**Лабараторная работа 4 (Проект)**

**Саенко Лев РИМ-220906**

*Вам необходимо выбрать один из пройденных инструментов автоматизации процесса машинного обучения (DVC/ClearML / MLFlow / Airflow), реализовать с его помощью процесс обучения любой выбранной вами (или созданной самостоятельно) модели. В зависимости от используемого инструмента определить решаемые экспериментом задачи - работа с различными датасетами, либо подбор оптимальных гиперпараметров, либо сравнение результатов обработки данных различными алгоритмами, либо др. Пояснить цель и ход своих экспериментов.*

*В качестве решения прикрепить скриншоты созданных задач в выбранном инструменте, файлы скриптов по работе с данными и обучению модели, а также файл с небольшим пояснением содержимого экспериментов.*

Код всего проекта – <https://github.com/Basserti/MLOPS_course>

**Описание**

Был выбран:

* Jenkins – оркестратор
* DVC – версионирование дата сета и pipeline обучения
* 5 виртуальных машин на Debian 11
* Putty и notepad++ (с аддоном на ssh)
* Flask
* CatBoost
* Dataset взят из соревнования в качестве примера данных.

Данные мануально (вручную) помещаются в клонированный гит, в папку с данными (если изменили исходные данные, не предпроцессинг), DVC выступает в роле пайплайна обучения. Происходит предпроцессинг данных, потом обучение и замер метрики, данные отправляются в DVC моделью, а код, параметры обучения, конфиги итд, в гит. Jenkins получает репо с гита, скачивает данные и модель с DVC и тут есть развилка:

* Тест метрики (чтобы была меньше 1 200 000 (MAE))
* Обучение модели с новыми параметрами, которые можно менять прямо в гите.

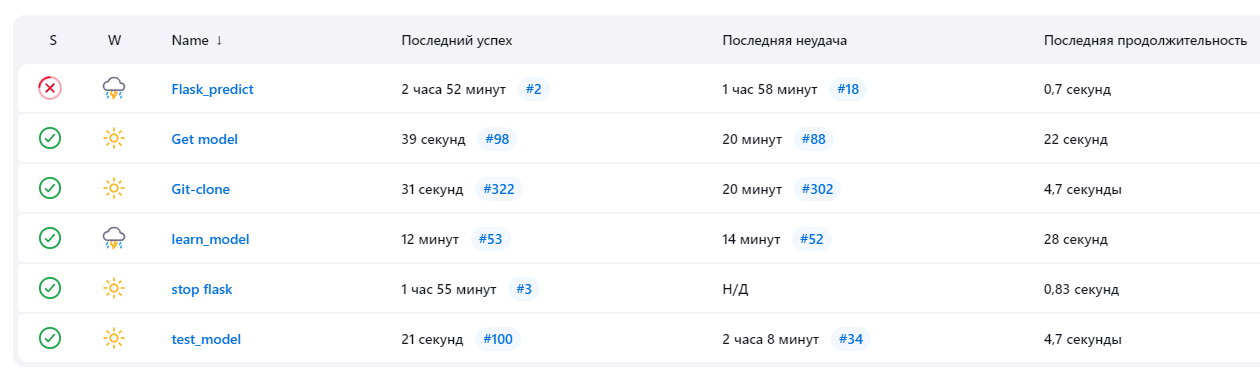
Гипер подбор не реализовывал, также решил не использовать ClearML/AirFlow/MLFlow, так как они тяжёлые и требовательные. А 5 виртуальных машин – это не лёгкий труд для 4 ядер и 12 гб оперативной памяти (файл подкачке в винде не ограничивал, и ssd часто жаловался, что всё пространство занято (12 гб дополнительно имеет вес)).

После одного из выбранного пункта на развилке можно запустить Flask, который при запросе вернёт результат прогноза.

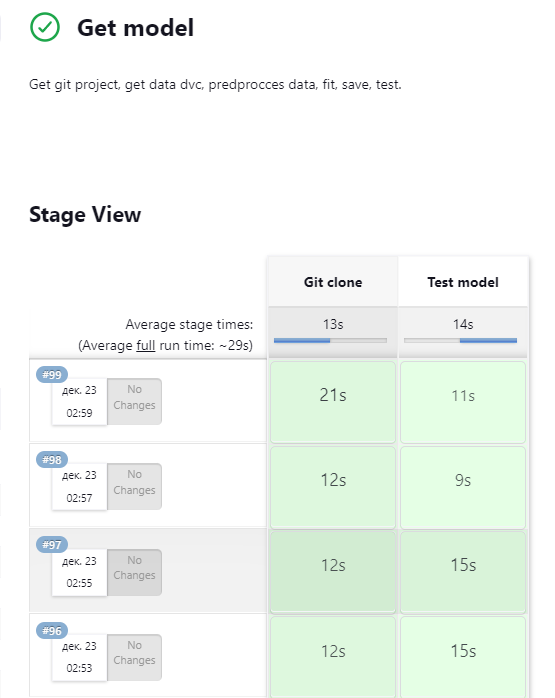
Тем самым создан «почти» автоматический пайплайн от момента получения данных, до запуска API и вывода результатов.

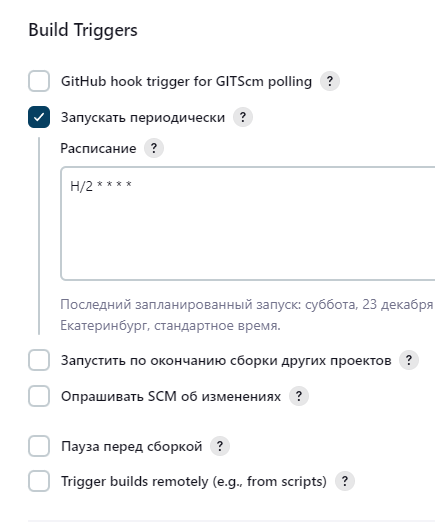
Ниже представлены скриншоты из Jenkins и результат работы FlaskAPI. А также описания .

**Скриншоты:**



Jenkins – job’ы и pipeline.

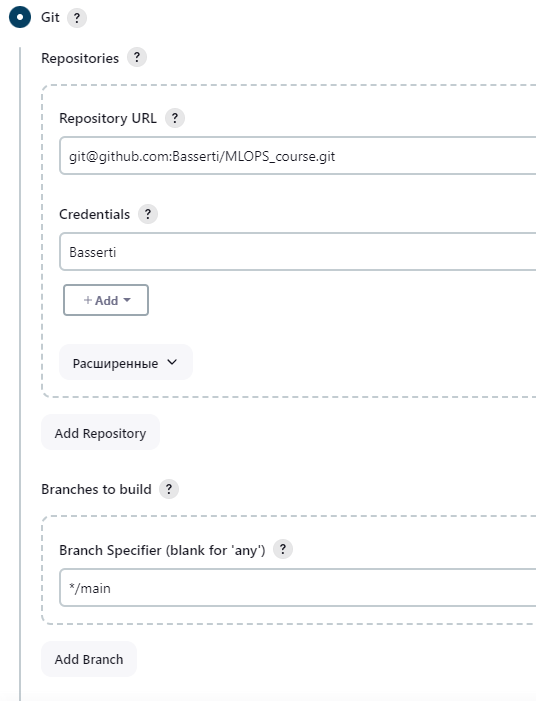


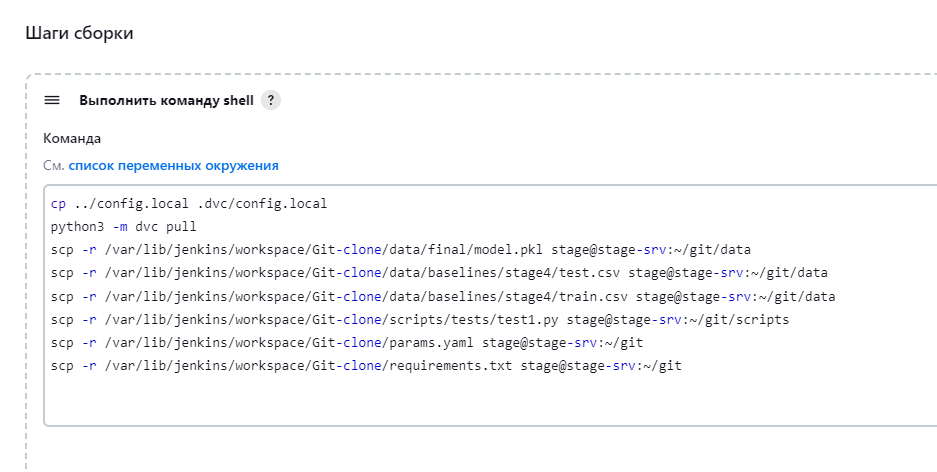


Pipeline запускается часто (только для тестов/в рамках выполнения проекта), обычно раз в день (вечер/ночь).



Pipeline получения модели с гита, загрузкой из локального dvc и тесты модели.

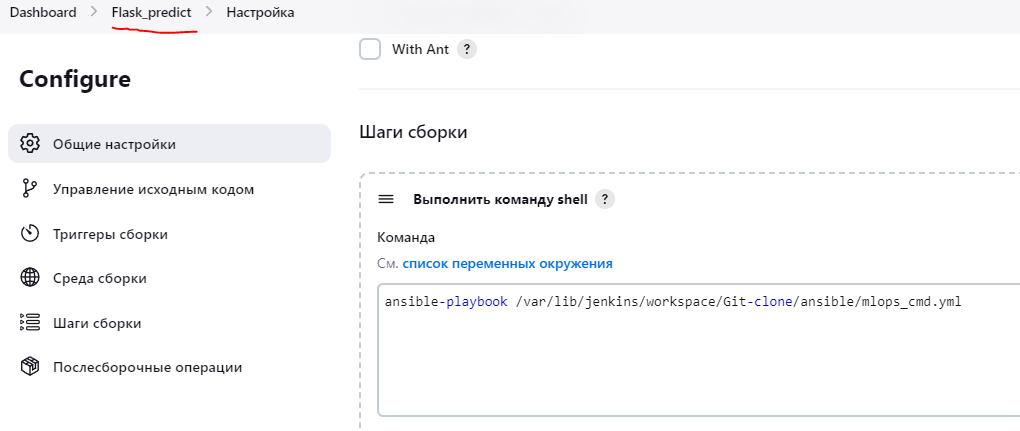




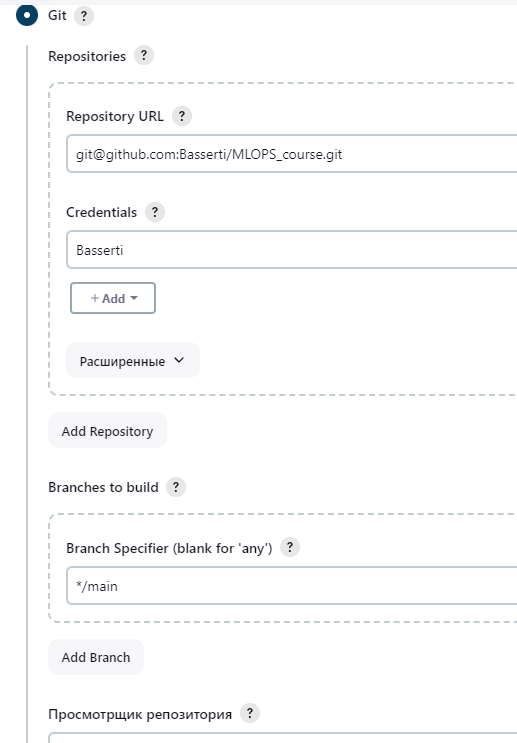
Git-clone. Клонируем гит, закидываем локальный конфиг dvc внутрь, чтобы работал dvc (там пароль от пользователя на машине data-srv)



Test model. Запускаем тест модели на stage-srv



Flask Run







Обучение модели через DVC. Меняем значения в файле params.yaml на гите, после запускаем вручную job в Jenkins. После обучения, модель выгружается в DVC и пушиться на гит (только dvc.lock, для получения модели используется dvc pull).

**DVC**

Файлы для DVC представлены в Git.

**Виртуалки**

* data-srv – DVC
* ml-srv – jupyter (пространство для разработки)
* prod-srv – Flask
* admin-srv – Jenkins
* stage-srv – Тест модели, обучение модели.

**Flask**

Flask выбран в качестве API, которое будет возвращать результат предсказания.

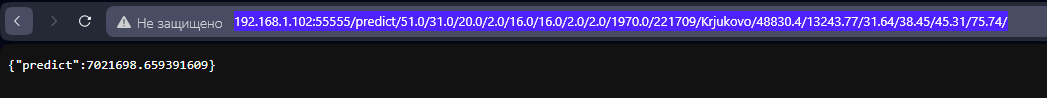
Пример запроса:

```

http://192.168.1.102:55555/predict/51.0/31.0/20.0/2.0/16.0/16.0/2.0/2.0/1970.0/221709/Krjukovo/48830.4/13243.77/31.64/38.45/45.31/75.74/

```

Ответ:



**Итог**

Построен пайплайн обучения с выгрузкой в dvc и git,