logout from Ubuntu: Logout
- Remove empty directory: rm directory_name
- Remove non empty directory: rm -rf directory_name
  We use the rm command to delete a directory that is not empty. The syntax is:
  rm -rf dir-name
  rm -rf /path/to/dir/name

  Be careful when you use the rm command with -r and -f options. The -r option remove directories and their contents recursively including all files. The -f option to rm command ignore nonexistent files and arguments, never prompt for anything. There is no undo option. So you have to be very careful with rm -rf command. Let us see some examples.

- Copy one directory to another directory: cp -r dir1 dir2
  https://linuxize.com/post/how-to-copy-files-and-directories-in-linux/
- Move files and directory to directory: mv dir1 dir1
  https://linuxize.com/post/how-to-move-files-in-linux-with-mv-command/
  Move multiple files: mv file1 file2 dir1

- Install python libraries on ubuntu
  Okay so here's what I did and it worked
  1. sudo apt-get update
  2. sudo apt install python3-pip
  3. pip install requirements.txt

  sudo just gives elevated permissions . your current profile does not have permissions to install stuff
  apt-get is the command-line tool for handling packages and provides functions such as installing, removing, and updating packages on a system with a single operation
  Kevin: That's just the way to install libraries for ubuntu (non-Python)

- Download files: wget file_link
- Make a requirement.txt file in a pipenv virtual environment: **pip freeze > requirements.txt**
  https://stackoverflow.com/questions/51845562/how-to-freeze-a-requirement-with-pipenv

Training Model

Sunday, May 22, 2022 1:40 AM

Use 2021 Green Taxi Data From https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

JAN https://s3.amazonaws.com/nyc-tlc/trip+data/green_tripdata_2021-01.parquet
FEB https://s3.amazonaws.com/nyc-tlc/trip+data/green_tripdata_2021-02.parquet

Create a folder on ubuntu desktop and download this data on that:

mkdir data
cd data/
wget https://s3.amazonaws.com/nyc-tlc/trip+data/green_tripdata_2021-01.parquet
Wget https://s3.amazonaws.com/nyc-tlc/trip+data/green_tripdata_2021-02.parquet

```
(base) ubuntu@ip-172-31-12-129:~/data$ wget https://s3.amazonaws.com/nyc-tlc/tri
p+data/green_tripdata_2021-02.parquet
--2022-05-21 20:14:53--  https://s3.amazonaws.com/nyc-tlc/trip+data/green_tripda
ta_2021-02.parquet
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.217.198.208
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.217.198.208|:443... connect
ed.
HTTP request sent, awaiting response... 200 OK
Length: 1145679 (1.1M) [binary/octet-stream]
Saving to: 'green_tripdata_2021-02.parquet'

green_tripdata_2021 100%[===================>]   1.09M  --.-KB/s    in 0.02s

2022-05-21 20:14:53 (64.0 MB/s) - 'green_tripdata_2021-02.parquet' saved [114567
9/1145679]

(base) ubuntu@ip-172-31-12-129:~/data$ ls
green_tripdata_2021-01.parquet  green_tripdata_2021-02.parquet
(base) ubuntu@ip-172-31-12-129:~/data$
```

Improving/**Productionizing** Jupyter Notebooks:

ML FLOW

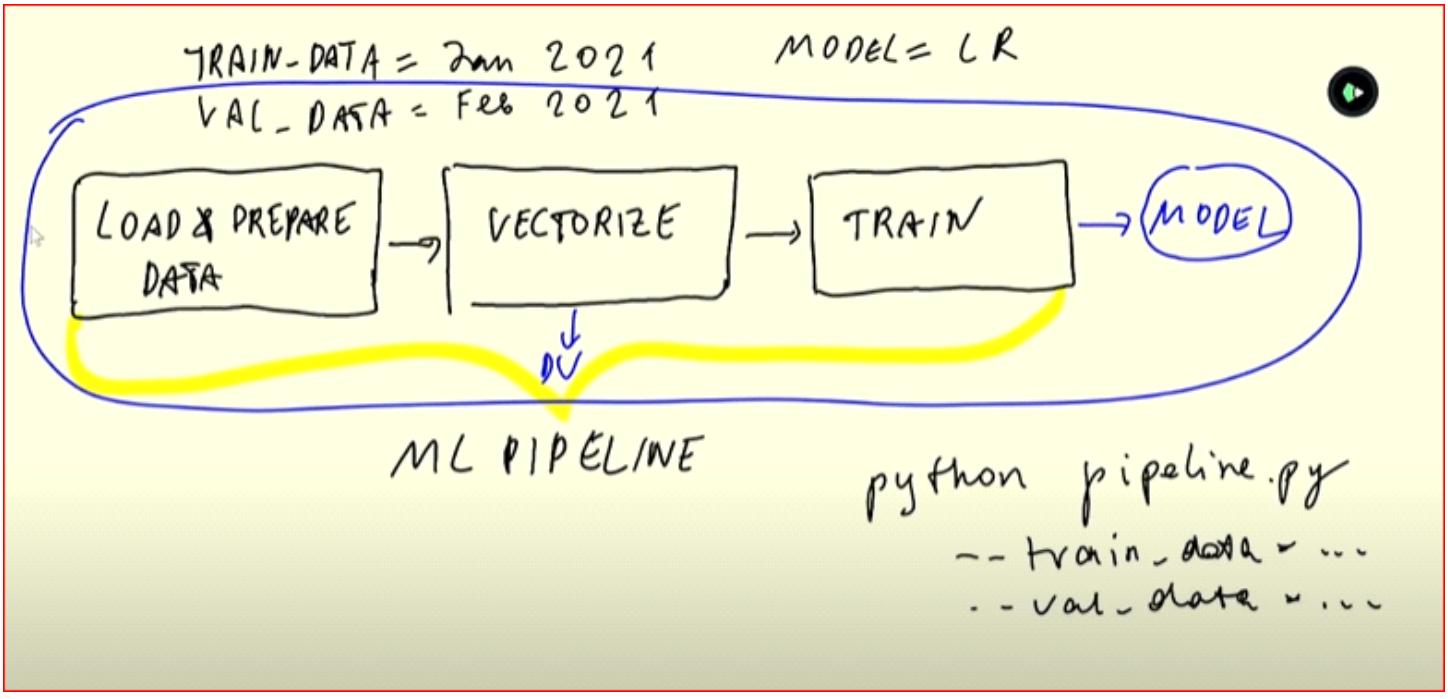
We need a tool to help us remember history and maintain logs. ML Flow comes into picture for that

Experiment Tracker: We try a lot of models by executing and changing various cells but next time we open the notebooks, we don't have the saved information. We can save the model performance in experiment tracker where we'd be able to compare for example how Ridge, Lasso and Linear regressions performed against each other

Model Registry: We save the final model as a bin file using pickle. What about other models. We can maintain a model registry to save models.

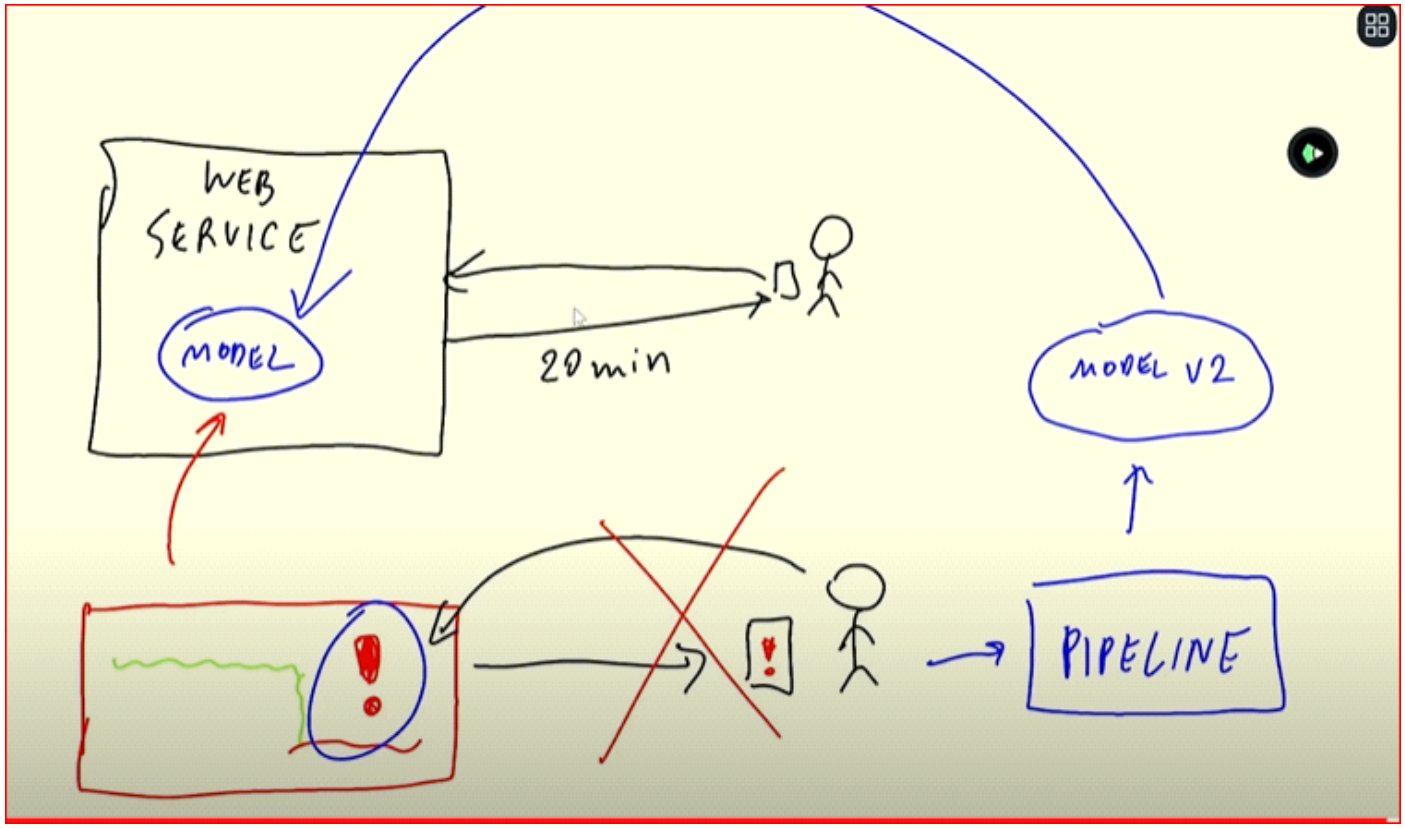
ML PIPELINES

Maintain the code in a parametrized, reusable form



Automating training pipelines. We set some parameters(model, train data, test data) and produce a python script that will handle this pipeline. We can package this pipeb line in a docker.

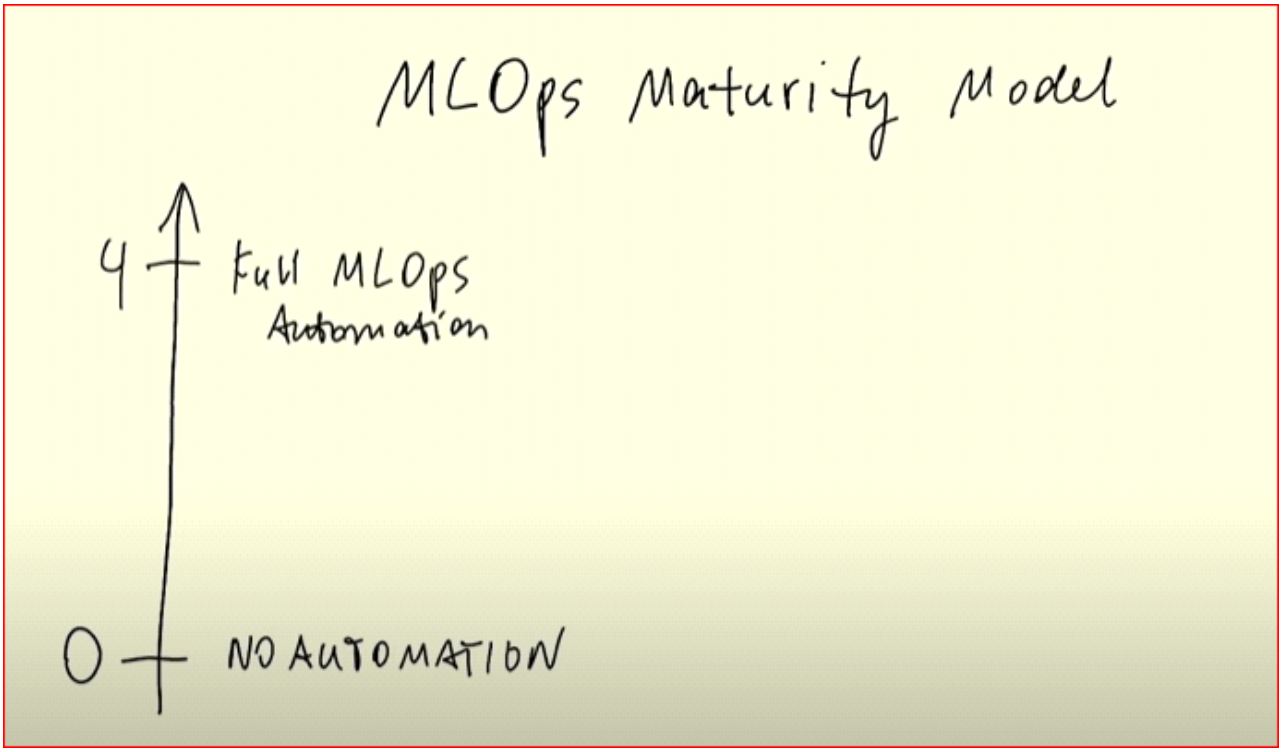
MODEL PERFORMANCE TRACKING + DEPLOYMENT



User interacts with Web Service. Then We have the model performance tracker on bottom left. If performance goes down, we can build a pipeline that handles it, retrain a new model and feeds it to the web service. This is all the purpose of automation.

MATURITY MODEL LEVELS
https://docs.microsoft.com/en-us/azure/architecture/example-scenario/mlops/mlops-maturity-model

When doing MLOPs the idea is to automate all the processes. There are many maturity levels that are there: partially automating to fully automating(no human element - model would automatically retrain if the performance drops. The picture above is for the full MLOPs automation : Level 4.



Level 0: No MLOPs automation

- No pipeline, no experiment tracking, no automation
- Technical: All in Jupyter Notebook
- Use Case: POC Projects
- Workforce: Data Scientist who work alone and are not a part of any engineering team

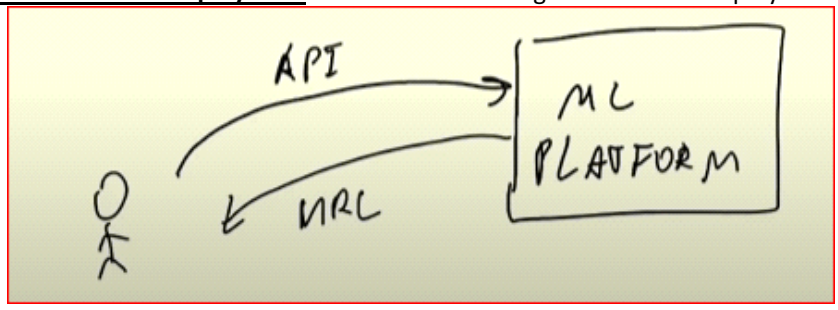
Level 1: Devops, No MLOPs - Good Engineering but no Machine Learning

- They are not specific to Machine Learning. They are not aware of ML handling: No experiment tracking, No reproducibility, DS separated from engineers
- Technical: Releases are automated, we have CI/CD pipelines, unit and integration tests, Track operational Metrics
- Use Case: Going straight from POC to production
- Workforce: DS(work alone and then pass on to engineers). Engineers deploy models using web services

Level 2: Automated Training

- Training pipeline, Experiment tracking, Model Registry, Low friction deployment
- Use Case: When we have 2-3 ML projects, we can use this stage as the investment in deployment would make sense
- Workforce: DS work together with Engineers.

Level 3: Automated Deployment - Automated Training + Automated Deployment (No human needed to deploy models)



- Human calls an API to ML platform(which has models) and the ML platform returns with a response, Model monitoring(how well the model is performing)
- Technical: A/B Tests(if we have two versions of models, which one is better)
- Use Case:
- Workforce: DS(work alone and then pass on to engineers). Engineers deploy models using web services

Level 4: Full MLOps Automation - Automated Training + Automated Deployment + Automated Retraining

- Model deployed in web service -> Model Monitoring -> If model is underperforming -> New training pipeline is automatically triggered -> New model is fed to the web service
- Discussed in photo above this Maturity Module

Revise:
What I do
Preprocess(Df):
    Opretation on df

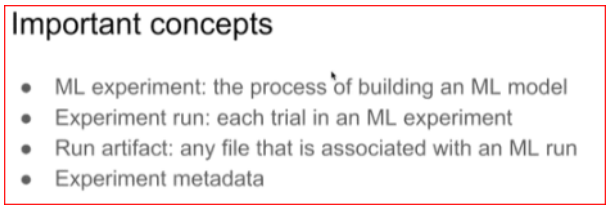
This gives
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.

DO:
Process(file_path):
    Df = read_file

M2: Experiment Tracking & Model Registry with MLFLOW

Tuesday, May 24, 2022 1:06 PM

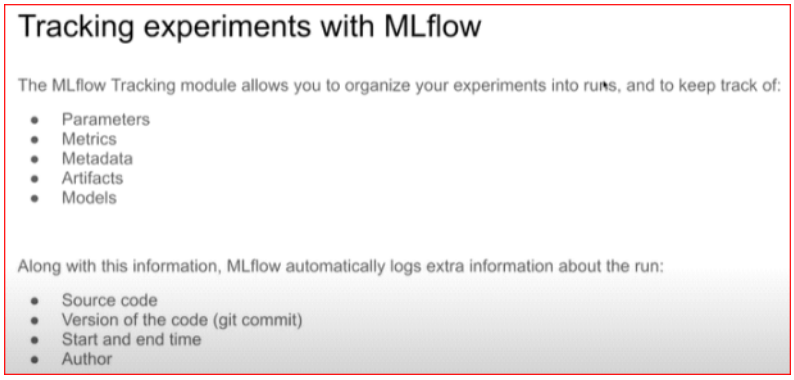
Best Notes: https://gist.github.com/Qfl3x/ccff6b0708358c040e437d52af0c2e43



Experiment tracking is the process of keep track of all the relevant information from an ML experiment, which includes:
- Source Code
- Environment
- Data
- Model
- Hyperparameters
- Metrics

Why is experiment tracking so important?
- Reproducibility
- Organization : Collaboration (everybody should know where to find what). If somebody needs to check how the model has been performing in the last few months
- Optimization : Other open source tools provide optimization as well but it's not automated

We will use **MLFLOW** for this : *An open source platform for the machine learning lifecycle*
In practice it is just a **Python Package** that can be installed using pip and it contains four main modules
- Tracking
- Models
- Model Registry
- Projects



Setting up MLFLOW in a new Conda environment:
1. Create conda environment: **conda create -n exp-tracking-env python=3.8.5**
2. Change C: drive to D: drive: **D:**
3. Change directory to the folder where there is requirements.txt file present - it has all the needed packages in line separated formt: **cd directory_path**
4. Check where is conda environment installed: **conda info --envs**
5. Activate conda environment (we are in D directory as of now so need to provide full path): **conda activate C:\Users\akshit73\anaconda3\envs\exp-tracking-env**
6. Install libraries: **pip install -r requirements.txt** (r is for recursive installation)
7. Launch MLFLow: **mlflow ui --backend-store-uri sqlite:///mlflow.db**

Manual Logging vs Auto Logging

Manual:

Parameters (7)

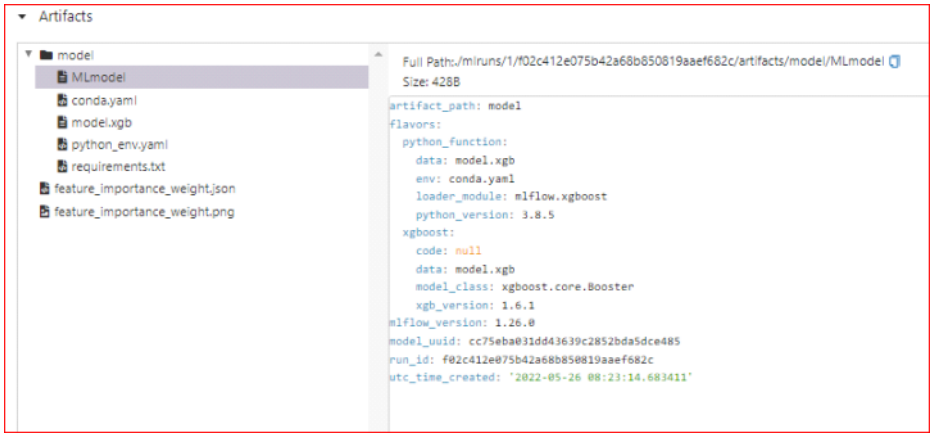
Name	Value
learning_rate	0.5025282456558428
max_depth	11
min_child_weight	0.6512069358884192
objective	reg:linear
reg_alpha	0.028366066476343543
reg_lambda	0.004560675852156512
seed	42

Auto: Logs much more parameters

Parameters (12)

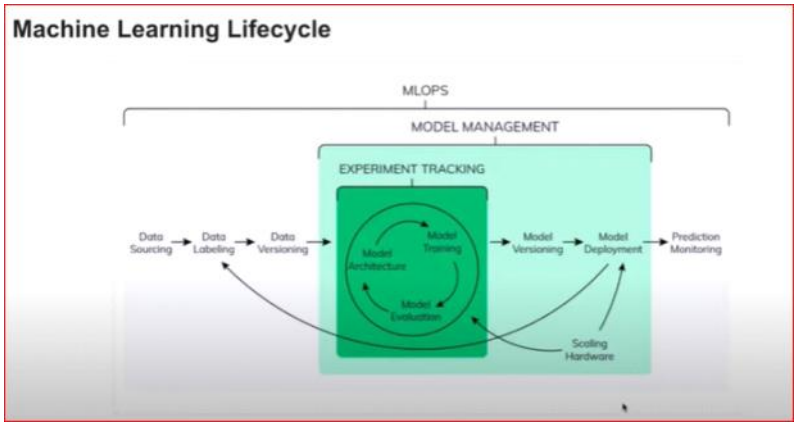
Name	Value
custom_metric	None
early_stopping_rounds	50
learning_rate	0.5025282456558428
max_depth	11
maximize	None
min_child_weight	0.6512069358884192
num_boost_round	1000
objective	reg:linear
reg_alpha	0.028366066476343543
reg_lambda	0.004560675852156512
seed	42
verbose_eval	True

MLFlow saves all the information we need to run the model

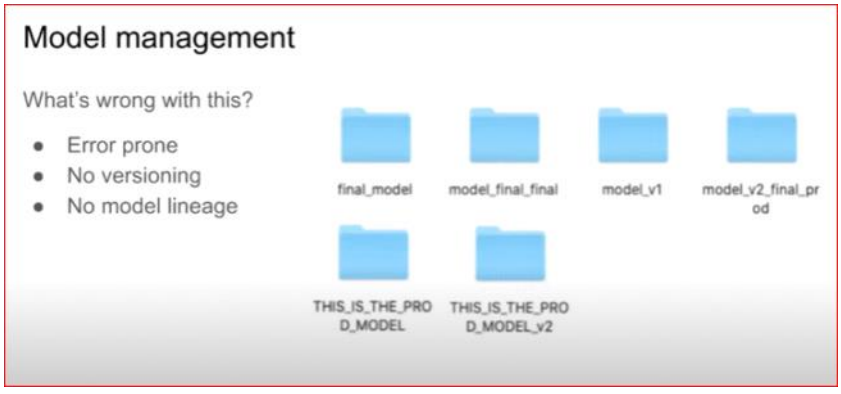


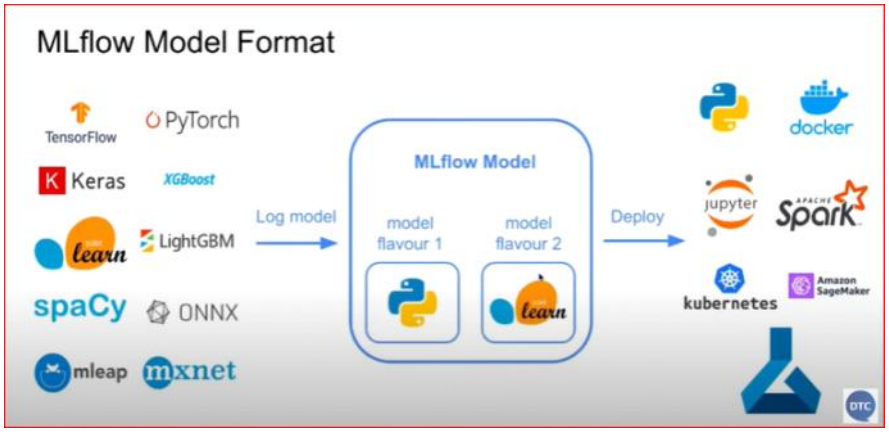
But this was saved as a part of autologging. If we want to do that with frameworks not supported by autologging, we can read how to do that by clicking the model button on top. It will show the code for that.

Model Management using MlFlow

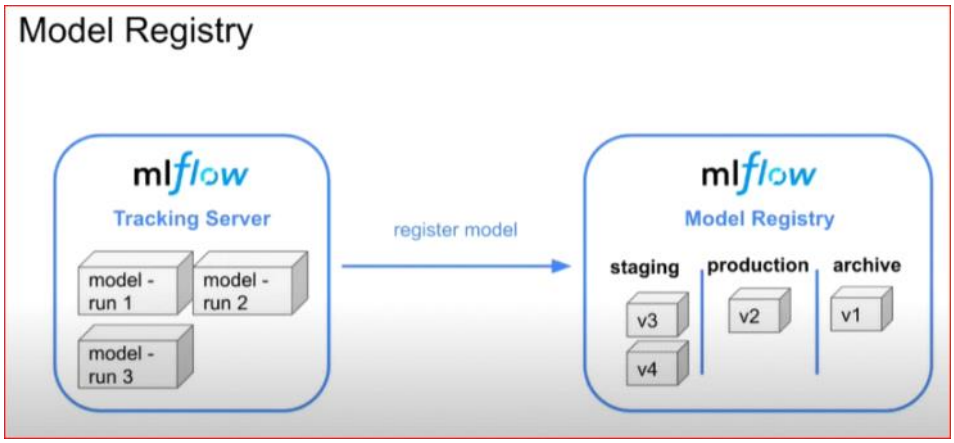


Once we finish experiment tracking, we are happy with one model(the best one). Now it's time to save it and have some version with it.





Model Registry

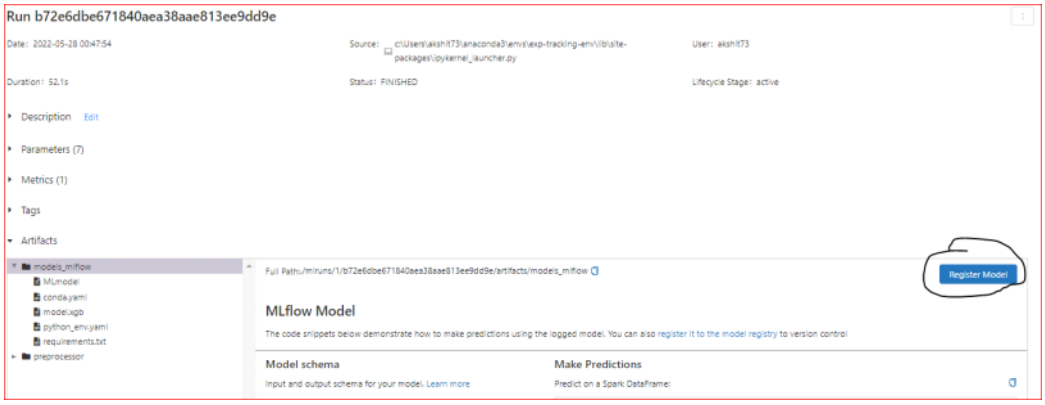


Model registry is not used to deploy models. It only shows us the status of what's happening in staging/production. We actually connect model registry to production server to get this information. Then we can move the models from staging to production or from archive to production.

How do we decide which model to put in production? Consider these parameters:

1. Training time
2. RMSE
3. Model Size

Once we like a model based on this which is ready to be moved into production, we click on "register model"

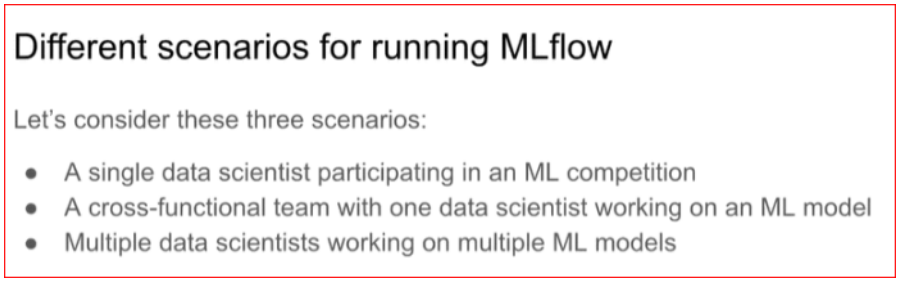


Once we register it, it will appear in the "**models**" tab of MLFLOW. We can save multiple version of model(Xgboost , Gradient Boost) under the same model registry
Name and add tags to those model versions (xgboost, gradient boost).

Artifacts don't show up in experiments but show up in models? Why?
Same lines: we used .db in train.py but uri server link in other .py files. Why?
How is it flowing? Diagram?
Nothing appeared in artifacts

Try the same flow using mlflow server and then Yang Cao's point might be valid

MLFLOW in Practice



1. Single DS participating in ML competition:
   Model Registry isn't required and would have no use since there is no production.

Command to run mlflow UI(run it from where you're running the script): **mlflow ui**

- *Tracking server:* no (use MLFLOW UI)
- *Backend store:* Local filesystem (by default in **MLRUNS** folder)
  Metrics and Params: log_metric() and log_params()
  It's saved in MLRUNS -> 0(experiment ID) -> Run_ID -> metrics
  It's saved in MLRUNS -> 0(experiment ID) -> Run_ID -> params
- *Artifact store:* Local filesystem (by default in **MLRUNS** folder)
  Model: Log_model(model_object , artifact_path = 'models')
  It's saved in MLRUNS -> 0 -> Run_ID -> Artifacts/models

Cannot access model registry when backend and artifact store is LOCAL

2. Cross functional team with one DS working on ML model
   Model Registry may or may not be required
   Command to run mlflow UI(run it from where you're running the script):
   mlflow server --backend-store-uri sqlite://backend.db --default-artifact-root ./artifacts_local
   (mlflow will be launched at http://127.0.0.1:5000

   Mlflow.set_tracking_uri("http://127.0.0.1:5000")

   The experiments can be explore locally accessing the local tracking server

   - *Tracking server:* yes, local server
   - *Backend store:* sqlite database
     Mlflow will use the backend store associated with http://127.0.0.1:5000
     Metrics and Params: log_metric() and log_params()
     It's saved in backend.db
     It's saved in backend.db
   - *Artifact store:* Local filesystem (**artifacts_local** folder instead of mlruns folder)
     Mlflow will use the artifact store associated with http://127.0.0.1:5000
     Model: Log_model(model_object , artifact_path = 'models')
     It's saved in artifacts_local -> 0 -> Run_ID -> Artifacts/models

   Can access model registry

3. Multiple DS working on multiple ML models: Model Registry is required
   The main idea in this one is to share whatever we're doing and for that we'll need something remote for both backend store and artifacts so we can share it

   Command to run mlflow UI(run it from where you're running the script):
   mlflow server -h 0.0.0.0 --backend-store-uri postrgres://DB_USER:DB_PASSWORD@DB_ENDPOINT:5432/DB_NAME
   --default-artifact-root s3://S3_BUCKET_NAME

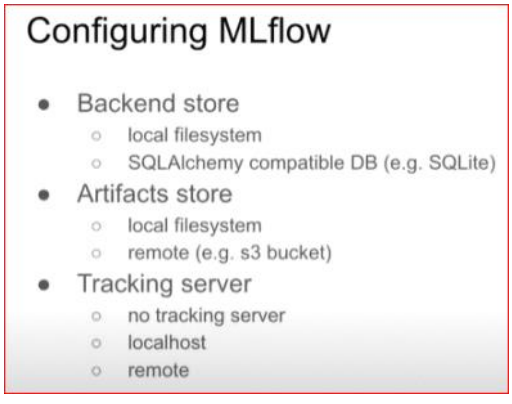
   TRACKING_SERVER_HOST: AWS LINK
   Mlflow.set_tracking_uri(f"http://{TRACKING_SERVER_HOST}:5000")

   The experiments can be explore locally accessing the remote tracking server

   - *Tracking server:* yes, remote server , EC2
   - *Backend store:* postgresql database
     Metrics and Params: log_metric() and log_params()
     It's saved in postgresql AWS
     It's saved in postgresql AWS
   - *Artifact store:* s3 bucket
     Model: Log_model(model_object , artifact_path = 'models')
     It's saved in S3 bucket -> 0 -> Run_ID -> Artifacts/models

   Can access model registry

Configuring MLFLOW



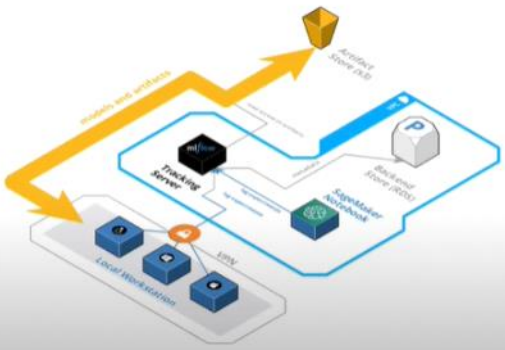
1. Backend Store: Save all metadata of experiment (metrics, parameters, tags etc.). By default, MLFlow assumes we want to save this information locally and it will create a folder in our local filesystem. Alternatively, we can use SQL Lite DB and this enables us to use model registry. By default is it is the folder **MLRUNS**
2. Artifact Store: Now we can save artifacts on local or remote.
3. Tracking Server:
   - No tracking server: For case 1: individual DS working on a competition
   - Local host:

Remote tracking server

The tracking server can be easily deployed to the cloud

Some benefits:

- Share experiments with other data scientists
- Collaborate with others to build and deploy models
- Give more visibility of the data science efforts



Issues and Limitations of MLFLOW

Issues with running a remote (shared) MLflow server

- Security
  - Restrict access to the server (e.g. access through VPN)
- Scalability
  - Check Deploy MLflow on AWS Fargate
  - Check MLflow at Company Scale by Jean-Denis Lesage
- Isolation
  - Define standard for naming experiments, models and a set of default tags
  - Restrict access to artifacts (e.g. use s3 buckets living in different AWS accounts)

MLflow limitations (and when not to use it)

- **Authentication & Users**: The open source version of MLflow doesn't provide any sort of authentication
- **Data versioning**: to ensure full reproducibility we need to version the data used to train the model. MLflow doesn't provide a built-in solution for that but there are a few ways to deal with this limitation
- **Model/Data Monitoring & Alerting**: this is outside of the scope of MLflow and currently there are more suitable tools for doing this

MLflow alternatives

There are some paid alternatives to MLflow:

- **Neptune**
- **Comet**
- **Weights & Biases**
- and many more

M3: Workflow Orchestration with Prefect

Saturday, June 4, 2022 11:44 PM

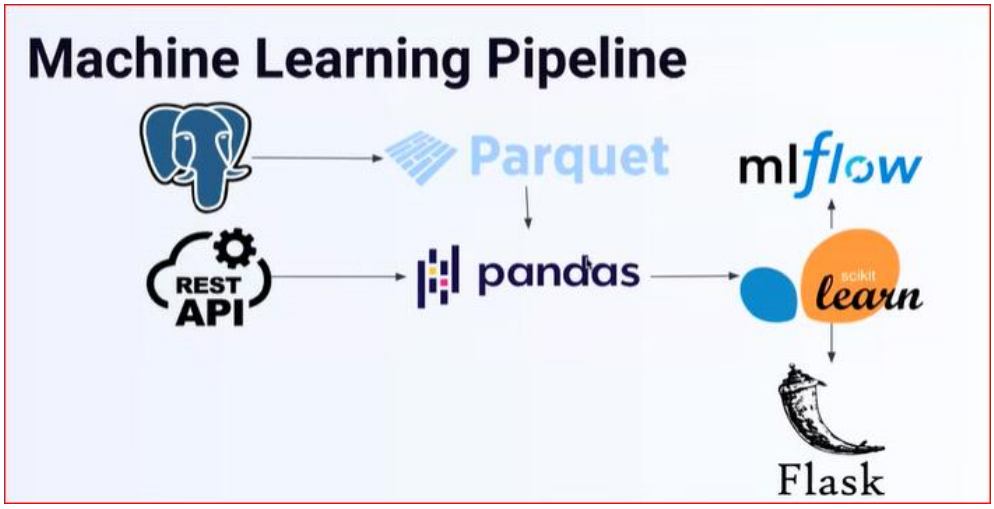
NOTES: https://gist.github.com/Qfl3x/8dd69b8173f027b9468016c118f3b6a5

Overview : Why Prefect

What is workflow Orchestration: Set of tools that schedule and monitor work that we want to accomplish. Eg: We want to run a machine learning pipeline every week and if a model fails, then we to introduce some observability to know why it failed.

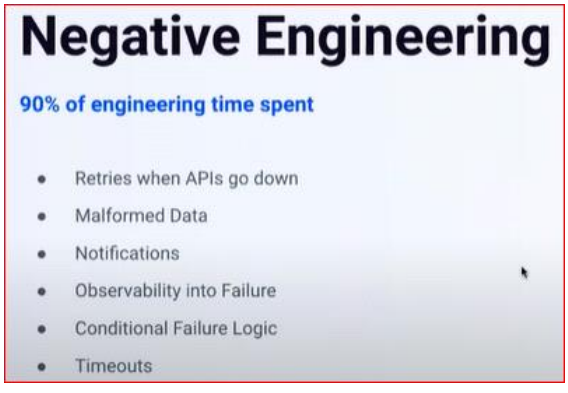
Additional benefit: Our goal is also to minimize the impact of potential errors or in another words fail gracefully(have a solid failure mechanism: what all should run if something fails)

Classic Machine Learning Pipeline:



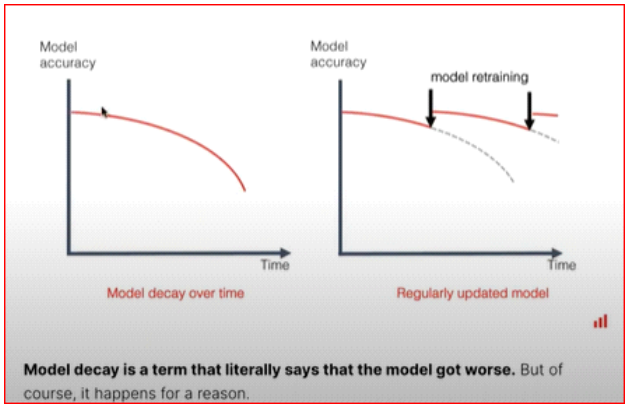
We have a **postgreSQL database** where we perform some joins to generate a **parquet file**. We use **pandas** to ingest that parquet file and combine it with some **API data** that we're pulling. Then we pass this data to **scikit learn** to train a model. Once we train the model, we register the model artifact on **mlflow.** We also deploy the model on **Flask** based on certain conditions.

Now, in all these, things can go wrong anywhere in multiple forms (Rest API not loading, potgreSQL going through maintanance, wrong Excel file format). When we think about it, we spend 90% of our job time to deal with the potential errors happening. This is what **negative engineering is : coding against failure/coding against the negative effects happening.**



Using Prefect, we try to reduce this negative engineering time so we can focus more on models and business time.

Introduction to Prefect



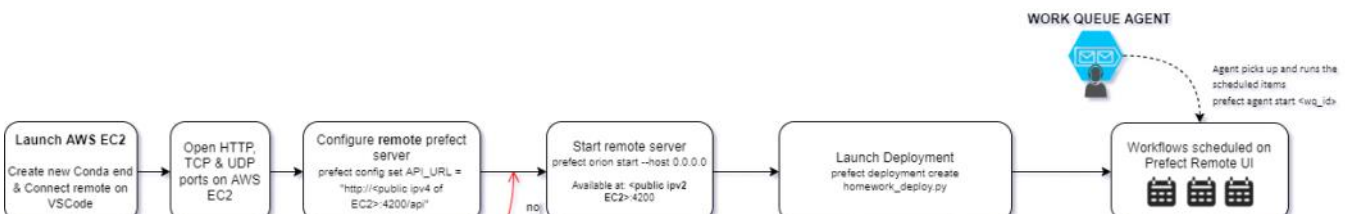
Model decaying is known as drift and to avoid drift, we should retrain our model.

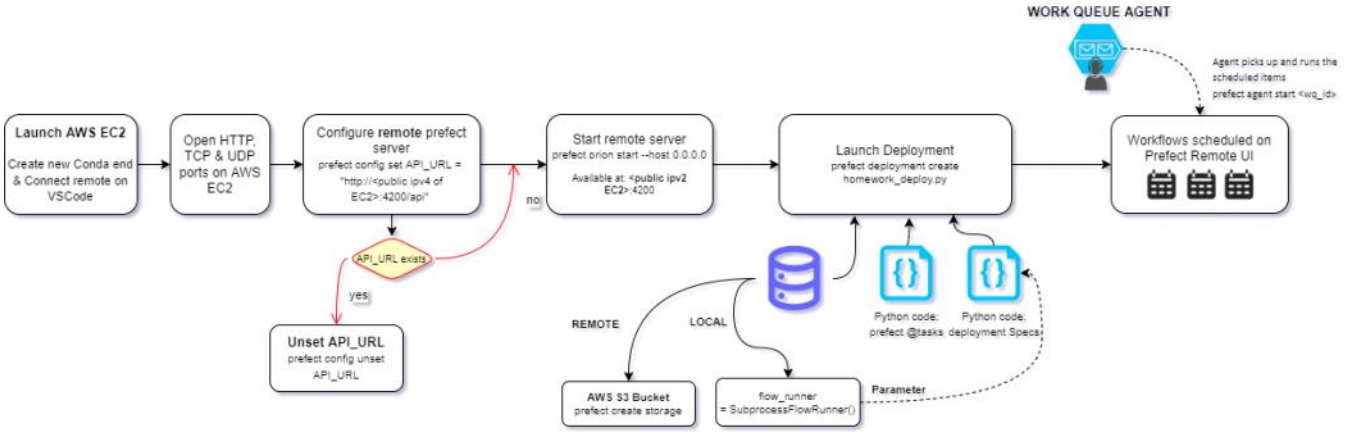
Steps to make a flow

1. Modularize/re-factor the code into functions
2. Assign decorators : **@task** to the functions
   By changing functions to tasks, we get observability and logs out of it(they appear on the dashboard)
3. Run a flow calling those functions
   The way we run the flow is to assign a **@flow** decorator to the main function
4. Perfect orion script (to show everything on UI)

Flow's default running mechanism is **concurrent,** we can make it sequential.

WEEK 3 SUMMARY





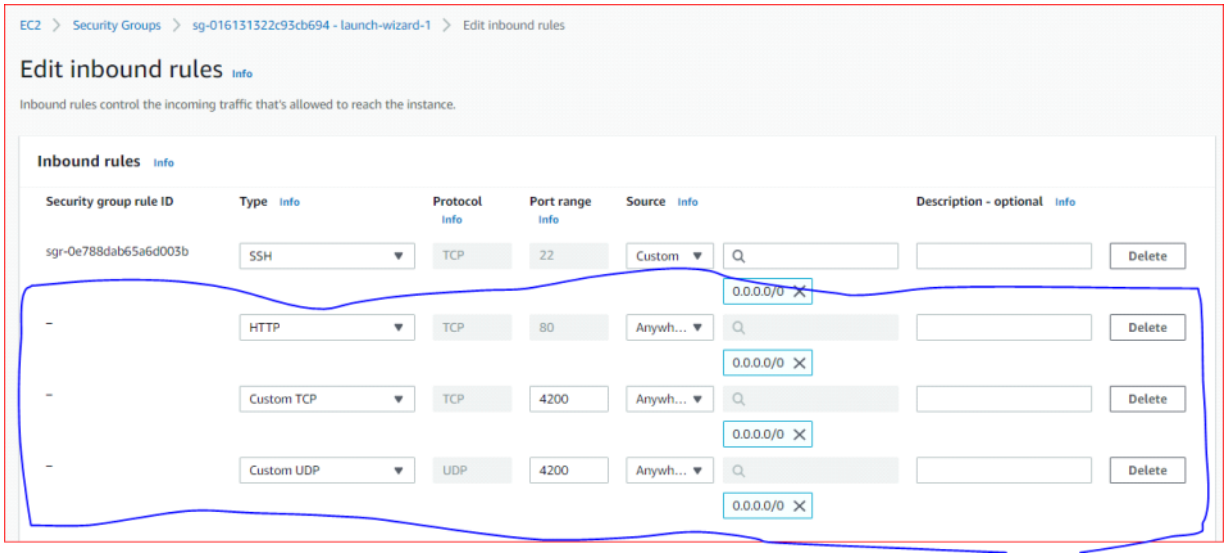
@Akshit Miglani

Using environment env-prefect for this work

1. Launch EC2 instance (ssh to public IPV4 : **mlops-zoomcamp**)
2. Create new conda environment. Install requirements using requirement.txt
3. Launch VS code on local and connect **mlops-zoomcamp**
4. Open HTTP, TCP, UDP ports in security settings of EC2 instance of AWS
5. Use EC2 IP address to spin a **remote prefect server**
   prefect config set PREFECT_ORION_UI_API_URL="http://**3.234.255.76**:4200/api"
   prefect config set PREFECT_ORION_UI_API_URL="http://<<public ipv4 of EC2):4200/api"
   Do **Prefect config view** to check settings
6. Start the remote server **prefect orion start --host 0.0.0.0**
   0.0.0.0 was our source in the security settings of EC2
   The prefect remote server will be available at **3.234.255.76**:4200
7. In homework.py we have all the flow and task
8. In homework.deploy.py we have the deployment settings in which we will also mention when to run the scheduled flows (for eg: every 15th day of month : this will be done using **cron deployment**). But we don't have a remote storage so for local storage we will have:
   flow_runner = SubprocessFlowRunner()
9. On VSCODE(connected to mlops-zoomcamp), launch the deployment: prefect deployment create homework_deploy.py
10. Go to the prefect server UI, add a work queue for an agent to run these tasks
11. Launch the agent : prefect agent start <work_queue_id>
12. Check the status of available runs. Hours let us see how far in future we want to look the scheduled runs
    prefect work-queue preview d84f9ad6-f3f3-4451-ae26-b28334c8cb94 --hours 1000000

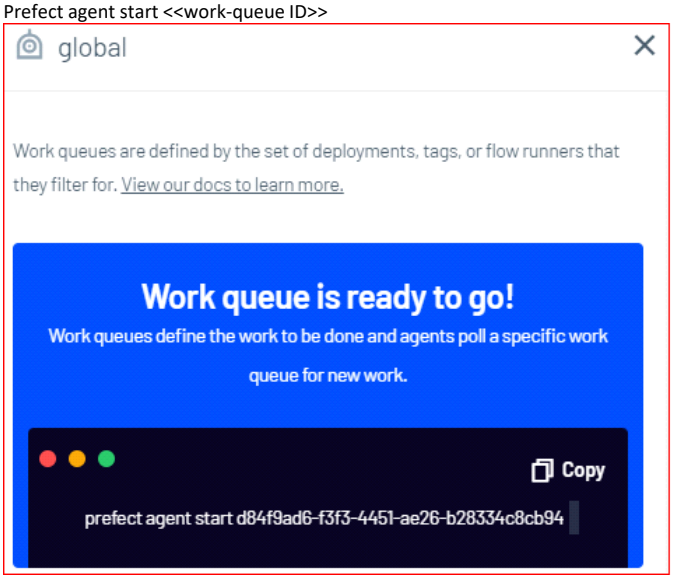
Local Instance of Prefect:
Simply do **prefect orion start**

Remote Prefect:
We need to open port on AWS EC2 instance to access the servers over SSH, HTTP or HTTPS.



To get things running on schedule we need three components:

1. Storage(saves the file and saves it somewhere (local/remote) docker/kubernete need s3 storage
2. Get the scheduled runs on prefect (**cron deployment**)
3. To run these schedules runs, we need an agent to run them. For this we create a **work-queue**
   We need an agent to run the work-queue
   Prefect agent start <<work-queue ID>>



If we have docker this agent will spin up the docker and run the tasks on that.

Questions/ More Insights

Hi All, I just completed week 3(pretty late) and I have a few thoughts on my mind that I'd love more clarity on before I move ahead:

1. I didn't understand the usage of .result(). My current understanding is that if we have a function returning more than two parameters, we're required to use .result(). I didn't understand "we use .result() when mixing and matching of normal functions with task functions"

   The issue is that the PrefectFuture is not yet indexable so you can't do this syntax: x, y = (1,2) which is standard Python. We call .result() to create that tuple so we can write that expression. The important thing is that you need to call the .result() to work with other Python code like if-else

2. Following video 3.4, I ran the prefect server on AWS EC2 and it was accessible on my local browser(windows) by accessing EC2_ipaddress:4200 . In the video at 7:07, Kevin does the prefect config set URL on the local machine. Why is this needed? I can already access the prefect on my local without doing this?

   Here is what I did following video 3,4:
   1. Connect to AWS EC2
   2. prefect config set PREFECT_ORION_UI_API_URL="http://<**EC2IPV4**>):4200/api" on AWS EC2
   3. prefect orion start --host 0.0.0.0 on AWS EC2

   After this prefect was running on EC2(I believe) and it was accessible on my local machine at the address **EC2IPV4**:4200Right after this, at 7:07, you again set up the prefect config set PREFECT_ORION_UI_API_URL="http://<**EC2IPV4**>):4200/api" on the local machine. My question is that I could access the remote server on EC2 on local before doing this, so why do this?

   There are two env variables here. The first is Orion API which I think you understand. The local needs to hit the EC2 instance. The second one is telling the UI what API to hit because it uses localhost too by default. If you don't set that, the UI will be blank when you load it

   Also, on the local machine you set PREFECT_API_URL and on remote instance we

set PREFECT_ORION_UI_API_URL. The commands were different. I am just reiterating kevin's point because @Akshit Miglani copied the same command for EC2 and local host in his previous message.

3. Working with prefect in industry, how exactly are we dealing with model decay(re-training the model when it goes wrong) here?

   That is more related to the Evidently AI section. Prefect is general purpose and you can just specify actions to take if you see drift

4. If we decided to use Docker, where do we use it? Would it be somewhere with the work queue?

   The DeploymentSpec took SubprocessFlowRunner, it can take DockerFlowRunner where you specify an image.

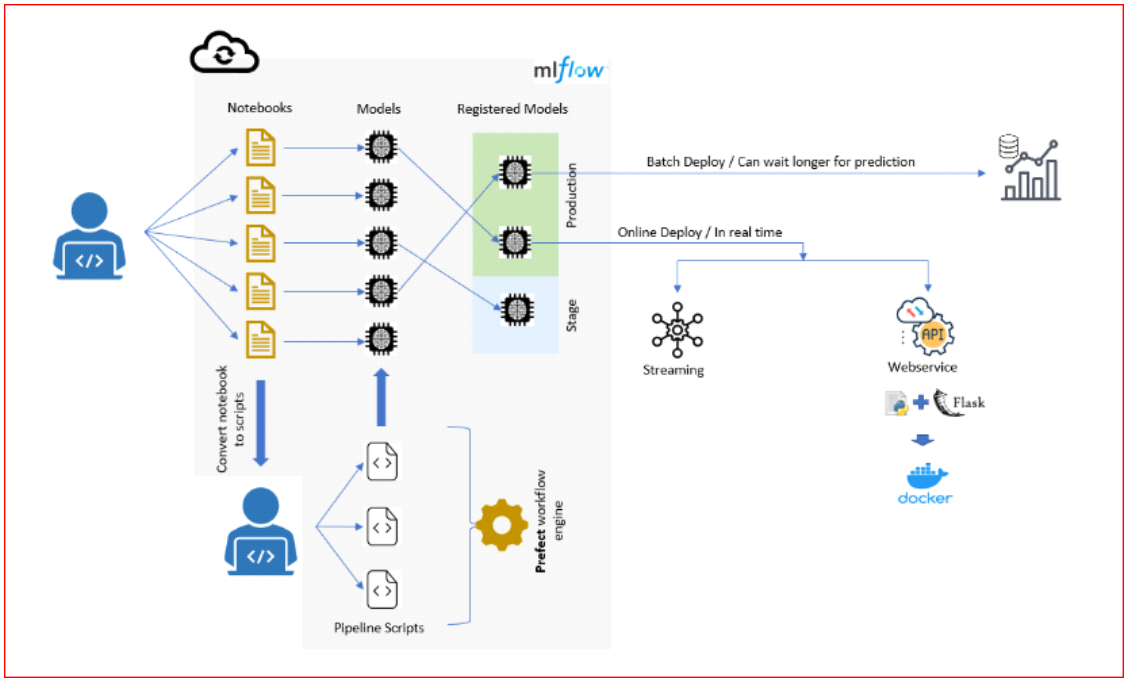
5. How would prefect and MLFlow work together? Prefect is tracking workflows and MLFlow is tracking model details. How do we use these together in the industry?

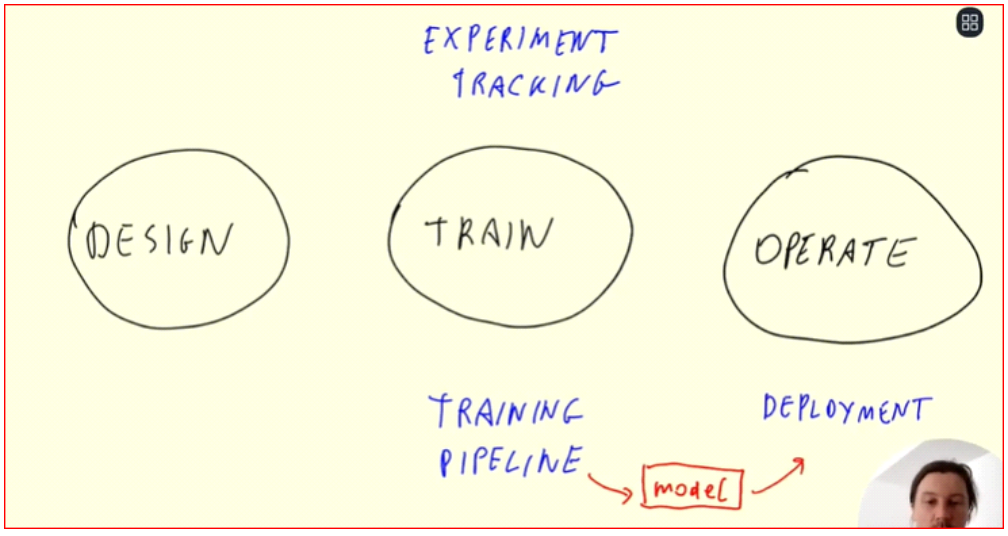
   Prefect will be general purpose and MLFlow will be specific. MLFlow has a nice API to promote models based on certain metrics, you can use this functionality inside a task

M4: Model Deployment

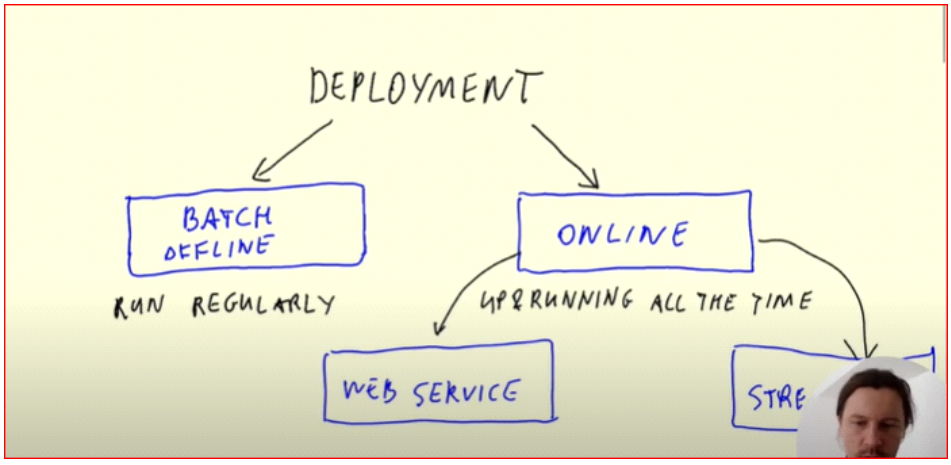
Wednesday, June 15, 2022 11:09 PM

https://gist.github.com/Qfl3x/de2a9b98a370749a4b17a4c94ef46185
https://github.com/BPrasad123/MLOps_Zoomcamp/tree/main/Week4



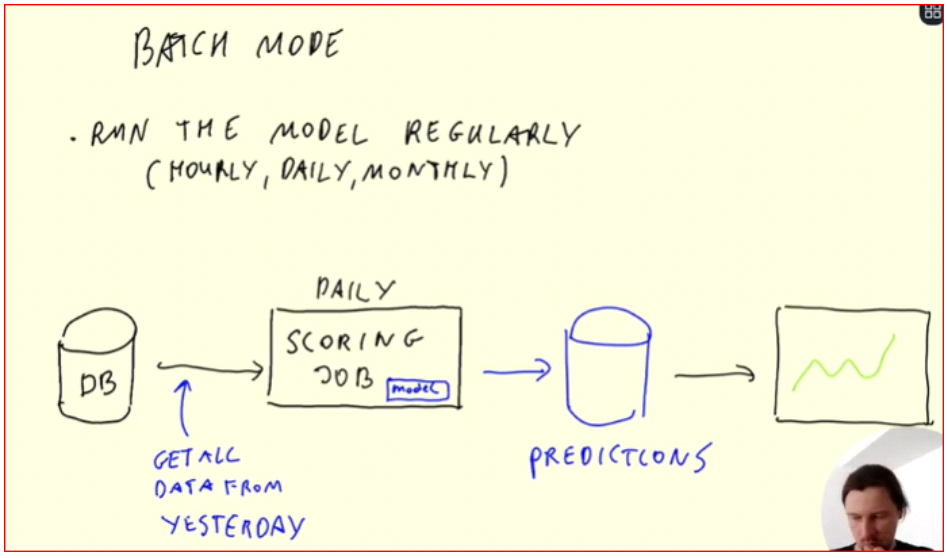


So far we have done everything before Operate. In this module we will explore Deployment.

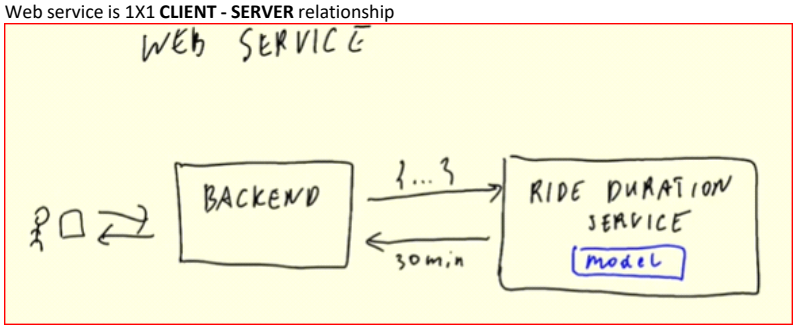


Web Service: We send a request and receive back predictions through response.

TYPES



Web service is 1X1 **CLIENT - SERVER** relationship

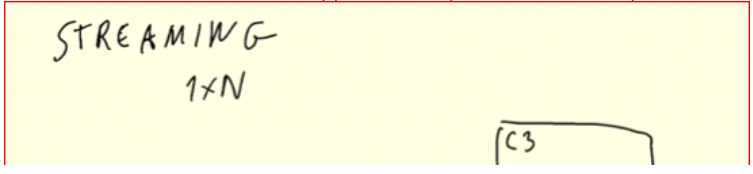


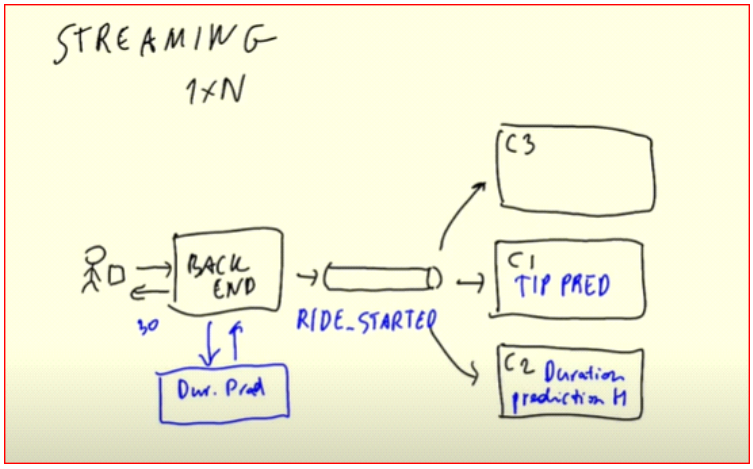
CLIENT - Backend

User interacts with APP Front End. Front End interacts with Backend. Backend sends request to web service(working all the time). Web service replies back with predictions. Backend gives the response to our app which we see.

Streaming - 1XMANY or MANY to MANY - **Consumer Producer** relationship

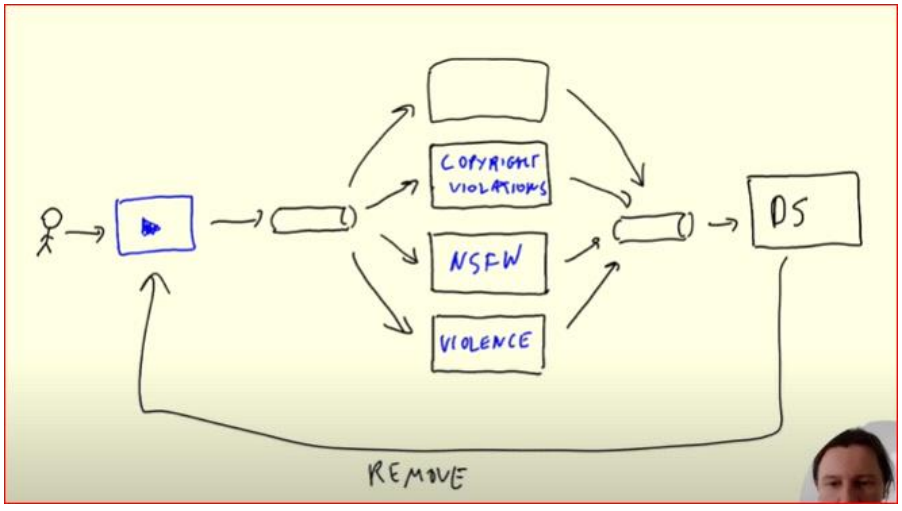
The difference here is that backend only pushes the request. It doesn't directly interact with consumers.





The duration prediction on the left is to show a quick prediction. The one on the right actually gives a more accurate prediction.
Producer: User -> App -> Backend
Event/Stream
Consumer: 3 services

Example: Youtube content moderation



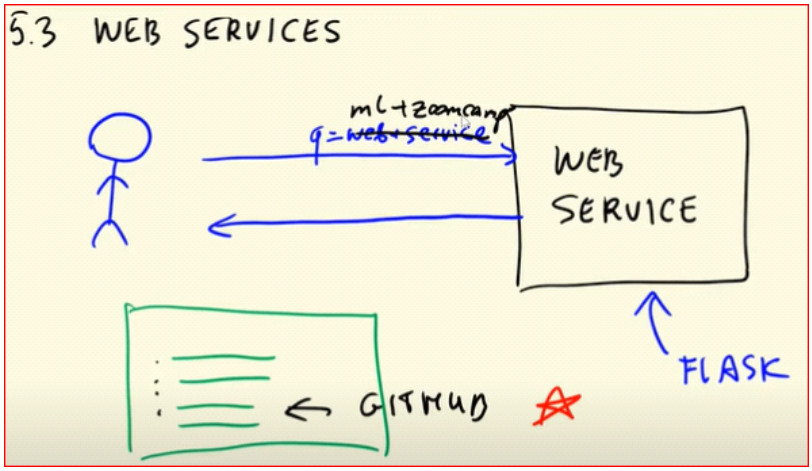
User uploads a video. A streaming event runs. The 3 services check if the content is bad. They report their responses to the **Decision Service** which replies back to Youtube to REMOVE the video.
Advantage of this architecture: It's scalable: we can add another service without disturbing anything

Deploying models to WEBSERVICE using Flask and Docker

ML Zoomcamp 5.3 - Web Services: Introduction to Flask

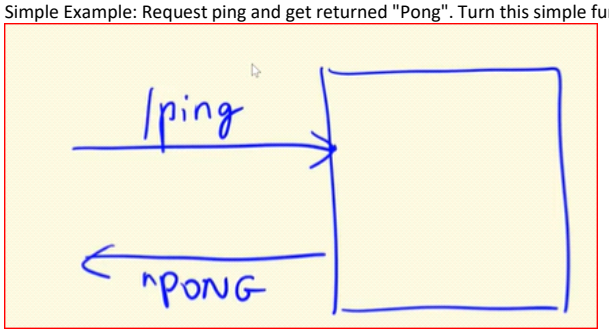
https://github.com/alexeygrigorev/mlbookcamp-code/blob/master/course-zoomcamp/05-deployment/03-flask-intro.md

https://gist.github.com/Qfl3x/de2a9b98a370749a4b17a4c94ef46185 (BEST NOTES)



Example: When we search "mlzoomcamp" on google, it passes the query (ml+zoomcamp) to Google and Google returns us back a list of pages. We can deploy such service using FLASK.

Simple Example: Request ping and get returned "Pong". Turn this simple function into Flask web service

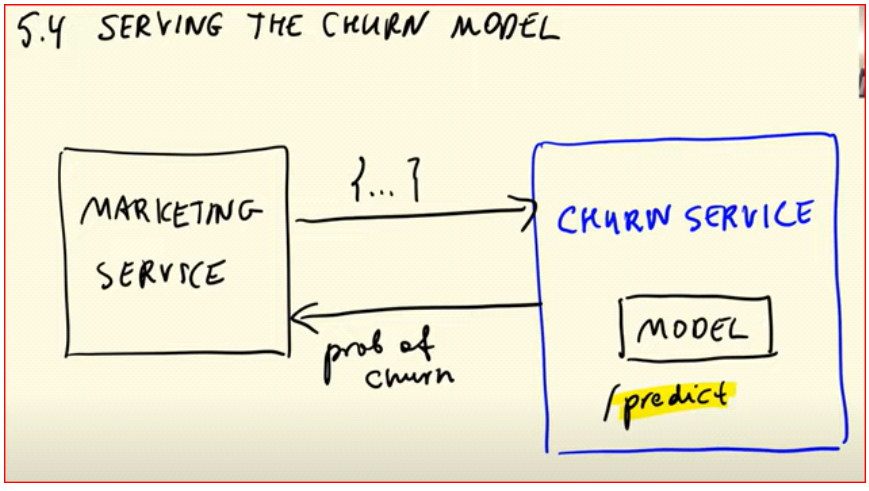


plan.md | ping.py | train.py | predict.py

ping.py

```python
from flask import Flask

app = Flask('ping')

@app.route('/ping', methods=['GET'])
def ping():
    return "PONG"

if __name__ == "__main__":
    app.run(debug=True, host='0.0.0.0', port=9696)
```

Decorators are just ways to add functionalities to functions.



Marketing service would pass on the customer parameters. We need to write a predict function which would return back the predictions using the parameters we got.

Component 1: Predict function that takes in customer input(independent variables input) and returns the prediction using a model which we load through pickle.

WEB SERVICE IMPLEMENTATION

Docker File

FROM python:3.9-slim *#Existing docker image*
RUN pip install -U pip *#Update pip as certain packages (ex: XGBoost) need certain versions of pip*
RUN pip install pipenv *#Managing dependencies*

WORKDIR /app *#Creates and cd's into the /app directory on local*
COPY ["Pipfile", "Pipfile.lock", "./"] *#Copy the dependency setting into the current directory(app folder)*
RUN pipenv install --system --deploy *#--system installs the environment/dependencies in the parent OS in the container, --deploy makes sure Pipfile.lock is up-to-date and will crash if it isn't*
COPY ["predict.py", "lin_reg.bin", "./"] *#Copy model and python file(Flask app) to working directory(app folder)*
EXPOSE 9696 *#Open the docker container port for interaction with anything outside(local/cloud)*
ENTRYPOINT ["gunicorn", "--bind=0.0.0.0:9696", "predict:app"]

Building Docker Image

```
docker build -t ride-duration-prediction-service:v1 #docker build image_name:tag
```

And run the container that was built with:

```
docker run -it --rm -p 9696:9696 ride-duration-prediction-service:v1
```

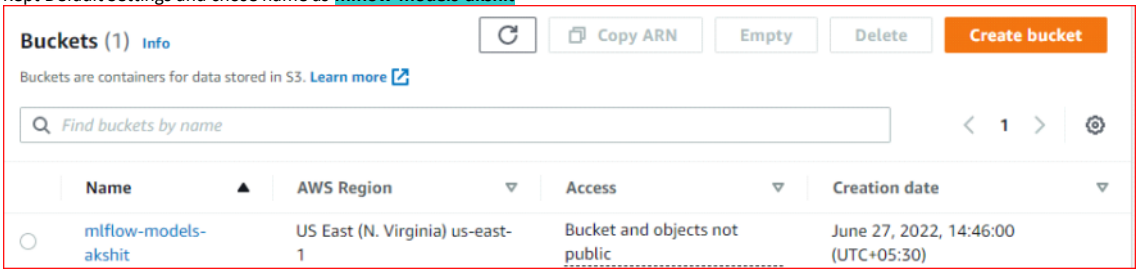
Now when we request predictions like earlier, we're instead calling the WGSI within the Docker Container.
Now we can deploy this docker container anywhere where it's supported

How to use MLflow on AWS to Better Track your Machine Learning Experiments
https://github.com/ahmedbesbes/mlflow/blob/main/README.md

PUSHING MODELS TO AWS s3 BUCKET

1. Launch EC2
2. Create s3 Bucket on AWS to store models there(we will pull models from there then)
   MLOps Zoomcamp 2.6 - MLflow in practice (26:00)

   Kept Default Settings and chose name as **mlflow-models-akshit**



3. Launch MLFLOW on ubuntu EC2 environment using this cloud bucket storage as artifact location

   Here if I try to push artifacts to s3 model I will get an error: NoCredentialsError: Unable to locate credentials
   https://app.slack.com/client/T01ATQK62F8/C02R98X7DS9/thread/C02R98X7DS9-1656325853.189499
   I have 2 options here :
   Either add the credentials, or set the bucket to public. Setting bucket to public is easy but it isn't industry standard/best practice

Adding credentials & connecting s3 bucket to mlflow server

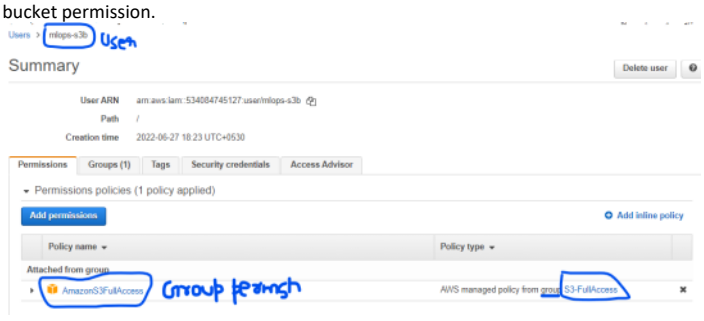
https://boto3.amazonaws.com/v1/documentation/api/latest/guide/credentials.html (boto3 and awscli)
https://gist.github.com/Qfl3x/8dd69b8173f027b9468016c118f3b6a5 (getting access keys)

The access keys are what will be used by boto3 (AWS' Python API tool) to connect with the AWS servers. If there are no Access Keys how can they make sure that have the righ t to access this Bucket? Maybe you're a malicious actor (Hacker for ex). The keys must be present for boto3 to talk to the AWS servers and they will provide access to the Bucket if you possess the right permissions. You can always set the Bucket as public so anyone can access it, now you don't need access key s because AWS won't care.

a. Go to AWS account -> security credentials -> Create User
Why are we creating a user(because we can get the access keys from our main account(root)'s security credentials as well?
It is recommended not to use your AWS root account for this kind of stuff. It can work, but it's not recommended because that account has root permissions. Instead add a user and give it just enough permissions to be able to do what it has to. Imagine that you're using some piece of software that you don't really know much, you can't trust it. If you give it root access it can do some very nasty stuff. It's a paradigm in Security, minimal trust. Don't trust anything with your root account except software that needs it like the AWS CLI.

When we create the user, it will give us the access keys**(Access key ID, Secret access key)** and we can save the same in a csv file.

b. Make a group and give it **Amazons2FullAccess** Permission. Now even if someone gets our key, all they'll be able to access will be our s3 bucket and nothing else as it only has s3 bucket permission.



We have got the access keys now.
Step 2 is to assign the Environment variables to those keys. export AWS_ACCESS_KEY_ID="...". Mlflow will use these keys from the variables to connect to AWS (through boto3). Then simply type the Bucket's URI in the mlflow server command and it should work. Now if you just do this, you'll have to create a new account and assign the keys every time you use MLflow.
To avoid that, either use the AWS CLI which stores the keys (Written in the docs I've given earlier for boto3), or write a **set_aws_keys.sh** script as follows:
<code>
#!/bin/bash
export AWS_ACCESS_KEY_ID="..."
export AWS_SECRET_ACCESS_KEY="..."
</code>
And execute this file with ./set_aws_keys.sh. It is preferable to use the aws CLI tool instead, much more secure.

c. Create a directory called **mlflow** and launch a virtual environment using **pipenv**
Sudo pip3 install pipenv
Sudo pip3 install venv
Pipenv install mlflow
Pipenv install awscli
Pipenv install boto3
Pipenv shell (Activate virtual environment)
Aws configure

```
(mlflow) (base) ubuntu@ip-172-31-12-129:~/mlflow$ aws configure
AWS Access Key ID [****************PTX4]: AKIAXYWPOO6TSKA6PTX4
AWS Secret Access Key [****************+a/3]: oEMRR3P29wH9Ktiogiwt+ZK2c3533tIY4c
em+a/3
Default region name [us-east-1]: us-east-1
Default output format [None]: json
```

Enter keys from saved csv file of the user we created in the last step

d. Launch mlflow server
Mlflow server --backend-store-uri=sqlite:///mlflow.db --default-artifact-root=s3://mlflow-models-akshit/

```
(base) ubuntu@ip-172-31-12-129:~$ mkdir mlflow
(base) ubuntu@ip-172-31-12-129:~$ cd mlflow/
(base) ubuntu@ip-172-31-12-129:~/mlflow$ mlflow server --backend-store-uri=sqlit
e:///mlflow.db --default-artifact-root=s3://mlflow-models-akshit/
```

This will show "listening on "http://127.0.0.1:5000/" but I won't be able to access it on local browser unless I **port forward 5000.**

Port	Local Address	Running Process	Origin
5000	localhost:5000		User Forwarded
8888	localhost:8888		User Forwarded

After this, my mlflow UI will open on local browser

e. Run the random-forest.ipyb script to push artifact to s3 bucket

```
import mlflow

mlflow.set_tracking_uri("http://127.0.0.1:5000")
mlflow.set_experiment("green-taxi-duration")
```

✓ 1.6s

```
2022/06/27 09:58:15 INFO mlflow.tracking.fluent: Experiment with name 'green-taxi-duration' does not exist. Creating a new experiment.

<Experiment: artifact_location='s3://mlflow-models-akshit/1', experiment_id='1', lifecycle_stage='active', name='green-taxi-duration', tags={}>
```

MIGHT BE USEFUL - ERROR SOLVING
https://github.com/mlflow/mlflow/issues/2150

BATCH DEPLOYMENT - CRON JOBS USING PREFECT

https://github.com/BPrasad123/MLOps_Zoomcamp/tree/main/Week4

Typical approach for deploying a model in batch model:

- Create a notebook/training script to train a model and save it
- Create a notebook to load the trained model and make prediction on the new data
- Convert the notebook to an inference script
- Clean and parameterize the script
- Schedule the inference script if required

Step 1: Train the model and save the artifacts

Connect to EC2 server and create a new virtual environment
Train the RF model

Step 2: Notebook for fetching the model and predicting

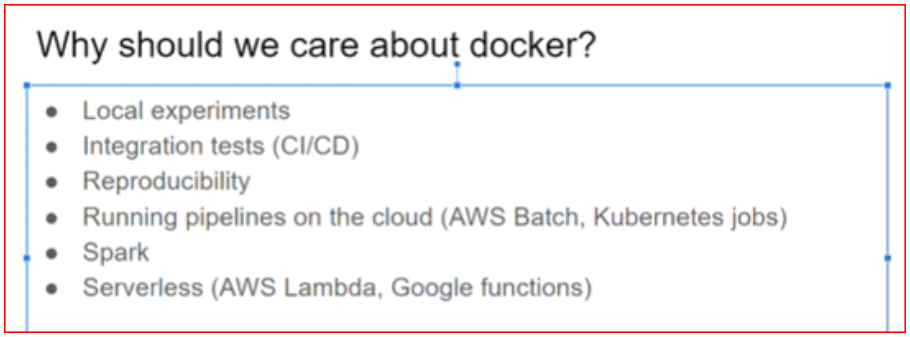
Once the notebook code successfully runs convert that to python script.
Clean and parameterize the prediction script score.py

Step 3: Dockerize the script
Make a dockerfile and package the script in that. In the script, we should have the model output predictions saved to the S3 bucket. Once we run the docker file, it will update the prefect workflow and save the output to S3 bucket

Docker

Thursday, June 16, 2022 11:36 PM

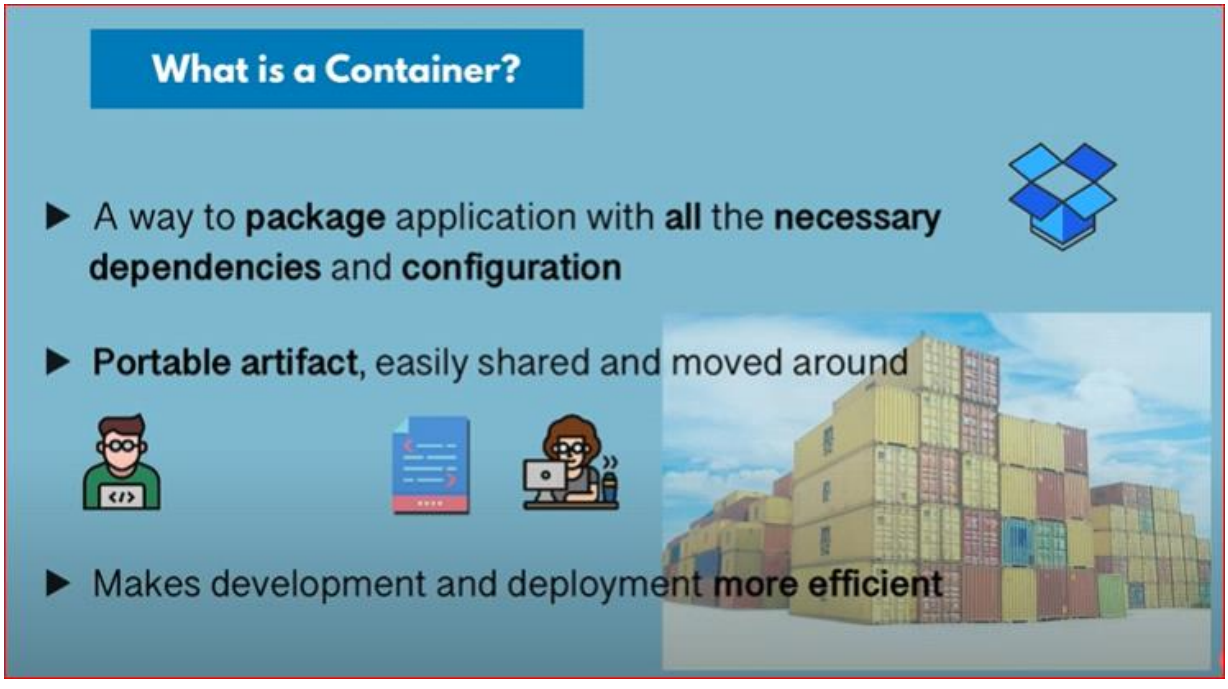
DOCKER



Installing Docker on Windows:

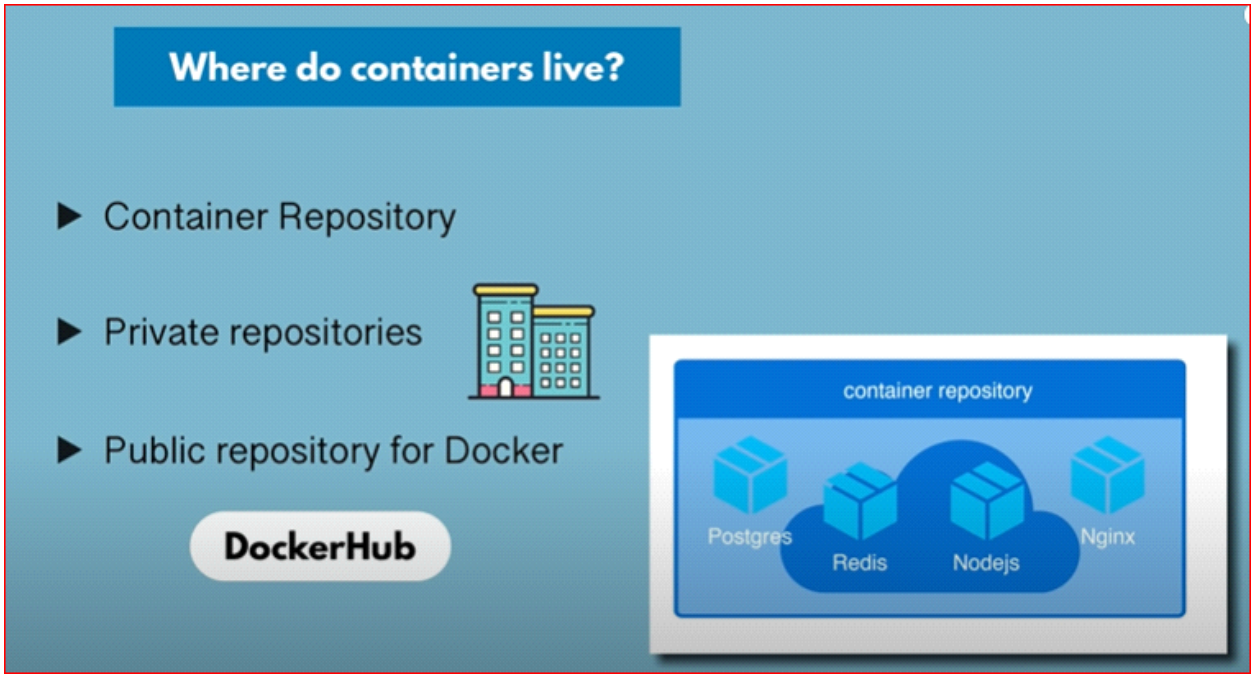
1. Check system requirements (**winver** on command prompt). Should be windows home 21G(or more) or windows pro. Docker only natively runs on windows 10
2. Go to task manager -> performance to see if virtualization is enabled
3. Download Docker Desktop App: https://docs.docker.com/desktop/windows/install/
4. Explicitly start docker by searching on windows

CONTAINER



Now since container is portable(we can share them and move them around), it must be stored somewhere/have a storage?

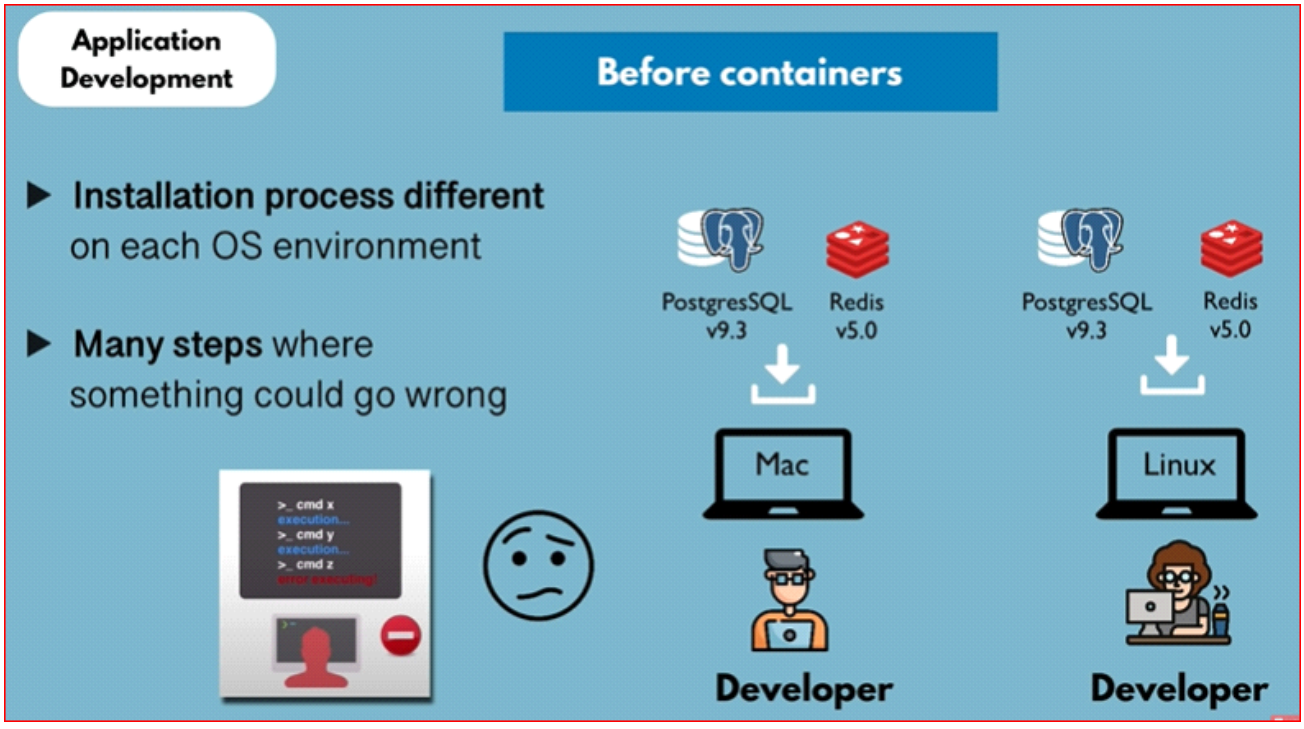
Containers live in **Container Repository.** Many companies have their own private repositories where they host/store a container.



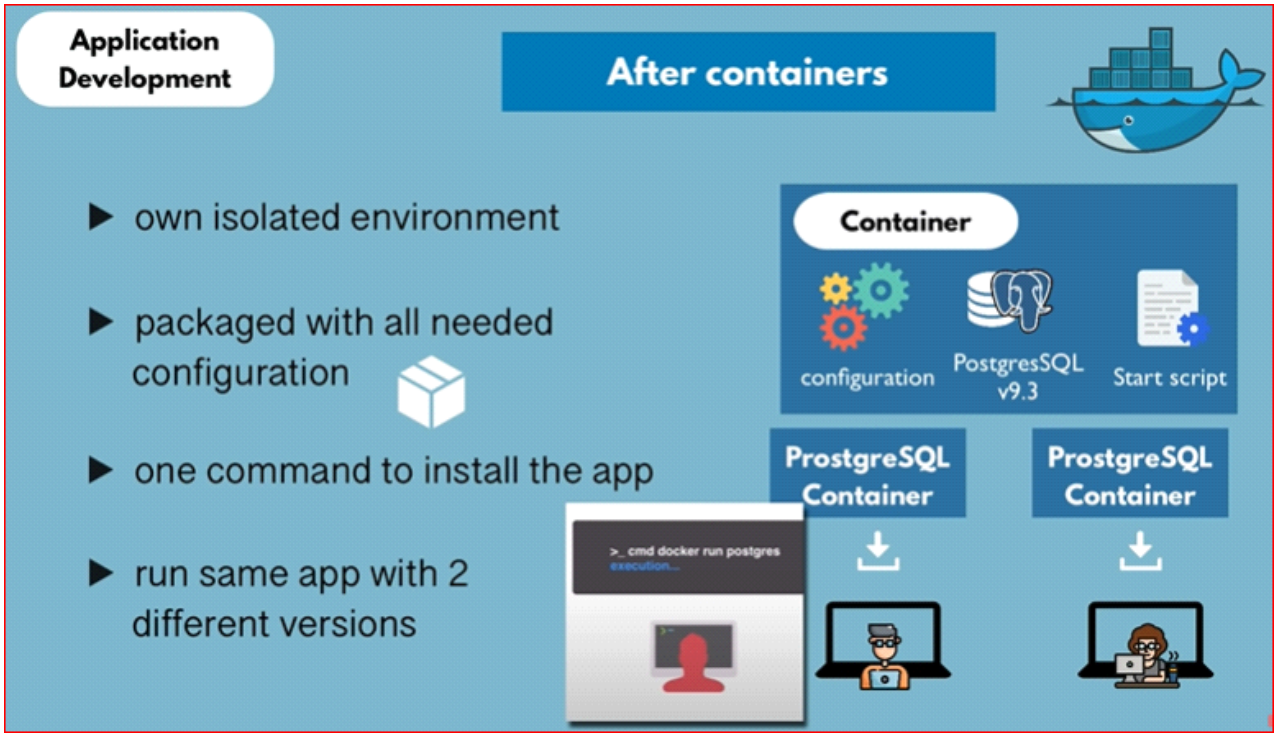
On DockerHub, we have images of so many companies/software(Jenkins etc)

How Dockers improve our Software Development process?

Before containers: There are multiple developers on a team and they'd have to install all the services on each of their systems and then try it out on their local environment(different OS). There are so many steps to install services and there's a possibility that it could go wrong somewhere. If we have 10 services, then we have to do this 10 times of each machine.

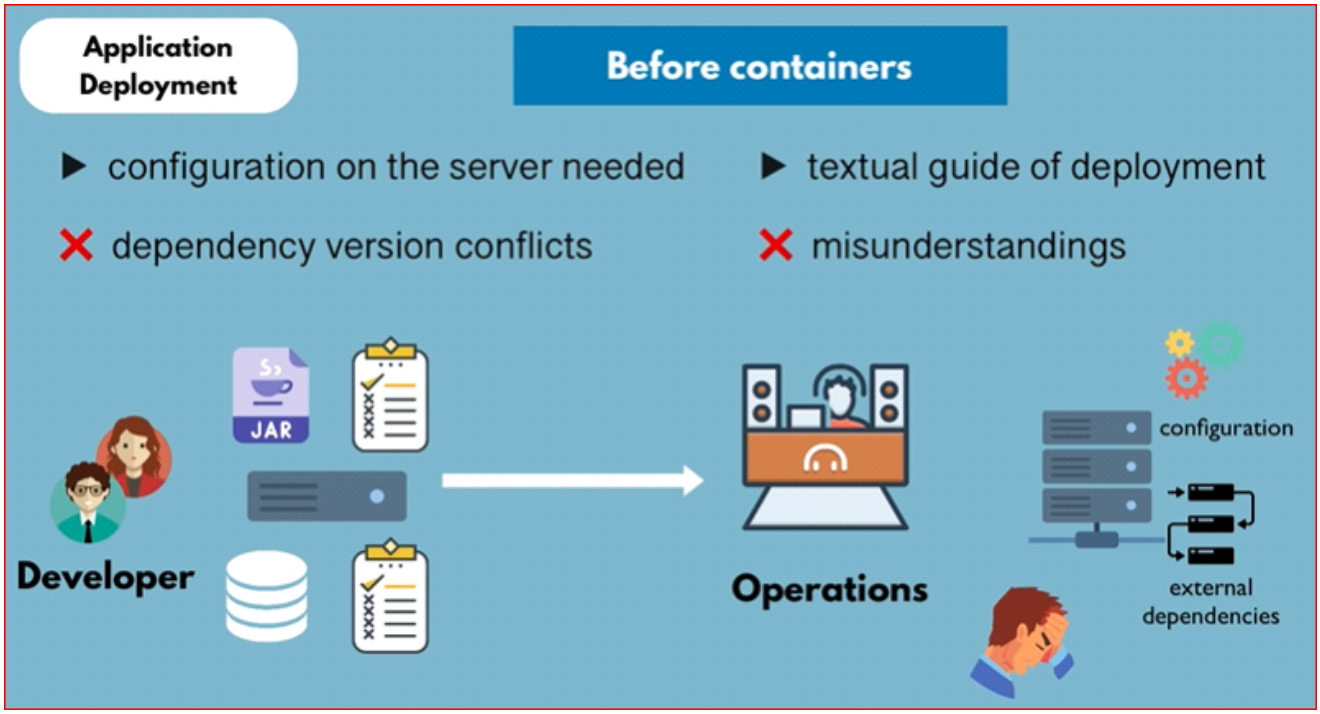


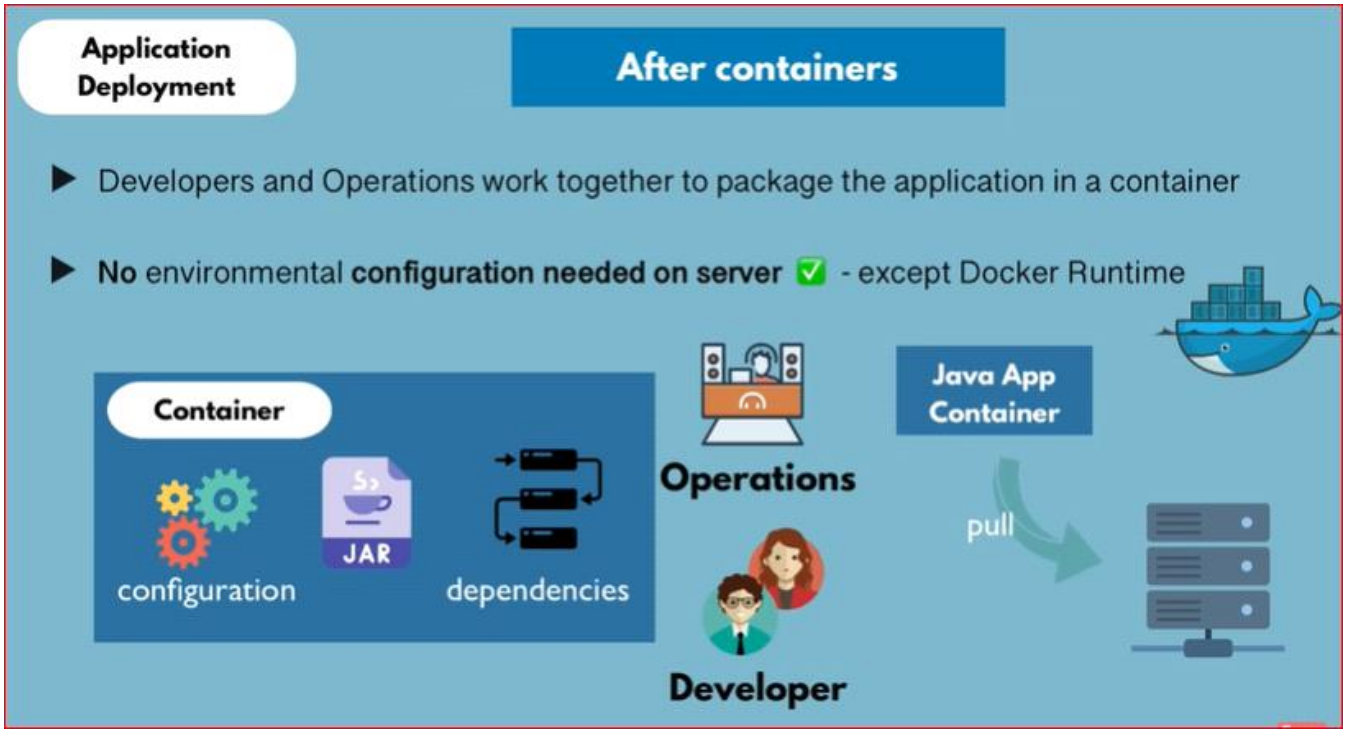
After Containers:



Docker command to install the container(one line) is same on all OS. This helps setting up our local environment easier, efficient and synchronous with all developers.

How Dockers improve our Deployment process?



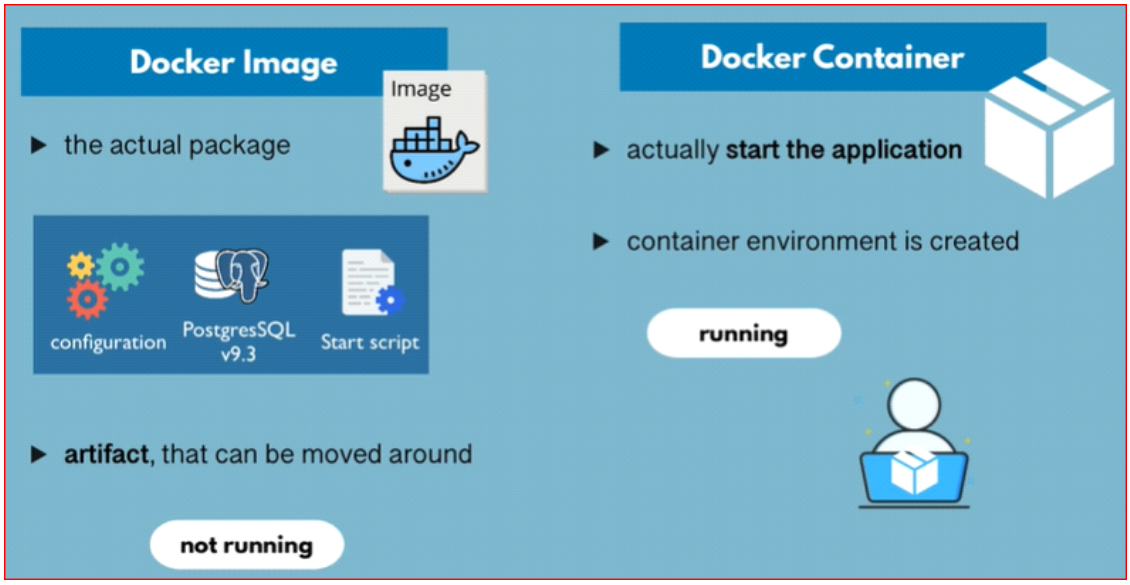


Operations and Developer work on the same team. No environmental configurations needed on the server.

Pulling containers from public repositories(docker hub)

1. Got to postgres public container and select a version (https://hub.docker.com/_/postgres)
2. **Docker run postgres:9.6** *(it first pulls the container and then starts it)*
3. The first line will say "unable to find locally" so it will go search on dockerhub for the postgres image
4. Docker containers are made up of layers(linux layer, X layer, Y layer etc..) so when we see "downloading" these are all layers/different images getting downloaded from dockerhub
5. **Docker ps :** check running containers. When we do docker ps, it shows postgres as an image

Container is when we pull the image on our local machine and start it. If we run the postgres, it's a container. When we don't run it, it's an image.



Advantage of using Layers

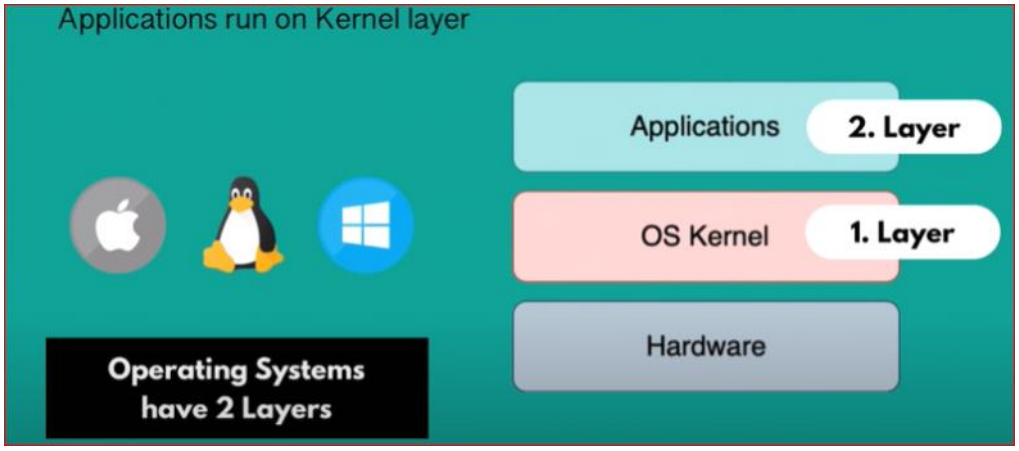
If we later run different version of postgres, the existing layers of the image(the ones downloaded with 9.6 version) won't download again. That saves us some time

```
Nanas-MBP:~ nanabiz$ docker run postgres:10.10
Unable to find image 'postgres:10.10' locally
10.10: Pulling from library/postgres
8f91359f1fff: Already exists
c6115f5efcde: Already exists
28a9c19d8188: Already exists
2da4beb7be31: Already exists
fb9ca792da89: Already exists
cedc20991511: Already exists
b866c2f2559e: Already exists
5d459cf6645c: Already exists
6de9d066d892: Downloading [==================>
401fcd8e29c4: Download complete
9b130e26214a: Download complete
1c048e77610c: Download complete
431b5e6c27b3: Download complete
4eca80d7c24a: Download complete
```

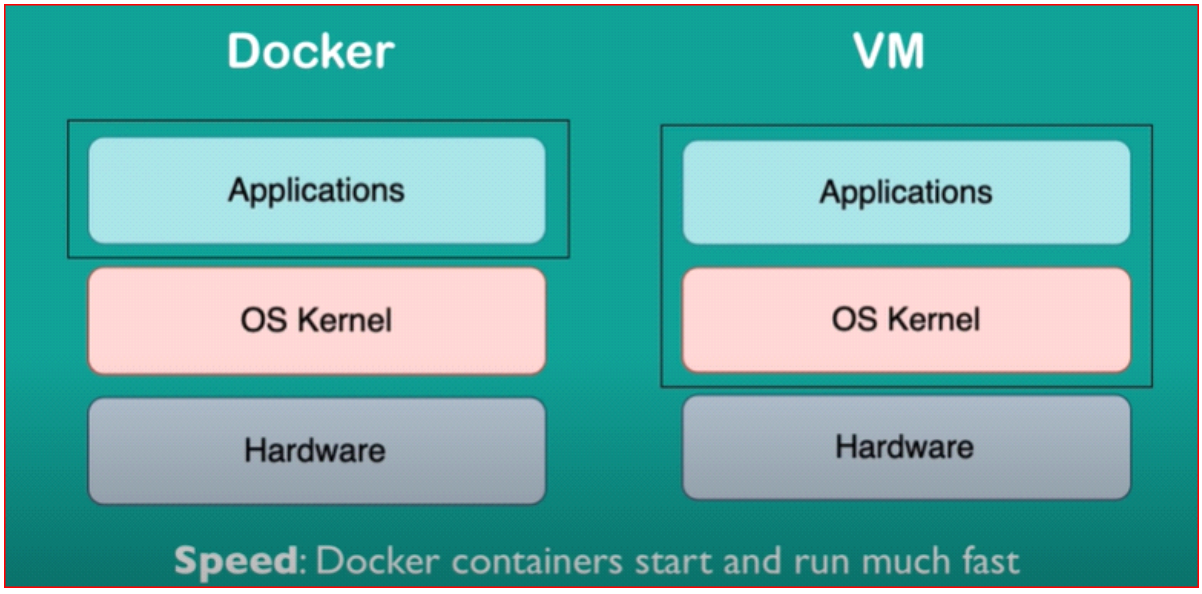
some layers already exist ✅

How is Docker different from Virtual Machine?

How OS works: Kernel talks with Hardware. Application runs on Kernel layer.
For example: Linux has so many distributions(Ubuntu, Debian, CentOS) and all look different **but they have all the same OS Kernel: Linux**

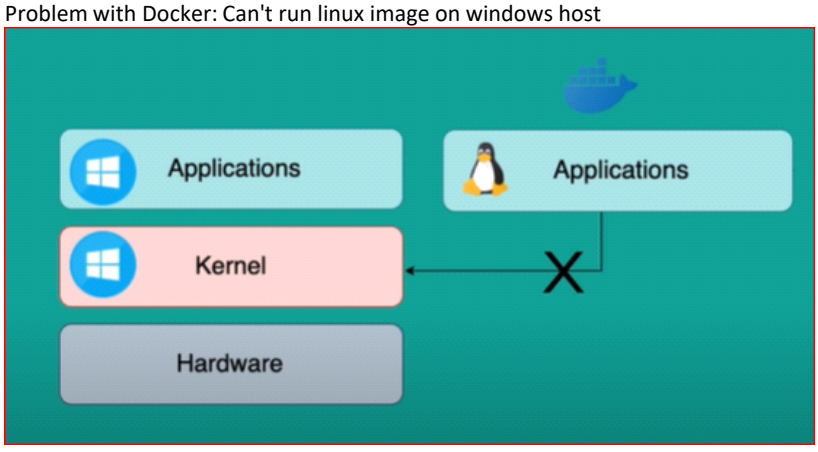


Both are virtualization tools.. The difference is in what part they virtualize. VM has its own kernel so its virtualizes the entire system. When we download VM image on our system. It doesn't use our host. It uses its own system.



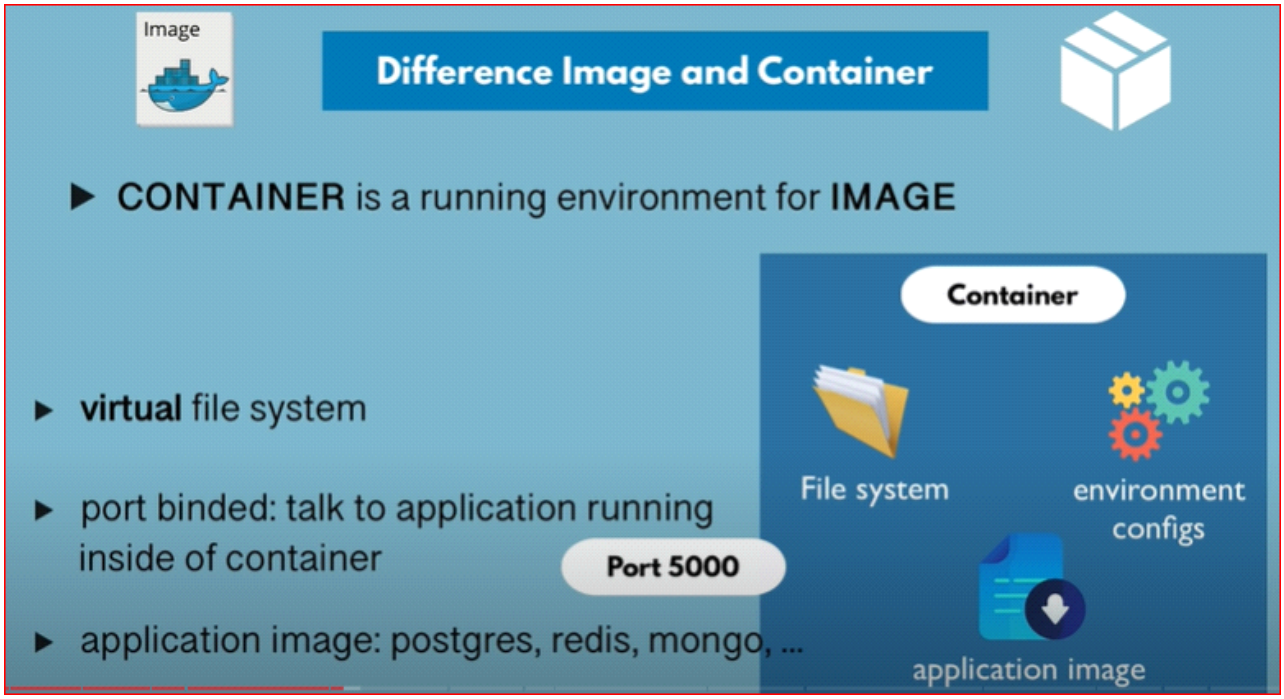
Difference

1. Size: **(Docker wins)** They are just a few MBs in size but VM images are GBs in size.
2. Speed: **(Docker wins)** They run quicker as VM has to load a totally different OS
3. Compatibility: **(VM wins)** VM of any OS can run on any OS host. Can't do that with docker
   Problem with Docker: Can't run linux image on windows host



Solution to this: Install Docker ToolBox which takes care of this

BASIC DOCKER COMMANDS



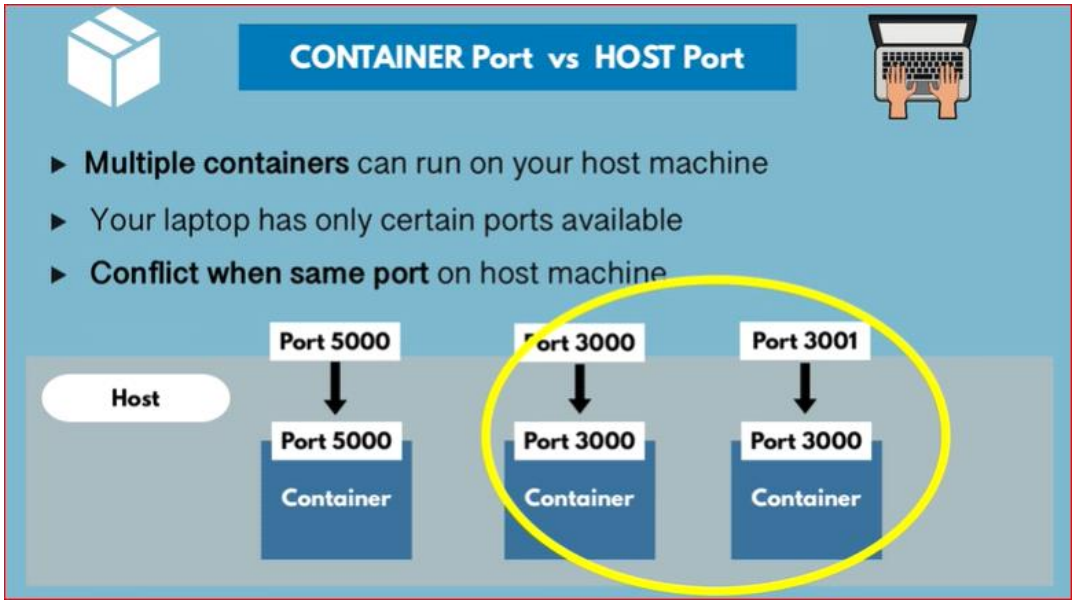
Containers provides the environment for the image to run, such as virtual file system to store logs/other stuff and env configs to set things and port to run it on **(listen to incoming requests)**.

Docker run = Docker pull + Docker Start

1. Docker pull redis
   This downloads the image. Redis is not running. Only the image is downloaded.
   Now we need reddis running so our application can connect to it
2. Docker run redis
   This will start the image in the container. Container is the running version of image
3. Docker ps
   Check running containers
4. Docker run -d redis
   Run container in a detached mode: basically it lets us use the same terminal. Otherwise we have to start new terminal as a container starts on the existing one
5. Docker stop <docker id>
   Docker start <docker id>
   Restart container
6. Docker ps -a
   Lists all running **and stopped** containers

What if we have two different version of reddit running on two different containers?
Catch here: Both will have the same **container port**.

How it works?
Our laptop has certain ports and we have to bind them to container's ports to run the containers.



However, here the container port might be same but our host port is different.

Mentioned situation:

```
[~]$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
cfec85d7bbec        redis:4.0           "docker-entrypoint..."   12 seconds ago      Up 14 seconds       6379/tcp            pedantic_raman
8381867e8242        redis               "docker-entrypoint..."   16 minutes ago      Up 9 minutes        6379/tcp            heuristic_newton
[~]$
```

Same container ports. However, here we have not yet bind the port to any of our host(laptop) port

Solution to run two different version of redis: Choose different host ports
Docker -p<Host port> : <Container port> <image name>
Docker run -p6000:6379 redis (first container - first version of redis)
Docker run -p6001:6379 redis:4.0 (second container - second version of redis)

Both of them are bound to different ports on our laptop.

Difference between docker start and docker run

Docker run: create new container
Docker start: restart a stopped container

Docker run starts a new containers by mentioning image name. We tell docker at the beginning what kind of container with what attributes we want.

Once the container is created and let's say we stop it. We can start is using **docker start** and it will retain all of its properties mentioned at time of **docker run**

DEBUGGING CONTAINERS

Docker logs <container ID>

What if we don't want to remember container IDs and have our own docker name(instead of a default name) instead to remember it by.

Docker run -d -p6001:6379 **--name redis-older** redis:4.0

```
CONTAINER ID        IMAGE               COMMAND                  CREATED                  STATUS              PORTS                    NAMES
5b7d84a59a7c        redis:4.0           "docker-entrypoint..."   Less than a second ago   Up 2 seconds        0.0.0.0:6001->6379/tcp   redis-older
d7a6ef66e6da        redis               "docker-entrypoint..."   15 minutes ago           Up 15 minutes       0.0.0.0:6000->6379/tcp   heuristic_ardinghelli
```

Docker logs redis-older (much more comfortable to view logs)

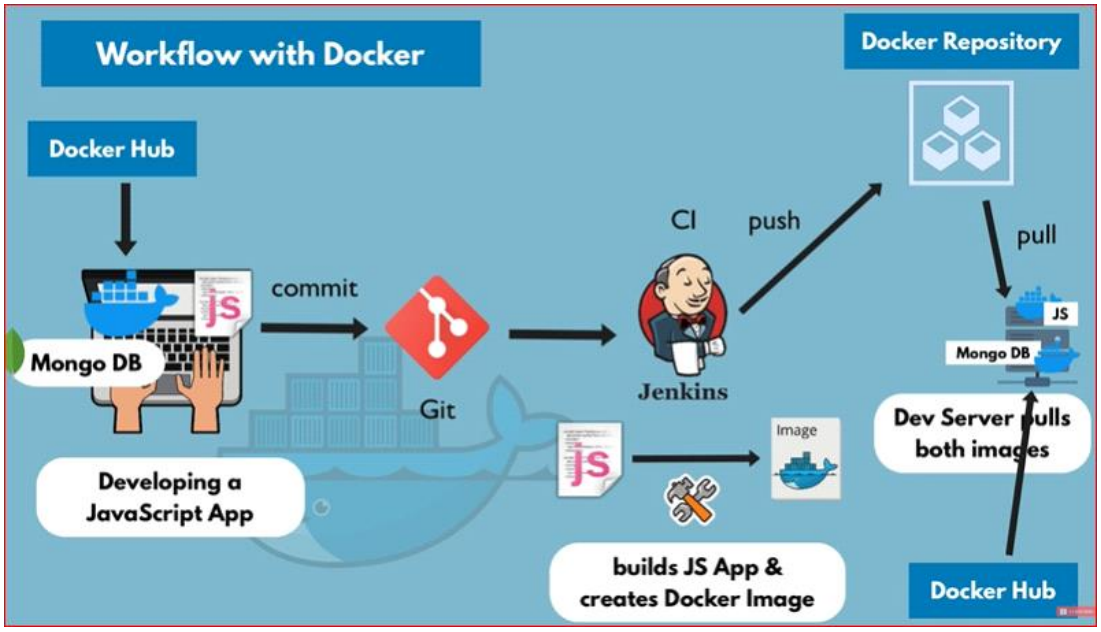
Getting inside container(to view files/logs/env variables/folders)

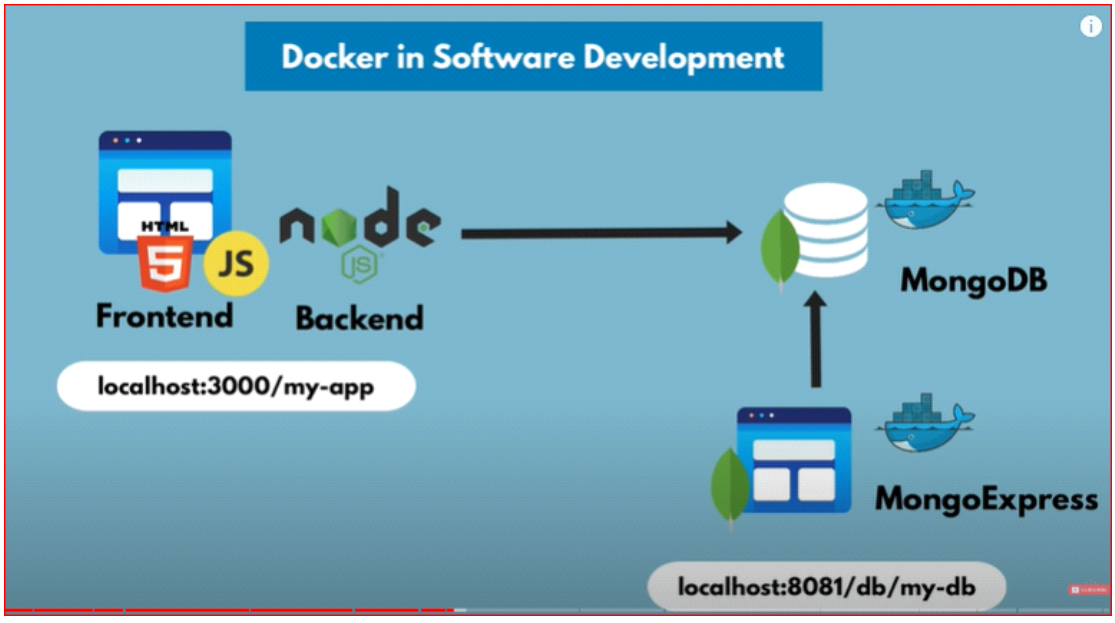
Docker exec -it <container ID> /bin/bash

```
[~]$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED              STATUS              PORTS                    NAMES
cae903a74202        redis               "docker-entrypoint..."   About a minute ago   Up About a minute   0.0.0.0:6000->6379/tcp   redis-lates
5b7d84a59a7c        redis:4.0           "docker-entrypoint..."   About a minute ago   Up About a minute   0.0.0.0:6001->6379/tcp   redis-older
[~]$ docker exec -it cae903a74202 /bin/bash
root@cae903a74202:/data# ls
root@cae903a74202:/data# pwd
/data
root@cae903a74202:/data# cd /
root@cae903a74202:/# ls
bin  boot  data  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@cae903a74202:/# env
no_proxy=*.local, 169.254/16
HOSTNAME=cae903a74202
REDIS_DOWNLOAD_SHA=6624841267e142c5d5d5be292d705f8fb6070677687c5aad1645421a936d22b3
PWD=/
HOME=/root
REDIS_VERSION=5.0.6
GOSU_VERSION=1.11
TERM=xterm
REDIS_DOWNLOAD_URL=http://download.redis.io/releases/redis-5.0.6.tar.gz
SHLVL=1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
_=/usr/bin/env
OLDPWD=/data
root@cae903a74202:/# exit
exit
```

DOCKER IN PRACTICE

1. Our JS app uses MongoDB and instead of downloading MongoDB on our system, we just download a docker image of that.
2. Commit JS application to GIT which would trigger a **continuous integration** - Jenkins Build
3. (JS Artifact -> Docker Image) Jenkins build will produce artifacts from our JS application and create docker image of that.
4. Push the Jenkins build to a private Docker repository(usually companies have them)
5. Now the docker **JS image** created in step 3(present in Jenkins build) has to be deployed to a development server. The server will pull the **JS image** from docker repository and MongoDB from the DockerHub.



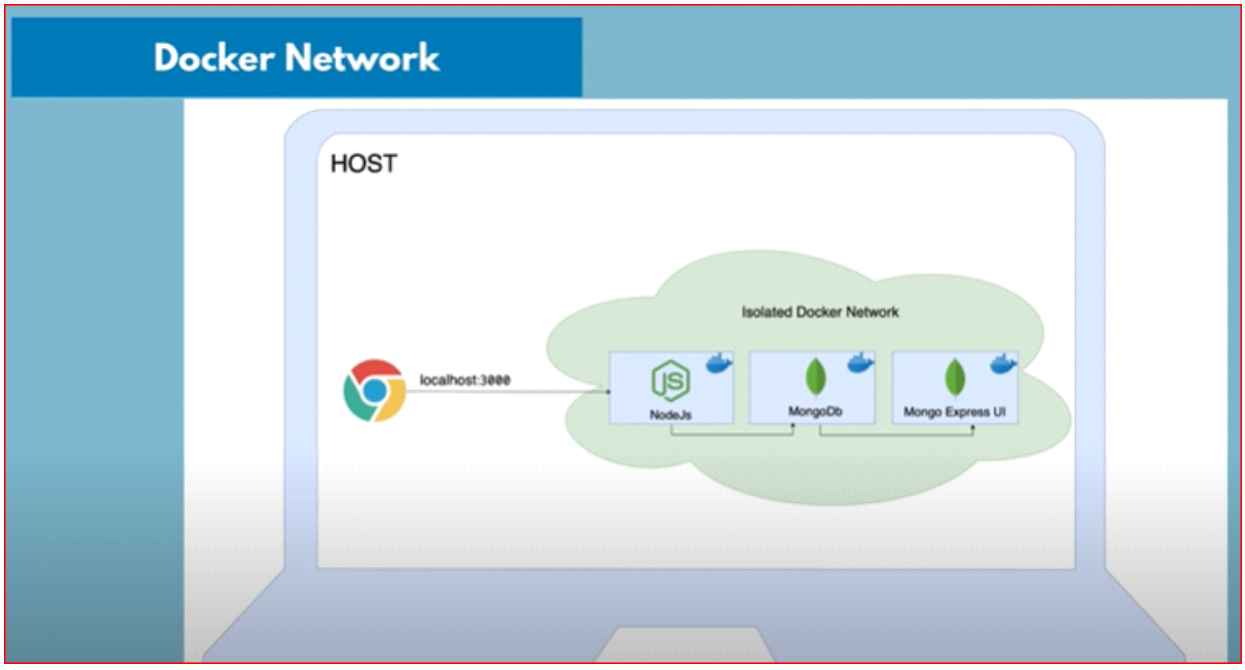


In practice:
JS: Front end
NodeJS: Back end
DB: MongoDB Docker, MongoExpress Docker(just a UI for commands - don't have to use terminal if we have UI). In this UI we'll be able to see all the updates to our application in the database much easier.

We need to connect the MongoExpress to MongoDB container **+** connect the DB to our JS app

Docker Network: Docker creates its isolated docker network where containers are running in

- If we have two container run on same docker network, they can connect with each other just by using container name - without any local host/port number etc.
- We can then connect our JS app to this network using the localhost and port number. When we package our application, our browser(which is not a part of docker network) can access the application using the same local host and port number which was used to connect Node JS to MongoDB Dockers.



.

Check docker networks: **docker network ls** (there are 4 types)
Create docker network: **Docker network create mongo-network**
Have mongodb and expressUI run in the above created network: We normally used to run "docker run mongo" but now we'll add some environmental variables and also specify the network we created above

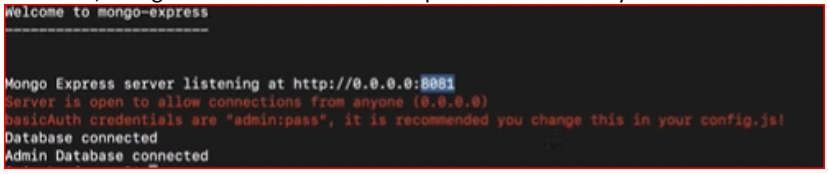
MONGO

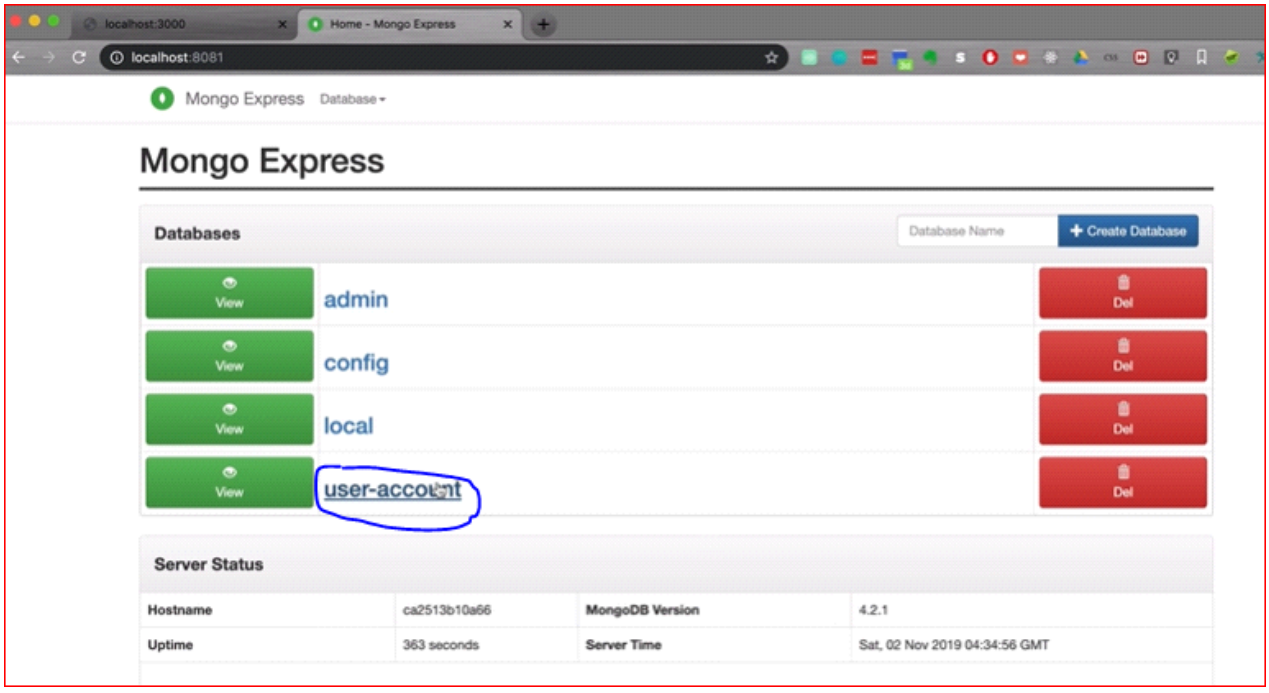
```
docker run -d \                                  #detach mode
-p 27017:27017 \                                 #opening port on the host
-e MONGO_INITB_ROOT_USERNAME=admin \             #setting up env variables/overriding default ones
-e MONGO_INITB_ROOT_USERNAME=password \ #setting up env variables/overriding default ones
--name mongodb \                                 #overwrite name of container
--network mongo-network \                        #specify mongo network on which the container will run
mongo                                            #image name - start the mongo container
```

MONGO EXPRESS

```
 docker run -d \
   -p 8081:8081 \
   -e ME_CONFIG_MONGODB_ADMINUSERNAME =admin
   -e ME_CONFIG_MONGODB_ADMINUSERNAME =password
   --name mongo-express \
   -net mongo-network \
   -p 8081:8081 \
   -p ME_CONFIG_MONGODB_SERVER= mongodb \ (container name)
   mongo-express
```

After this, mongo db UI will be available at port 8081 on local system.





We created a DB using UI and now we will connect that with our JS app.
Connecting to this DB(user-account) from NodeJS : Give URI(local host + port) of DB to NodeJS

```
[simple-js-app]$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                     NAMES
4130276c9ea0        mongo-express       "tini -- /docker-e..."   29 hours ago        Up 7 hours          0.0.0.0:8081->8081/tcp    mongo-express
ca2513b10a66        mongo               "docker-entrypoint..."   29 hours ago        Up 7 hours          0.0.0.0:27017->27017/tcp  mongodb
```

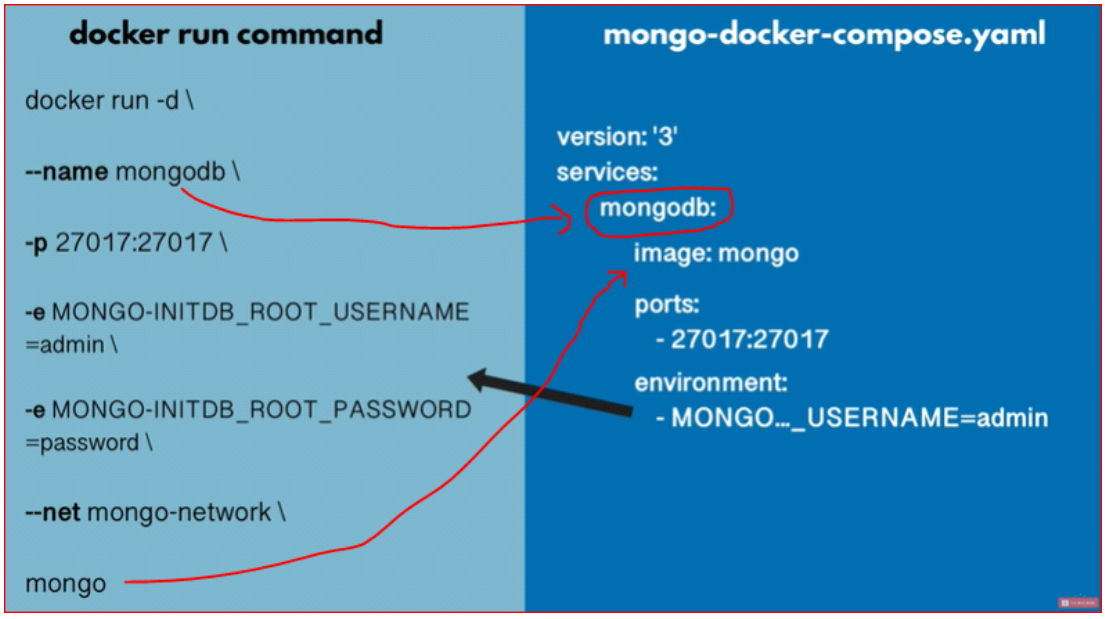
Connecting data base in Nodejs code
Var MongoClient = require('mongodb').MongoClient;
MongoClient.connect('mongodb://admin:password@localhost:**27017**", function(err, client){ get request from DB}

SUMMARY OF WHAT WE DID:

1. Created Fronend page on JS
2. Used MongoDB using docker and MongoDBExpress using Docker for UI
3. Created a DockerNetwork
4. **Run** both MongoDB and MongoDBExpress on it using a **common name** which would connect them both
5. Made a DB using MongoExpressUI (Accessed using localhost)
6. Connect that DB to NodeJS app in the script

A potential problem in this: Docker run command were so big to run everytime

Solution: **DOCKER COMPOSE: writing run commands in a structured way**

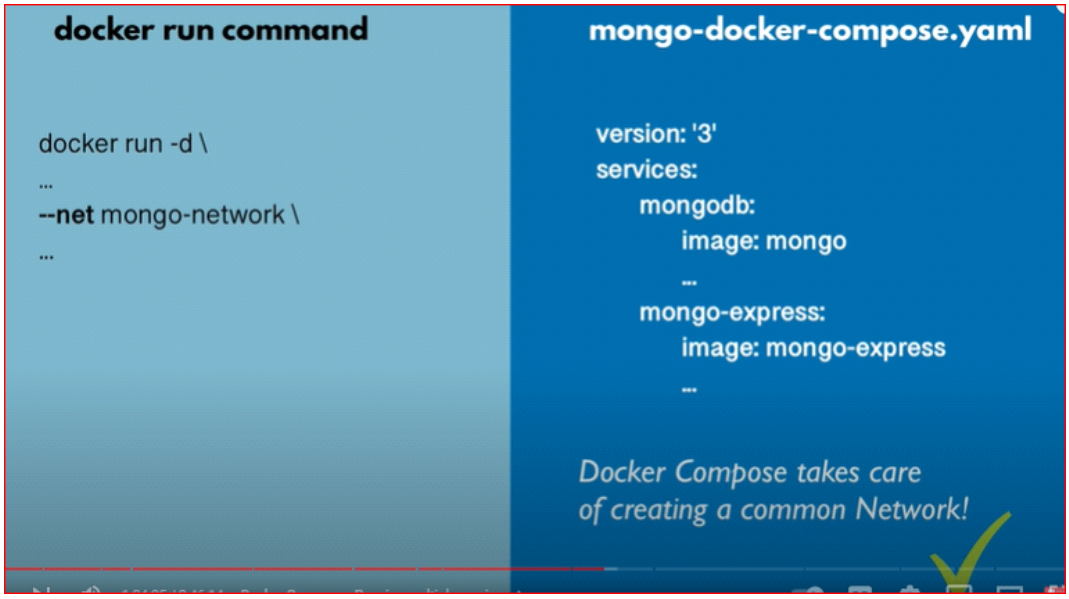


Docker compose is a structured way to contain very normal common run commands

- We can change code anytime(change port, add configs)

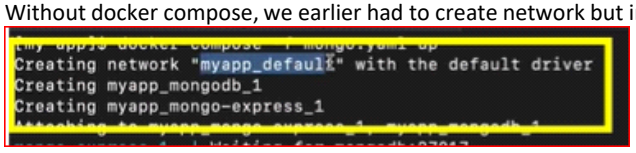
CATCH: We don't have to create a network and mention it in containers

Docker compose will take care of that



Docker-compose -f mongo.yaml up : Start the containers(listed in mongo.yaml):

Without docker compose, we earlier had to create network but in case of docker compose, it does that for us



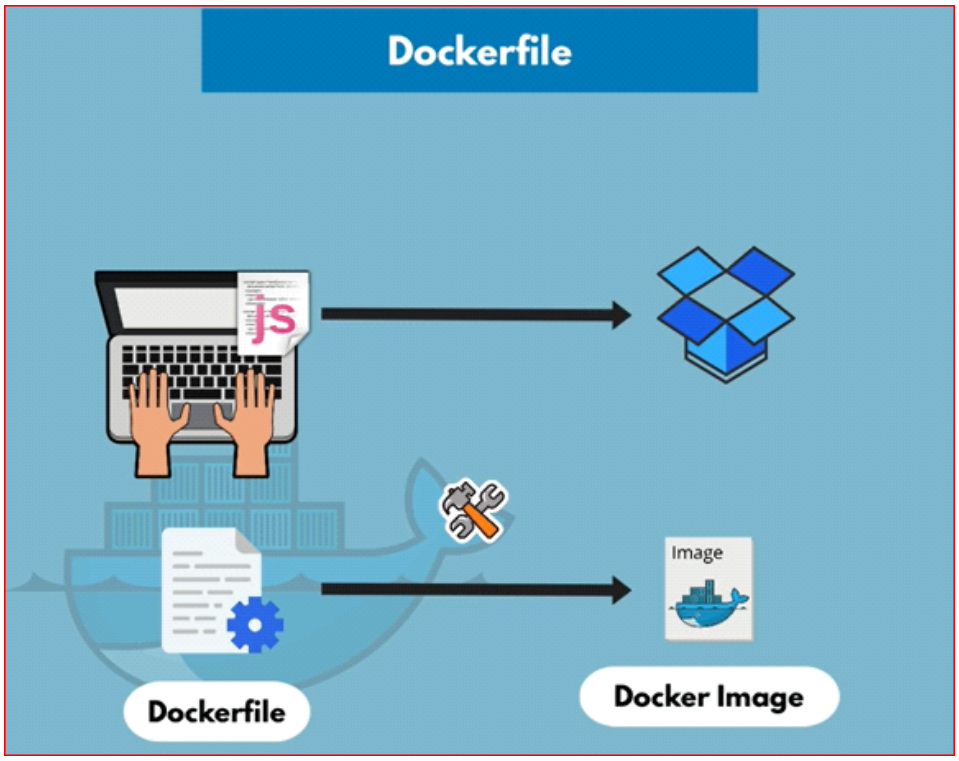
Myapp_mongodb_1 and myapp_mongo-express_1 are container names.

NOTE: When we restart a container, everything we configured in that container is gone(data is lost - **no data persistence**).

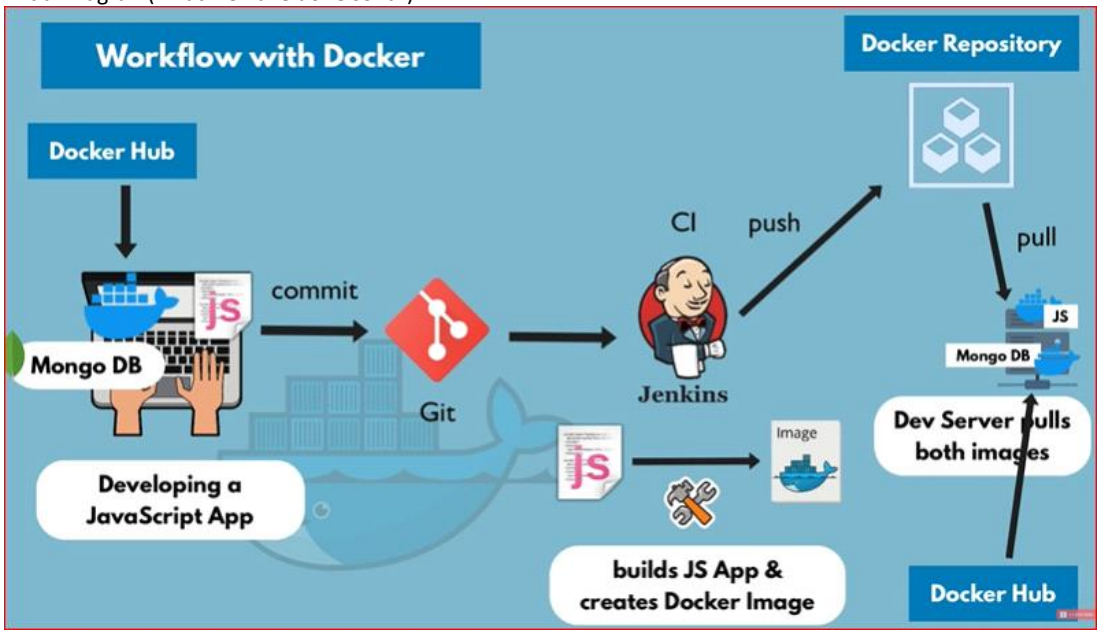
Docker-compose -f mongo.yaml down : Stopping containers(it stops the network as well)

DOCKER FILE (The name must be **Dockerfile**)

Docker file is a blue print for building images



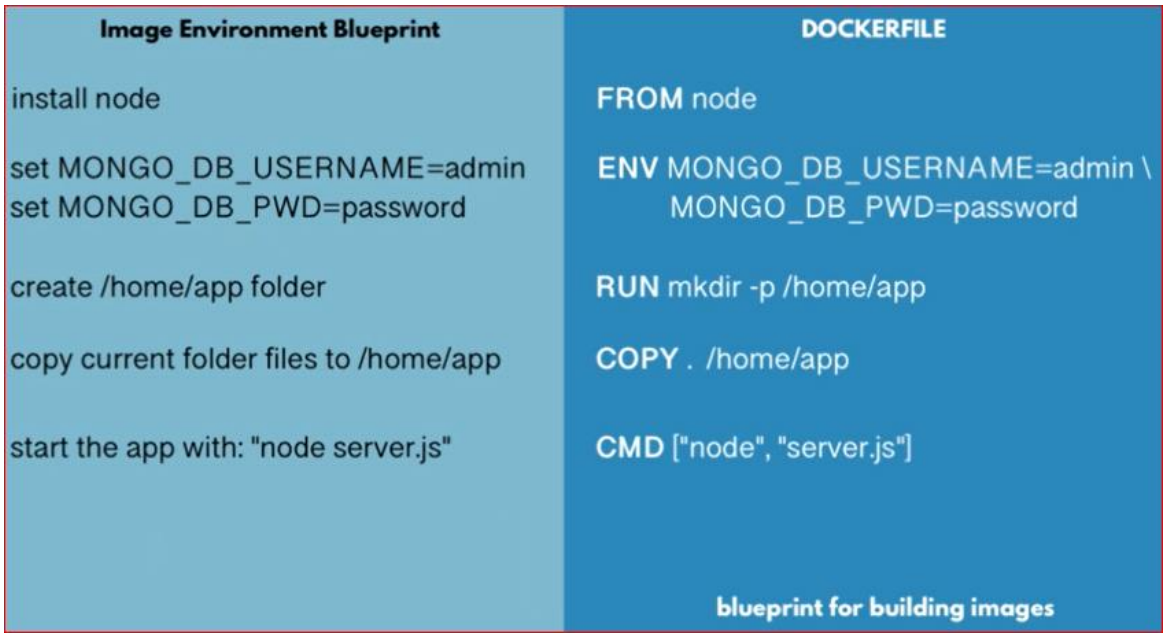
Initial Diagram(what we have done so far)



What we have done so far is use mongoDB from docker and create a JS app. Now what would Jenkins do? It builds the JS app and pushes it to our company's docker repository.

We are going to simulate Jenkin's behavior and build a docker image.

BLUE PRINT



All the commands will apply to the docker container, none of them will affect the laptop/host environment

FROM node *#We always have to base it on another image - node in our case*
ENV MONGO_DB_USERNAME = admin \
MONGO_DB_PWD = password *#Optionally set environmental variables inside of image environment. Better to do it in docker compose step as if anything changes, it's better to change it while we run it than when we build the app*
RUN mkdir -p /home/app *#Using **run** we can execute any Linux command. Our command makes a directory. NOTE: this directory will be made inside the container, not on laptop/host*
COPY . /home/app *#Now you might ask why can't we just use copy command of linux with the run command. Because the **COPY** command of docker file executes on the host machine. **COPY SOURCE(local files) TARGET(inside the container)***
CMD ["node", "/home/app/server.js"] *#starts the app with the command "node /home/app/server.js". What is the difference between RUN and CMD? CMD is an entry point command - JUST ONE. We can have multiple RUN commands.*

Building Docker Image
Docker build -t myapp:1.0 . (docker build -t app_name:version path_to_docker_file(dot means current directory))
THIS ABOVE STEP IS WHAT JENKINS WOULD DO(Build docker image from dockerfile)

Run Docker Image
Docker run myapp

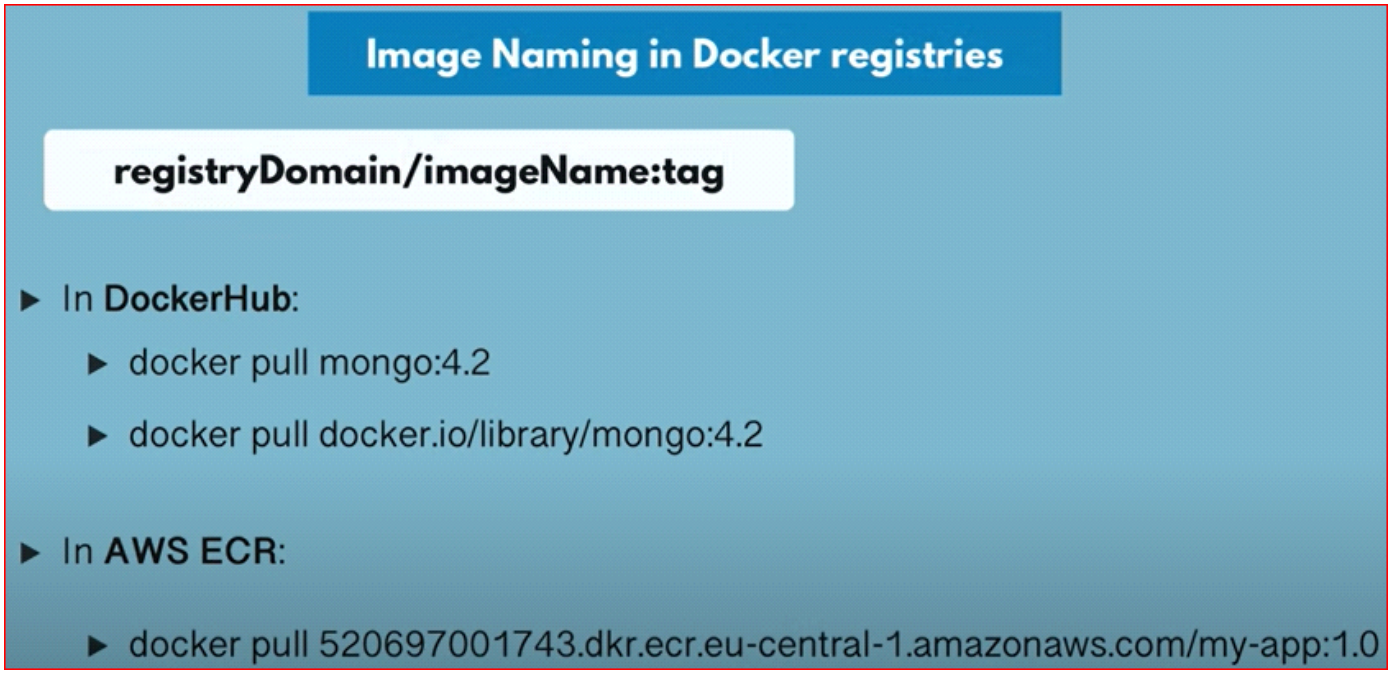
How to get inside the container and see what's happening?

Docker exec -it docker_id /bin/sh *#-it is for interactive terminal*
Exit to leave terminal

IF WE DECIDE TO CHANGE ANYTHING IN DOCKERFILE, WE NEED TO REBUILD THE IMAGE

1. Docker ps (see the container_id of our app)
2. Docker stop container_id (Stop the container)
3. Docker rm container_id (Remove the container)
4. Docker images (see image ID)
5. Docker rmi image_id (Remove image)
6. Docker build -t myapp:1.0 (Build again)
7. Docker run myapp:1.0 (Run again)

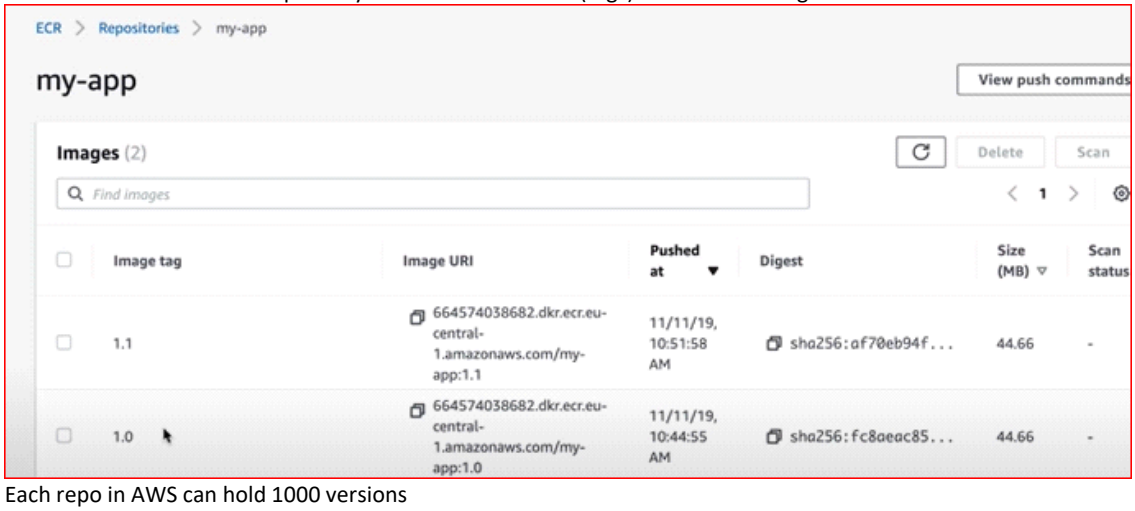
WE BUILT A DOCKER IMAGE. TIME TO PUSH IT TO DOCKER REPOSITORY



Usually this is how image name works but when we do it from docker hub, we can avoid writing docker.io/library/, it's a special case

NOTE: AWS WILL PROVIDE US THE COMMANDS ON THEIR UI UNDER "AWS PUSH"

1. Create a private docker repository on AWS - also called a **docker registry**
   Use ECR(elastic container registry) service on AWS
   Create a repository
   NOTE: HOW ECR WORKS
   For each image, we create a separate repository
   What we store inside that repository are different versions(tags) of the same image

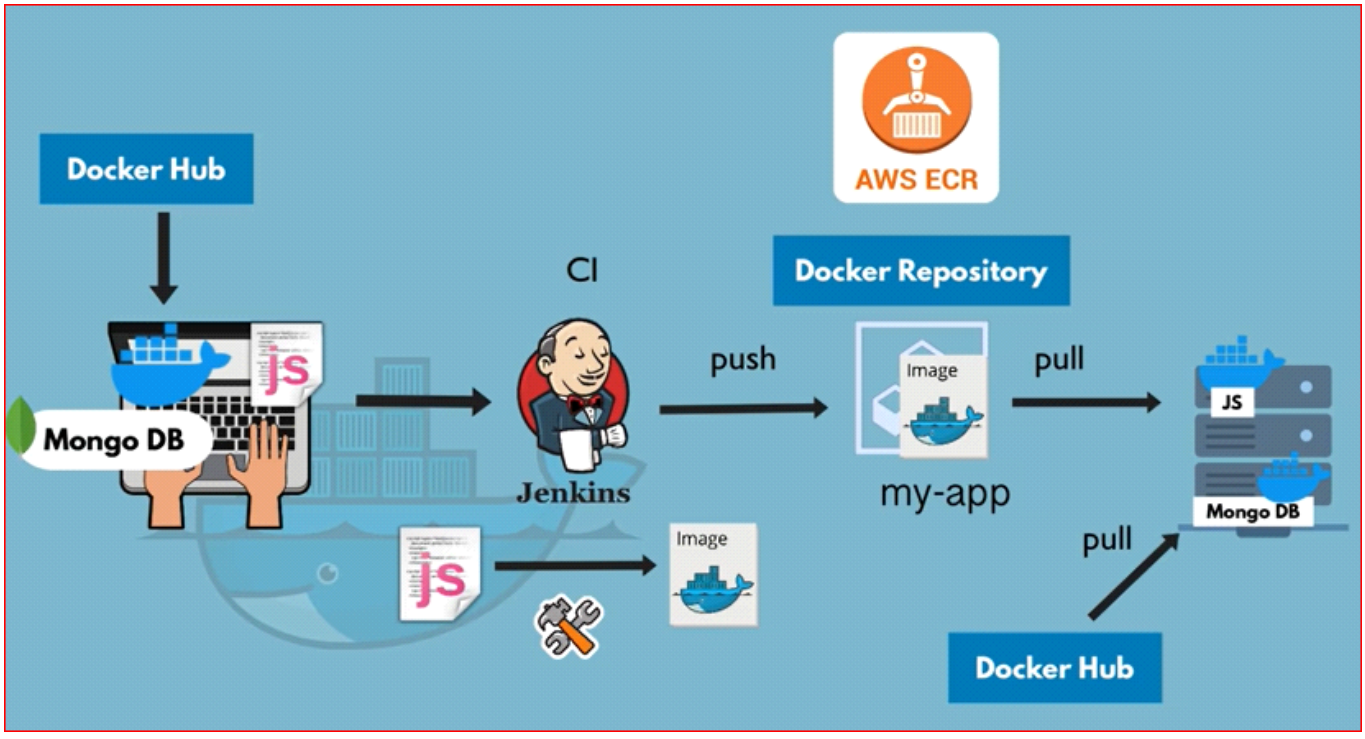


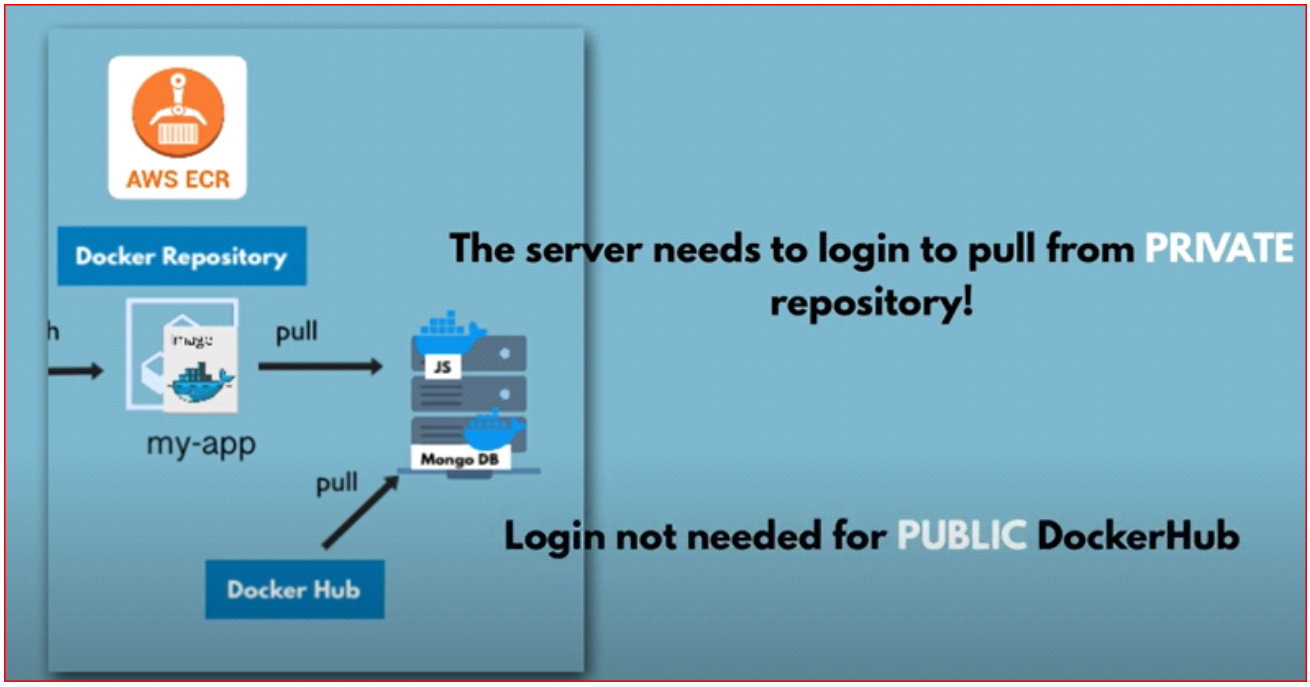
   Each repo in AWS can hold 1000 versions
2. Push image into repository
   a. Docker login
      Pre-req: install **awscli** and **have credentials configured**
      (aws ecr get-login --no-include-email --region eu-central-1)
   b. Tag my image (rename the image and makes a copy to include the aws name in our tag, we'll see how this helps in next step)
      Docker tag my-app:latest registryname/my-app:latest

```
[~]$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
my-app              1.0                 83fdc778a892        2 days ago          116 MB
node                13-alpine           f20a6d8b6721        3 days ago          105 MB
mongo               latest              965553e202a4        10 days ago         363 MB
mongo-express       latest              597a2912329c        13 days ago         97.6 MB
redis               4.0                 e187e861db44        3 weeks ago         89.2 MB
redis               latest              de25a81a5a0b        3 weeks ago         98.2 MB
postgres            9.6.5               bd287e105bc1        2 years ago         266 MB
[~]$ docker tag my-app:1.0 664574038682.dkr.ecr.eu-central-1.amazonaws.com/my-app:1.0
[~]$ docker images
REPOSITORY                                                  TAG                 IMAGE ID            CREATED
664574038682.dkr.ecr.eu-central-1.amazonaws.com/my-app      1.0                 83fdc778a892        2 days ago
my-app                                                      1.0                 83fdc778a892        2 days ago
node                                                        13-alpine           f20a6d8b6721        3 days ago
mongo                                                       latest              965553e202a4        10 days ago
mongo-express                                               latest              597a2912329c        13 days ago
redis                                                       4.0                 e187e861db44        3 weeks ago
redis                                                       latest              de25a81a5a0b        3 weeks ago
postgres                                                    9.6.5               bd287e105bc1        2 years ago
```

c. Push the image to private docker repository
If we do just docker push myapp:1.0, it will try to push it to dockerhub(by default) and it won't work. We need to tell it to push to the AWS private repo. This is where the tag we created helps us
Docker push registryname/my-app:latest

OK NOW WE GOT THE DOCKER IMAGE IN AWS PRIVATE REPO, WE NEED TO **DEPLOY THE CONTAINERIZED APP**

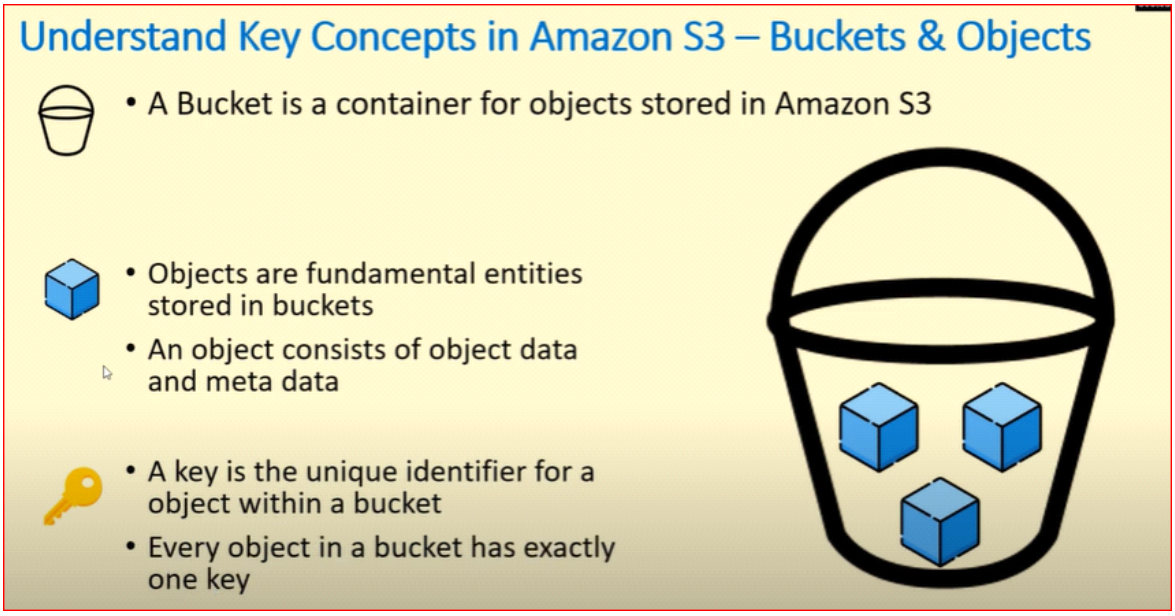



AWS ECR
Docker Repository
pull
my-app
JS
Mongo DB
pull
Docker Hub
The server needs to login to pull from PRIVATE repository!
Login not needed for PUBLIC DockerHub

AWS

Monday, June 27, 2022 5:52 PM

S3 Bucket



FLASK

Friday, July 1, 2022 2:26 PM

Overview

Flask is a web application framework written in Python
Flask Overview : **https://hackersandslackers.com/flask-routes/**

Flask's "Hello world!"

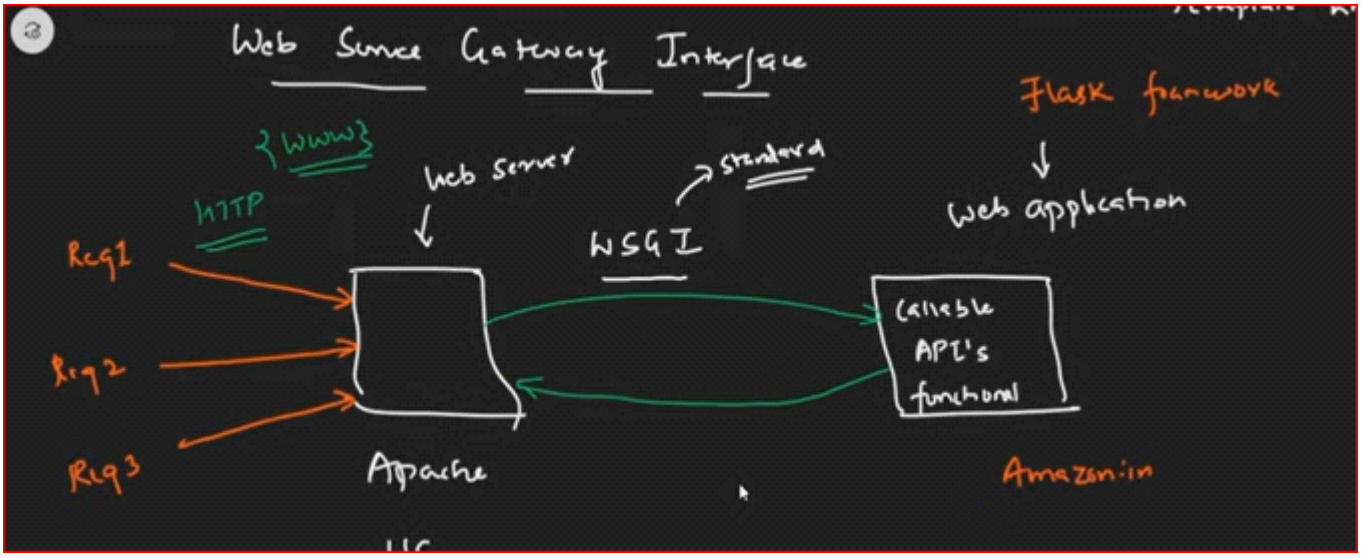
```
from flask import Flask
app = Flask(__name__)


@app.route("/")
def home():
    return "Hello World!"
```

Short Explanation:
Whenever we trigger the page inside @app.route(**"/"**), the function below it will get triggered.

Long Explanation:
Running Flask means, basically, that you are running a webserver, also known as an HTTP server. The Flask server will respond if a request is made for the URL defined in the route. This happens either when you visit the URL in your browser, or when some other HTTP client tries to access that URL.

The argument to app.route() is the *path* component of the URL. If you run a Flask server, the hostname of the machine it runs on will be the *host* component. The port it listens on will be the *port*. The *scheme* will be http. So if you ran the above example on a machine located at 52.12.34.56, listening on port 8088, you could reach the endpoint by going to http://52.12.34.56:8088/. You could also add a query-string or a fragment-id - the latter would be ignored, the former would be seen by the Flask server. So you could change the Python code in the function hello_world so that it returns different output depending on the query-string.

If you were using Flask in a formal production setting, you could have other servers doing things like proxying or load balancing involved in your setup. So potentially your users might visit a URL with a hostname which points to your load balancer, and that would pass on the request to Flask, possibly changing the URL in various ways. At the moment, you shouldn't worry about that. Just try and see if you can run a server and load the right page to see the text 'Hello World!' in your browser.




```python
app = Flask('churn')

@app.route('/predict', methods=['POST'])
def predict():
    customer = request.get_json()

    X = dv.transform([customer])
    y_pred = model.predict_proba(X)[0, 1]
    churn = y_pred >= 0.5

    result = {
        'churn_probability': y_pred,
        'churn': churn
    }

    return jsonify(result)
```

How this function translates: We have a flask app **churn** which will run on localhost_url/**predict** and the below function will act when there is a **POST** request passed on the predict url. So this function will basically handle the POST request.
In the POST request, we need to supply customer information in JSON format which this function will store in **Customer = request.get_json**() and then return the result as JSON.

Okay, so let's try sending a request to this app(routed hosted URL).

We have a response in the first cell which we send as a POST request to the app(stored in the variable url = https://localhost:9696/predict).

Okay so one more thing: we get a WARNING

There are many WSGI servers available in Python. Eg: Gunicorn

Pip install gunicorn
Gunicorn --bind 0.0.0.0:9696 predict:app

We mention --bind 0.0.0.0:9696 because gunicorn won't execute the if __name__ == '__main__' function so we're just re-specifying the URL. **predict** is the name of our python file

NOTE: gunicorn is a mac/linux feature and doesn't work on windows
Solution on windows: waitress
pip install waitress
Waitress-serve --listen=0.0.0.0:9696 predict:app

VIRTUAL ENVS & DOCKER

Saturday, July 2, 2022 4:14 PM

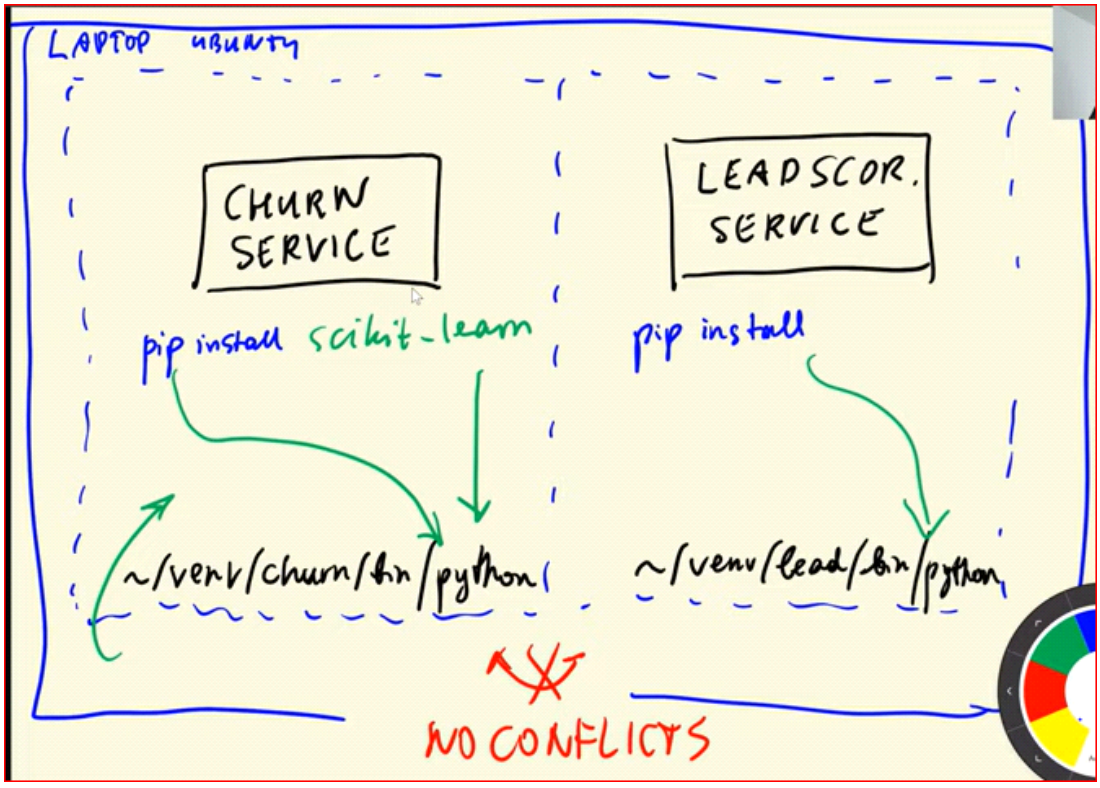
The need for Virtual Environment: Different services can require different version of same libraries

3 ways to create virtual environment in Python

https://medium.com/@krishnaregmi/pipenv-vs-virtualenv-vs-conda-environment-3dde3f6869ed

Pipenv was created due to many shortcomings of virtualenv such as it not making a distinction if project dependency and the dependencies of the project dependency, not having mechanism to distinguish dev and production needs etc.

To install new packages do `pip install package` , and pipenv will automatically add the package to the pipenv file that's called `Pipfile`



The virtual env of churn service uses the python this virtual env has and installs the sckit-learn using this specific python only.

Ways:

1. **Venv**
2. **Conda**
   Create: conda create -n ml-zoomcamp python=3.8
   Activate: conda activate ml-zoomcamp
   Install Libraries: conda install numpy pandas seaborn
3. **Pipenv**
   Install pipenv: Pip install pipenv (it creates a virtual environment of the folder we're in)
   Install Libraries: pipenv install scikit_learn==1.0.4 (Creates pipfile and pipfile.lock)

Activate: pipenv shell
Exit: CTRL +C

Pipfile: saves library versions and give us [packages] and [dev-packages] options. [dev-packages] option is for installing something that's required for development but we don't want to push it to cloud(deployment site).
Note: Dev packages in pipenv: Sometimes we want to add certain packages but only in the current environment without writing them to the production environment such as testing packages. We use pipenv install --dev instead of pipenv install.

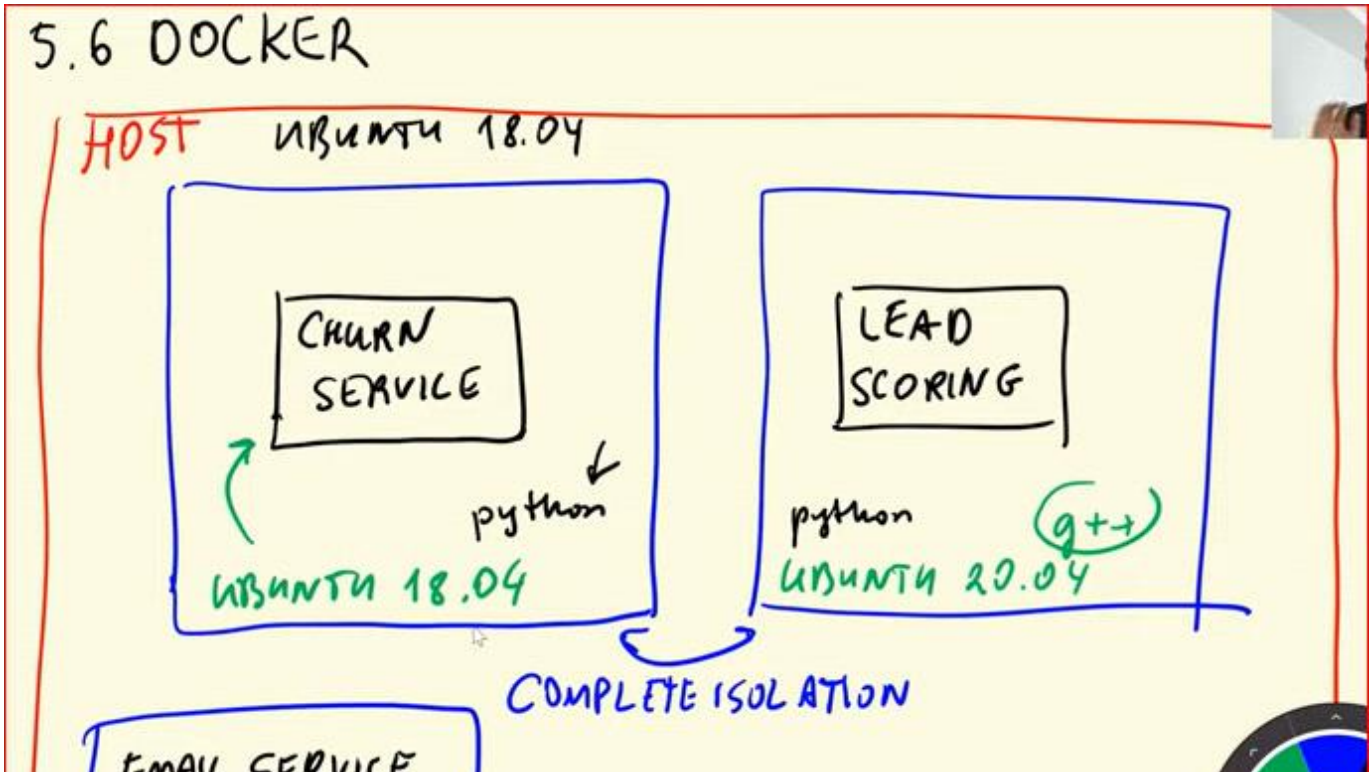
Pipfile.lock: helps with reproducability. Pinning dependencies to avoid compatibility issues: when we use this virtual env on a different machine, we need to make sure all the dependencies are same so pipfile.lock just saves the hashes of libraries installed

4. **Poetry**

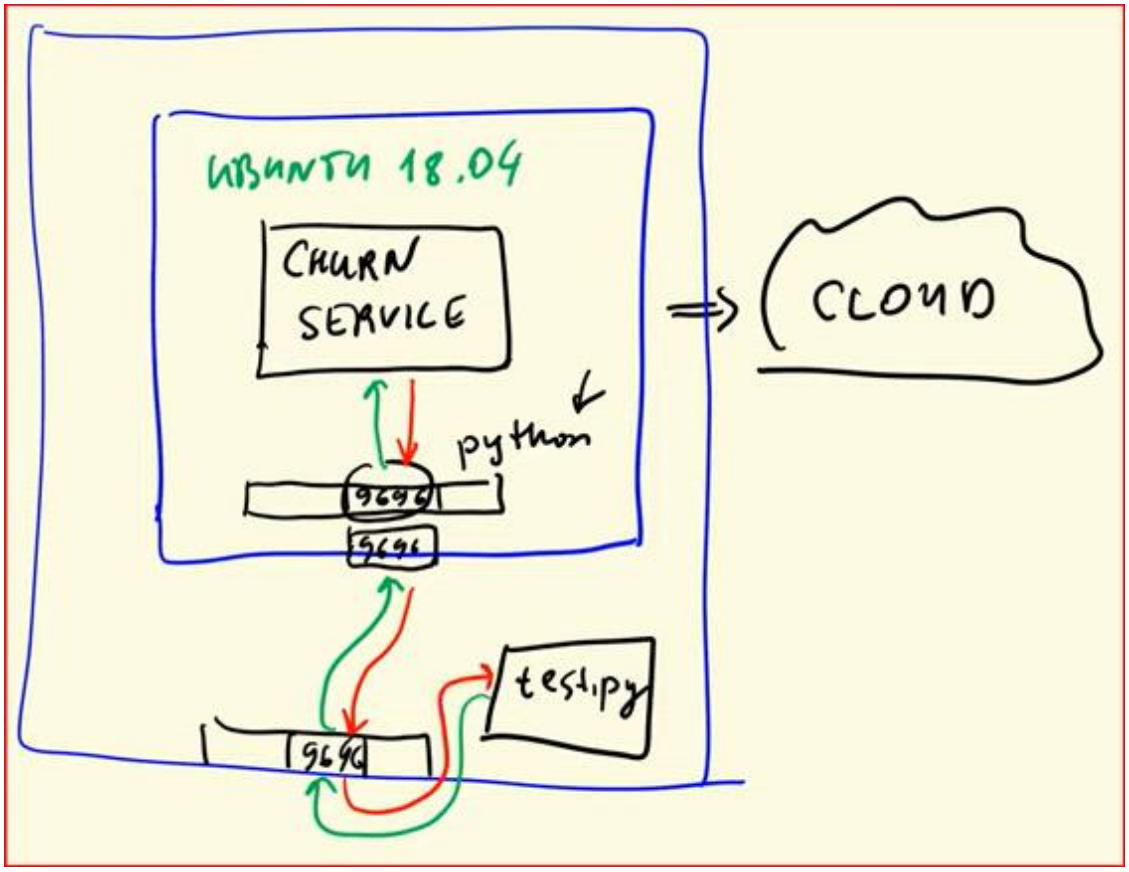
DOCKER

ML Zoomcamp 5.6 - Environment Management: Docker

Covered
- Why we need docker
- Running a Python image with docker
- Dockerfile
- Building a docker image
- Running this newly built docker image



- We discussed Virtual Environment which can create 2 spaces on the same laptop. Docker takes it to next level by having even different operating systems.
- Deployment to cloud



1. We got python docker image from dockerhub
2. We built our own image on top of that using some basic code in dockerfile. We put our models and supporting files to the docker
3. We ran the docker
4. Now we need to expose this docker to our local machine and we do that by opening up the port and connecting it to our local machine
5. Our local machine would sent request to the port, which will send request to the churn service and we'll get the response back from the same route

Now instead of managing anything on local, we can also do this on cloud.

[ML Zoomcamp 5.7 - Deployment To The Cloud: AWS Elastic Beanstalk (Optional)]

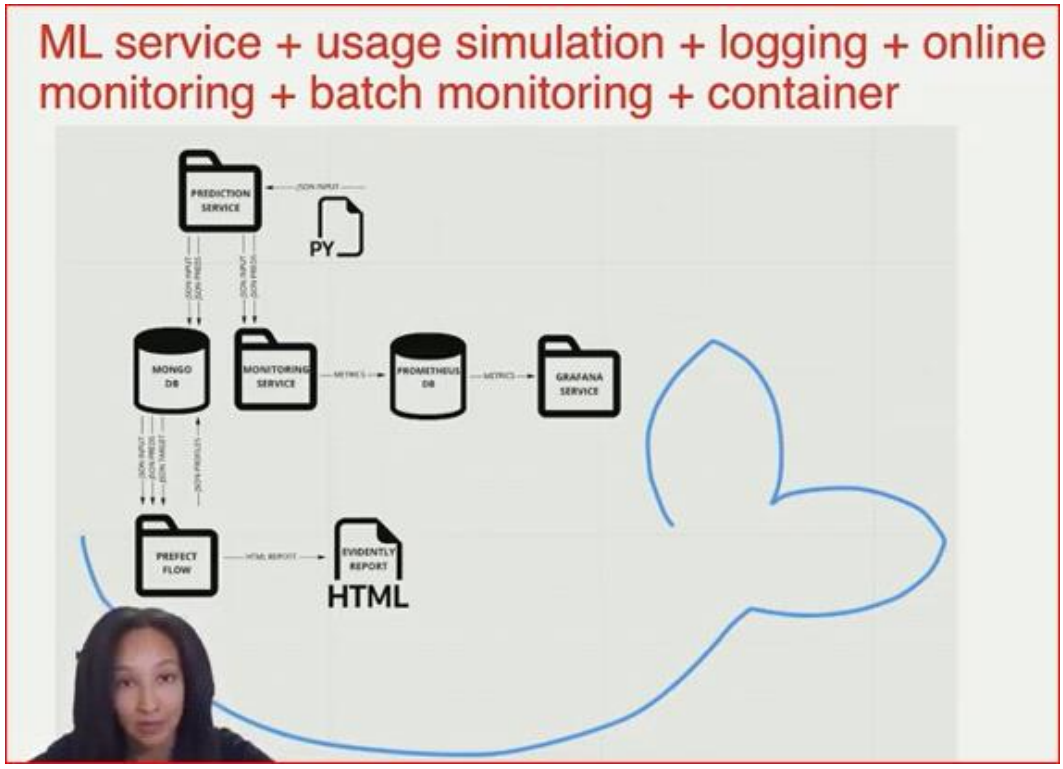
M5: Model Monitoring in Prod

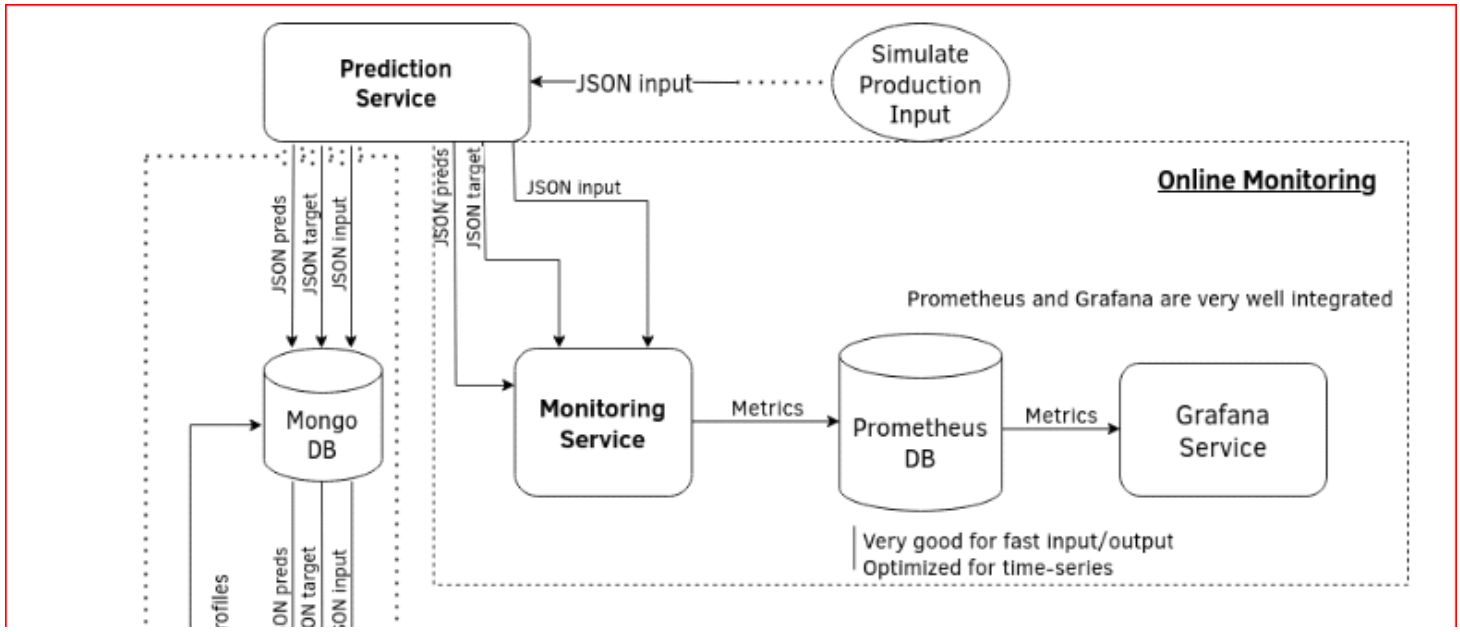
Wednesday, July 6, 2022 12:46 AM

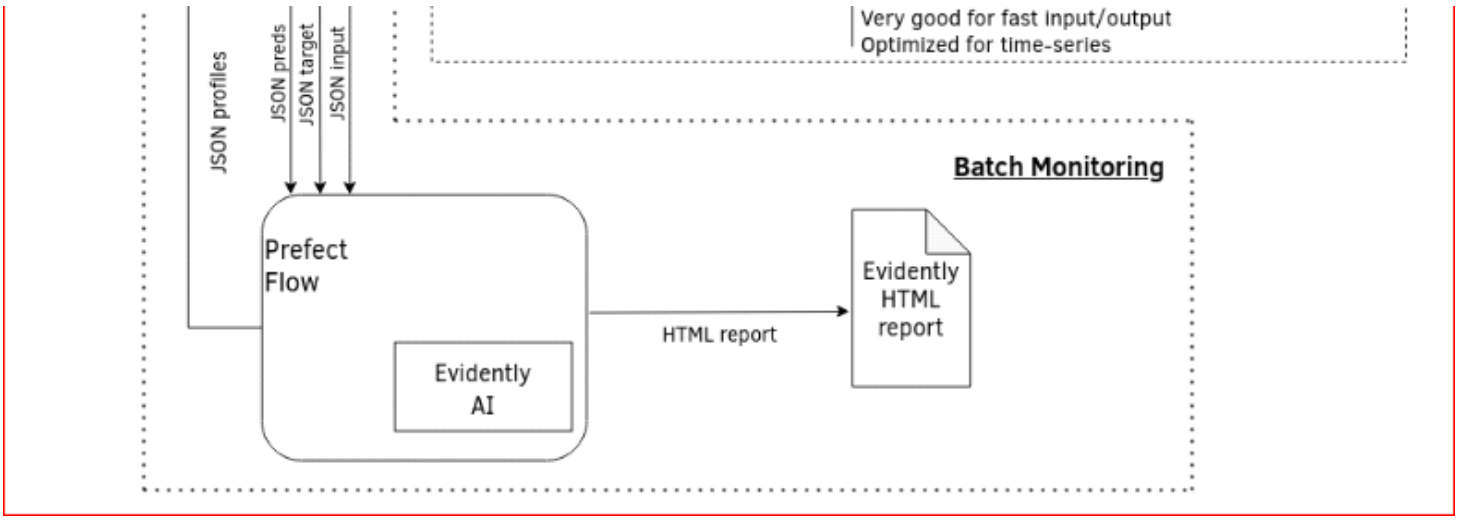
BEST NOTES : https://gist.github.com/Qfl3x/aa6b1bec35fb645ded0371c46e8aafd1

Why? Model models degrade and need monitoring

In the last module, we have built a prediction service that takes in a JSON data and gives out predictions in JSON format. Now we need to monitor this.







Why use Evidently?

Prometheus provides time series storage and a query language.
Grafana provides dashboarding and alerting. Neither Prometheus nor Grafana calculate any metrics on their own. One is a place to store metrics, another to visualise them. When you use Prometheus/Grafana for operational software monitoring (e.g. things like latency), you'd typically use a client library to instrument your code. Prometheus then scrapes the metrics from the end-point. With ML-focused monitoring, you need a way to collect ML-focused metrics and push them to Prometheus. This is what Evidently does in the set-up: it provides the library of ML-focused and data-focused metrics and a way to collect them and define the logic. You could, potentially, just write all your predictions logs to Prometheus (not an optimal choice) and then construct a giant PromQL query to calculate something like statistical test for data drift. But it would be very hard to maintain.

Useful comment on structuring project:

You could have more services depending on what you are trying to do..Ideally you need a prediction service,
a monitoring service (especially if you want live dashboards), data store service if you need super fast write and read, and more or less..If you want real time monitoring, then more services like prometheus, grafana, mongo, evidently and prediction service..If you want batch, and no live dashboards you can get away with prediction service, mongo dbYou may also not need mongo db, if you have a data warehouse , your prediction service can write to it, not recommended for real time, as this will be slow write..Ideally your training should be separate.. The only thing that will be shared between your training and prediction service, is a location where your model is stored..So separate compose files, because you may not want prediction service to wait until your model is trained.. You may be live with one version, and training should happen independently, then prediction service picks up on new model, as soon its deployed.