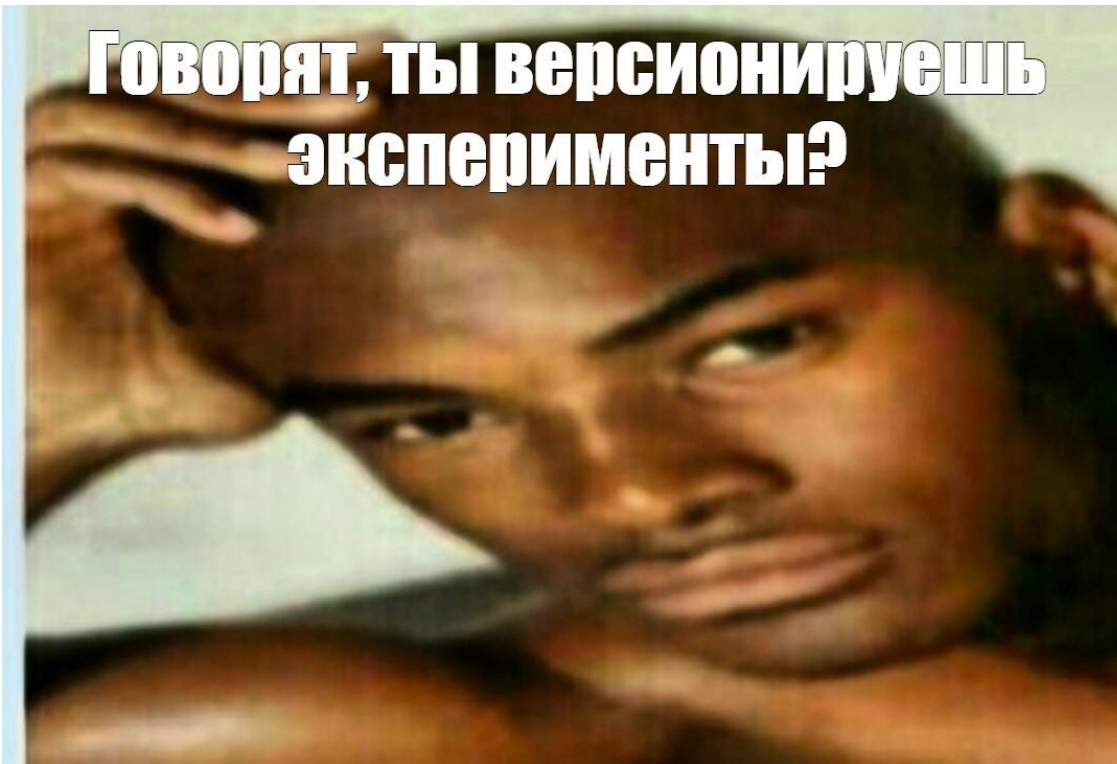


Версионирование данных и моделек



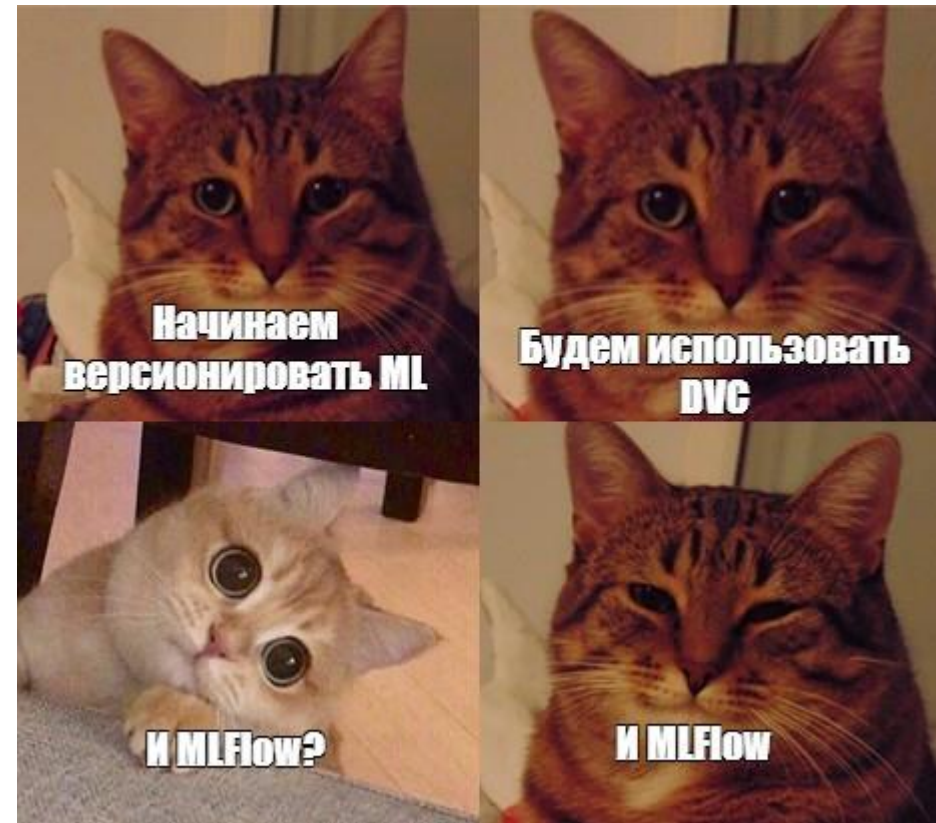
Борисенко Глеб
07.11.2024

О чем говорили в прошлый раз

- Фронт
- Стримлит

О чем поговорим сегодня

- Зачем версионировать данные и модельки
- Ведение экспериментов
- DVC
- MLFlow



Зачем нужно версионировать данные и модельки

- Хранить данные и модельки – возможность повторно воспользоваться ими
- Версионирование – одна из составляющих для возможности воспроизвести эксперимент.
- Версионировать – означает возможность хранить несколько версий одного и того же объекта данных и переключаться между версиями.

Важный моментик

- Удаление - это специальный маркер, а не фактическое удаление
- Изменение объекта - по сути новый объект

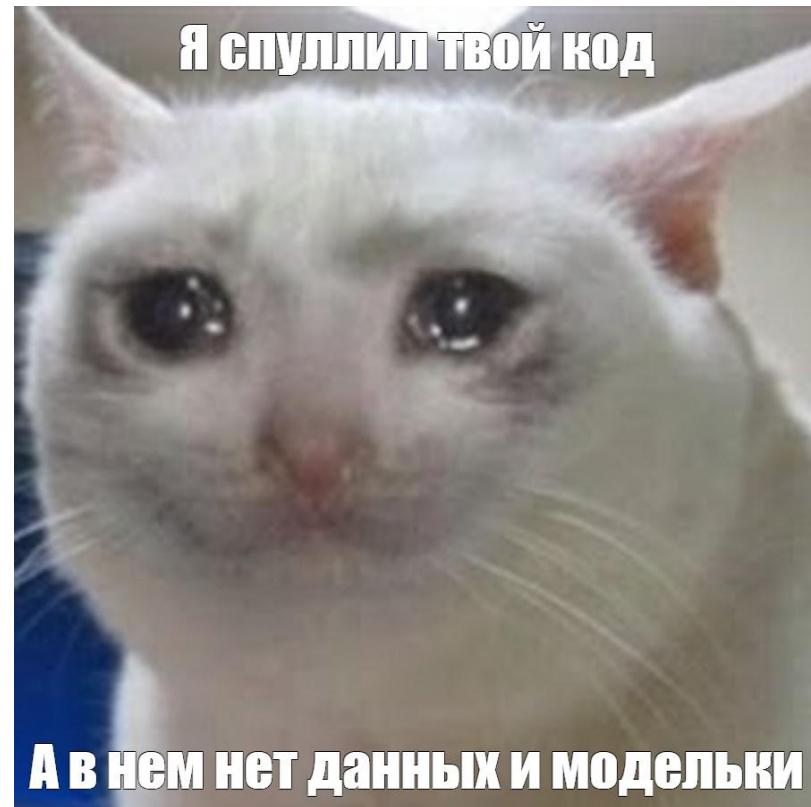
Примеры, когда она нужно

- Делаем экспериментов до задницы – во всем этом нужно суметь разобраться потом:
 - Что делали?
 - Что было на входе?
 - Когда делали?
 - Что получили?
- Моделька переобучилась по расписанию – получилась какаха, нужно откатиться



Командная работа

- Все накидывают на вентилятор экспериментов
- Хочется делать это КОМАНДНО и не запутаться
- Нужны версии



Версионирование в файловой системе

- git не помощник в версионировании данных
- При решении задачи версионирования в файловой системе хорошо принять решение о разделении данных на чанки
- Некоторые файловые системы имеют встроенные системы версионирования
- Для универсальности – проект DVC

Версионирование данных в базах данных

- Данные меняются - в каждой таблице есть поле версии и признак того что объект удален.
- Изменение - новый объект. Удаление объекта - метка
- Использование триггеров - перегружает и замедляет базу (ок, если это не продакшен)
- Только инкрементальные изменения

Версионирование моделей

- Моделька – по сути, те же самые данные
- Гиперпараметры и веса
- Либо артефакт – файл или набор файлов

Итак, надеюсь, вы поняли

- Вы поняли, зачем нужно версионировать данные и модельки
- Приступим к инструментам



DVC

- Мощщщщщная штука
- По сути, аддон к гиту (GitOps, все дела)
- Позволяет версионировать данные с помощью гита
- Храним данные в одном месте – на S3, HDFS, etc., а версионируем метаданные в гите

ОСНОВЫ

- Одна из основных концепций здесь - кэш dvc.
- Это просто директория `files/md5/`, где все файлы хранятся по md5 хэшам. Причем хранятся они разбитые по директориям:

```
.dvc/cache  
└─ 22  
    └─ a1a2931c8370d3aeedd7183606fd7f
```

ОСНОВЫ

- В самом `.dvc` файле будет храниться информация об этом хэше, размер, количество файлов если это директория, и локальное название файла относительно расположения `.dvc` файла.
- Пример такого файла:

```
outs:  
- md5: ff18477f8960bbc5925da5021e4be0ed.dir  
  size: 43464104813  
  nfiles: 152115  
  hash: md5  
  path: image_segmentation
```

ОСНОВЫ

- Заметили, что там не просо хэш, а ``.dir``?
- Если датасет - это не один файл, а директория, то файл в кэше будет иметь дополнительно расширение ``.dir`` и хранить информацию о всех файлах в директория.
- Таким образом получается, что ``.dvc`` файлик указывает на ``.dir`` файл, который в свою очередь указывает на остальные файлы в директории.

ОСНОВЫ

- Возникает вопрос, а как с этим работать?

ff/18477f8960bbc5925da5021e4be0ed
01/005a38cfbf84d0207020e689e0ede2

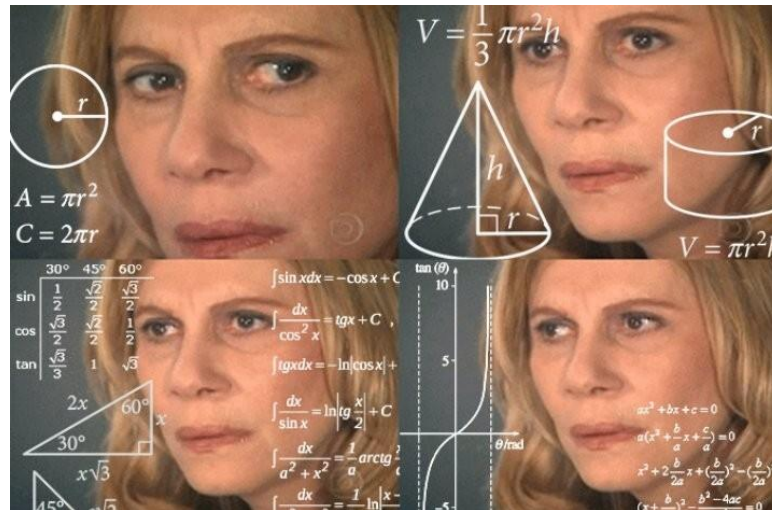


ОСНОВЫ

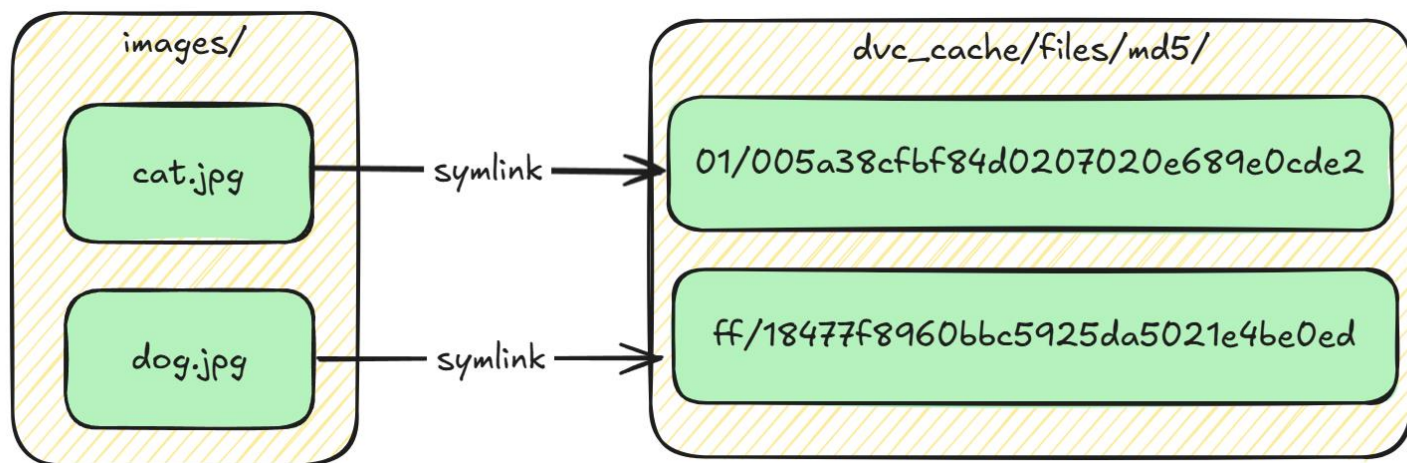
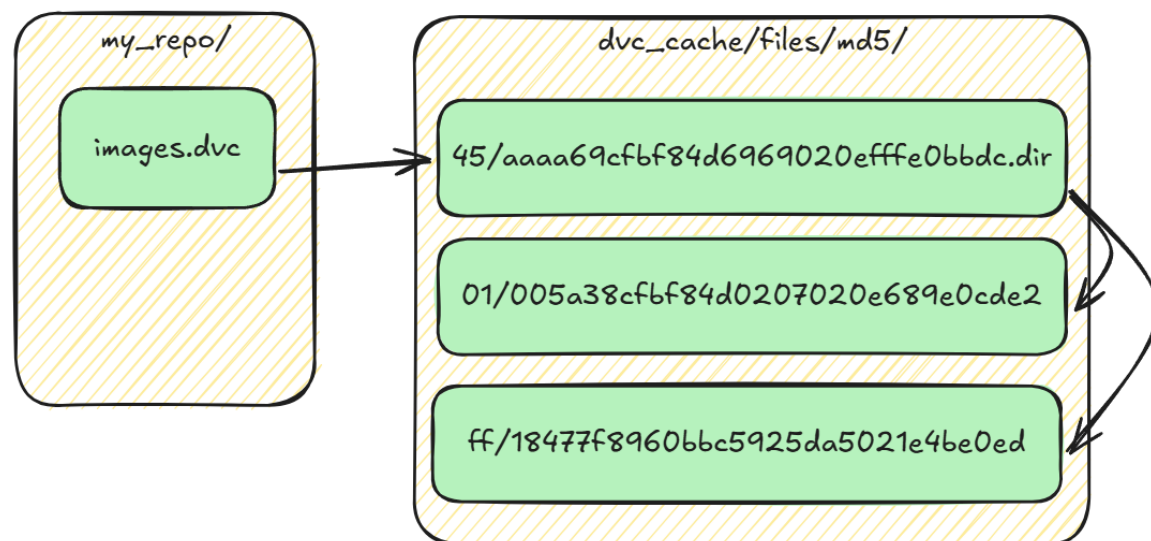
- А здесь dvc умеет создавать ссылки на файлы в кэше. Это могут symlink, hardlink, reflink (лучший вариант, но поддерживает пока что очень мало файловых систем), cору (тупо копирование).
- По умолчанию это cору, но это дело настраивается. И получается, что локально "под ногами" у вас не сами файлы (либо сами, если это стратегия cору), а ссылки на файлы, которые хранятся в кэше.

ОСНОВЫ

- Причем директория кэша настраивается. Т.е. это может быть не `.dvc``, а какая-то общая для всех репозиториев папка.
- Так, у вас может быть куча репозиториев с одинаковыми данными, но физически будет только одна копия данных в кэше, а все репозитории будут на них ссылаться.



ОСНОВЫ



ОСНОВЫ

- `dvc init` – создает файлы метаданных в `.dvc/`



Добавление данных

- `dvc add <my_data>` - добавление локальных данных в локальный кэш.
 - DVC посчитает md5-хэши всех файлов, скопирует их в кэш, создаст локальный индекс к этому кэшу (чтобы не перерасчитывать их повторно и хранить информацию о локальных ссылках), создаст ссылки в соответствии с выбранной стратегией (symlink, copy, etc.), создаст `.dvc` файл вашего датасета и пропишет пути к локальным данным (которые теперь ссылки) в `.gitignore`.
 - ВАЖНЫЙ МОМЕНТ: если выбрана стратегия не copy, то после копирования данных в кэш (но до создания ссылок) ваши локальные файлы будут удалены, и это операция обычно долгая. Здесь есть "хаки", упрощающие жизнь, но про них позднее.
- `git add . && git commit -m "New data"` - добавляем созданный `.dvc` файл в версионирование гита.

Допустим. А где s3?

- Синхронизация с s3 тогда осуществляется с помощью синхронизации dvc-кэша с таким же на s3
- То есть по определенному пути в s3 будут храниться файлы в таком же виде, как у вас локально в папке dvc кэша.
- Связь с s3 же указывается в файле ``.dvc/config`` в репозитории.
- В нем указываются все remote-ы - места на s3, где могут быть расположены данные: это путь в виде бакет/префикс и эндпоинт урл.

Допустим. А где s3?

```
[core]
  remote = my-minio
['remote "my-minio"']
  url = s3://dvc-bucket/
  endpointurl = https://my-minio:8000
```

- Креды к s3 также прописываются в dvc, но храниться они будут в таком же файле, только чисто локально, чтобы кредиты не утекли в gitlab.

Как ~~рулить~~ хранить на s3

- `dvc remote add -d storage s3://mybucket/dvcstore`
- `git add .dvc/config`
- `git commit -m "Configure remote storage"`
- `dvc push` - dvc посмотрит, какие файлы из кэша ассоциируются с вашими `.dvc` файликами "под ногами", посмотрит и сравнит, что есть уже на s3 и чего нет, и загрузит недостающее на s3.
- `git push` - отправляем изменения репозитория с `.dvc` файликом на gitlab

Как скачать и изменить версию

- `git clone gachi_experiment`
- И... ----->

Изменить версию:

- `git checkout`
- `dvc checkout`



Как посмотреть без гита

- `dvc list https://github.com/gigachad/gachi_experiment data`
- `dvc get https://github.com/gigachad/gachi_experiment \`
 `data/big_duck`
- `dvc import https://github.com/gigachad/gachi_experiment \`
 `data/big_duck`

Под капотом будет клонирование репы, так что доступ все равно нужен.

Можно использовать ssh ссылки.

Есть и python api

Так, что за импорт?

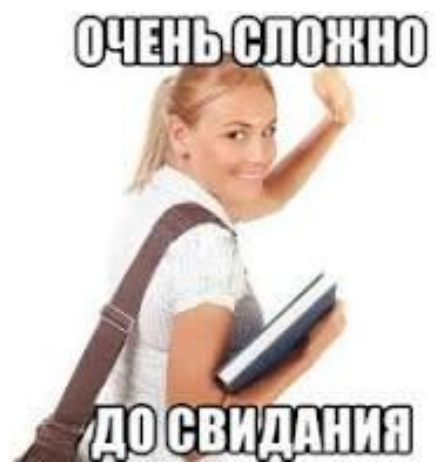
- DVC также позволяет создавать "файлы-ссылки" на другие .dvc репозитории. Они выглядят так:

```
md5: 2ae04fb8b10ab77b54bba024efdab137
frozen: true
deps:
- path: MNIST
  repo:
    url: https://gitlab.ru/myteam/dvc-datasets.git
    rev: gleb_dungeon_master
    rev_lock: c49177ebd26f48bd9c525aa472c80dc99a60b6ee
outs:
- md5: 563f931b5aa3d66aaee35ca9ea59ff3e.dir
  size: 66544770
  nfiles: 8
  hash: md5
  path: MNIST
```

- В такой датасет вы добавлять новые данные не сможете, но у вас появляется опция делать `dvc update` - dvc посмотрит, изменился ли `.dvc` файл в репозитории, откуда импортировали датасет, и если он изменился, то обновит импортированный `.dvc` файл и данные, с ним ассоциированные. Импортировать датасет можно с помощью команды `dvc import --rev <branch> <repo> <path>`.

Концепция Data Registry

- Единный гит репозиторий, где находятся только датасеты (*ну и может связанная с ними информация/дока*)
- По сути, низкоуровневый Feature Store
- Очень хорошо ложится в концепцию GitOps
- В основном полезно в DL, так как нет нормальных фиче сторов для нетабличных данных



Что там еще есть

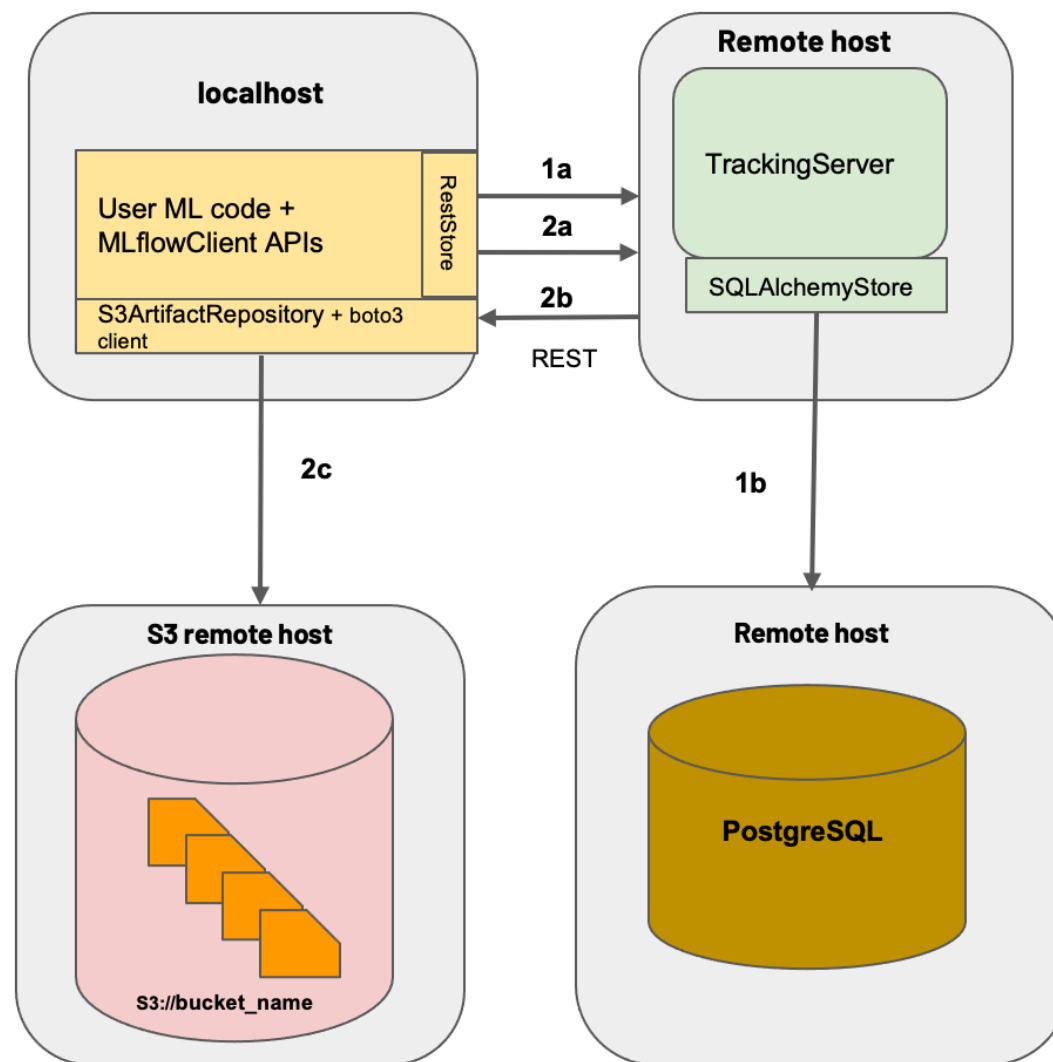
- DVC Pipelines
- DVC Metrics, Plots
- DVC Experiments

MLFlow

- Тоже мощная штука
- Позволяет логировать параметры, графики, модельки, все это хранить
- И все это аккуратно посмотреть в красивом UI



Как чаще всего это используют



MLFlow (Tracking) в питоне

```
mlflow.set_experiment("my-experiment-1")

with mlflow.start_run():

    X, y = load_iris(return_X_y=True)

    params = {"C": 0.1, "random_state": 42}
    mlflow.log_params(params)

    lr = LogisticRegression(**params).fit(X, y)
    y_pred = lr.predict(X)
    mlflow.log_metric("accuracy", accuracy_score(y, y_pred))

    mlflow.sklearn.log_model(lr, artifact_path="models")
    print(f"default artifacts URI: '{mlflow.get_artifact_uri()}'")

from mlflow.tracking import MlflowClient

client = MlflowClient()
client.list_registered_models()
```



MLFlow (Registry) в питоне 2

```
model_name = "nyc-taxi-regressor"
latest_versions = client.get_latest_versions(name=model_name)

for version in latest_versions:
    print(f"version: {version.version}, stage: {version.current_stage}")
```

```
version: 1, stage: Staging
version: 2, stage: Production
version: 4, stage: None
```

```
model_version = 4
new_stage = "Staging"
client.transition_model_version_stage(
    name=model_name,
    version=model_version,
    stage=new_stage,
    archive_existing_versions=False
)
```

```
run_id = "b8904012c84343b5bf8ee72aa8f0f402"
model_uri = f"runs://{run_id}/model"
mlflow.register_model(model_uri=model_uri, name="nyc-taxi-regressor")
```


```
Registered model 'nyc-taxi-regressor' already exists. Creating a new version of this model...
2022/05/19 16:47:17 INFO mlflow.tracking._model_registry.client: Waiting up to 300 seconds for model version to finish creation.
del name: nyc-taxi-regressor, version 4
Created version '4' of model 'nyc-taxi-regressor'.
```



UI 1

	Date	User	Source	Version	Parameters		Metrics		
					alpha	l1_ratio	mae	r2	rmse
<input type="checkbox"/>	2018-06-04 23:00:10	mlflow	train.py	05e956	1	1	0.649	0.04	0.862
<input type="checkbox"/>	2018-06-04 23:00:10	mlflow	train.py	05e956	1	0.5	0.648	0.046	0.859
<input type="checkbox"/>	2018-06-04 23:00:10	mlflow	train.py	05e956	1	0.2	0.628	0.125	0.823
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	1	0	0.619	0.176	0.799
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	1	0.648	0.046	0.859
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0.5	0.628	0.127	0.822
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0.2	0.621	0.171	0.801
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0	0.615	0.199	0.787
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0	1	0.578	0.288	0.742
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0	0.5	0.578	0.288	0.742
<input type="checkbox"/>	2018-06-04 23:00:09	mlflow	train.py	05e956	0	0.2	0.578	0.288	0.742
<input type="checkbox"/>	2018-06-04 23:00:08	mlflow	train.py	05e956	0	0	0.578	0.288	0.742

UI 2

 [Github](#) [Docs](#)

Run 7c1a0d5c42844dcdb8f5191146925174

Experiment Name: Default

Start Time: 2018-06-04 23:47:22

Source: train.py

Git Commit: 3aa48cffe58b8d9d69f5

User: mlflow

Duration: 145ms

▼ Parameters

Name	Value
alpha	0
l1_ratio	0

▼ Metrics

Name	Value
mae	0.578
r2	0.288
rmse	0.742

► Tags

▼ Artifacts

▼ model

MLmodel


model.pkl

Full Path: /Users/mlflow/mlflow-prototype/mlruns/0/7c1a0d5c42844dcdb8f5191146925174/artifacts/model/MLmodel

Size: 259B

```
artifact_path: model
flavors:
  python_function:
    data: model.pkl
    loader_module: mlflow.sklearn
sklearn:
  pickled_model: model.pkl
  sklearn_version: 0.19.1
run_id: 7c1a0d5c42844dcdb8f5191146925174
utc_time_created: '2018-06-05 06:47:22.757025'
```

UI 3



GitHub Docs

Registered Models

search model name

Name	Latest Version	Staging	Production	Last Modified
Model A	Version 1	Version 1	—	2019-10-16 22:51:19
Model B	Version 1	—	—	2019-10-16 22:51:52

< 1 >

Registered Models > Model A > Version 1

Registered At: 2019-10-17 13:38:51

Last Modified: 2019-11-12 09:56:00

▼ Description

Creator:

Source Run: Run b99a0fc567ae4d32994392c800c0b6ce

Stage: Staging

Transition to → Staging

Transition to → Production

Transition to → Archived

Что там есть еще, но используется реже

- MLFlow Projects – засовывание проекта как бы в пакет
- MLflow Models – помогает деплоить модельки; возможно, используется часто с каким-нибудь Seldon Core или KFServing, но не уверен

Минусы MLFlow

- Работает как говно
- Куча багов
- Очень прямолинейно, старо

Альтернативы

- ClearML (мощнее, круче, сейчас посмотрим)
- Neptune (дороже для команды и прикольнее)
- Wandb (Weights and Biases) (еще дороже и прикольнее)
- И другие

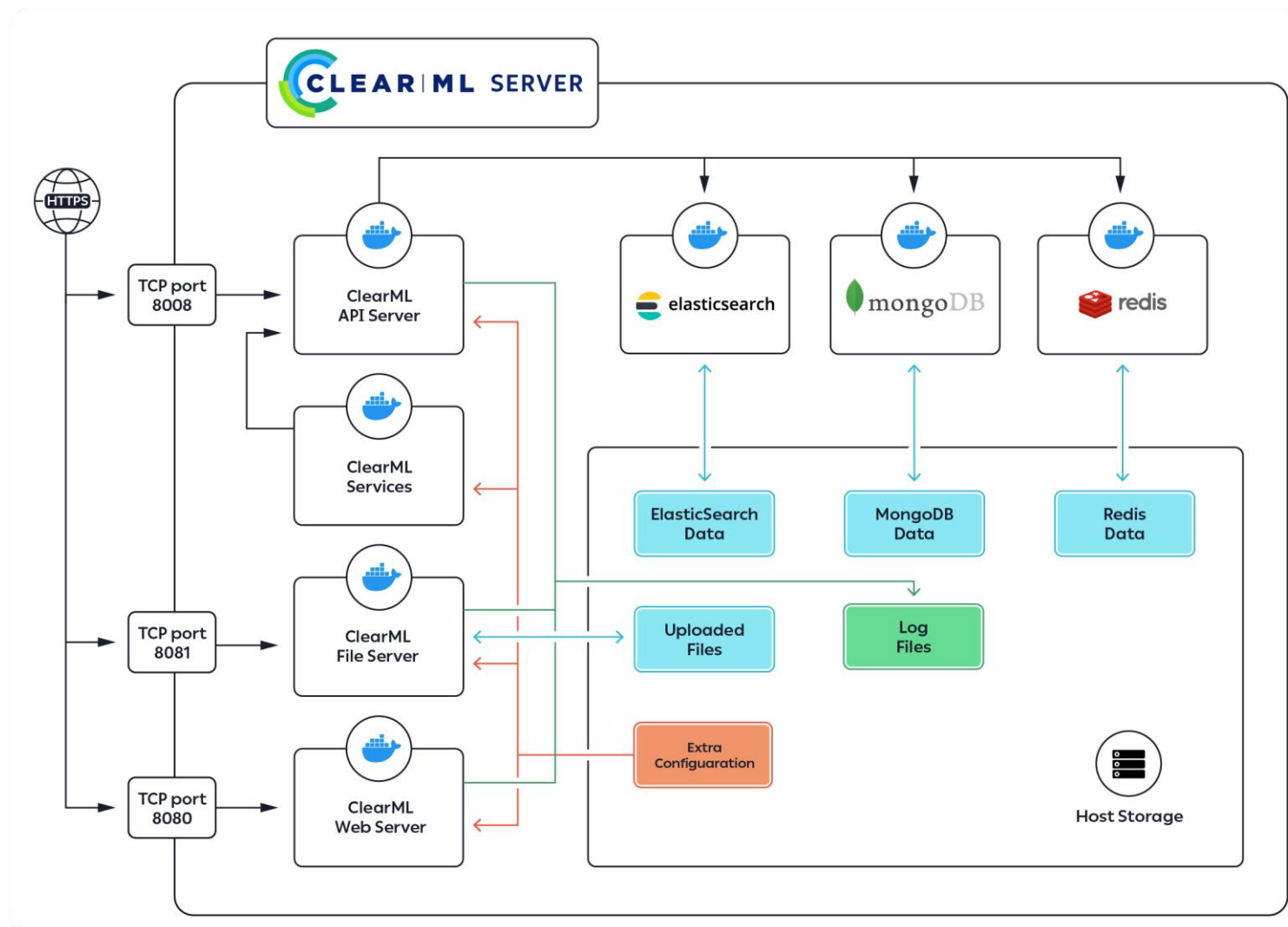


ClearML

- Более мощная штука, по сравнению с MLFlow
- Позволяет делать все то же самое, только лучше, удобнее
- Плюс еще дополнительные возможности в виде пайплайнов, датасетов, оптимизация гиперпараметров, кластера



Как это выглядит?



ClearML в коде (Part 1)

```
from clearml import Task

task = Task.init(project_name='great project', task_name='best experiment')

logger = task.get_logger()

logger.report_media("big_duck.png")

logger.report_histogram(hist_data)
```



Многие вещи (тензорборд, матплотлиб, параметры из гидры, аргументы ком. строки и другое) clearml сам трекает

ClearML в коде (Part 2)

```
from clearml import InputModel, OutputModel, Task

# Create an input model using the ClearML ID of a model already registered in the ClearML platform
input_model = InputModel(model_id="fd8b402e874549d6944eebd49e37eb7b")

# Connect the input model to the task
task.connect(input_model)

# Instantiate a Task
task = Task.init(project_name="myProject", task_name="myTask")

# Create an output model for the PyTorch framework
output_model = OutputModel(task=task, framework="PyTorch")

# Set the URI of the storage destination for uploaded model weight files
output_model.set_upload_destination(uri=models_upload_destination)

# Set the Label numeration
output_model.update_labels({'background': 0, 'label': 255})

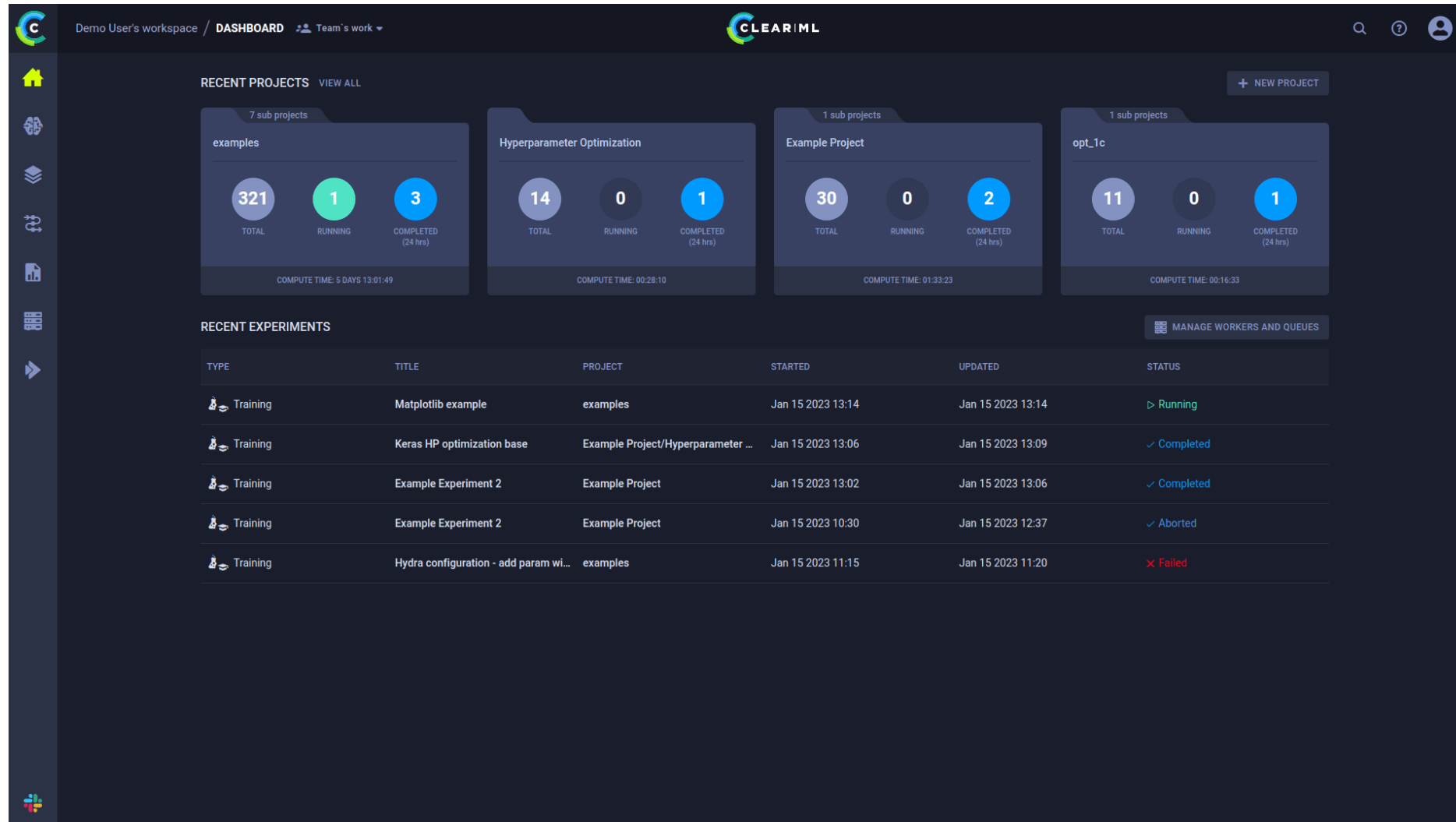
output_model.update_weights(weights_filename='models/model.pth')
```




Модели тут

- Здесь нет простого автоматического версионирования модели, как, например, в mlflow, но это и хорошо.
- Каждая модель связана здесь с проектом - директорией в каталоге. И у каждого проекта могут быть самые разные модели, которые неправильно называть, например, версией "0.0.2" совершенно другой модели.
- Поэтому названия моделей вы называете каждый раз сами либо они генерируются автоматически, и они вполне могут совпадать (uid моделей будут разные).
- Связь обеспечивается через lineage модели - это информация о том, в каком эксперименте она была создана и в каких использована. То есть связь между моделями осуществляется косвенно, через эксперименты.

UI 1



UI 2

 pytorch mnist train

EXAMPLE

ID 7d0526bd ...

EXECUTION

CONFIGURATION

ARTIFACTS

INFO

CONSOLE

SCALARS

PLOTS

DEBUG SAMPLES

DETAILS

UNCOMMITTED CHANGES

INSTALLED PACKAGES

CONTAINER


SOURCE CODE

REPOSITORY	https://github.com/allegroai/clearml.git
BRANCH NAME	Latest in branch master
SCRIPT PATH	pytorch_mnist.py
WORKING DIRECTORY	examples/frameworks/pytorch
BINARY	python3.6

OUTPUT

DESTINATION	
LOG LEVEL	INFO

UI 3





MNIST Inference


+ ADD TAG

DRAFT

ID 1a2b4f83 ...







GENERAL

NETWORK


LABELS

METADATA

LINEAGE

SCALARS

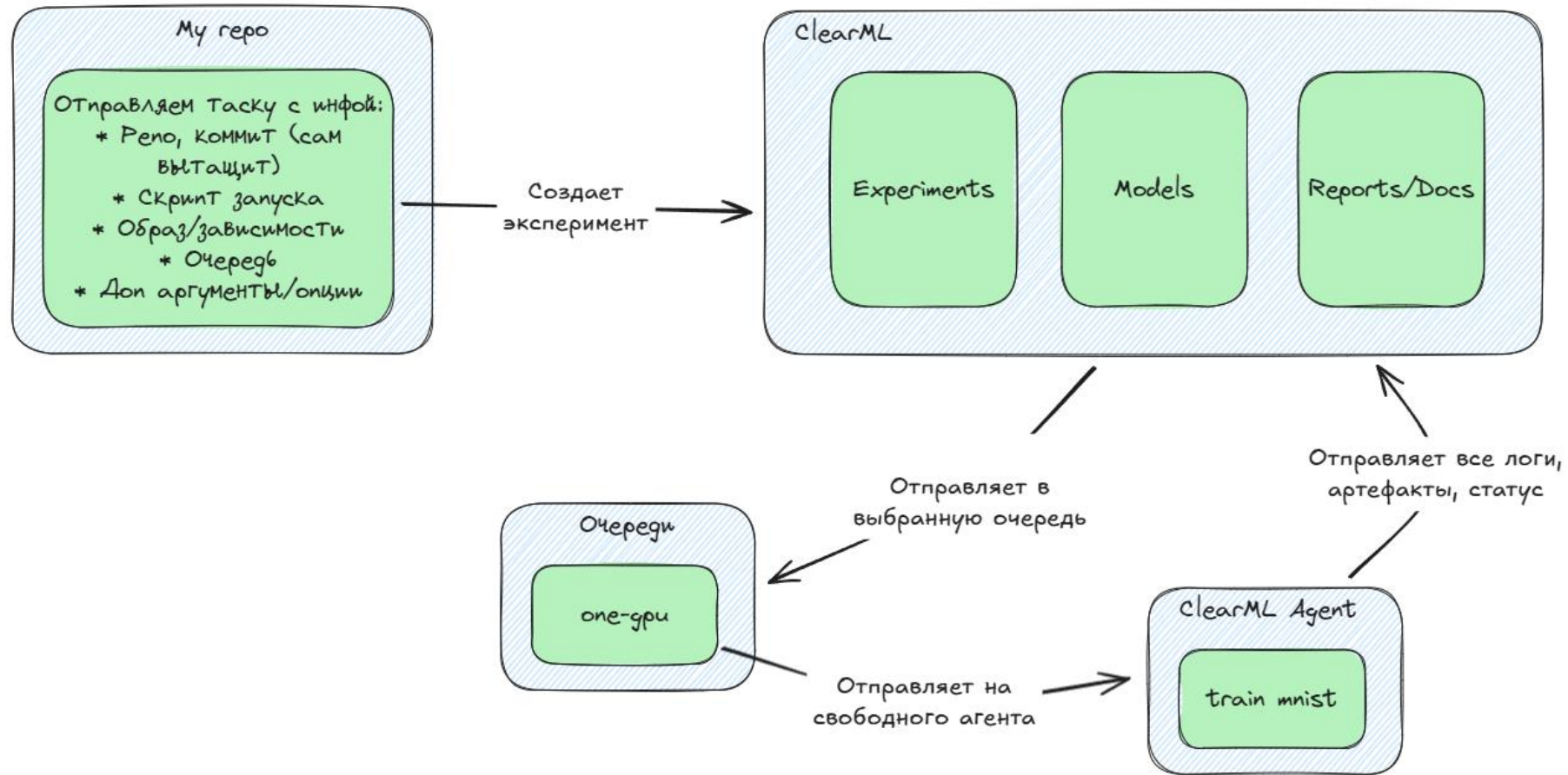
PLOTS

CREATED AT:	Mar 17 2023 1:37
UPDATED AT:	Mar 17 2023 1:37
FRAMEWORK:	PyTorch
STATUS:	Draft
MODEL URL:	file:///tmp/tmp63q3uz5v/input.pt 
USER:	Demo User
ARCHIVED:	No
PROJECT:	d61399d4-7ac5-4b3e-a18b-d25fc450dbc4
DESCRIPTION:	Imported by task id: 2402cf770a394c5883876eee6f4d7e14

Доп. фичи

- Версионирование датасетов – лично мне больше нравится DVC, так как тот прозрачнее и удобнее
- Выч. Кластер – ClearML позволяет развернуть «сеть агентов», на которых можно выполнять ваши task-и; прикольная фича для обеспечения воспроизводимости и вообще контроля экспериментов
- Пайплайны – пайплайны, выполняющиеся на кластере; те же Dagster, AirFlow или аналоги лучше с этим справляются обычно, но как вариант для чисто своих делишек, вполне сойдет
- Оптимизация гиперпараметров – прикольная штука, которая создание тасок берет на себя, а если есть сеть агентов – то вообще топчик
- Отчеты – можно писать MD отчеты с импортами данных из экстов.

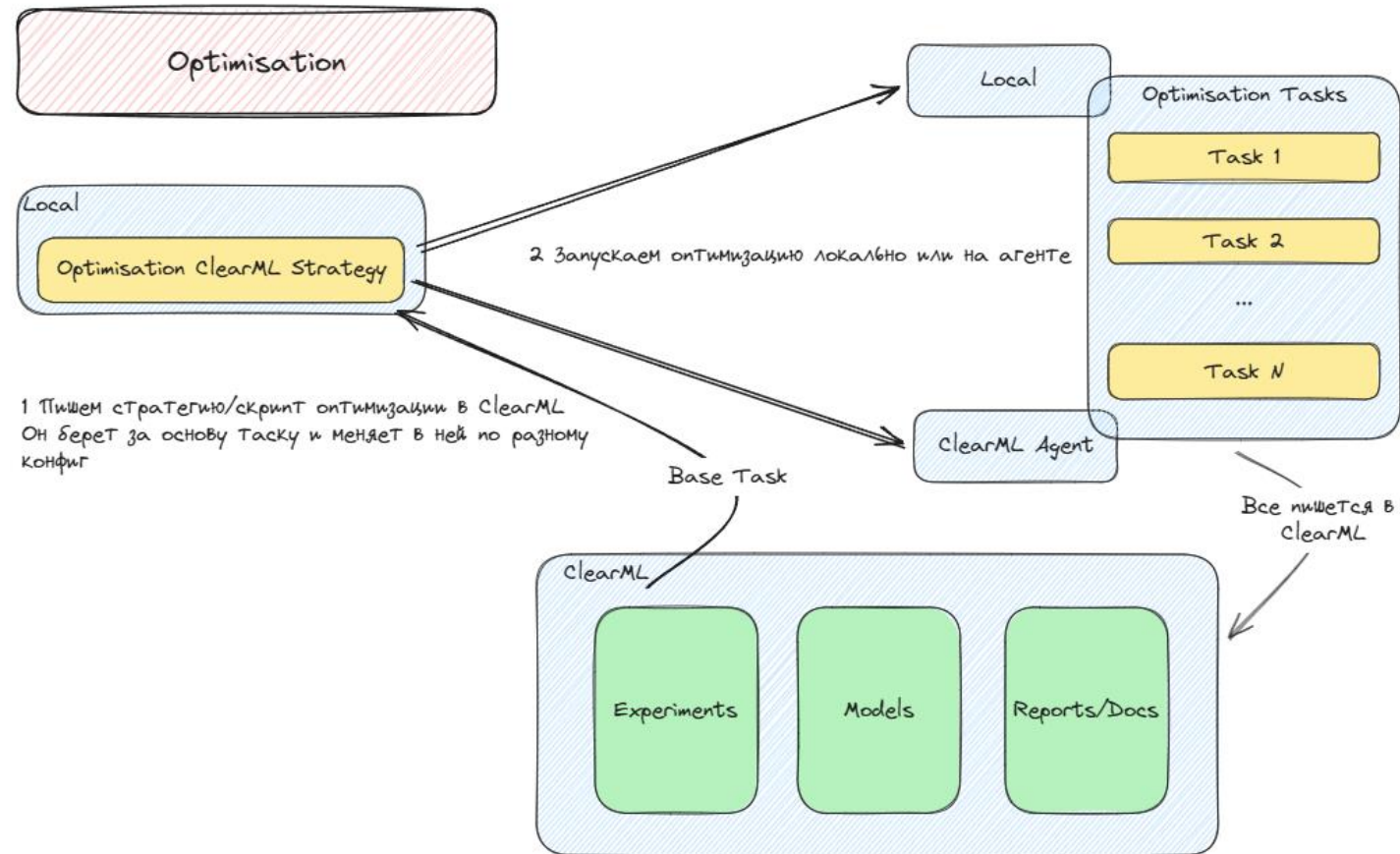
Немного больше про агентов



Немного больше про агентов

- Можно указывать кучу всего, весьма гибко
- Непросто настроить, но это того стоит
- Можно запускать агентов на компе процессом, в докере, в кубере
- Агенты могут быть в нескольких очередях
- Отправлять задачу в очередь можно через UI, CLI, Python
- Через UI все параметры можно менять и отправлять по новой

Оптимизация на агентах



DVC и ClearML можно использовать вместе

- ClearML – версионирование моделей и параметров
- DVC – версионирование данных
- Ну и Git – версионирование кода



Выводы

- Воспроизводимость эксперимента - это не только гиперпараметры и код, но модельки и данные
- Возможность лучше контролировать результат
- DVC и ClearML- наши друзья

Смотрите в следующей серии (не решил еще с порядком)

- Тестирование и его виды
- Pytest

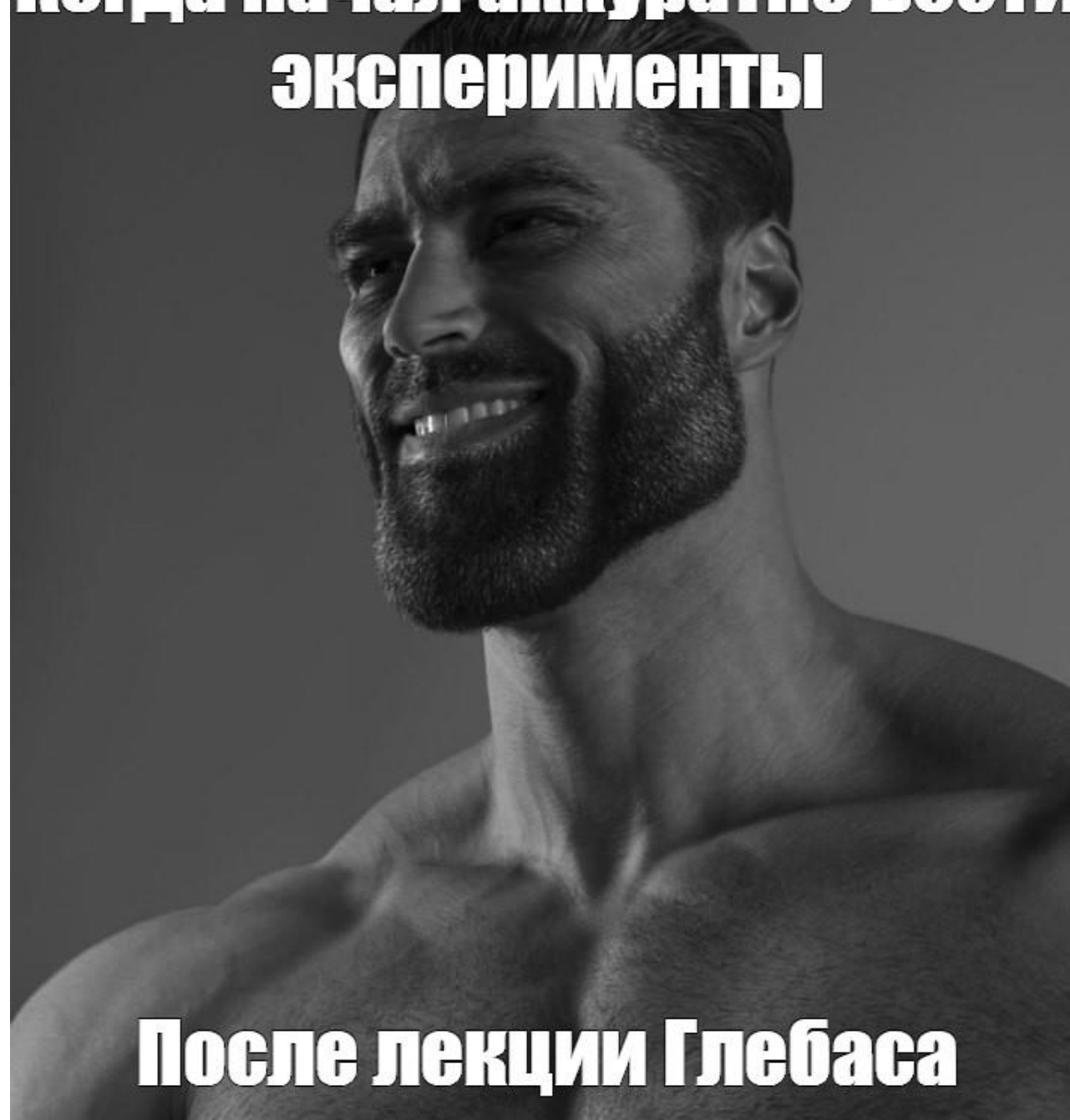
ЛИБО

- Пайплайны данных

ЛИБО

- Лекция про код по следам ДЗ

**Когда начал аккуратно вести
эксперименты**



После лекции Глебаса