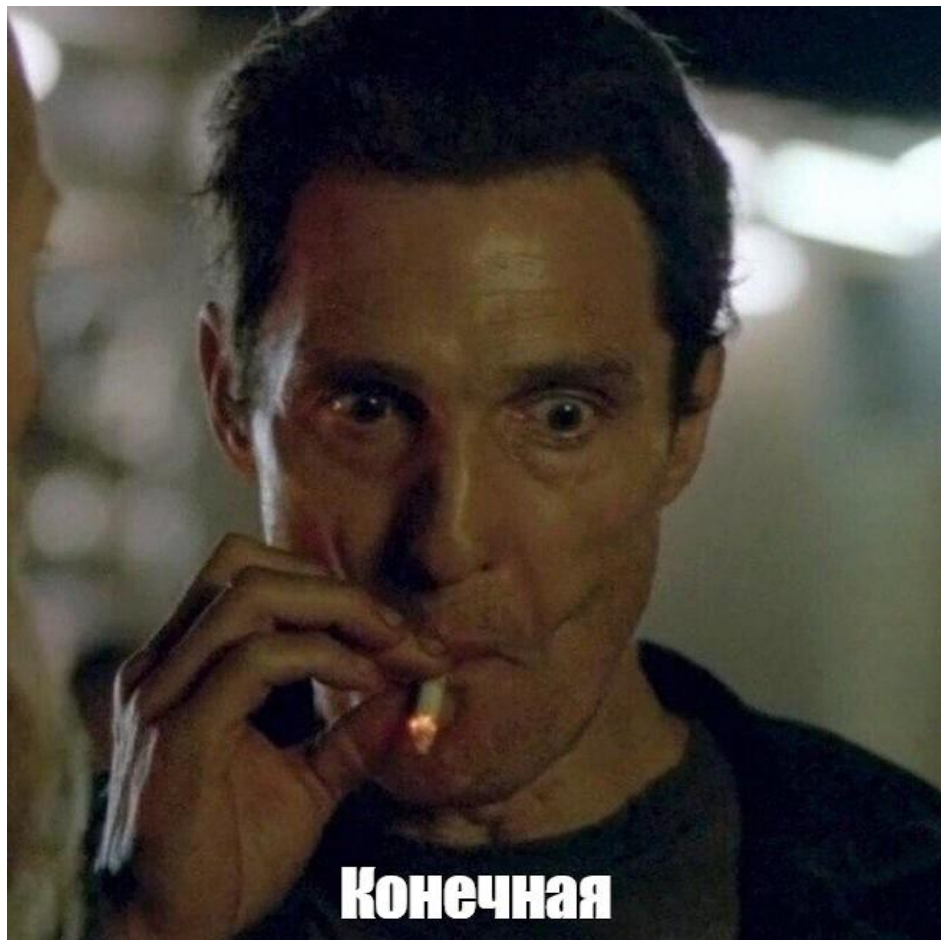


Feature Store, Triton, архитектура



Борисенко Глеб

ФТиАД2022

Конечная ☹

Feature Store

Широкие витрины

“Одноразовые”
датасеты и фичи

vs.

Feature Store

Общий каталог
фичей-преагрегатов

Сложное ведение
документации

vs.

Работа с данными и
документирование в
одном окне

Нет четкого разделения
ролей участников

vs.

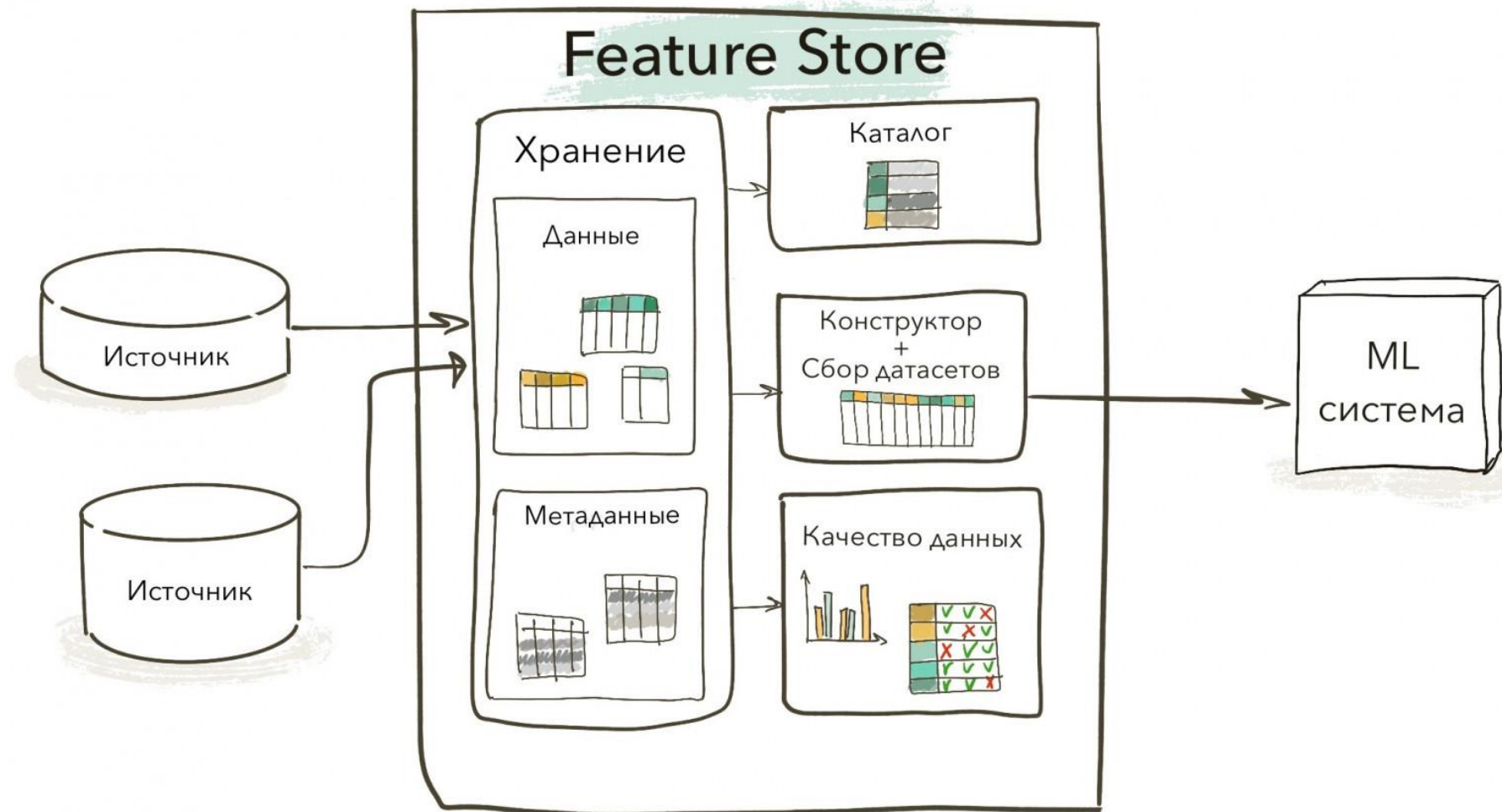
DE и DS по разные
стороны “прилавка”:
первые готовят фичи и
поддерживают работу “магазина”;
вторые — выступают в роли
потребителей

Нерациональное
использование ресурсов

vs.

Благодаря преагрегатам,
процесс перестроения
датасетов занимает меньше
ресурсов

Еще картинка



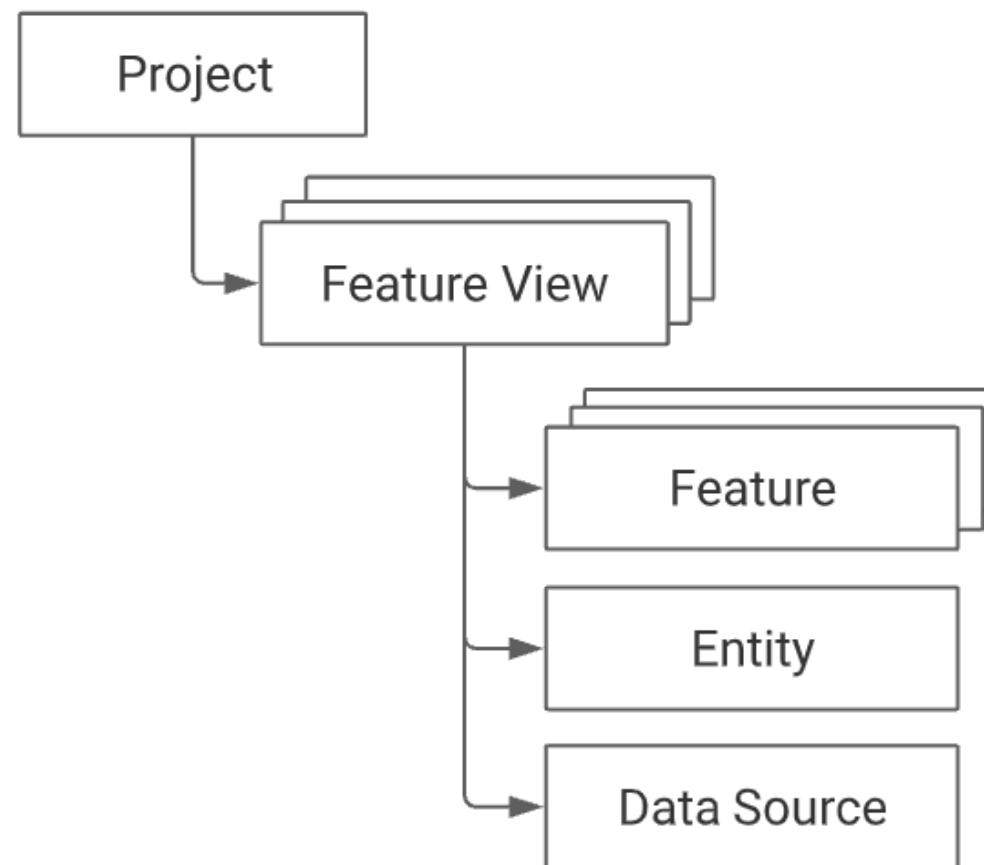
Что за предагрегат?

- Фичи по возможности хранятся в максимально атомарном виде.
- Фичи сгруппированы с некоторой минимально-необходимой гранулярностью: данные агрегируются за день/неделю/месяц.

Feast

- Feast is likely not the right tool if you:
 - are in an organization that's just getting started with ML and is not yet sure what the business impact of ML is
 - Rely primarily on unstructured data
 - need very low latency feature retrieval (e.g. p99 feature retrieval << 10ms)
 - have a small team to support a large number of use cases

Концепты Feast

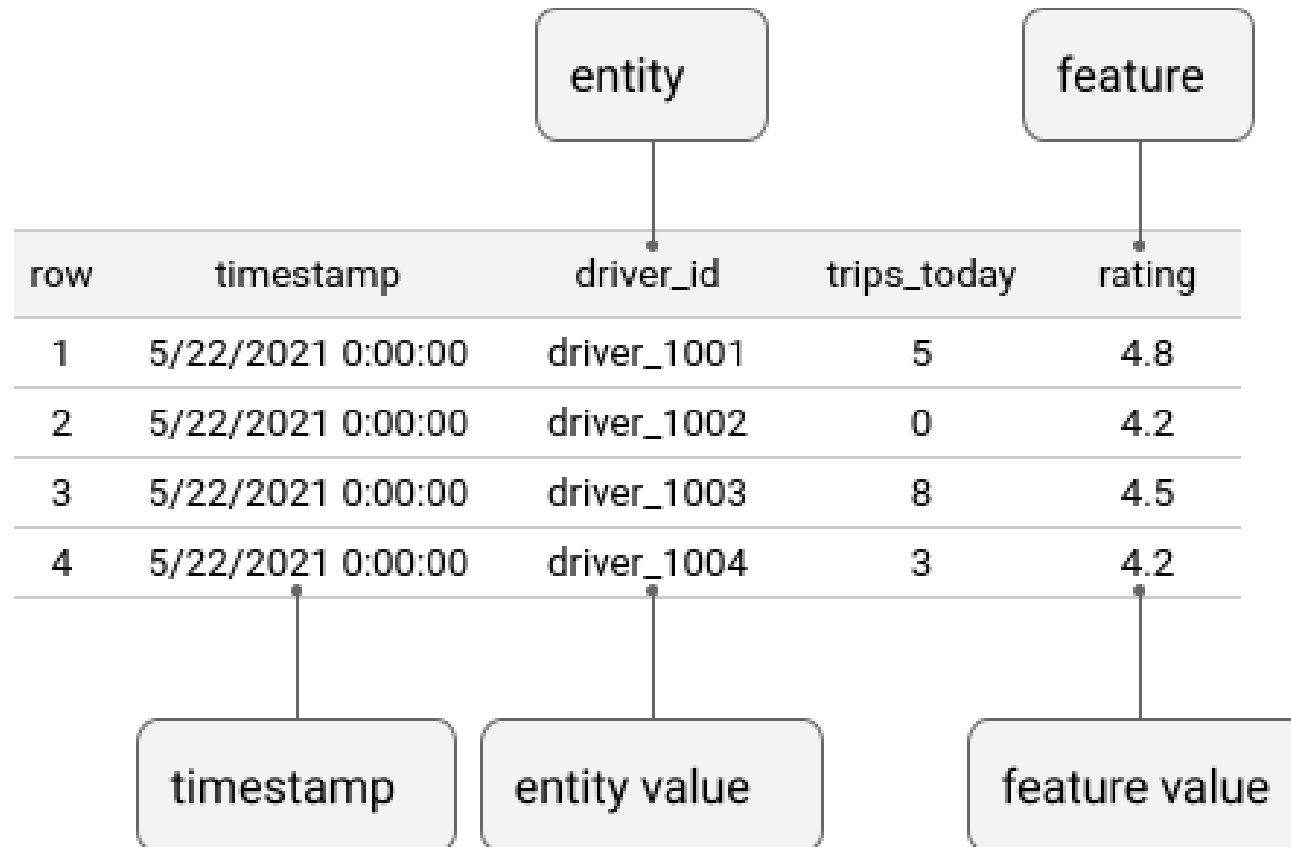


Entity

entity key

row	timestamp	customer_id	driver_id	trips_today	rating
1	5/22/2021 0:00:00	customer_A	driver_1001	5	4.8
2	5/22/2021 0:00:00	customer_B	driver_1002	0	4.2
3	5/22/2021 0:00:00	customer_C	driver_1003	8	4.5
4	5/22/2021 0:00:00	customer_D	driver_1004	3	4.2

Feature View



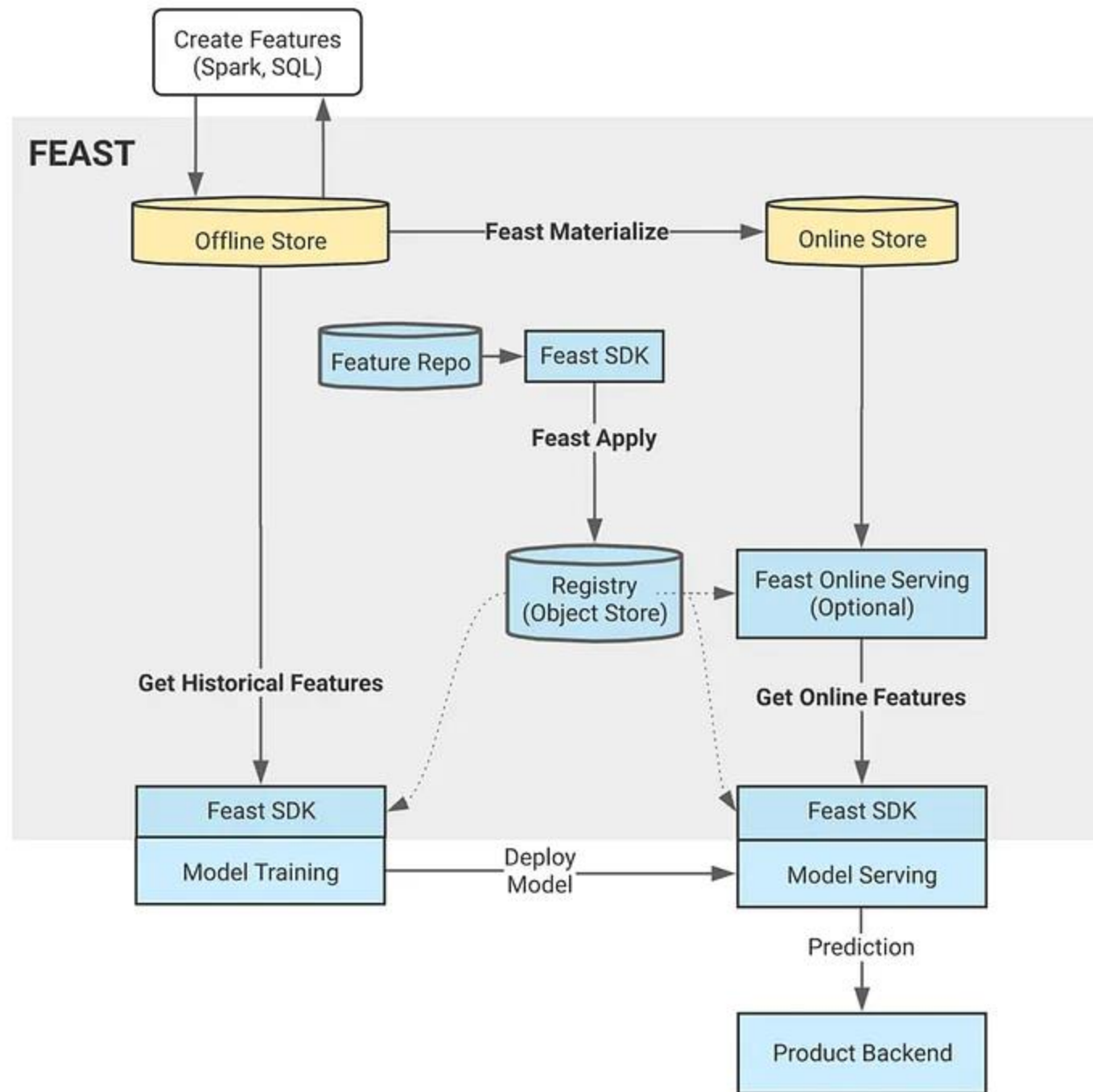
Feature View

```
from feast import BigQuerySource, Entity, FeatureView, Field
from feast.types import Float32, Int64

driver = Entity(name="driver", join_keys=["driver_id"])

driver_stats_fv = FeatureView(
    name="driver_activity",
    entities=[driver],
    schema=[
        Field(name="trips_today", dtype=Int64),
        Field(name="rating", dtype=Float32),
    ],
    source=BigQuerySource(
        table="feast-oss.demo_data.driver_activity"
    )
)
```

Опять картинка



Feature Service и получение фичей

```
from driver_ratings_feature_view import driver_ratings_fv
from driver_trips_feature_view import driver_stats_fv

driver_stats_fs = FeatureService(
    name="driver_activity",
    features=[driver_stats_fv, driver_ratings_fv[["lifetime_rating"]]]
)
```

```
from feast import FeatureStore
feature_store = FeatureStore('.') # Initialize the feature store

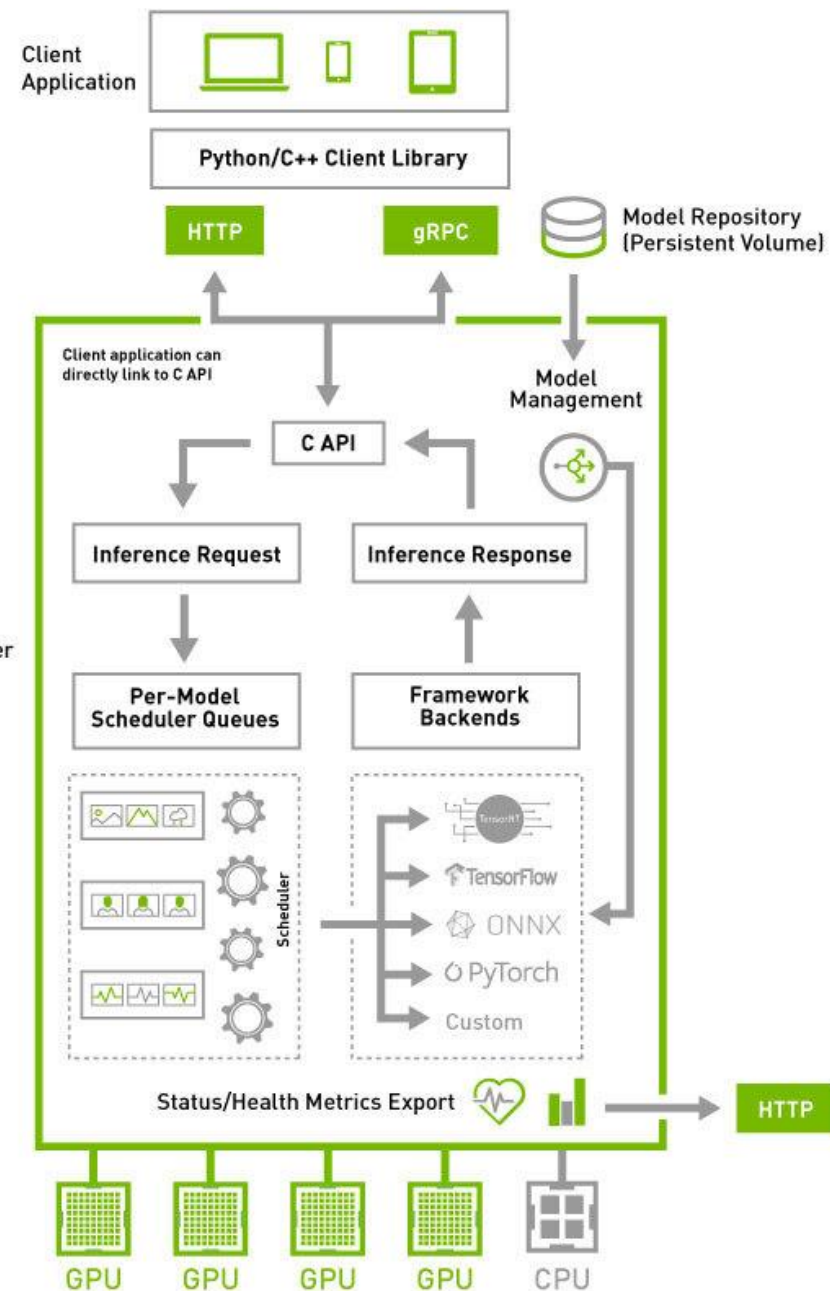
feature_service = feature_store.get_feature_service("driver_activity")
feature_store.get_historical_features(features=feature_service, entity_df=entity_df)
```

Прямое получение фичей

```
entity_df = pd.DataFrame.from_dict(  
    {  
        "driver_id": [1001, 1002, 1003, 1004, 1001],  
        "event_timestamp": [  
            datetime(2021, 4, 12, 10, 59, 42),  
            datetime(2021, 4, 12, 8, 12, 10),  
            datetime(2021, 4, 12, 16, 40, 26),  
            datetime(2021, 4, 12, 15, 1, 12),  
            datetime.now()  
        ]  
    }  
)  
training_df = store.get_historical_features(  
    entity_df=entity_df,  
    features=store.get_feature_service("model_v1"),  
)  
.to_df()  
print(training_df.head())
```

Triton Inference Server

NVIDIA Triton Inference Server

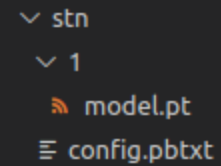


Фичи

- Параллельный запуск моделей
- Поддержка кучи фреймворков
- Динамические батчи
- Некоторые бэкэнды освобождают память, когда не используются
- Есть последовательные батчи
- Сервисы (модели) могут быть Stateful

Конфиг модели torchscript

```
backend: "pytorch"
max_batch_size: 32
input [
  {
    name: "input__0"
    data_type: TYPE_FP32
    dims: [ 3, 24, 94 ]
  }
]
output [
  {
    name: "output__0"
    data_type: TYPE_FP32
    dims: [ 3, 24, 94 ]
  }
]
```



```

  ▾ stn
    ▾ 1
      📄 model.pt
      ≡ config.pbtxt
```

Python backend

```
def initialize(self, args):
    self.model_config = json.loads(args["model_config"])
    self.model = load_yolo(
        settings.YOLO.WEIGHTS, settings.YOLO.CONFIDENCE, torch.device("cuda")
    )
```

```
backend: "python"
max_batch_size: 0
input [
  {
    name: "input__0"
    data_type: TYPE_UINT8
    dims: [ -1, -1, 3 ]
  }
]
output [
  {
    name: "output__0"
    data_type: TYPE_FP32
    dims: [ -1, 3, 24, 94 ]
  },
  {
    name: "output__1"
    data_type: TYPE_FP32
    dims: [ -1, 4 ]
  }
]
```

```
def execute(
    self, requests: List[pb_utils.InferenceRequest]
) -> List[pb_utils.InferenceRequest]:

    responses = []

    for request in requests:
        image = pb_utils.get_input_tensor_by_name(request, "input__0").as_numpy()
        image = prepare_detection_input(image)

        detection = self.model(image, size=settings.YOLO.PREDICT_SIZE)

        df_results = detection.pandas().xyxy[0]
        img_plates = prepare_recognition_input(
            df_results, image, return_torch=False
        )

        out_tensor_0 = pb_utils.Tensor(
            "output__0", img_plates.astype(self.output0_dtype)
        )
        out_tensor_1 = pb_utils.Tensor(
            "output__1",
            df_results[["xmin", "ymin", "xmax", "ymax"]].
            to_numpy().
            astype(self.output1_dtype),
        )

        inference_response = pb_utils.InferenceResponse(
            output_tensors=[out_tensor_0, out_tensor_1]
        )
        responses.append(inference_response)

    return responses
```


Граф вычислений

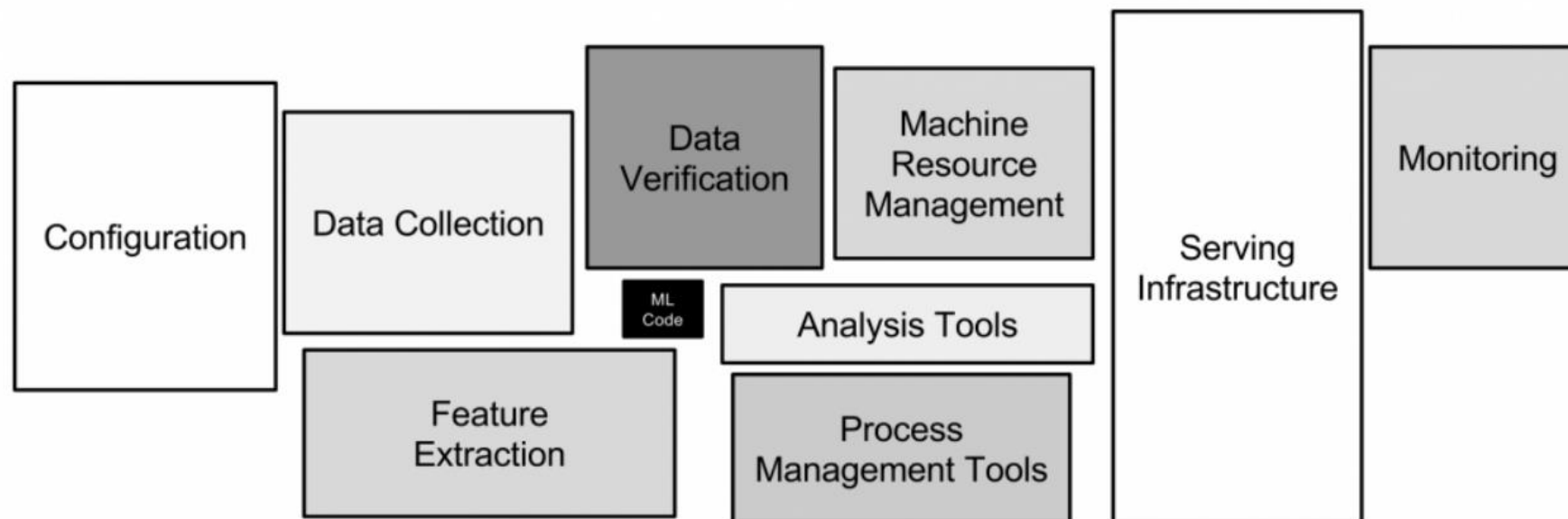
```
cropped_images, coordinates = self.predict(  
    model_name="yolo",  
    ...  
)  
  
num_plates = from_dlpack(coordinates.to_dlpack()).shape[0]  
if num_plates == 0:  
    ...  
    continue  
  
(plate_features,) = self.predict(  
    model_name="stn",  
    ...  
)  
  
(text_features,) = self.predict(  
    model_name="lprnet",  
    ...  
)
```

```
backend: "python"  
max_batch_size: 0  
input [  
    {  
        name: "input__0"  
        data_type: TYPE_UINT8  
        dims: [ -1, -1, 3 ]  
    }  
]  
output [  
    {  
        name: "coordinates"  
        data_type: TYPE_FP32  
        dims: [ -1, 4 ]  
    },  
    {  
        name: "texts"  
        data_type: TYPE_STRING  
        dims: [ -1, 1 ]  
    }  
]
```

Вы успешны

Model	Version	Status
lprnet	1	READY
plate_recognition	1	READY
stn	1	READY
yolo	1	READY

Архитектура



Все это вместе образует **Систему**.

Архитектура - это устройство таких систем.

Зачем нам думать об этом?

0. Архитектура описывает конечный предмет, который мы должны построить
1. Выбор архитектуры диктует слабые и сильные стороны решения и пути его модификации в будущем
2. Плохая архитектура породит технический долг и приведет к проблемам в её работе, поддержке, и доработке
3. Для эффективной работы в команде нужно общее понимание, как именно устроена ваша система

Best Practices for ML Engineering by Martin Zinkevich

- To make great products: **do machine learning like the great engineer you are, not like the great machine learning expert you aren't .**

От требований к архитектуре

- Требования к расчету фичей
 - на лету или предварительно?
 - с помощью каких программных средств?
- Требования к обучению
 - как часто?
 - на каких ресурсах?
- Требования к выдаче прогнозов
 - каким способом?
 - как быстро?
 - минимальное допустимое качество?
- Tradeoff скорость/надежность разработки
- Требования к автономности
- Требования к доступной кастомизации
- Требования к обработке приватных данных

И прочее, прочее, прочее

Препроцессинг фичей

- Обычно есть два подхода:
 - Считаем фичи в режиме реального времени
 - Требуется рассчитывать некоторые фичи заранее
- Использование имеющихся источников порождает задачи интеграции и накладывает на них ограничения - например, по времени ответа.
- Feature Store здесь очень помогает

Обучение

- Наиболее распространено переобучение по расписанию - запускаем код с помощью cronjob, планировщиков задач
- Еще вариант – обучение на лету (например, с помощью Kafka, Flint)

Выдача прогнозов - сервис

- Как сервис:
 - Просто: Пишем REST API обертку на питоне
 - Быстро: Переписываем код на Go/C++, используем gRPC
- Как зависимость
- С Precomputation-ом
- По требованию (через брокер и очереди)

Когда ML - дополнение

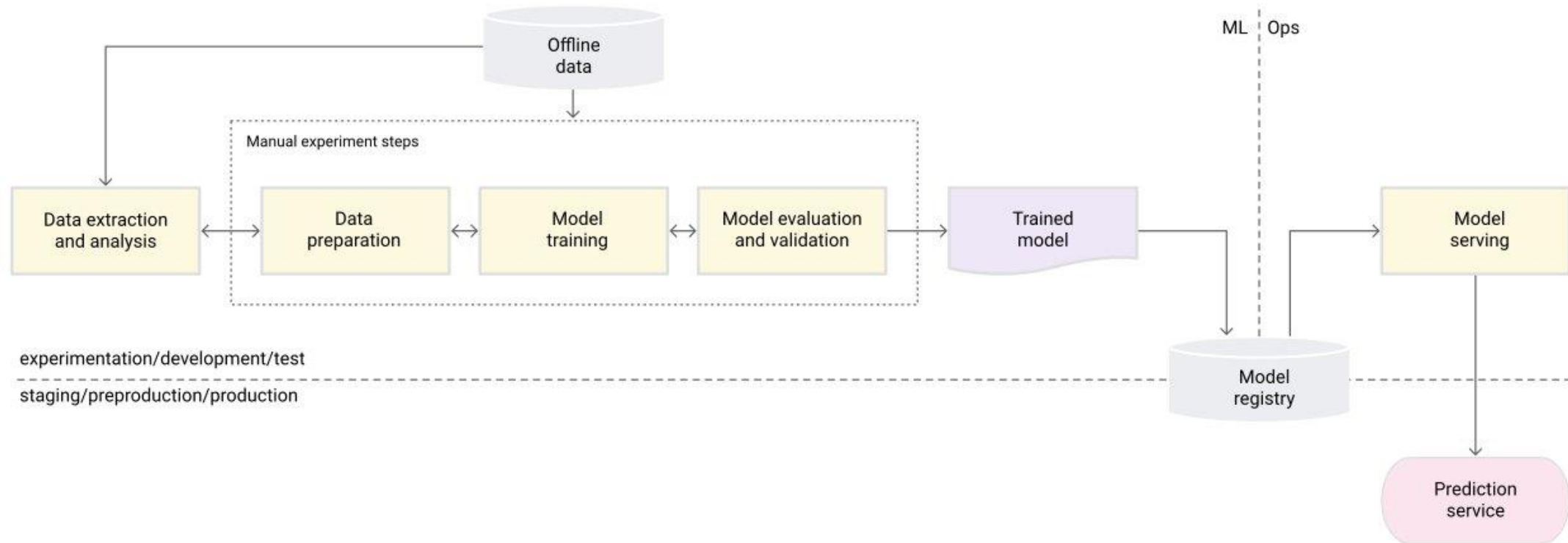
- Часто ML-системы возникают не на пустом месте, а поверх уже существующей инфраструктуры:
 - 1. Уже есть Data Warehouse
 - 2. Уже есть бэкенд и устоявшиеся подходы к разработке
- В таком случае типично использование имеющихся инструментов и подходов.
- Пример - использование Airflow для запуска пайплайнов по обучению моделей, когда он используется в компании для ETL.
- Или деплой на AWS Sagemaker/OpenShift/Heroku, когда они используются разработчиками, и так далее.
- Это может быть как плохо (бывают неудобные инструменты), так и хорошо (не нужно возводить все с нуля).

Итак

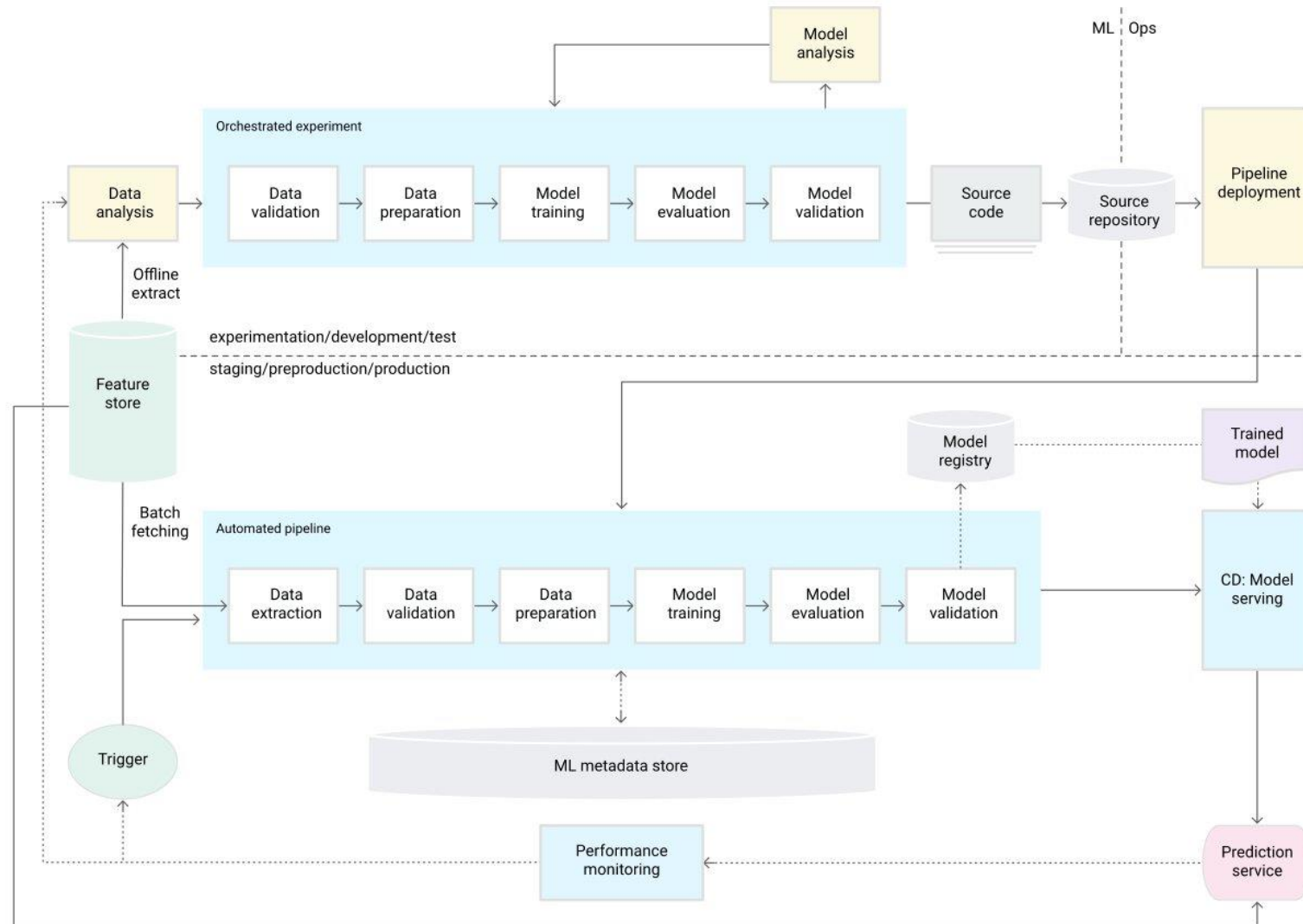
- Архитектура - это описание и устройство программных систем
- Начинать разговор об архитектуре удобно с набора требований к системе. Одни из основных ML-специфичных - это требования к расчету фичей, к обучению моделей и к выдаче прогнозов.

MLOps пайплайн (назад в прошлое)

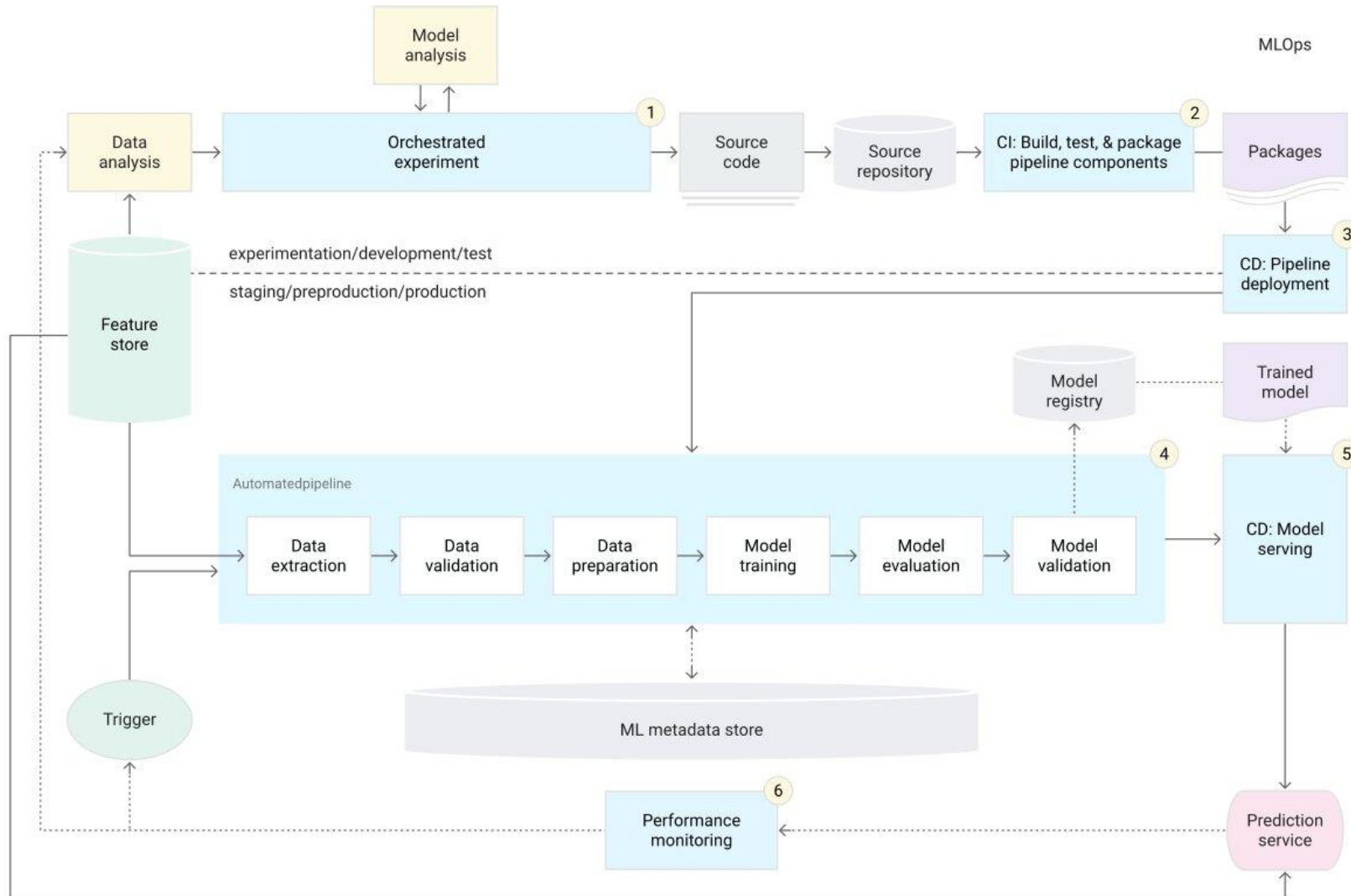
MLOps level 0: Manual process



MLOps Level 1: автоматизация ML пайплайна



MLOps Level 2: автоматизация CI/CD пайплайна



Спасибо вам за ваше внимание и интерес на курсе :)

- Спасибо, что слушали :3
- Домашки обязательно тщательно допроверяю в ближайшее время, что у вас было достаточно времени для их правки
- Ооочень жду комментариев, критики, оценки от вас (можно и желательно в личку в телеге, не стесняйтесь писать правду :D)
- Еще раз спасибо :3