# **MLOps как информационная система**

https://selectel.ru/blog/mlops-infosystem/

После разбора проекта появляется соблазн рассматривать MLOps-платформу как простой набор программных ML-компонентов, но лучше этого не делать. Есть великое множество важных аспектов, реализация которых напрямую влияет на работоспособность всех ML-процессов.

В качестве цельной модели можно обратиться к концепции информационной системы.

Если изучить архитектурные аспекты информационных систем, то можно прийти к следующему набору составных элементов:

* hardware (железо),
* software (софт),
* data (данные),
* networks (сетевая связность)
* people (команда),
* procedures (процессы).

Хочется добавить еще один элемент — knowledge, но это больше про сохранение опыта команды в процессе построения и эксплуатации ML-платформы. Одно дело, когда сотрудники построили платформу и успешно ею пользуются. Другое — когда этот опыт можно передать новым сотрудникам и они смогут дальше развивать платформу. Причем будут понимать, какие решения при ее развитии были приняты и почему.

Чтобы детальнее описать всю сложность ML-платформы, необходимо рассмотреть каждый из элементов ИС.

Железо

Далеко не все участники рынка осознанно подходят к выбору аппаратного обеспечения для ML-системы. Особенно это заметно в разрезе выбора GPU. Часто используется то оборудование, на котором построен pipeline, — потом этот паттерн просто перетекает в следующие проекты. Впрочем, это не всегда плохо: например, это обосновано, если переписывать pipeline очень дорого.

Если у небольшой компании есть возможность купить себе сервер для ML, то в большинстве случаев она выберет машину с GPU потребительского сегмента (RTX 2000/3000).

Теперь разберем этот выбор:

* Для не самых ресурсоемких задач такие серверы подойдут.
* Если появляется потребность в большом количестве видеопамяти, сразу возникают сложности. Нужно учитывать, что в память должны поместиться модель и какое-то количество данных для нее. Можно подстроить размер порции данных для обучения или инференса, чтобы уложиться в отведенное место.
* Если вес модели достигает десятков гигабайт (например, компания разрабатывает голосовых роботов) и данные для нее необходимо доставлять, выбранный сервер точно не подойдет. Популярных альтернатив тут три: Nvidia Tesla A30 на 24 ГБ и Nvidia Tesla A100 на 40 и 80 ГБ. Если и этого мало, на помощь придет объединение нескольких GPU через NVLink или NVSwitch.

С памятью GPU немного разобрались. А как подбирать CPU и RAM?

Можно посмотреть на примеры конфигураций серверов у провайдеров, но и это не гарантирует, что они подойдут для специфических задач компании. Опросы компаний показывают, что размер RAM подбирается под размер максимального датасета, чтобы повысить скорость его загрузки в GPU. GPU — самый дорогой ресурс в сервере, поэтому в идеале он не должен простаивать.

CPU чаще всего нужен для предобработки и загрузки данных в RAM. В зависимости от специфики эксперимента могут задействоваться как все ядра CPU, так и малая часть. Некоторые для каждой высокопроизводительной GPU выделяют от 12 до 24 vCPU, а кому-то хватает и 8. Точное число ядер можно получить только экспериментами.

Дополнительную головную боль, опять-таки, привносит необходимость пользоваться GPU в Kubernetes, так как он де-факто является стандартом для реализации ML-систем. Головная боль заключается в необходимости (в понятиях Kubernetes) превратить GPU в доступный ресурс. Без этого пользоваться видеокартой будет невозможно. Для этого необходимо разобраться с Nvidia GPU Operator или его отдельным компонентом — Device Plugin. И раз до этого дошло, хорошо бы еще правильно аннотировать ноды лейблами. Даже не все Kubernetes-админы про такое слышали.

Софт

Разговор про программное обеспечение необходимо разделить на несколько частей:

* ML-специфичные программы,
* вспомогательные программы.

Под ML-специфичными понимаются программные решения, реализующие части функциональностей компонентов схемы MLOps. Описать ML-часть программными компонентами можно так.

Процесс экспериментирования можно выстроить с помощью любого из решений:

* ClearML,
* Kubeflow,
* MLFlow,
* guild.ai,
* polyaxon и т.д.

Автоматизацию переобучения моделей можно реализовать с помощью

* Kubeflow,
* ClearML,
* Airflow,
* Dagster,
* Prefect и т.д.

Serving и Deploy моделей можно реализовать с помощью:

* Seldon Core,
* KServe,
* BentoML,
* и т.д.

Если проект уже требует продуманной работы с фичами, самым популярным решением является [Feast](https://feast.dev/%22%20%5Ct%20%22_blank), у которого есть [альтернативы](https://www.featurestore.org/). Впрочем, чаще компании пишут что-то свое, так как производительности готовых решений недостаточно.

Получается, есть целый стек ML-компонентов, которые нужно уметь настраивать, поддерживать и эксплуатировать. Часть про «настраивать и поддерживать» требует использования вспомогательного ПО.

Например:

* Kubernetes — в качестве среды оркестрации контейнеров.
* KeyCloak — в качестве единой точки авторизации пользователей.
* Grafana, Prometheus, Loki — в качестве средств мониторинга и сбора логов.
* Traefik — в качестве средства управления трафиком кластера.
* Балансировщик нагрузки для входящих запросов в кластер и пр.

Помимо этого, не следует забывать про различные сертификаты безопасности, доменные имена, базы данных для используемых компонентов, каскадные кэши для датасетов, файловые шары и многое другое.

На изучение и настройку всего в рамках одной системы может уйти очень много времени и сил. Просто поставить на сервере ClearML не поможет.

Данные

И снова про данные. В этом блоке логично сослаться на различные реализации существующих платформ хранения данных, чтобы интересующиеся смогли вместе с MLOps познать модный нынче терминологический словарь Data-инженеров.

Итак, данные для ML могут храниться в сложных архитектурах:

* Data Warehouse,
* Data Lake,
* Data Lakehouse,
* Data Mesh и т.д.

Чтобы построить их, требуется огромное количество ресурсов, так что в этой статье только про MLOps.

Сетевая связность

Сейчас кластеры Kubernetes довольно часто расположены у какого-нибудь провайдера. Есть команды, у которых in-house-инфраструктура, но и там она в отдельной серверной, достаточно удаленной от команды.

Проблема в том, что данные, на которых обучается модель, должны загружаться на конкретный сервер, где происходит непосредственно процесс обучения. А если сетевой канал недостаточно широкий, приходится долго ждать загрузки необходимых данных. Это сильно увеличивает время ожидания результатов экспериментов.

Получается, нужно придумывать различные сетевые архитектуры, чтобы данные передавались быстрее, или внедрять  кэши в кластер. Если данные in-house, а вычислительные ресурсы у провайдера, то может потребоваться отдельный сетевой канал, чтобы сохранить производительности при такой распределенной архитектуре.

Процедуры

Здесь нужны актуальные инструкции и регламенты, по которым будет работать команда и учиться новые участники.

Тут вспоминается ITIL со своими лучшими практиками управления IT-сервисами, но альтернативы в мире MLOps пока нет. Если CRISP-DM уже выглядит устаревшим, то [CRISP-ML](https://arxiv.org/pdf/2003.05155.pdf) пока недостаточно развит. Можно найти его перечень процессов, но подробного описания самих процессов недостаточно.

Для начала можно предложить подумать про следующие регламенты:

* версионирования кода (модели, эксперименты, workflows),
* версионирования датасетов,
* версионирования файлов моделей,
* версионирование environments,
* логирование метрик,
* использование  инфраструктуры,
* мониторинга метрик моделей.

В любом случае, каждая команда выстраивает собственные адаптированные процессы, даже если изначально пользуется готовым подходом.

Команда

Если в команде нет выделенной роли, которая отвечает за работоспособность стека ML-технологий, то не стоит отчаиваться -– сейчас это пока норма. MLOps-инженеры пока еще могут считаться единорогами, почти как Angular-разработчики.

На сегодняшний день чаще всего задачи внедрения ML-систем отдают DevOps-специалистам. Если они отвечают за CI/CD, пусть еще и ML-пайплайны себе заберут. Естественно, у подхода есть минусы:

* деплой ML-моделей отличается от деплоя кода,
* Serving и Deploy требуют знаний специфических инструментов,
* нужно разбираться в Kubernetes на высоком уровне, а такие специалисты и без ML на вес золота,
* нужно использовать IaaС и дружить с Terraform,
* вишенкой является процесс Continuous Training, который подразумевает написание Python-скриптов для автоматизации взаимодействия разных компонентов в оркестраторах.

Как минимум нашему DevOps-специалисту потребуется много времени на изучение всех процессов и технологий. В одной версии идеального мира MLOps-специалист — следующая стадия развития DevOps-инженера. Когда уже все Terraform-файлы написал, сконфигурировал с помощью Ansible и в Kubernetes через Argo CD запустил.

Полезно, когда MLOps-специалист понимает мир ML-разработки, знает сложности и может аргументированно корректировать пайплайны. Таким образом, в другой версии идеального мира MLOps-инженер — ML-разработчик, который не только ML-модели готов обучать, но и с инфраструктурой разбираться.

На данный момент идеальный MLOps-инженер представляется как «воин дракона»: ученик, учитель, data scientist, backend developer, ML-инженер, data-инженер, devops, software developer и остальное в одном человеке. Только где такого найти?

Мы рассмотрели концепцию MLOps — через элементы информационной системы.