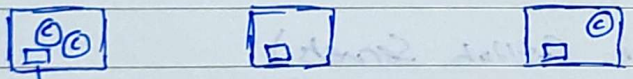
Kubernetes (K8S)

=> Google's container orchestration tool.
=> Increasingly micro-service oriented applications nowadays
=> Kubernetes used to manage stuff between them

=> Expectations
   -> Availability (of services) should be high / no-downtime
   => Scalability or high performance
   => Disaster Recovery (Backup & Restore)

ARCHITECTURE:

-> Master Node
(control plane)
(A machine, not container)



Kublet Service

Worker nodes

- Everything in a Kubernetes cluster is a container.
- Every node has a service running known as Kublet service.
  i) keeps track of machine stats and communicates to master.
- None of our work/containers is done/are run on master

STUFF RUNNING ON MASTER:

1) API Server:
   - Entry point into the K8S cluster.

2) Controller Manager (CM)
   - Keeps track of what is happening in the cluster

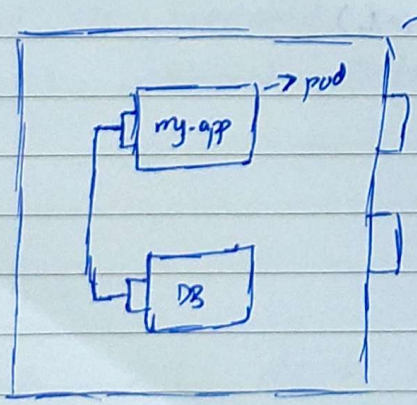
3) Scheduler
   - Ensures pods placement

4) et-cd
   - K8S backup store (All others log their activity here)

Accessing K8S:
i) UI ii) CLI iii) API

POD:  -> pod

- Smallest unit in K8S cluster (for what it works)
- An abstraction over container.
- Usually 1 container per pod (very rare multiple)
- Each pod gets its own ip address



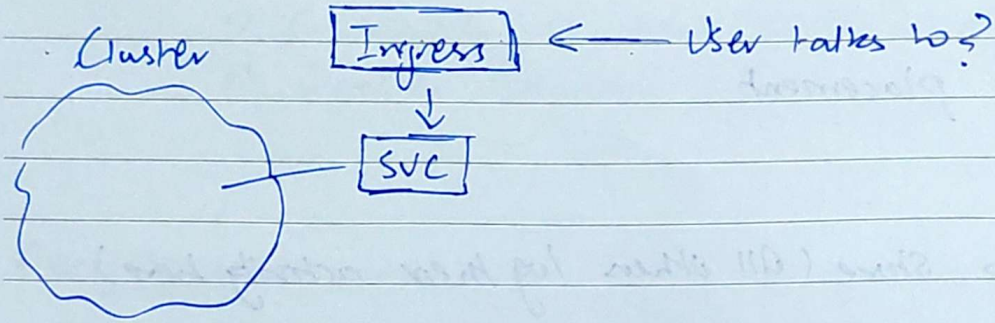
- Pods are ephemeral (temporary). Can be created/destroyed. Part of a normal process of scaling & down scaling.

MiniKube

- Each pod gets a new IP on creation.
- This creates a problem, any other pod talking with it would be confused (when scaled / de-scaled)

LIFETIME of SVC and Pods are not linked

2-9-2025



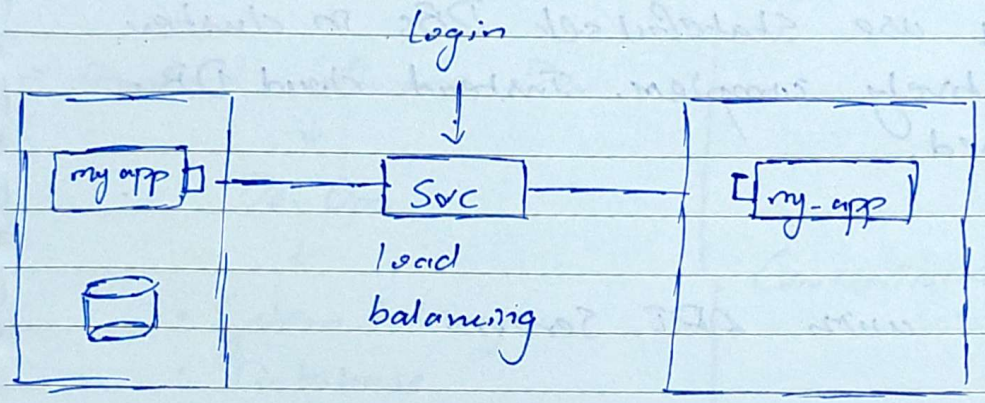
INGRESS:

CONFIG-MAP:
- Contains external configuration to your app
- Counterpart of environment variables / URLs of Docker Compose
- Readable isn't a problem

SECRET: (base 64, encoding ≠ secret-)
- Isn't human readable.

VOLUMES:

- Can be within & outside the node
- K8S doesn't maintain data persistence by default. Has to be maintained by user.



- Distribution of load isn't a problem with a stateless login page

- ~~How to use~~ ~~ensure~~ We now have an issue ← - what if we have a DB that needs state maintenance

K8s Keywords:

1) Deployment : When you need stateless deployment.

2) Stateful set : When you need stateful deployment.

=> SVC is needed to interact with pods, so if a pod runs without interaction, we don't need SVC.
↳ for eg a crawler that gets data from website and put in DB.

=> Each SVC deals with one type of pods (multiple instance of that thing)

Statefulset /

=> Deployment done through configuration.yaml (Yaml)

↓

← a blueprint for working (you can't directly give commands to create pods, you write in files).

Abstraction of pods.

=> We usually don't use Stateful set DBs in cluster since that is relatively complex. Instead cloud DBs like S3 are used.

CONFIGURATION.YAML:

- Communicates with API_Server.

Format: (rough)

1 - metadata
    name: _____

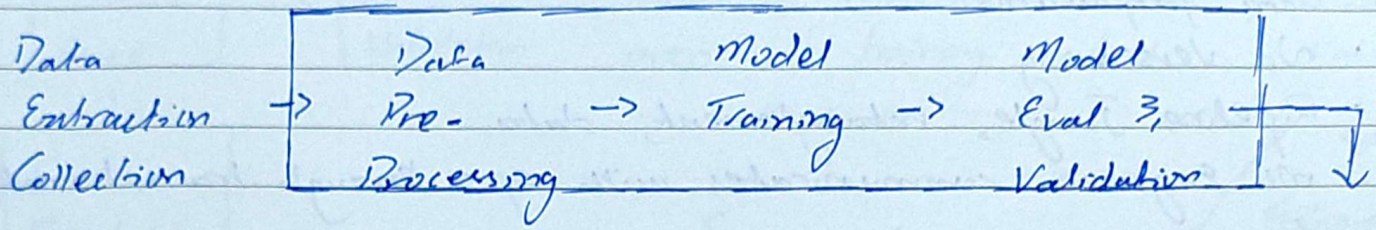
2 - spec
    replicas:
    image:
    selection:
    template:

3 - Status
    Desired ≠ Status

Hidden Technical Debt in Systems (2015) -> Google (Scully)

9-10-2025

MLOPS



=> Standard Software Dev

↓

DevOps

↓

- System latency
- Database
- Other conventional monitoring stuff etc

} CONVENTIONAL

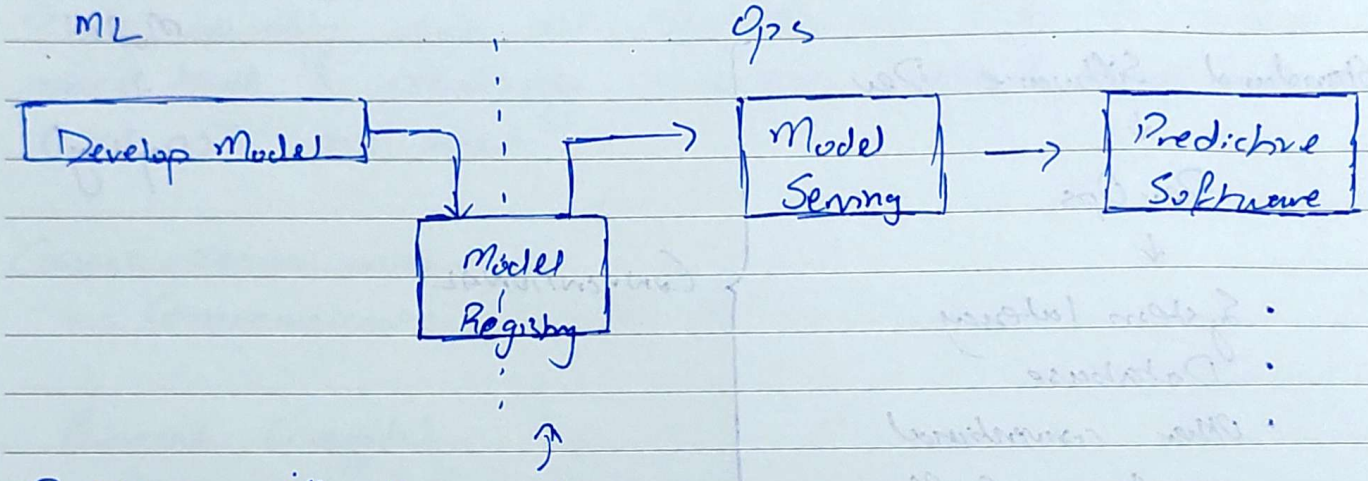
=> What's more in ML ?

↓

Performance Monitoring

Level 0: (Manual)

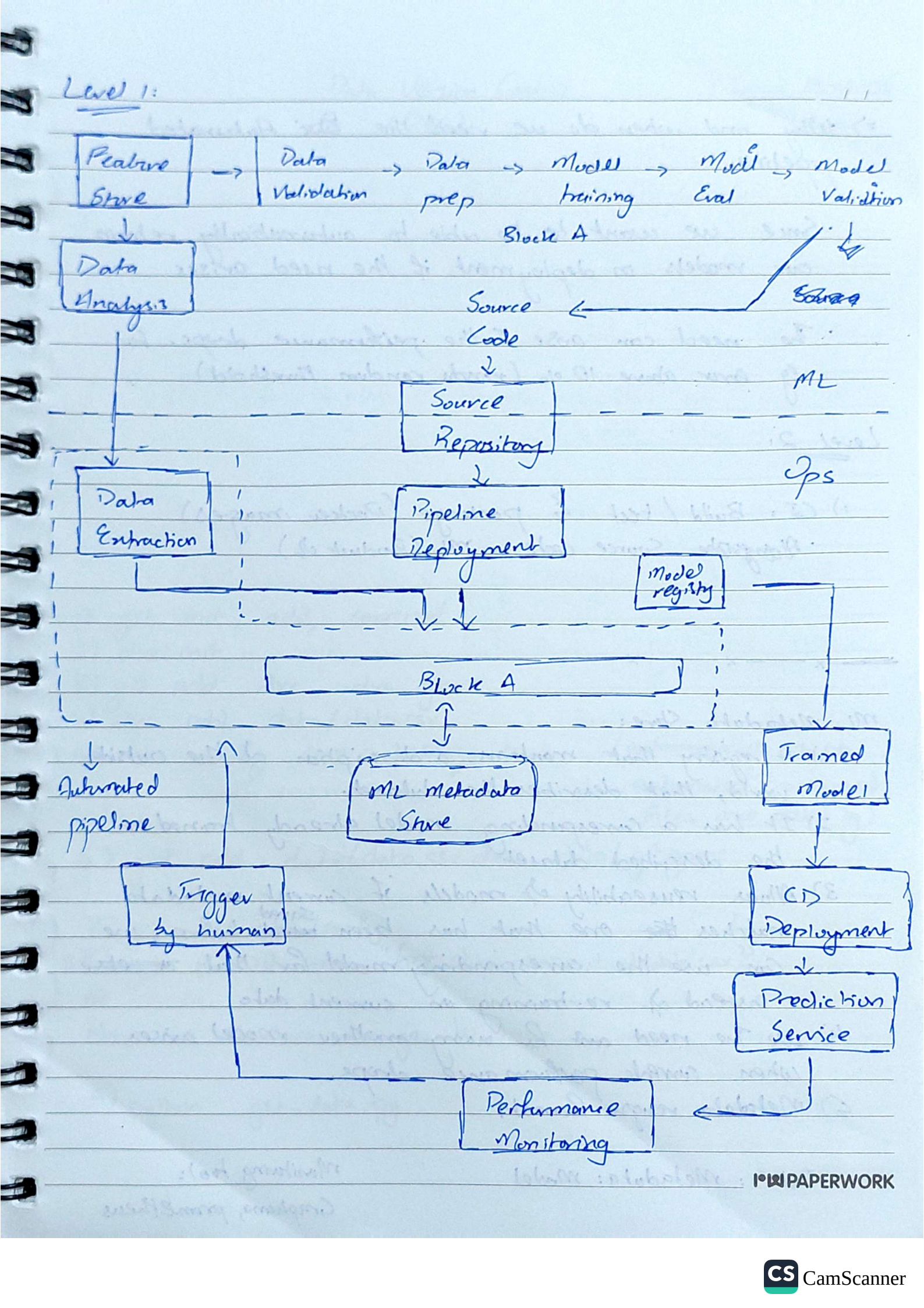
- Data preparation
- No versioning
- Pipeline Jungle, entanglement, data
- ML Engineer communicates with ops through trained models



Problems with this system:

- Ops just gets a file, no mechanism for transfer of intern information.
- No Rollback etc. (Personal addition point)

Level 1:



=> Why and when do we need the ~~Stvi~~ Automated pipeline

- Since we want to be able to automatically retrain our models in deployment if the need arises
- The need can arise if the performance drops. For eg error above 10 % (~~random~~ random threshold)

Level 2:

1) CI: Build / test & package (Docker images)
   - ~~Always~~ the Source code in ML (Product of)

ML Metadata Store:

1) A registry that maintains a description of the outside world, that describes the dataset.
2) It has a corresponding model already trained on the described dataset
3) Allows reuseability of models if current metadata matches ~~the~~ one that has been ~~used~~ saved before, we can use the corresponding model for that ~~or else~~ instead of re-training on current data
4) ~~We~~ The need ~~are~~ for using another model arises when current performance drops.
5) Metadata rough format:

Date : Metadata : Model

Monitoring tool:
Graphana, prometheus

Data Version Control (DVC)

14-10-25

Why?

Why not git?

Commands:

1) git init, add, commit
2) dvc init
3) git add .dvc .dvc_ignore
4) dvc add data/data.csv
5) dvc config core.autostage true (~~automatically~~ starts automatic tracking of files once added ig)
6) git add data/data.csv.dvc data/.gitignore
7) git add .dvc/config
8) git commit -m "first data commit of data.csv"

Now we introduce change to track

9) python get_data.py

=> Git hasn't recognized change, although dvc has noticed change due to due to auto-staging

=> ~~But auto-STAGING, we still have to add new data~~

dvc status

10) dvc add data/data.csv

=>

11) Now git noticed a change in data.csv.dvc (since it is tracking that)

11) git add data/data.csv.dvc

12) git commit -m "data is doubled"

=> Now if we wanna go back to previous data

13) dvc logs, find id of what data you want

14) git checkout <id> // Changes data.csv.dvc

15) dvc checkout // Changes data.csv by looking at changed data.csv.dvc

=> DVC isn't technically a version control system, the version controlling is done by git, it just reflects the changes since git isn't meant to handle large data. and other reasons

ls -lah data/data*csv

Now we also need a remote location to hold our data, so that others can collaborate (dvc push & stuff)

1) pip install dvc-s3

2) dvc remote add gdrive-remote gdrive://<UUID>

UUID = Universal Unique Identifier
(for google drive)

=> Open any google drive folder, in the URL
folder/<UUID>

-> Config file has been updated

3) git add .dvc/config

4) git commit -m "remote storage added"

5) dvc push -r gdrive-remote.

=> Failed to push since we didn't authenticate dvc to push on google drive

=> Let's authenticate:

1) Google Cloud (go to)
2) Create a project
3) APIs & Services
4) Enable API & Services
5) Enable google drive API
6) Now you have the dashboard
7) Create OAuth client ID
8) Desktop app, name
9) Now you have a client ID
10) Now you have to download json which has client ID & secret

PAPERWORK

6) dvc remote modify gdrive-remote gdrive_client_id <client id>

7) " " " " gdrive_client_secret <secret>

8) dvc push -r gdrive-remote

——— x ——— x ——— x

=> The other collaborator will now do

7) git pull

8) dvc pull // after sorting credentials.

——— x ——— x ——— x

Possible Error:

1) First user did git push but not dvc push.
2) Second user does git pull, dvc pull. Will get error related to this.

DVC Scripting YAML

16-10-25

⇒ We want to automate the retraining process on new data.

⇒ You don't need to manually dvc push dvc pull

1) dvc repro

2) dvc params show
dvc metrics show
dvc metrics diff

Issues:

1) Dependency didn't change, but the output of that stage changed

2) We add "externally_changed" file in dependencies which triggers the whole pipeline.

yaml

get_data: python get_data.py

preprocess:

train: cmd: python train.py, deps: , params: train.epochs

outs: metrics:
- metrics.json

Q: Why is train.py stage running on every "dvc repro"?

=> It only updates md5 values when stage is completed.

=> If md5 hashes are different, the stage is run again when "dvc repro"

=> but when stage was run it wouldn't complete since it didn't find all the files it needed.

=> So the stage didn't complete, the hash didn't synchronize & the train stage is triggered every time because of the hash difference

GitHub Action yaml explanation:

```
run: |
  make install
  dvc repro
  git fetch --prune
  dvc metrics diff --md master > report.md
  gh pr comment $((github.event.pull_request.number))
    -F report.md
```

Explanation:

1) Install dependencies
2) Re-run training (whole yaml of dvc). We now have new statistics.
3) git fetch --prune fetches all branches except the one it is running on (to compare old metrics with new one).

4)

=> You can have a specific file which is purely for triggering GitHub actions.

=> ~~Where~~ When you need to repro, which is done in GitHub actions, we can push to that file which will trigger GitHub Actions.

=> Later we'll automate the trigger based on monitoring. (Grophana) etc.

EXPERIMENT TRACKING

23-10-25

=> We have hundreds of parameter (hyperparameter level)
=> So infinite combos
=> How to find the best ones?

CONCEPTS:

1) ML Experiment
2) Experiment Run
3) Run Artifact
4) Experiment Metadata

WHY IS IT IMPORTANT:

1) Reproducibility
2) Organization
3) Optimization

WHAT'S THE PROBLEM WITH SPREADSHEETS:

1) Error prone
2) No standard format
3) Visibility & Collaboration

SOLUTION 1: ML FLOW: (Python package)

1) Tracking
2) Models
3) Model Registry : Model versioning ig
4) Projects : A complete unit which is pushed on gitHub and can be run by just one command. No manual repo cloning and stuff.

Mlflow

Storing the stuff:

1) Local Repositories on your device. Different repositories for different projects and a dashboard tying them together (problem: not accessible for other people).

2) Server based. Can use through: (Live ip needed)
   1) ~~Live ip~~
   2)

mlflow ui

mlflow server --host=127.0.0.1 --port:

Demo of mlflow: (Python file)

1) ~~Write~~ pip install mlflow
2) Import
3) Initialize (Where you can also set server ip, or local)
4) Normal training
5) Use mlflow syntax to log what you want
6) ~~Also, calculate~~

Auto Log:

~~Impo~~ => Don't have to manually set ~~which~~ which parameters to log and any code
=> Just call autolog, it will detect model & do it for you

Tracking:

Hyper parameter tuning using Gridsearch in MLFlow:

⇒ Example
- 3 hyper parameters
- 3 values for each
- Run one
- Compare & pick best combo (automatically have track as well).

Plots:
1) Without steps for measures
2) With steps for showing improvement or & reduction in metrics.

PROJECTS:

1) MLproject # project config file
2) conda.yaml
3) python_env.yml
4) requirements.txt
5) train.py.

} Standardize environment
- Run from anywhere
- Experiment in diff envs environments

// mlflow run . -e projects -P n_estimators=200 -P max_depth=10

// Can also run a github repo (and its commit version) which are compliant with MLflow

// mlflow.run (can run through python as well).

// Can give any environment or runtime, local / conda

Models

=> Saving models (Pytorch, sklearn ...)