Дизайн систем машинного обучения

8. Развертывание

https://ods.ai/tracks/ml-system-design-23

План курса

1) ~~Машинное обучение на практике~~
2) ~~Основы проектирования ML-систем~~
3) ~~Обучающие данные (лекция 3 из 2022 г)~~
4) ~~Подготовка и отбор признаков (лекция 4 из 2022 г)~~
5) ~~Выбор и обучение ML-модели (лекция 5 из 2022 г)~~
6) ~~Оценка качества модели (лекция 7 из 2022 г)~~
7) ~~Улучшение модели через данные~~
8) **Развертывание — Вы находитесь здесь**
9) Диагностика ошибок и отказов
10) Жизненный цикл модели
11) Потоковые данные
12) Языковые модели в продуктовом окружении
13) Временные ряды и графы
14) Безопасность, этика и восстание машин
15) Интеграция в бизнес-процессы

Развертывание модели

- Данные
  - Где хранятся данные
  - Как данные готовятся
- Вычисления
  - Где конкретно будет работать ML-модель
  - Когда будут рассчитываться предсказания модели
- Коммуникация
  - Как модель получит запрос пользователя
  - Как пользователь получит ответ модели

Offline vs Real-time data

- Offline data у вас уже есть
  - Можете очистить, преобразовать
  - Заранее посчитать эмбеддинги
  - Пользователь прислал данные, можно обработать их потом
- Real-time data
  - Пользователь прислал данные, нужно обработать их сейчас

Online vs Batch vs Streaming processing

- Online Processing (by demand)
  - Пользователь присылает один запрос
  - Модель делает предсказание
- Batch processing
  - Собираем много запросов
  - Обрабатываем их вместе
- Streaming processing
  - Пользователь постоянно присылает данные
  - Модель постоянно выдает предсказания

Cloud vs Edge computing

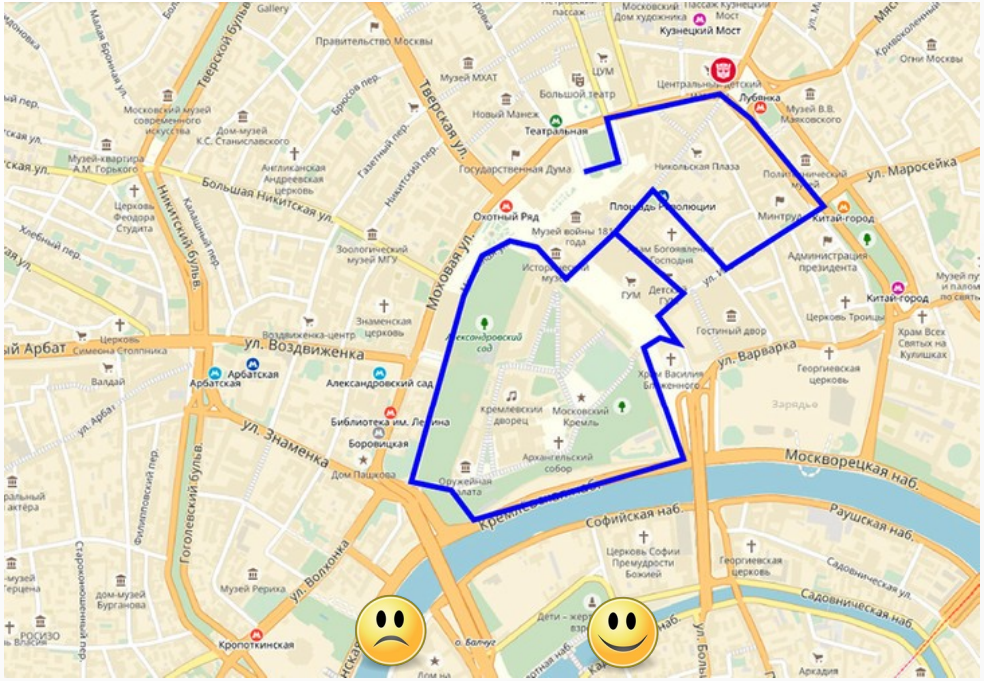
- Cloud computing
  - Модель работает на выделенных серверах
    в облаке или вашем датацентре, постепенно разница стирается
- Edge computing →
  - Модель работает на конечном устройстве
  - Например, на телефоне пользователя
  - Или на компьютере пользователя
  - Или внутри «умной колонки»
  - Или внутри камеры видеонаблюдения
  - Или в браузере - см Tensorflow JS

Задача

- Делаем приложение, рекомендующее маршрут для прогулки
- На основе данных
  - Последние посты пользователя в соцсетях + новости
  - Текущее местоположение, история местоположений
  - Профиль пользователя, календарь пользователя
  - Прогноз погоды, прогноз дорожных пробок
  - История рекомендаций и оценок
- Предлагаем маршрут прогулки

Пример: Маршрут прогулки

- Запускаем
- Видим карту
- Нравится:
  запускаем навигатор
- Не нравится:
  предлагаем еще



https://www.elle.ru/stil-zhizni/10-idealnyih-peshehodnyih-marshrutov-po-moskve/

Варианты развертывания модели

- Онлайн рекомендации (по запросу):
  - Локально, телефон рассчитывает маршрут
  - Запрос на сервер, сервер рассчитывает маршрут
- Пакетная рекомендация
  - Заранее считаем 10 рекомендаций для каждого пользователя
  - Загружаем на телефон при старте приложения
  - Или отдаем по запросу, как будто это онлайн рекомендация

Пример: Локальные рекомендации

- Безопасно — данные пользователя не передаются
- Может работать без интернета
- Дешево — пользователь сам покупает телефон

- Ограничения по вычислительной мощности
- Ограничения по месту
- Нет доступа к нашим данным на сервере
- Трудно обновлять модели

Пример: Онлайн рекомендации

- Большая пропускная способность:
  можем купить много серверов
- Сохраняем в тайне наши модели
- Собираем статистику и разметку
- Можем использовать ChatGPT ;-)

- Нужно платить за сервера
- Неравномерная нагрузка — все идут гулять в обед и вечером
- Если связь медленная, пользователю придется ждать

Пример: Пакетные рекомендации

- Сервер отвечает быстро
- Задержка модели не так важна — считаем все заранее

- Приходится считать много рекомендаций
- Много места для хранения рекомендаций
- Большая часть из них окажется не нужна
- Не можем мгновенно учитывать изменения — например, дождь, перекрытые дороги, аварии

Пример: Потоковые рекомендации

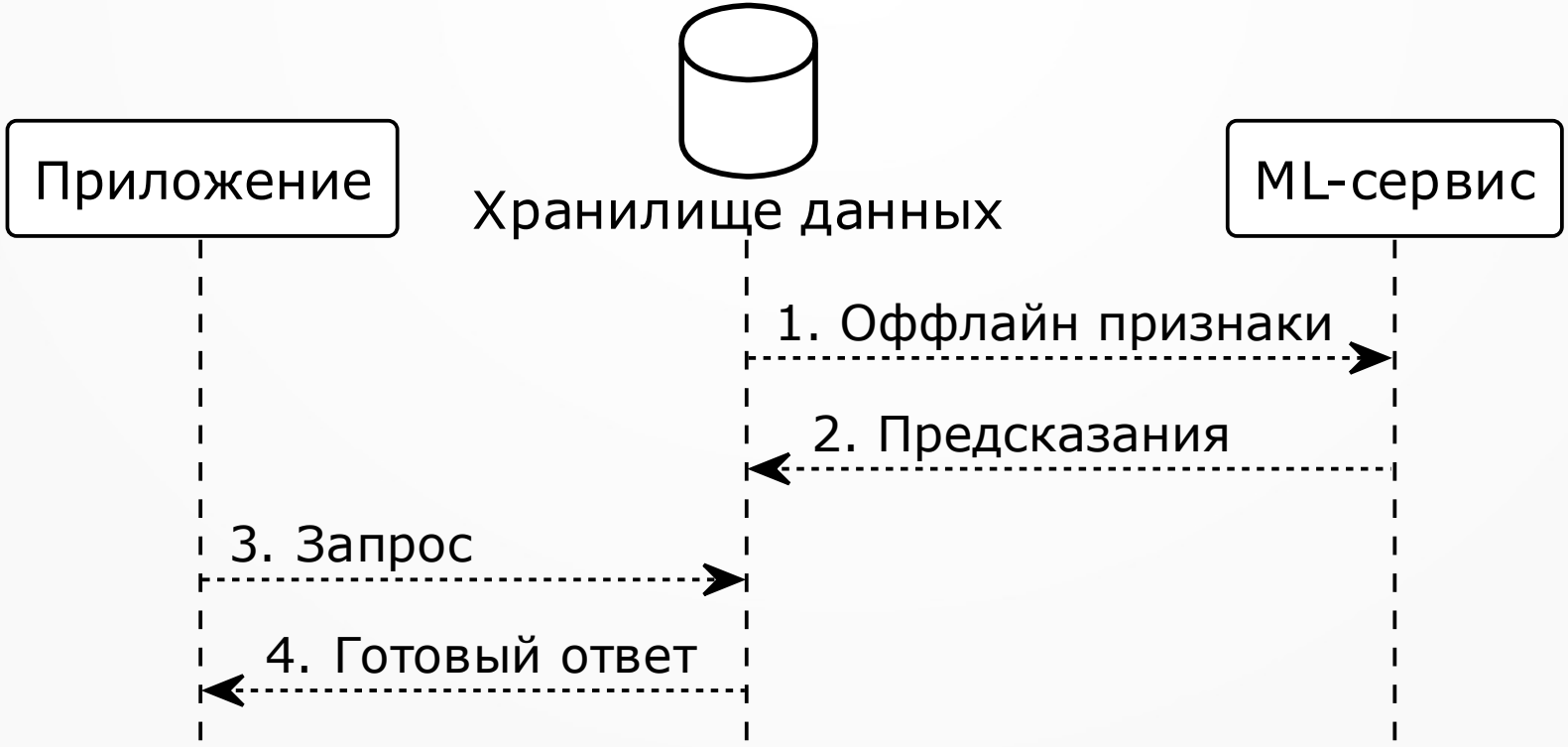
- Координаты передаются непрерывно
- Маршрут перестраивается на ходу
- Учитываем все новые данные

- Большая нагрузка на сеть
- Большая вычислительная нагрузка

Шаблоны реализации

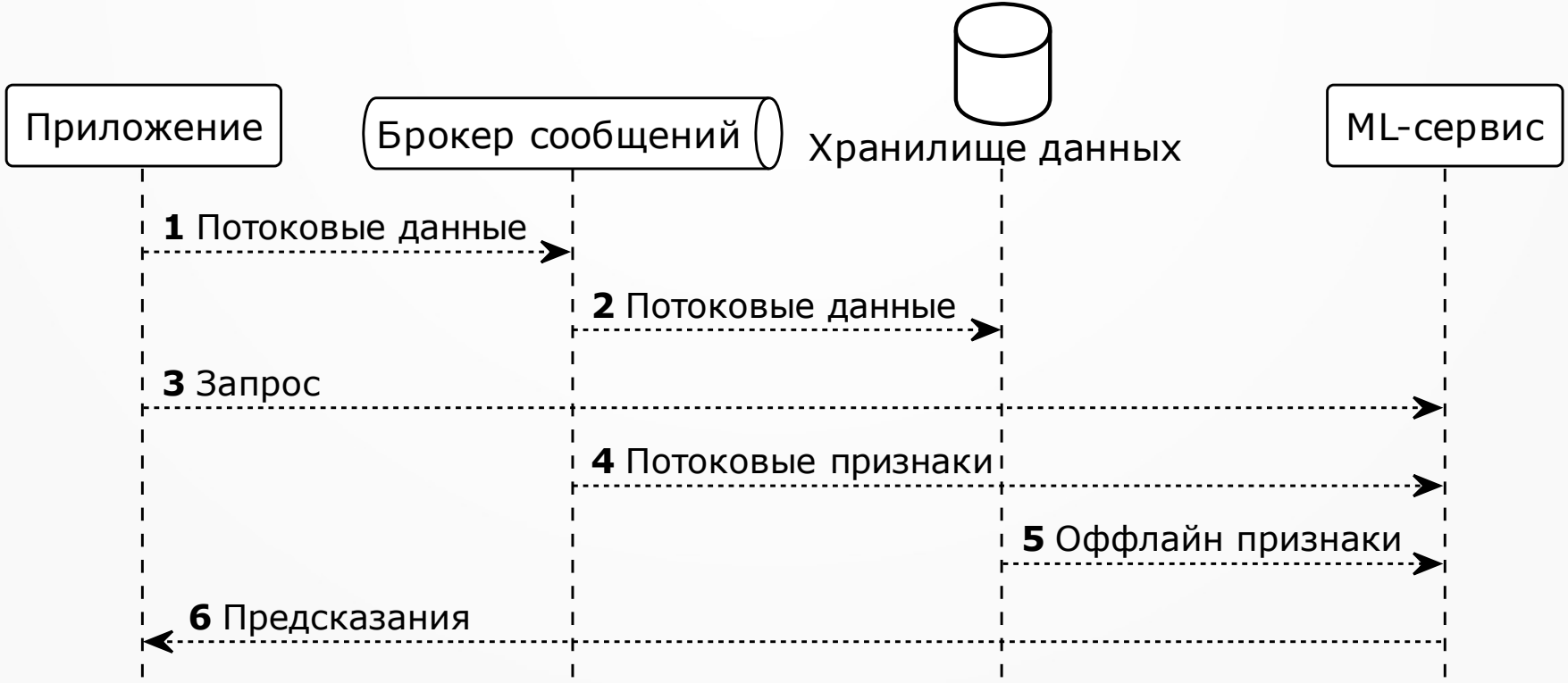
- Online Processing,
  offline data
- Online Processing,
  offline data + real-time data
- Streaming processing,
  offline data + real-time data

Пакетная обработка

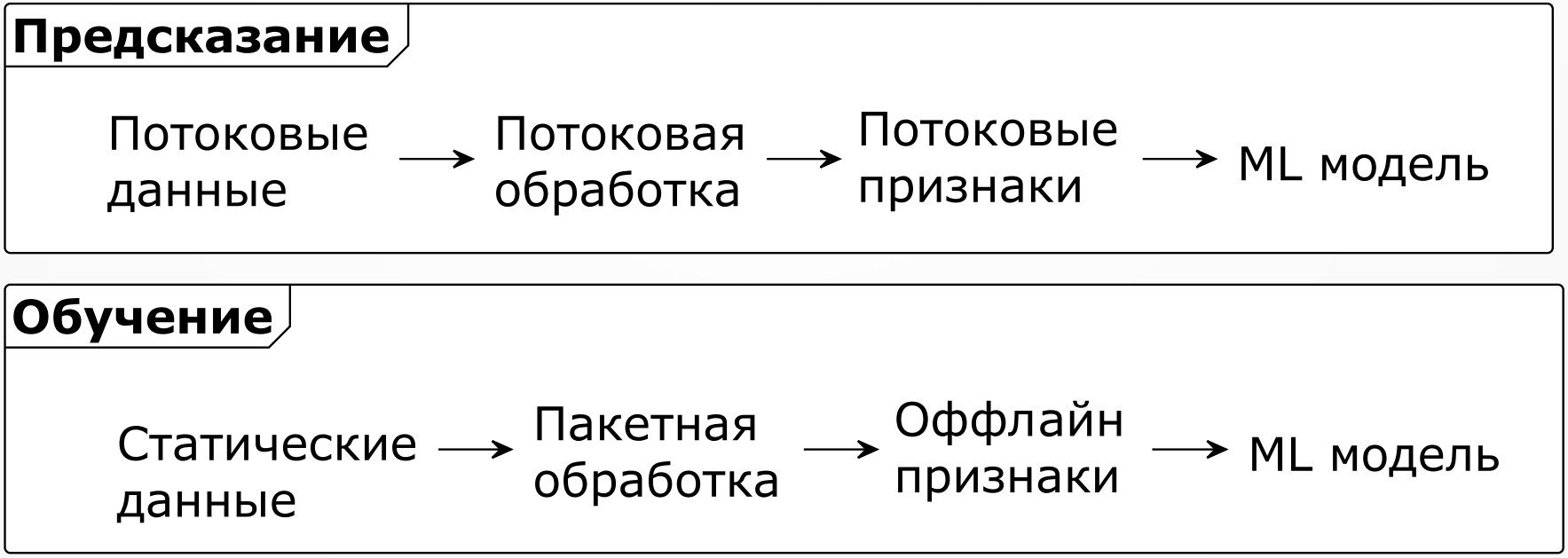


Онлайн предсказания

Потоковая обработка



Пайплайны обучения и инференса

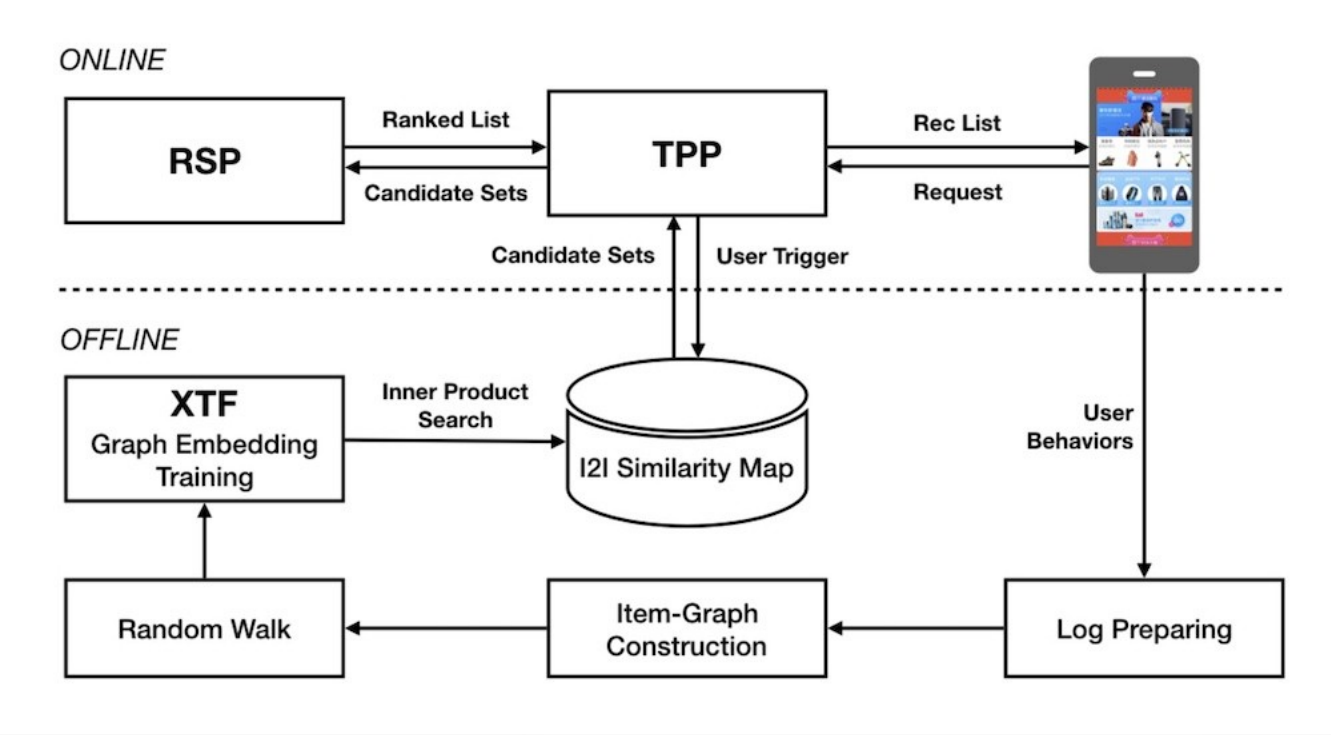


Batch prediction

- Быстрее
- Иногда дешевле
- Редко лучше

- Быстрее сети и компьютеры → онлайн предсказания
- Но — большие нейронки → оффлайн предсказания
- Но — ChatGPT → онлайн предсказания

Обычно применяются вместе



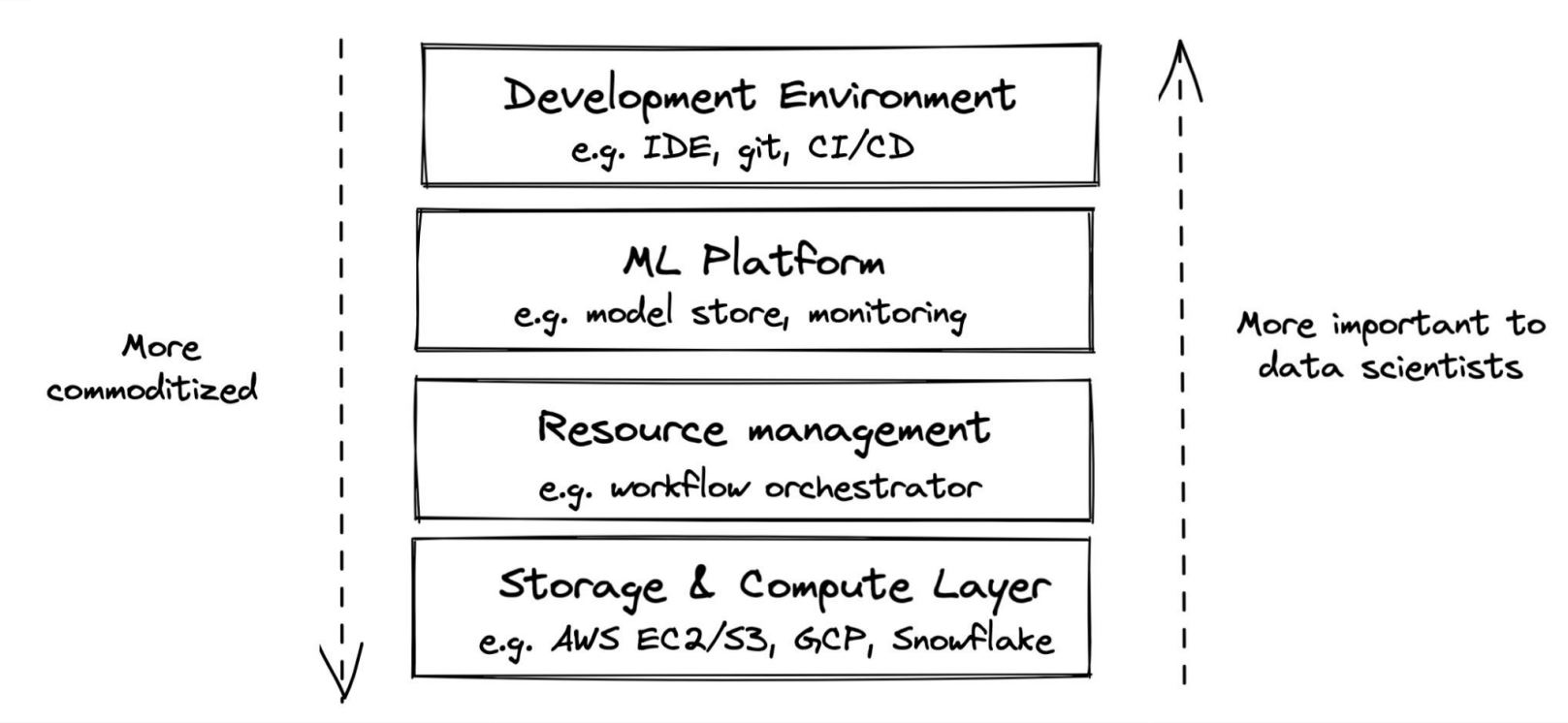
https://eugeneyan.com/writing/system-design-for-discovery/

ML инфраструктура и платформы

- Инфраструктура:
  комплекс взаимосвязанных структур или объектов, обеспечивающих функционирование системы

- ML инфраструктура:
  набор обеспечивающих систем и инструментов для поддержки жизненного цикла системы

Составные части инфраструктуры



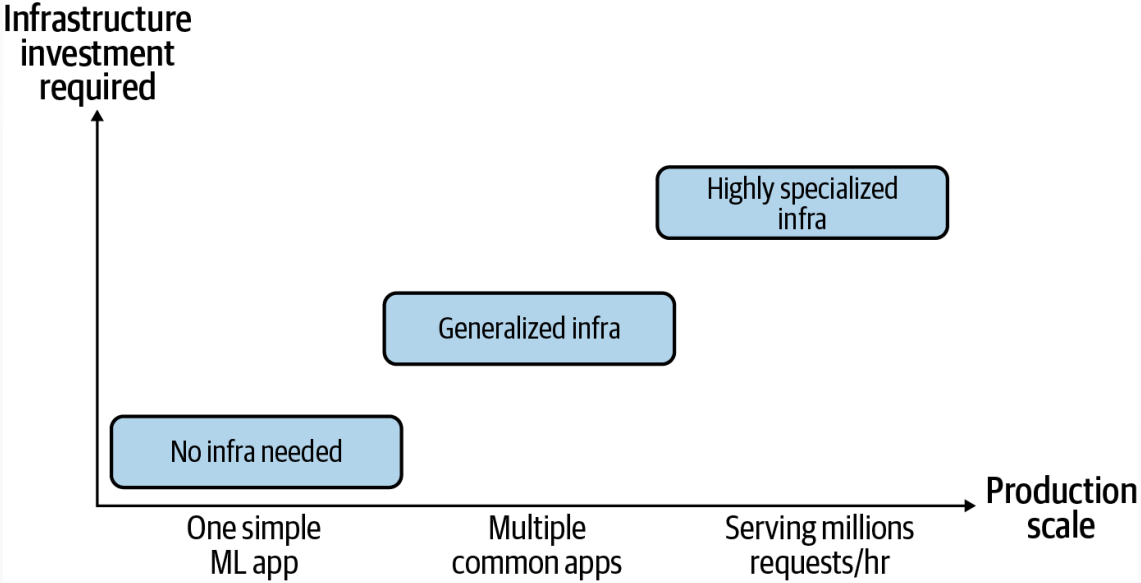
Chip Huyen CS 329

Какие проблемы решает инфраструктура

- Ускорение и воспроизводимость процессов
- Снижение порога входа для новых участников проекта
- Снижение затрат на разработку, развертывание и поддержку
- Экономия времени
- Повторное использование инженерных решений
- Единый набор подходов и инструментов

Разные потребности

- Большие компании разрабатывают свою инфраструктуру
- Маленькие стараются использовать готовую инфраструктуру



Chip Huyen CS 329

Уровень хранения

- Где хранятся данные, если их много
- HDD / SSD
- S3 / GS / Yandex Object Storage etc
- BigQuery
- Snowflake / RedShift

- Сравнительно дешевое место, дорогой трафик/обработка

Уровень вычислений

- CPU / GPU Baremetal servers
- vCPU / vGPU Virtual Servers
- App Runner / Cloud Run / Serverless Containers
- AWS Lambda / Cloud Functions / Yandex Cloud Functions
- k8s

Вычисления: процессор

- Тактовая частота
- Архитектура
- Количество ядер
- Hyper Threading →
- Размер кеша L1 L2 L3

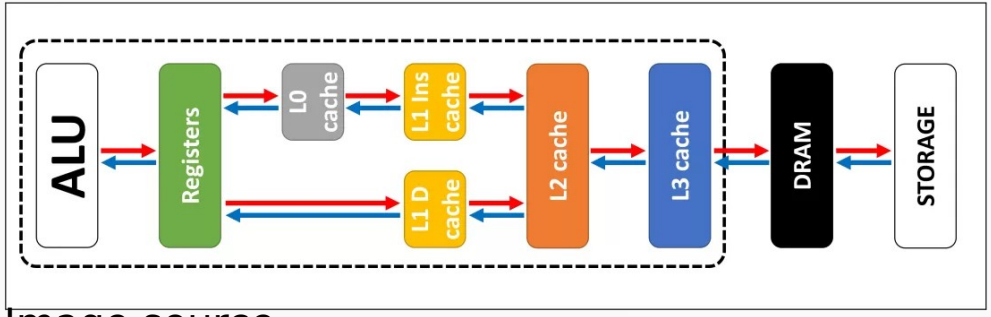
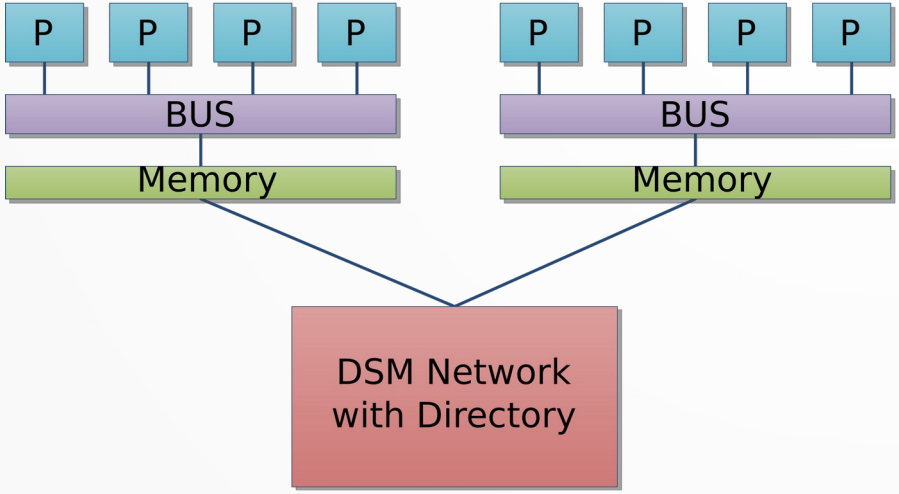


Image source
https://www.techspot.com/article/2066-cpu-l1-l2-l3-cache/

cat /proc/cpuinfo

```
processor       : 87
vendor_id       : GenuineIntel
cpu family      : 6
model           : 79
model name      : Intel(R) Xeon(R) CPU E5-2696 v4 @ 2.20GHz
stepping        : 1
microcode       : 0xb000040
cpu MHz         : 2134.134
cache size      : 56320 KB
physical id     : 1
siblings        : 44
core id         : 28
cpu cores       : 22
apicid          : 121
initial apicid  : 121
fpu             : yes
fpu_exception   : yes
cpuid level     : 20
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi
r ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadl
 vpid ept_ad fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm rdt_a rdseed adx smap intel_
vmx flags       : vnmi preemption_timer posted_intr invvpid ept_x_only ept_ad ept_1gb flexpriority apicv t
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs taa itlb_multihit m
bogomips        : 4401.63
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

NUMA

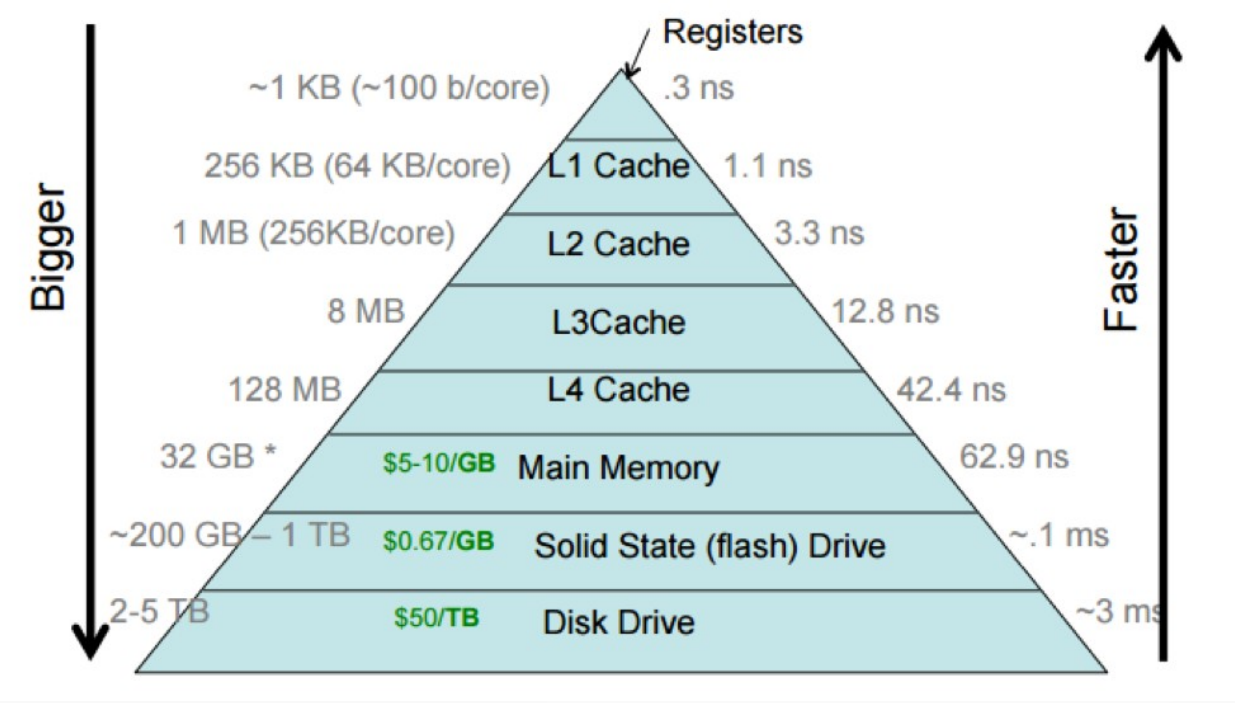



```
root@bora:~# lscpu | grep -i numa
NUMA node(s):                    2
NUMA node0 CPU(s):               0-21,44-65
NUMA node1 CPU(s):               22-43,66-87
```

https://ru.wikipedia.org/wiki/Non-Uniform_Memory_Access

https://www.baeldung.com/linux/numa-test-support

Вычисления: память



https://cs61.seas.harvard.edu/site/2018/Storage2/

Память

- RAM
  - Размер доступной памяти
  - Пропускная способность
- GPU RAM
  - Хранит модель и батчи
  - Можно «объединять» память нескольких GPU
  - Медленный обмен с основной RAM
- GPU часто простаивает из-за медленной загрузки данных

Скорость обмена с памятью

Memory Bandwidth

From the previous section, we have seen that Tensor Cores are very fast. So fast, in fact, that they are idle most of the time as they are waiting for memory to arrive from global memory. For example, during BERT Large training, which uses huge matrices — the larger, the better for Tensor Cores — we have a Tensor Core TFLOPS utilization of about 30%, meaning that 70% of the time, Tensor Cores are idle.

This means that when comparing two GPUs with Tensor Cores, one of the single best indicators for each GPU's performance is their memory bandwidth. For example, The A100 GPU has 1,555 GB/s memory bandwidth vs the 900 GB/s of the V100. As such, a basic estimate of speedup of an A100 vs V100 is 1555/900 = 1.73x.

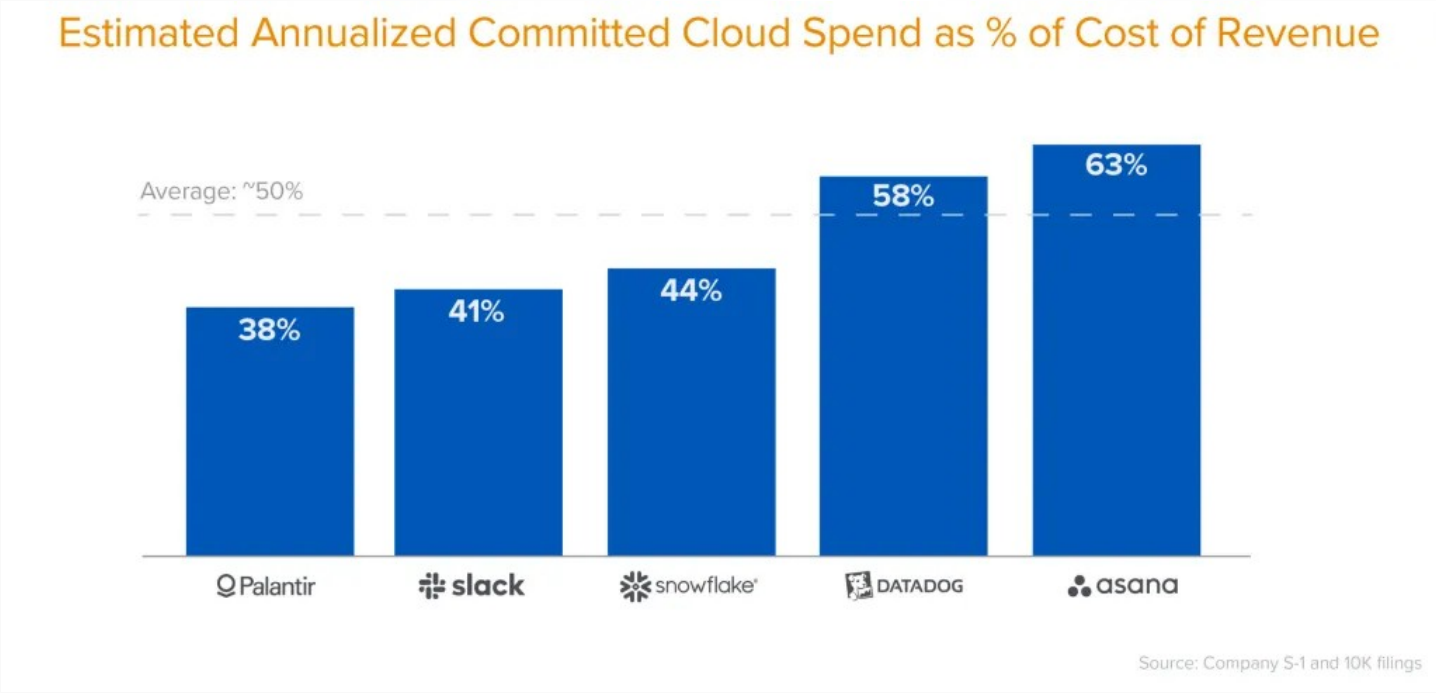
https://timdettmers.com/2020/09/07/which-gpu-for-deep-learning/

Свои сервера или облака

- Сервер в своей стойке на своей территории 1X
- Арендованный в стойке в датацентре 3X
- Арендованный в облаке 15x

- Облака — очень дорого
- Свои сервера обычно сильно недогружены
- Сервера надо покупать заранее (предвидение 90 LVL)
- Маленькие организации могут работать, арендуя сервера
- Стартапам и среднего размера бизнесу удобнее облака
- Большие организации частично мигрируют из облаков

Затраты на облака



https://a16z.com/2021/05/27/cost-of-cloud-paradox-market-cap-cloud-lifecycle-scale-growth-repatriation-optimization/

Запуск задач

- Запуск по расписанию (cron)
  - не отслеживает взаимосвязи задач
- Планировщик (SLURM)
  - Запускает задачи, управляет очередями задач
- Оркестратор (K8s)
  - Управляет ресурсами и координирует выполнение

Разделение условно,
планировщики умеют в оркестрацию,
оркестраторы умеют в планирование и расписание

Workflow management

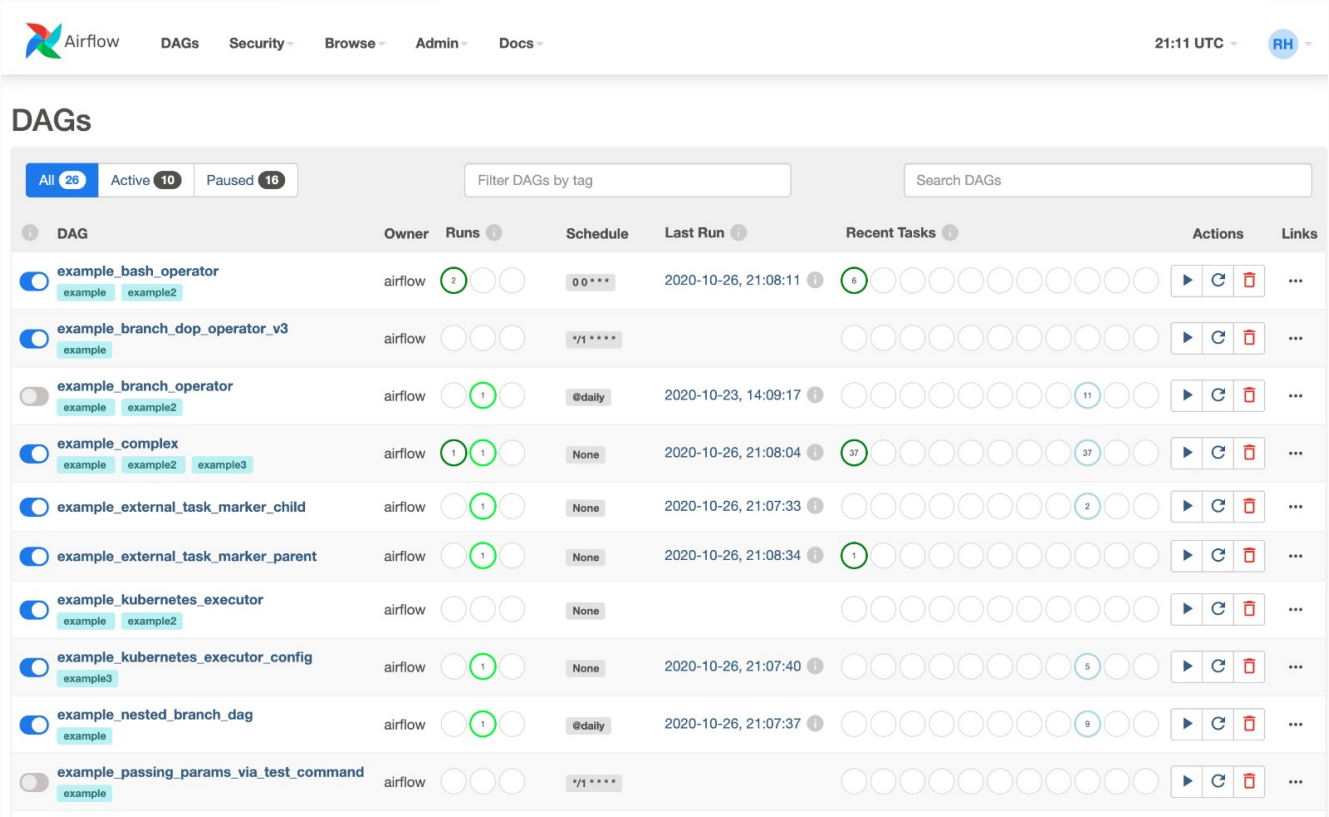
- Этап расчета — task
- Граф последовательного выполнения задач — DAG
- Одно выполнение графа — Flow
- Flow запускается вручную, по расписанию или по внешнему событию («сенсору»)
- Примеры — Airflow, Prefect, Dagster

Airflow

```python
from datetime import datetime

from airflow import DAG
from airflow.decorators import task
from airflow.operators.bash import BashOperator

# A DAG represents a workflow, a collection of tasks
with DAG(dag_id="demo", start_date=datetime(2022, 1, 1), schedule="0 0 * * *") as dag:

    # Tasks are represented as operators
    hello = BashOperator(task_id="hello", bash_command="echo hello")

    @task()
    def airflow():
        print("airflow")

    # Set dependencies between tasks
    hello >> airflow()
```

https://airflow.apache.org/docs/apache-airflow/stable/index.html

Airflow UI

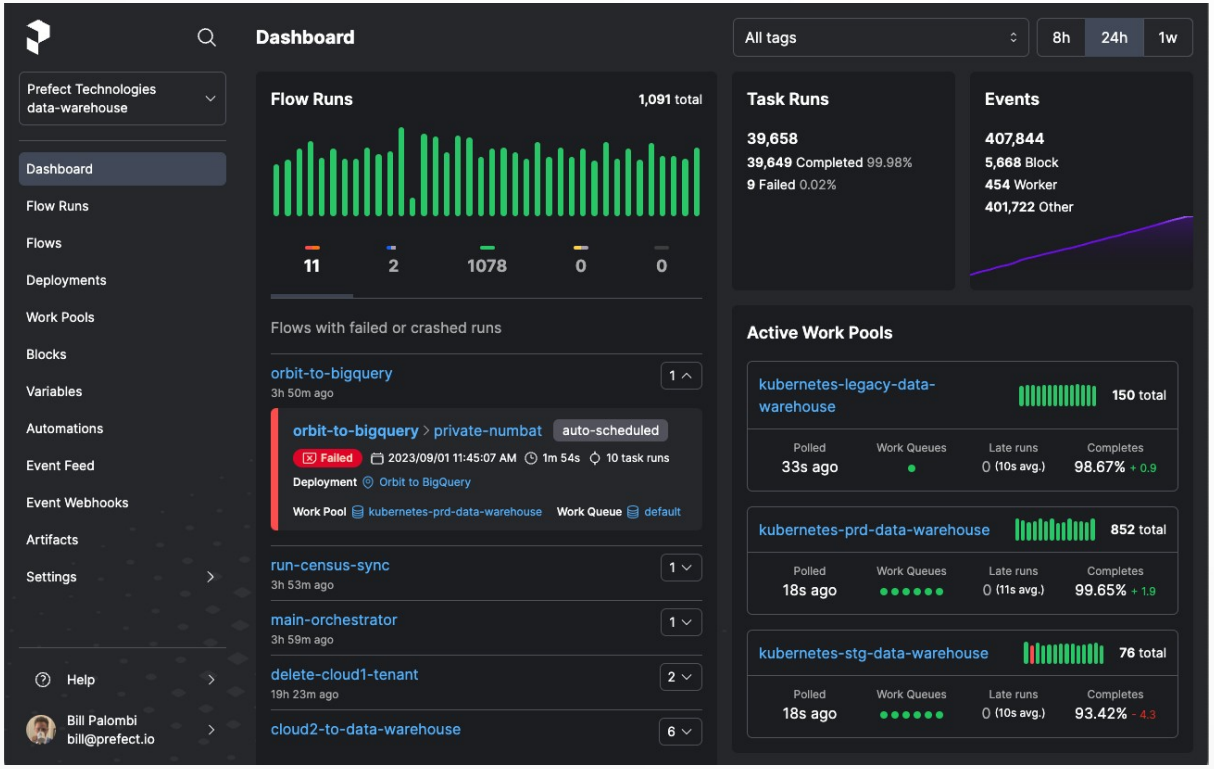

Airflow
DAGs
Security
Browse
Admin
Docs
21:11 UTC
RH
DAGs
All 26
Active 10
Paused 16
Filter DAGs by tag
Search DAGs
DAG
Owner
Runs
Schedule
Last Run
Recent Tasks
Actions
Links
example_bash_operator
example
example2
airflow
0 0 * * *
2020-10-26, 21:08:11
example_branch_dop_operator_v3
example
airflow
*/1 * * * *
example_branch_operator
example
example2
airflow
@daily
2020-10-23, 14:09:17
example_complex
example
example2
example3
airflow
None
2020-10-26, 21:08:04
example_external_task_marker_child
airflow
None
2020-10-26, 21:07:33
example_external_task_marker_parent
airflow
None
2020-10-26, 21:08:34
example_kubernetes_executor
example
example2
airflow
None
example_kubernetes_executor_config
example3
airflow
None
2020-10-26, 21:07:40
example_nested_branch_dag
example
airflow
@daily
2020-10-26, 21:07:37
example_passing_params_via_test_command
example
airflow
*/1 * * * *

Prefect.io

```python
import httpx
from datetime import timedelta
from prefect import flow, task, get_run_logger
from prefect.tasks import task_input_hash


@task(cache_key_fn=task_input_hash,
      cache_expiration=timedelta(hours=1),
      )
def get_url(url: str, params: dict = None):
    response = httpx.get(url, params=params)
    response.raise_for_status()
    return response.json()
```

https://docs.prefect.io/latest/tutorial/tasks/

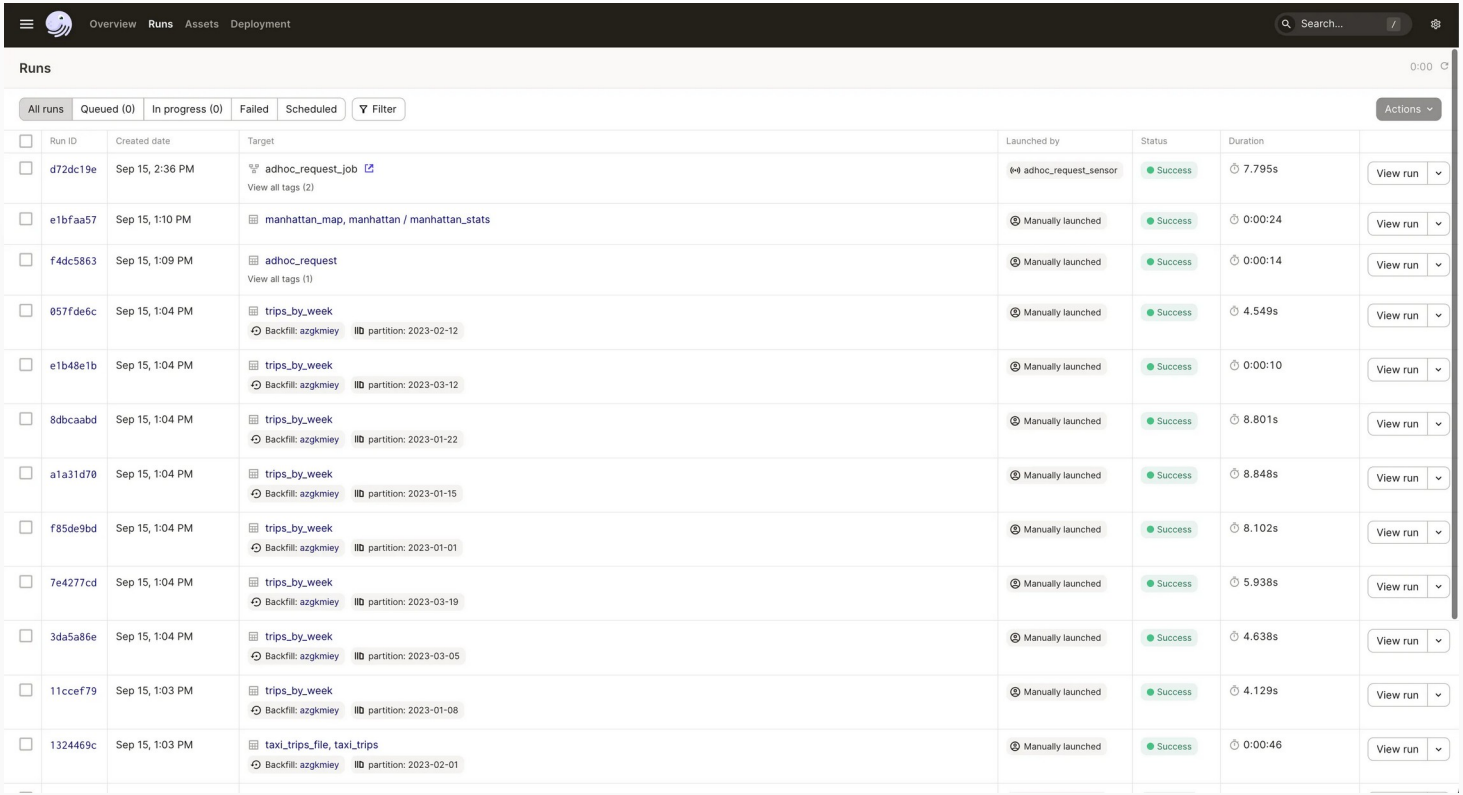
Prefect UI


Dashboard
All tags
8h
24h
1w
Prefect Technologies
data-warehouse
Dashboard
Flow Runs
Flows
Deployments
Work Pools
Blocks
Variables
Automations
Event Feed
Event Webhooks
Artifacts
Settings
Help
Bill Palombi
bill@prefect.io
Flow Runs
1,091 total
11
2
1078
0
0
Flows with failed or crashed runs
orbit-to-bigquery
3h 50m ago
1
orbit-to-bigquery > private-numbat
auto-scheduled
Failed
2023/09/01 11:45:07 AM
1m 54s
10 task runs
Deployment
Orbit to BigQuery
Work Pool
kubernetes-prd-data-warehouse
Work Queue
default
run-census-sync
3h 53m ago
1
main-orchestrator
3h 59m ago
1
delete-cloud1-tenant
19h 23m ago
2
cloud2-to-data-warehouse
6
Task Runs
39,658
39,649 Completed 99.98%
9 Failed 0.02%
Events
407,844
5,668 Block
454 Worker
401,722 Other
Active Work Pools
kubernetes-legacy-data-warehouse
150 total
Polled
33s ago
Work Queues
Late runs
0 (10s avg.)
Completes
98.67% + 0.9
kubernetes-prd-data-warehouse
852 total
Polled
18s ago
Work Queues
Late runs
0 (11s avg.)
Completes
99.65% + 1.9
kubernetes-stg-data-warehouse
76 total
Polled
18s ago
Work Queues
Late runs
0 (10s avg.)
Completes
93.42% - 4.3

DAGster

```python
from dagster import asset
from pandas import DataFrame, read_html, get_dummies
from sklearn.linear_model import LinearRegression as Regression

@asset
def country_stats() -> DataFrame:
    df = read_html("https://tinyurl.com/mry64ebh")[0]
    df.columns = ["country", "continent", "pop_change"]
    df["pop_change"] = df["pop_change"].str.rstrip("%").astype("float")
    return df

@asset
def change_model(country_stats: DataFrame) -> Regression:
    data = country_stats.dropna(subset=["pop_change"])
    dummies = get_dummies(data[["continent"]])
    return Regression().fit(dummies, data["pop_change"])

@asset
def continent_stats(country_stats: DataFrame, change_model: Regression) -> DataFrame:
    result = country_stats.groupby("continent").sum()
    result["pop_change_factor"] = change_model.coef_
    return result
```

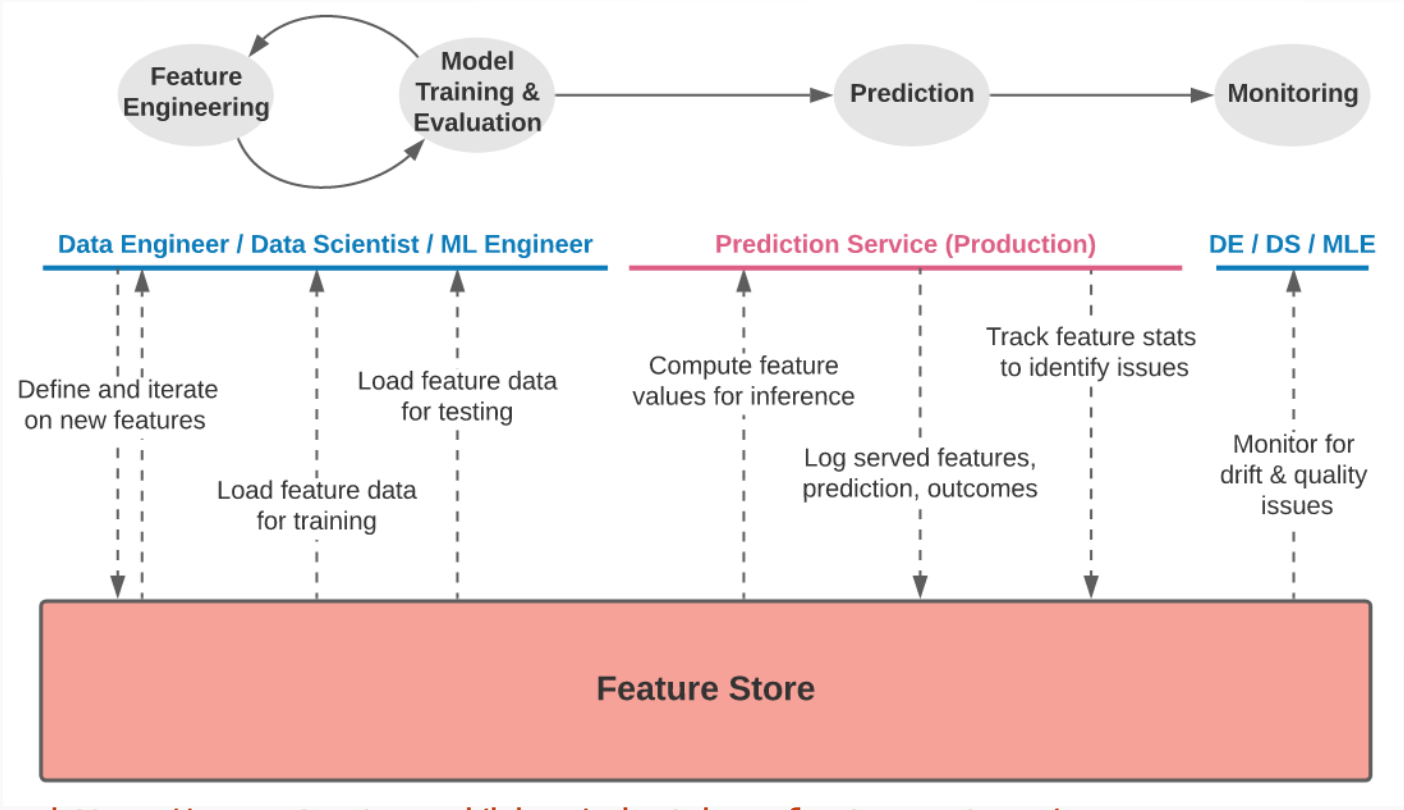
Dagit UI


Overview
Runs
Assets
Deployment
Search...
Runs
All runs
Queued (0)
In progress (0)
Failed
Scheduled
Filter
Actions
Run ID
Created date
Target
Launched by
Status
Duration
d72dc19e
Sep 15, 2:36 PM
adhoc_request_job
View all tags (2)
adhoc_request_sensor
Success
7.795s
View run
e1bfaa57
Sep 15, 1:10 PM
manhattan_map, manhattan / manhattan_stats
Manually launched
Success
0:00:24
View run
f4dc5863
Sep 15, 1:09 PM
adhoc_request
View all tags (1)
Manually launched
Success
0:00:14
View run
057fde6c
Sep 15, 1:04 PM
trips_by_week
Backfill: azgkmiey
partition: 2023-02-12
Manually launched
Success
4.549s
View run
e1b48e1b
Sep 15, 1:04 PM
trips_by_week
Backfill: azgkmiey
partition: 2023-03-12
Manually launched
Success
0:00:10
View run
8dbcaabd
Sep 15, 1:04 PM
trips_by_week
Backfill: azgkmiey
partition: 2023-01-22
Manually launched
Success
8.801s
View run
a1a31d70
Sep 15, 1:04 PM
trips_by_week
Backfill: azgkmiey
partition: 2023-01-15
Manually launched
Success
8.848s
View run
f85de9bd
Sep 15, 1:04 PM
trips_by_week
Backfill: azgkmiey
partition: 2023-01-01
Manually launched
Success
8.102s
View run
7e4277cd
Sep 15, 1:04 PM
trips_by_week
Backfill: azgkmiey
partition: 2023-03-19
Manually launched
Success
5.938s
View run
3da5a86e
Sep 15, 1:04 PM
trips_by_week
Backfill: azgkmiey
partition: 2023-03-05
Manually launched
Success
4.638s
View run
11ccef79
Sep 15, 1:03 PM
trips_by_week
Backfill: azgkmiey
partition: 2023-01-08
Manually launched
Success
4.129s
View run
1324469c
Sep 15, 1:03 PM
taxi_trips_file, taxi_trips
Backfill: azgkmiey
partition: 2023-02-01
Manually launched
Success
0:00:46
View run

ML платформы

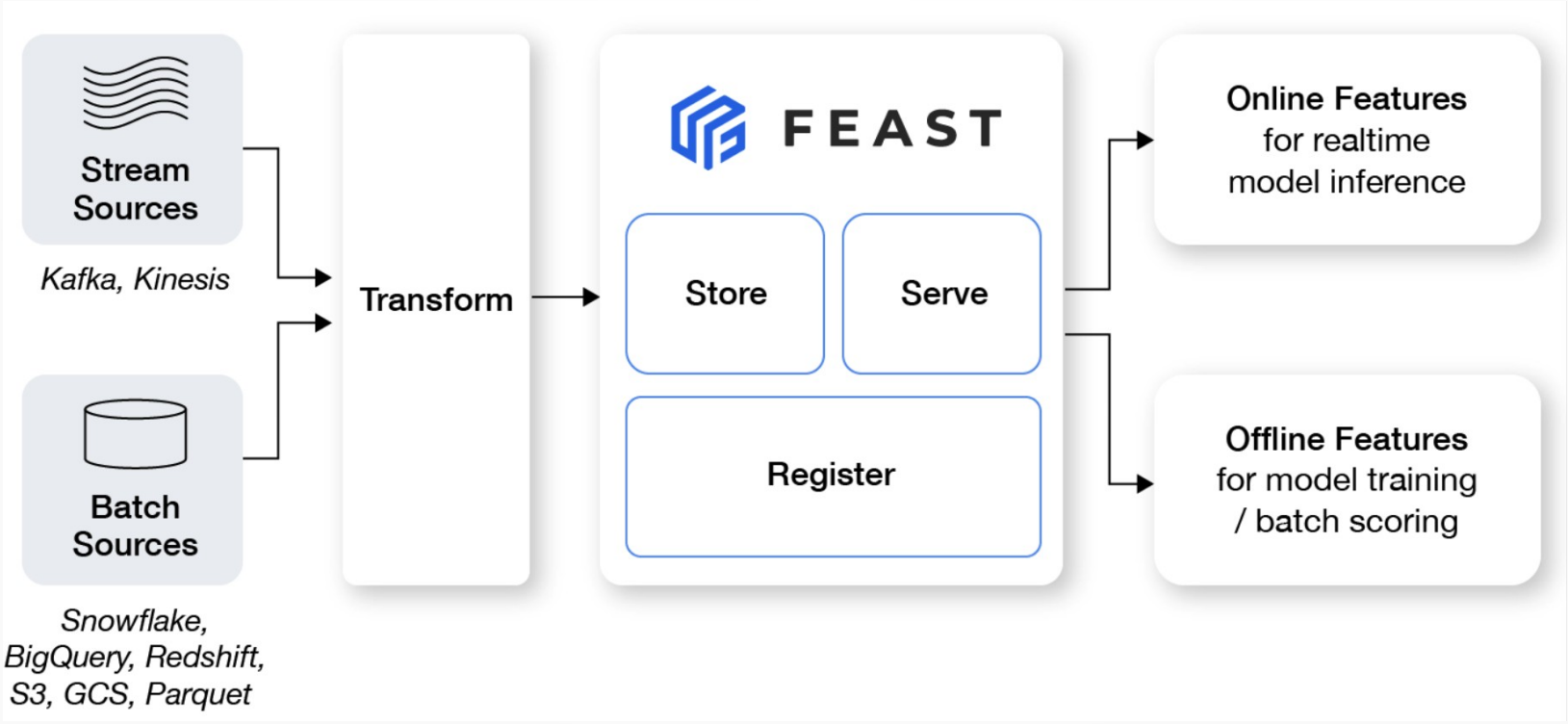
- Ключевые компоненты:
  - Развертывание
  - Хранилище моделей
  - Мониторинг
  - Отслеживание экспериментов
  - Метрики и дашборды
  - Feature Store

Feature Store



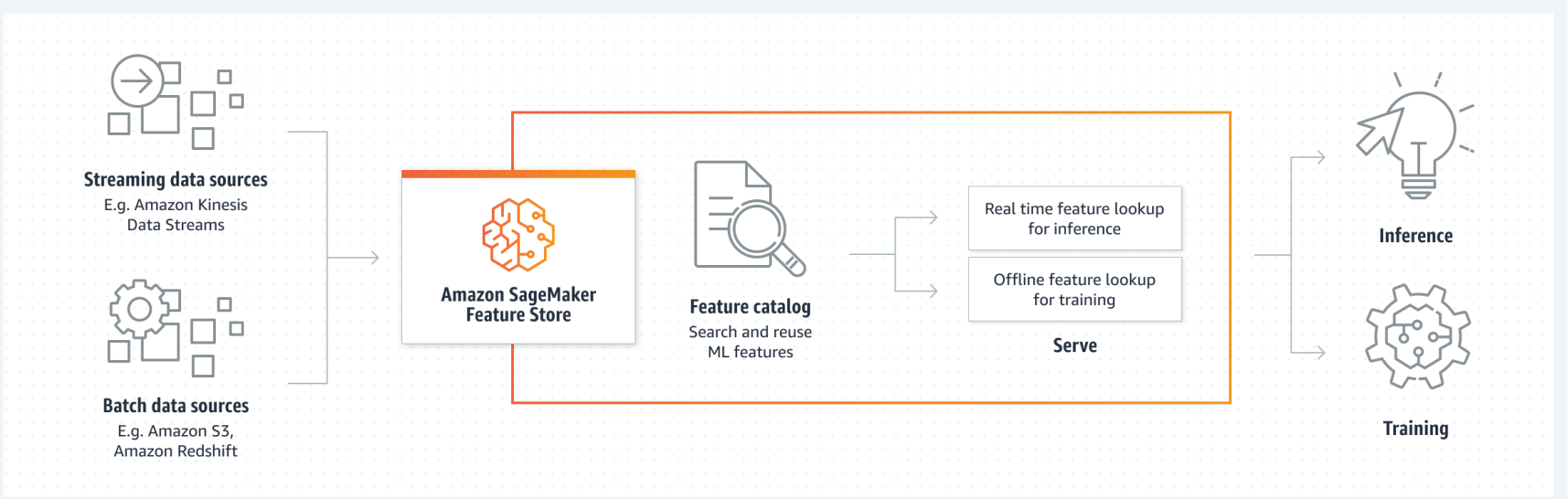
https://www.tecton.ai/blog/what-is-a-feature-store/

Feast



https://docs.feast.dev/

Amazon SageMaker



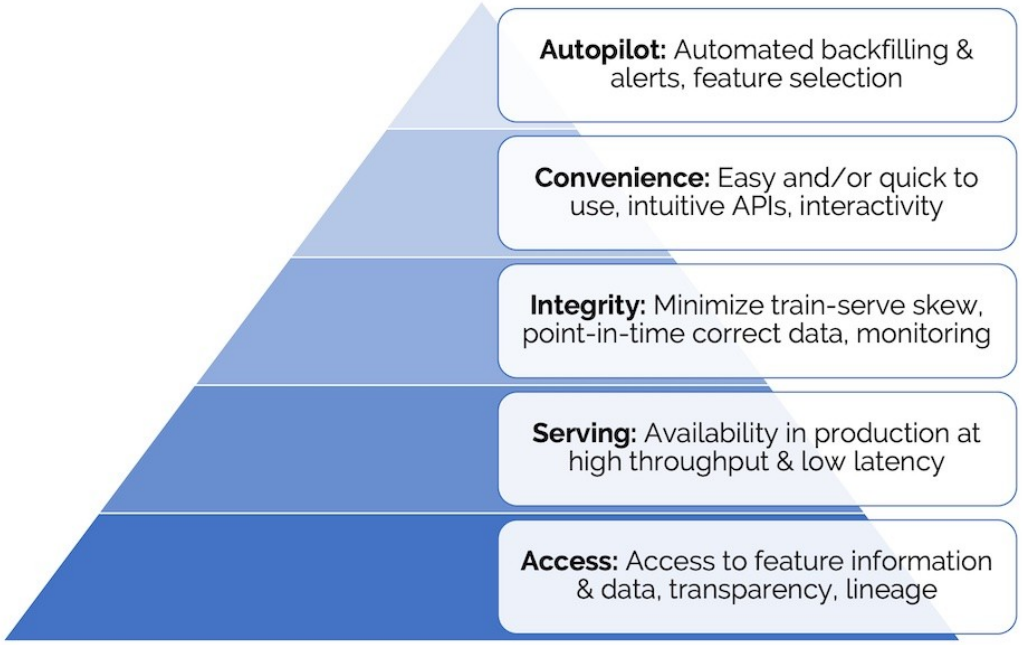
https://aws.amazon.com/sagemaker/feature-store/

И еще feature stores

- Hopsworks
- Google
- Databricks
- Featureform
- Tecton
- Continual

https://www.featurestore.org/

Хороший Feature Store



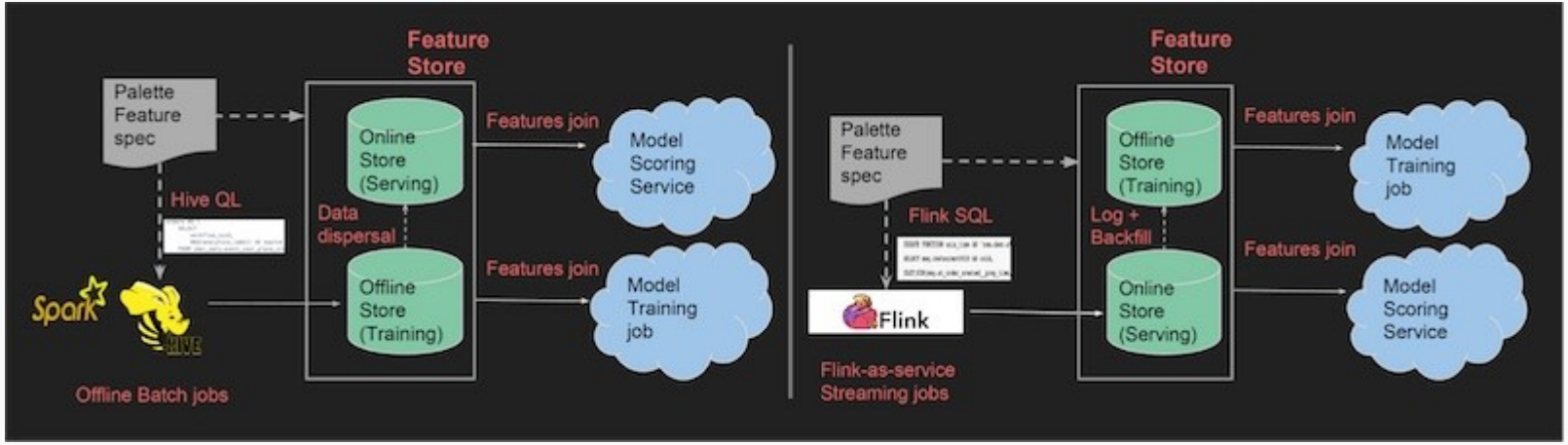
https://eugeneyan.com/writing/feature-stores/

FS: Доступ / Access

- Уменьшаем дублирование
- Облегчаем повторное использование
- Облегчаем совместное использование
- Хранилище, витрина данных:
  - Интерфейс между DS и DL
  - Реестр доступных данных
  - Прослеживаемость данных

FS: Раздача / Serving

- Высокая пропускная способность
- Низкая задержка
- Интеграция с хранилищами данных
- Трансформация данных на лету



https://eugeneyan.com/writing/feature-stores/

FS: Согласованность / Integrity

- Минимизировать разницу между обучением и инференсом
- Консистентность признаков
- Путешествия во времени
- Контроль качества данных

FS: Удобство / Convenience

- Удобное API
- Интеграция с используемыми инструментами
- Интерактивный доступ «поглядеть в данные»
- Самообслуживание
- Малые накладные затраты

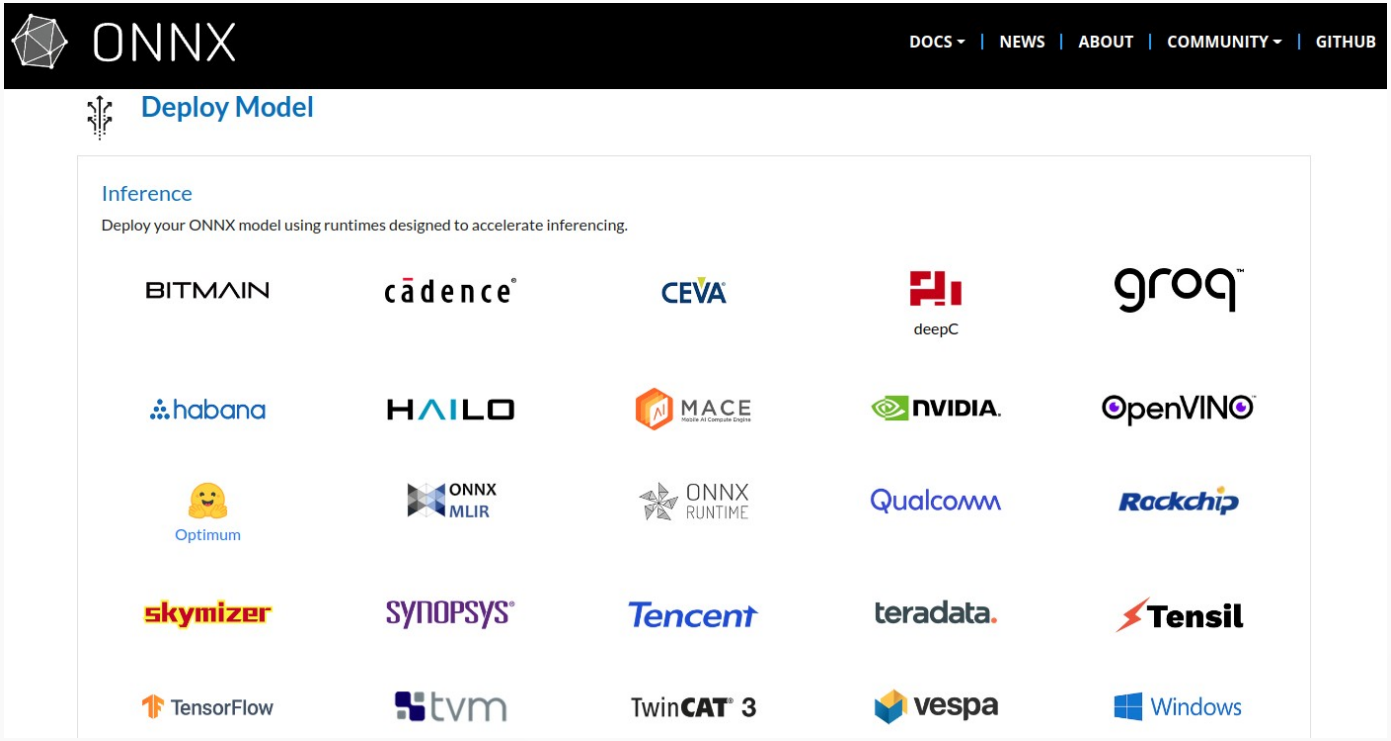
FS: Автоматизация / Autopilot

- Перезаливать данные
- Пересчитывать признаки
- Автодетекция аномалий в данных
- Автоподбор релевантных признаков
- Мониторинг качества данных
- Анализ использования признаков
- Всякое удобное

Обслуживание запросов

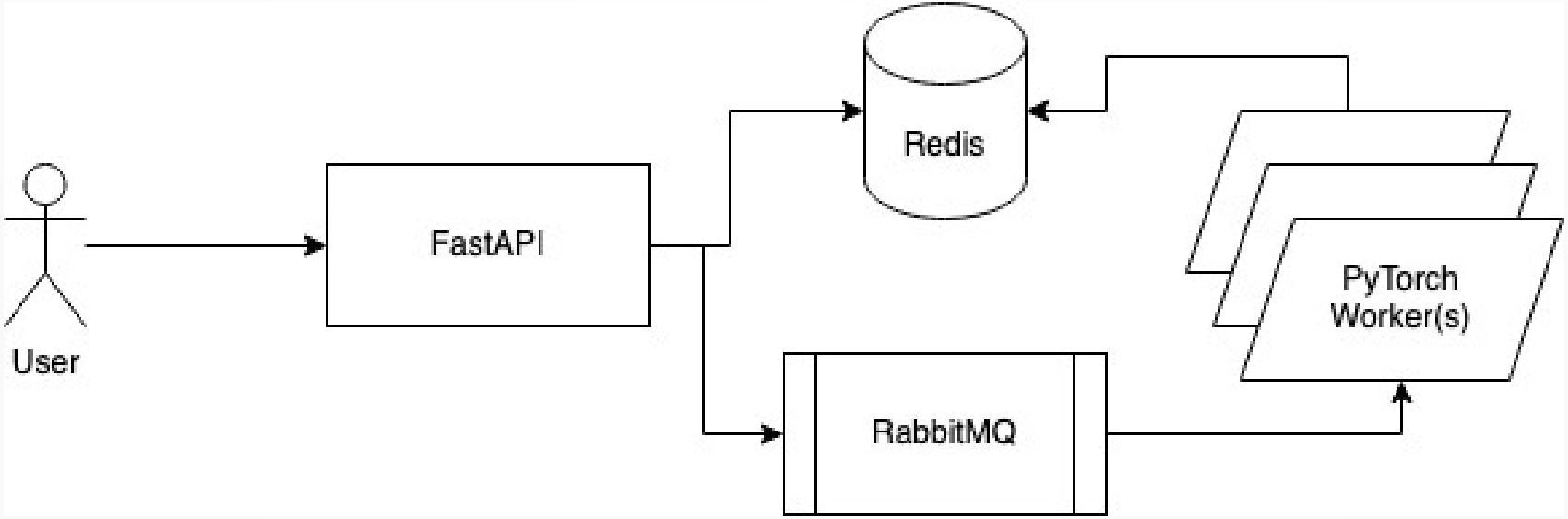
- Универсальные форматы, например onnx
- Сжатие / оптимизация моделей
- Велосипеды на FastApi
- Triton Inference Server
- TorchServe
- RayServe
- Serverless

ONNX



https://onnx.ai/

Велосипед на FastAI



- https://www.auroria.io/running-pytorch-models-for-inference-using-fastapi-rabbitmq-redis-docker/

NVIDIA Triton Inference Server

NVIDIA Triton

- What is NVIDIA Triton?
- Benefits
- Functionality
  - **Scalable AI**
  - Model Orchestration
  - LLM Inference
  - Model Analyzer
  - Tree-based Models
- Ecosystem Integrations
- Success Stories
- Resources
- Inference Newsletter

Explore the benefits.

Support for multiple frameworks.

Triton supports all major training and inference frameworks, such as TensorFlow, NVIDIA® TensorRT™, PyTorch, MXNet, Python, ONNX, XGBoost, scikit-learn, RandomForest, OpenVINO, custom C++, and more.

High-performance inference.

Triton supports all NVIDIA GPU-, x86-, Arm® CPU-, and AWS Inferentia-based inferencing. It offers dynamic batching, concurrent execution, optimal model configuration, model ensemble, and streaming audio/video inputs to maximize throughput and utilization.

Designed for DevOps and MLOps.

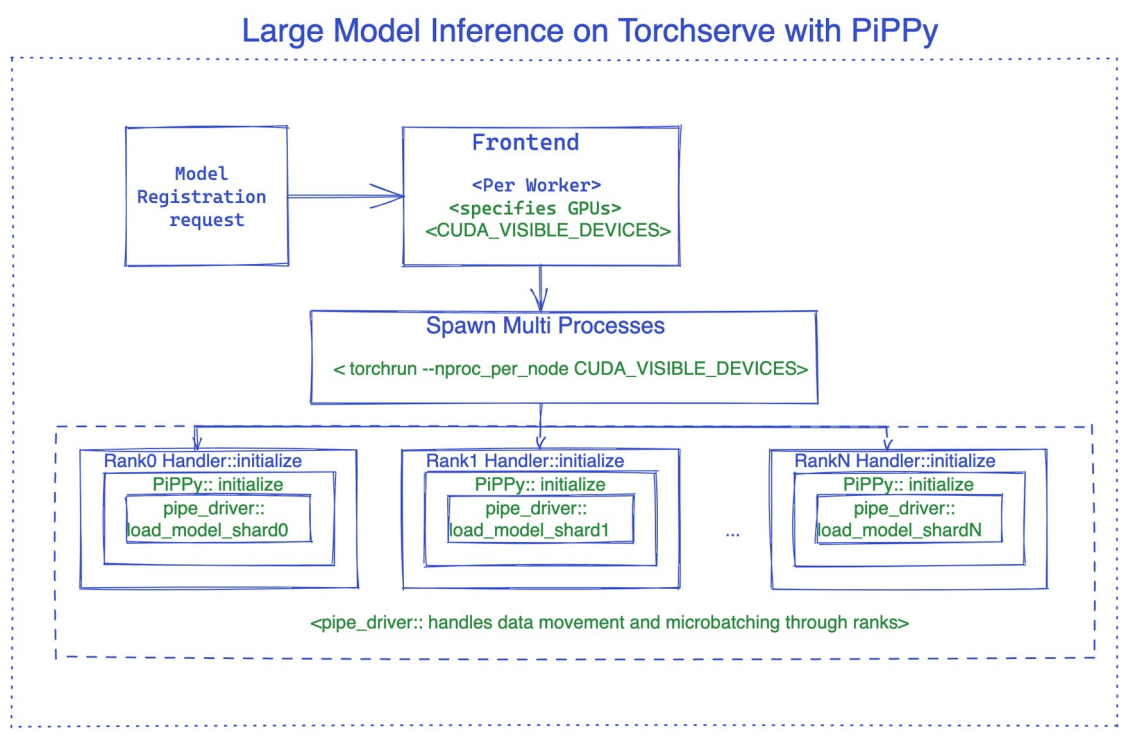
Triton integrates with Kubernetes for orchestration and scaling, exports Prometheus metrics for monitoring, supports live model updates, and can be used in all major public cloud AI and Kubernetes platforms. It's also integrated in many MLOps software solutions.

An integral part of NVIDIA AI.

The NVIDIA AI platform, which includes Triton, gives enterprises the compute power, tools, and algorithms they need to succeed in AI, accelerating workloads from speech recognition and recommender systems to medical imaging and improved logistics.

https://developer.nvidia.com/nvidia-triton-inference-server

TorchServe



https://pytorch.org/serve/large_model_inference.html

Ray Serve

Pythonic API

Configure your model serving declaratively in pure Python, without needing YAML or JSON configs.

Low latency, high throughput

Horizontally scale across hundreds of processes or machines, while keeping the overhead in single-digit milliseconds.

Multi-model composition

Easily compose multiple models, mix model serving with business logic, and independently scale components, without complex microservices.

Framework-agnostic

Use a single tool to serve all types of models — from PyTorch and Tensorflow to scikit-Learn models — and business logic.

FastAPI Integration

Scale an existing FastAPI server easily or define an HTTP interface for your model using its simple, elegant API.

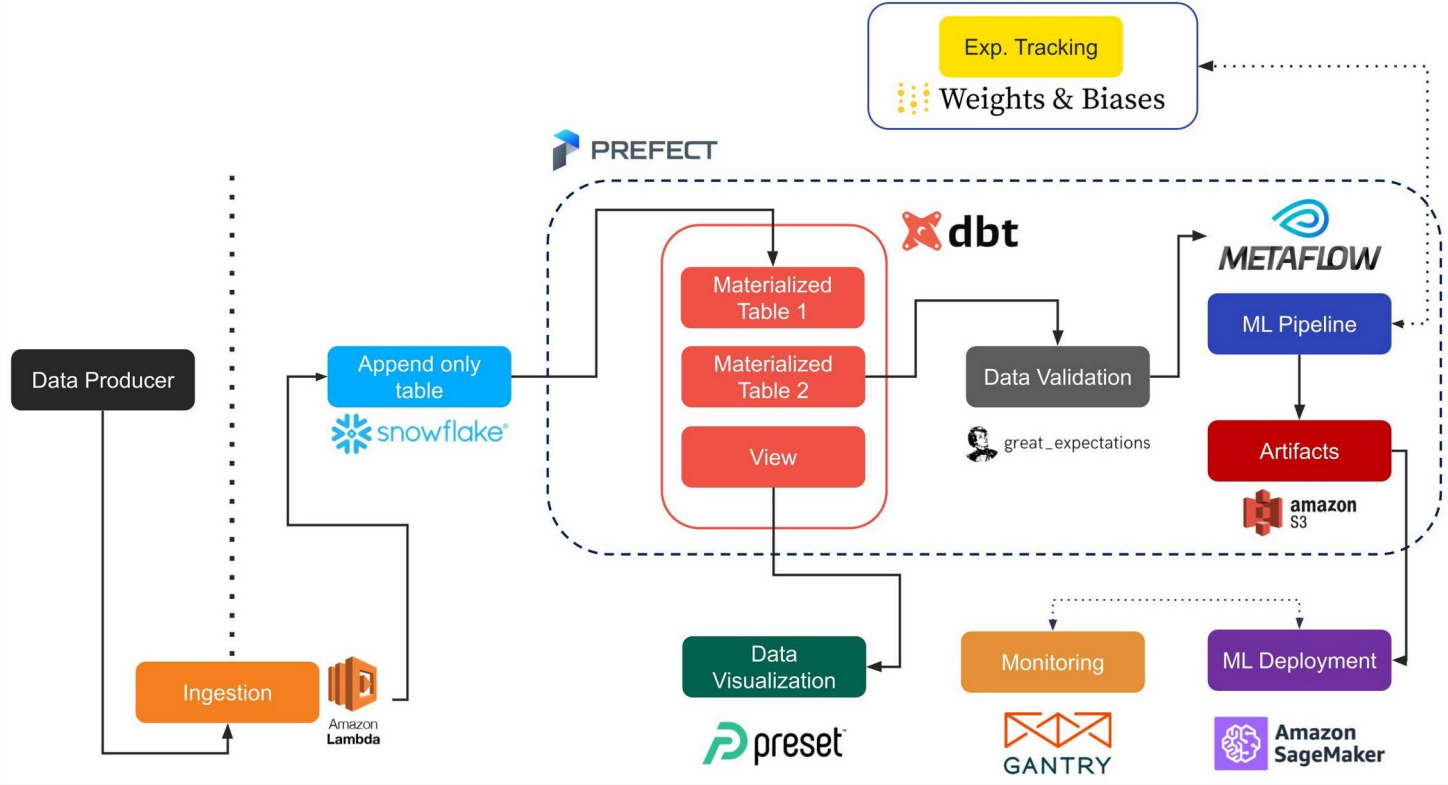
Native GPU support

Using GPUs is as simple as adding one line of Python code. Maximize hardware utilization by sharing CPUs or GPUs between different models.

https://docs.ray.io/en/latest/serve/index.html

https://www.anyscale.com/blog/serving-pytorch-models-with-fastapi-and-ray-serve

You Don't Need a Bigger Boat



https://github.com/jacopotagliabue/you-dont-need-a-bigger-boat

Дополнительные материалы

- Introduction to streaming for data scientists
- How We Scaled Bert To Serve 1+ Billion Daily Requests on CPUs
- You Do Not Need a Bigger Boat