

Подходы к промышленной реализации ML-решений

Вопросы

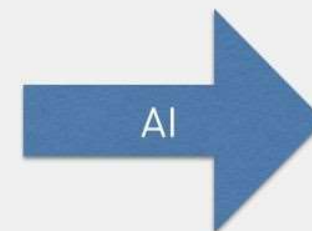
- 1.Пакетная обработка
- 2.Асинхронный режим
- 3.Синхронный режим

The background of the slide is a high-angle, aerial photograph of a city skyline, likely New York City, featuring numerous skyscrapers and dense urban development. The image is overlaid with a semi-transparent blue and green gradient. A network of thin, light blue lines connects various points across the gradient, creating a digital or data network aesthetic. The text "Примеры ML систем" is centered in the middle of the slide in a white, sans-serif font.

Примеры ML систем

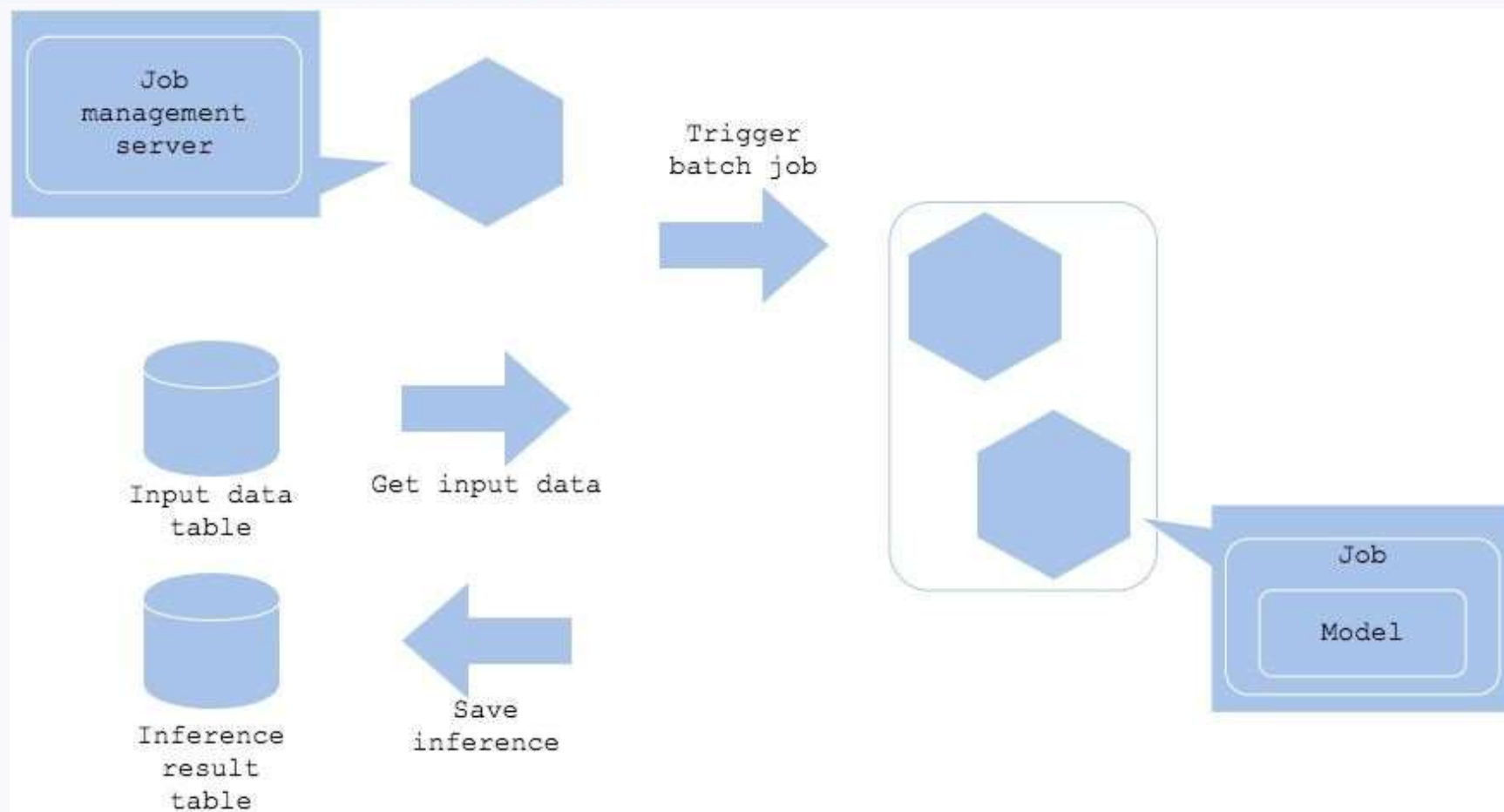
Классический банковский ML

Index	ПОЛ	ВОЗРАСТ	ЗП
0	М	23	300 к/сек
1	М	33	45 000 р/мес
2	М	34	15 000 р/мес
3	Ж	55	55 000 р/мес



PREDICT
1
0.5
0.1
0.7

Пакетная обработка (пакетный паттерн)



https://github.com/mercari/ml-system-design-pattern/blob/master/Serving-patterns/Batch-pattern/design_en.md

Пакетный паттерн - когда нужно использовать?

- Если не нужно получать результат в реальном или почти реальном времени
- Для массивной обработки данных
- Когда для корректной работы достаточно запускать процесс PREDICTION по расписанию
- Подходит для большей части ETL

Как это обычно выглядит?

- 1) Считаются фичи по данным за предыдущий день(**много джоб**)
- 2) Берем модель(обучаем модель) и делаем предсказание
- 3) Смотрим на предсказания, если их распределение норм, то заливаем в **прод**, если нет, то посылаем предупреждение, о том, что-то пошло не так.
- 4) Делать надо **каждый день**

Cron

Cron - unix утилита, используемая для периодического выполнения заданий в определенное время.

Регулярные действия описываются инструкциями в **crontab** и специальных каталогах

Crontab

```
worker1: ~ cronitor list
```

```
► Reading user "ubuntu" crontab
```

SCHEDULE	COMMAND
0 1 * * *	/var/app/edi/send_batch_invoices.py
0 2 * * *	/var/app/edi/send_batch_settlement.py
*/20 * * * *	/var/app/edi/reconciler.sh --full
* * * * *	/var/app/reporting/rollup minute
0 * * * *	/var/app/reporting/rollup hour
0 0 * * *	/var/app/reporting/rollup day
5 0 * * *	/var/app/reporting/rollup archive-ancient-data

```
► Reading /etc/crontab
```

SCHEDULE	COMMAND
0 * * * *	/tmp/foo.sh --systemcrontab

```
worker1: ~ █
```

Crontab

* * * * * /path/to/script.sh

Diagram illustrating the fields of a crontab entry:

- Minute (0- 59)
- Hour (0 - 23)
- Day of Month (1 - 31)
- Month of Year (1-12)
- Day Of Week (0-7) 0,7 are "Sunday"
- Command/Script to Execute

Как cron можно применить?

Помним, что нам нужно запустить несколько задач подряд

- 1) определяем время старта и конца каждой задачи и запускаем несколько cron-job
- 2) запикиваем все задачи в один cronjob

```
spark-submit job1
```

```
spark-submit job2
```

```
etc
```




Пример

Минусы cron

- трудно мониторить
- нет автоматических перезапусков
- нельзя явно задавать связи между задачами

```
#30 2 * * * spark-submit --master yarn --queue root.daily --executor-cores=8 --executor-memory=16G --class BanReport /mnt/h
#30 4 * * * spark-submit --master yarn --queue root.daily --executor-cores=4 --executor-memory=32G --class BanGenerator /mn
30 5 * * * spark2-submit --master yarn --queue root.daily --executor-cores=4 --executor-memory=8G --class etl.RequestOtahot
#30 6 * * * spark-submit --master yarn --queue root.daily --executor-cores=8 --executor-memory=16G --class etl.AvailScore /
#10 * * * * spark-submit --master yarn --executor-cores=4 --executor-memory=8G --class etl.DAPIToParquet /mnt/hadoop/spark-
#31 * * * * spark-submit --master yarn --executor-cores=4 --executor-memory=8G --class DiscrepancyHunter /mnt/hadoop/spark-
#33 * * * * spark-submit --master yarn --executor-cores=8 --executor-memory=16G --class DiscrepancyInStep /mnt/hadoop/spark
20 * * * * . /etc/profile; /mnt/hdfs/tools/spark-jars/exchange.py >> /root/exchange.log 2>&1
#11 * * * * spark2-submit --master yarn --queue root.hourly --executor-cores=4 --executor-memory=8G --class etl.RawDataConv
#26 * * * * spark-submit --master yarn --queue root.hourly --executor-cores=4 --executor-memory=8G --class etl.MinRates /mn
#32 * * * * spark2-submit --master yarn --queue root.hourly --executor-cores=4 --executor-memory=8G --class etl.Discrepancy
32 * * * * spark-submit --master yarn --queue root.hourly --executor-cores=4 --executor-memory=8G --class etl.DiscrepancyHu
#40 * * * * spark-submit --master yarn --queue root.hourly --executor-cores=4 --executor-memory=8G --class etl.RAIndex /mnt
```

Что нам хотелось бы?

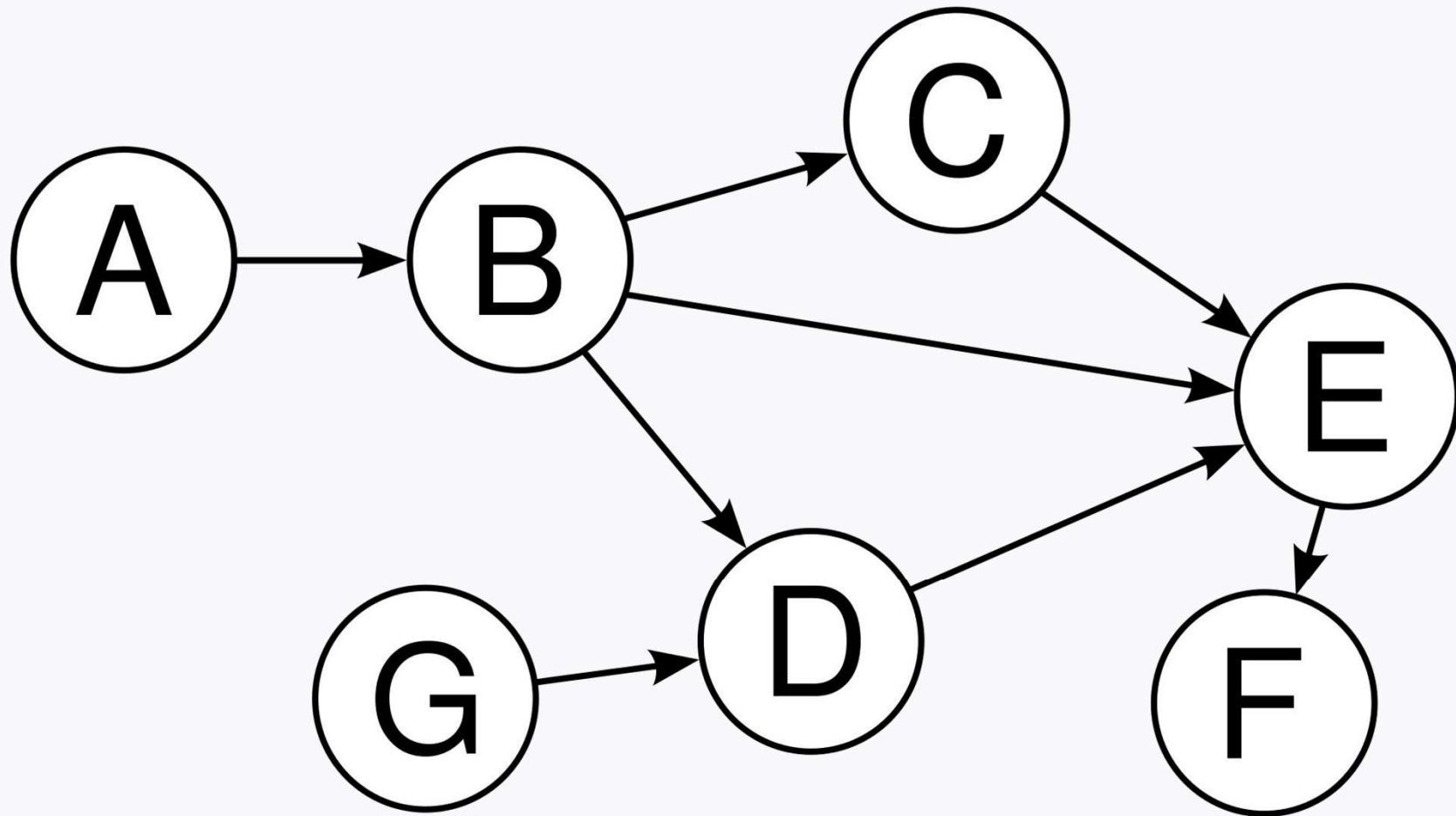
- возможность запускаться по расписанию
- запускать задачи друг за другом в зависимости от чего-либо
- мониторинг, если что-то упало и перезапуск
- декларативное описание графа выполнения

Оркестраторы

Оркестратор -- это штука, которая отвечает за:

- **Планирование задач** (когда запустить)
- **Управление зависимостями** (ждать пока выполнятся другие задачи перед запуском)
- **Репроцессинг** - легко перезапускать упавшие задачи и зависящие от неё
- **Мониторинг** - если задача упала, то нужно об этом уведомить

DAG - Directed Acyclic Graph



The background of the slide is an aerial photograph of a city skyline, likely New York City, featuring numerous skyscrapers. The image is overlaid with a semi-transparent blue and green gradient. A network of thin, light blue lines connects various points across the gradient, creating a digital or data-like pattern. The text "Apache Airflow" is centered in the middle of the slide in a white, sans-serif font.

Apache Airflow

Что такое Apache Airflow

Apache Airflow - платформа для автоматического управления задачами, их расписанием и мониторингом

Разрабатывается с 2014 года(изначально в Airbnb)

<https://github.com/apache/airflow>



Star

18.6k



Fork

7.3k



Как задаются DAG в Airflow?

- Даги в airflow задаются питоновским **кодом**
- Поддерживаются сложные штуки, вроде ветвлений и описания зависимостей из нескольких задач
- Легко расширяется кастомными задачами























```
dag = DAG(
    "easy pipeline",
    default_args=default_args,
    description="An easy airflow DAG",
    schedule_interval=timedelta(days=1),
)

task_1 = BashOperator(
    task_id="print_date",
    bash_command="date",
    dag=dag)

task_2 = BashOperator(
    task_id="sleep",
    depends_on_past=False,
    bash_command="sleep 5",
    retries=3,
    dag=dag
)

task_1 >> task_2
```


Интерфейс Airflow - даги

<div> AirFlow</div> <div>DAGs</div> <div>Tools ▾</div> <div>Browse ▾</div> <div>Admin ▾</div> <div>Docs ▾</div>				
DAGs				
DAG	Filepath	Owner	Task by State	Links
example1	example_dags/example1.py	airflow	<div><div>80</div><div>1</div><div>0</div></div>	<div></div>
example2	example_dags/example2.py	airflow	<div><div>128</div><div>10</div><div>0</div></div>	<div></div>
example3	example_dags/example3.py	airflow	<div><div>138</div><div>5</div><div>0</div></div>	<div></div>

Интерфейс Airflow - задачи

DAG: example2

Tree View

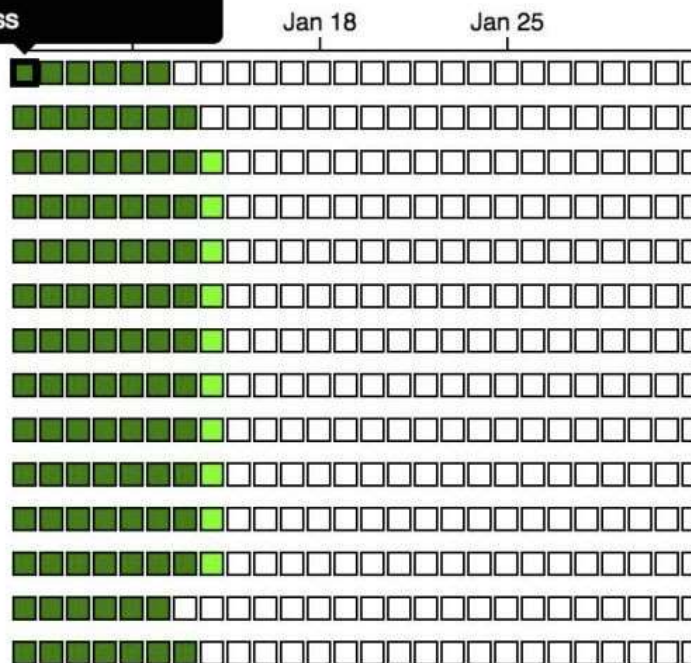
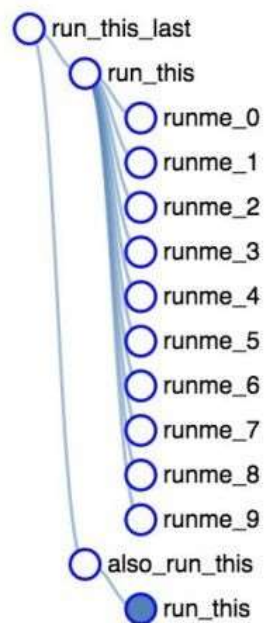
Graph View

Run: 2015-01-07T00:00:00
Started: 2015-02-01T20:22:22
Ended: 2015-02-01T20:22:22
Duration: 0
State: success

Landing Times

Gantt

Code



Как даги можно запускать?

- **по расписанию** (весь крон к нашим услугам)
- **через UI** (обычно используется, когда что-то упало)
- **Через API** (когда интегрируете airflow с внешними системами) - <http://airflow.apache.org/docs/stable/rest-api-ref.html>

Cron Presets

preset	meaning	cron
None	Don't schedule, use for exclusively "externally triggered" DAGs	
@once	Schedule once and only once	
@hourly	Run once an hour at the beginning of the hour	0 * * * *
@daily	Run once a day at midnight	0 0 * * *
@weekly	Run once a week at midnight on Sunday morning	0 0 * * 0
@monthly	Run once a month at midnight of the first day of the month	0 0 1 * *
@quarterly	Run once a quarter at midnight on the first day	0 0 1 */3 *
@yearly	Run once a year at midnight of January 1	0 0 1 1 *

The background of the slide is a high-angle, aerial photograph of a city skyline, likely New York City, featuring numerous skyscrapers and dense urban development. The image is overlaid with a semi-transparent blue and green gradient. A network of thin, light blue lines connects various points across the gradient, creating a digital or network-like pattern. The text "Альтернативные оркестраторы" is centered in white, sans-serif font.

Альтернативные оркестраторы

Luigi

Довольно старый оркестратор, очень похожий на airflow
не содержит функционал шедулинга

```
class MyTask1(luigi.Task):
    x = luigi.IntParameter()
    y = luigi.IntParameter(default=0)

    def run(self):
        print(self.x + self.y)

class MyTask2(luigi.Task):
    x = luigi.IntParameter()
    y = luigi.IntParameter(default=1)
    z = luigi.IntParameter(default=2)

    def run(self):
        print(self.x * self.y * self.z)

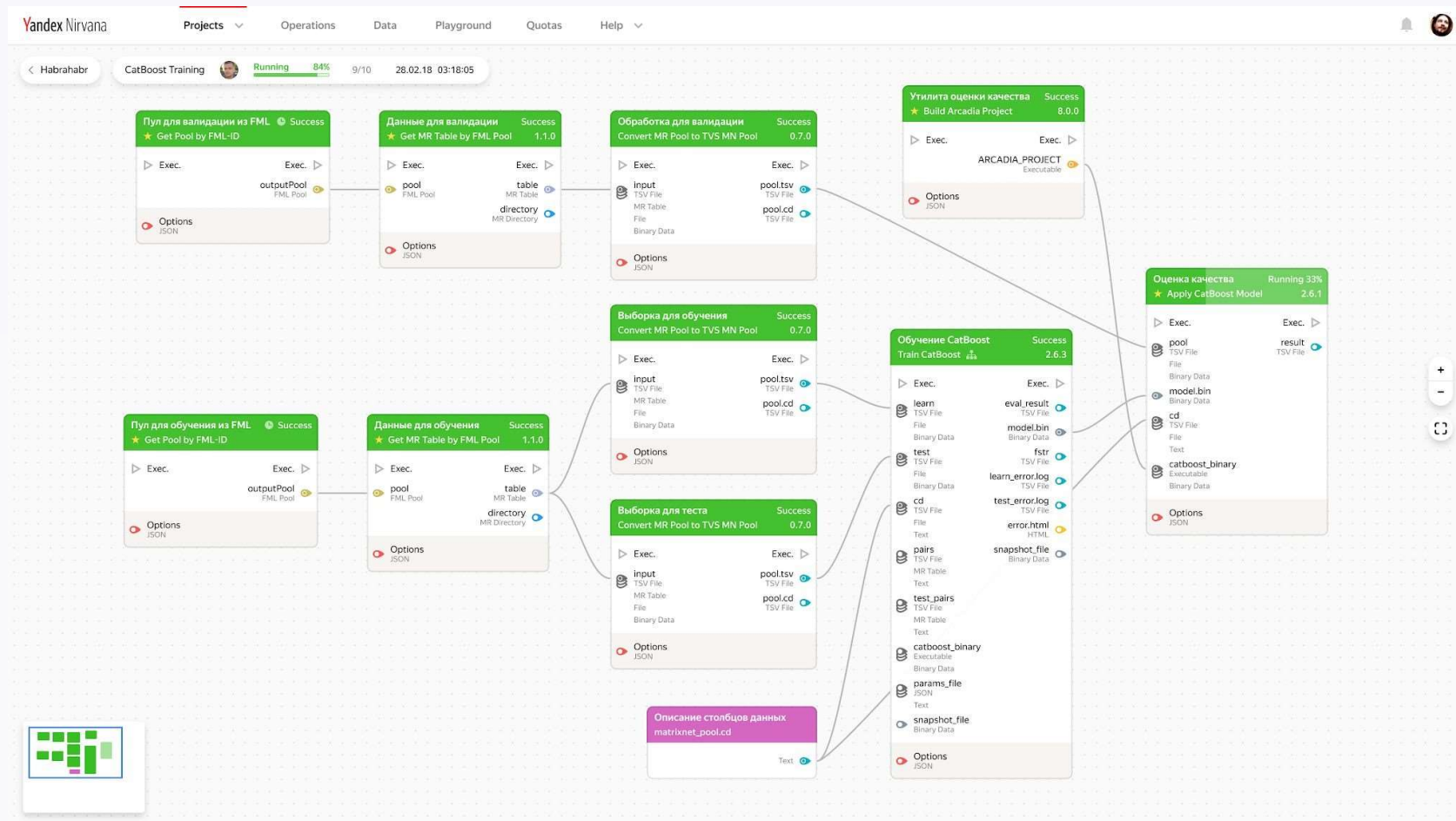
if __name__ == '__main__':
    luigi.build([MyTask1(x=10), MyTask2(x=15, z=3)])
```

Dagster

The screenshot displays the Dagster web interface in a browser window. The address bar shows the URL `localhost:3000/playground/airline_demo_ingest_pipeline/`. The interface is divided into several sections:

- Left Sidebar:** Contains navigation icons for Pipelines, Solids, Runs, Playground (active), and Schedules. The version number 0.7.6 is shown at the bottom.
- Header:** Shows the pipeline name `airline_demo_ingest_pipeline` and a 'Workspace' button.
- Main Editor:** Displays the pipeline configuration in YAML format. The configuration includes:
 - `target_folder: /tmp/dagster/airline_data/file_cache`
 - `pyspark:` section with `config:`, `spark_conf:`, `spark:`, `jars:`, and `packages:` (listing Databricks Spark Avro and Redshift connectors).
 - `solids:` section with a `process_q2_coupon_data` solid.
 - `inputs:` section with an `s3_coordinate` input.
 - `bucket: dagster-airline-demo-source-data`
 - `key: test/Origin and Destination Survey DB1BCoupon 2018 2.zip`
- Right Panel:** Contains a 'db_info' section with a 'config' field, a 'file_cache' section with a 'config' field, and a 'pyspark?' section with a 'config?' field. A tooltip提示 'Ctrl+Space to show auto-completions inline.' is visible.
- Bottom Panel:** Includes tabs for 'ERRORS', 'RUNTIME', 'RESOURCES', and 'SOLIDS'. The 'SOLIDS' tab is active, showing a list of solids: `april_on_time_s3_to_df`, `download_q2_sfo_weather`, `ingest_q2_sfo_weather`, `join_q2_data`, `june_on_time_s3_to_df`, `load_q2_on_time_data`, `load_q2_sfo_weather`, `master_cord_s3_to_df`, `may_on_time_s3_to_df`, `process_q2_coupon_data`, `process_q2_market_data`, `process_q2_ticket_data`, and `process_sfo_weather_data`. A 'Start Execution' button is located at the bottom right.

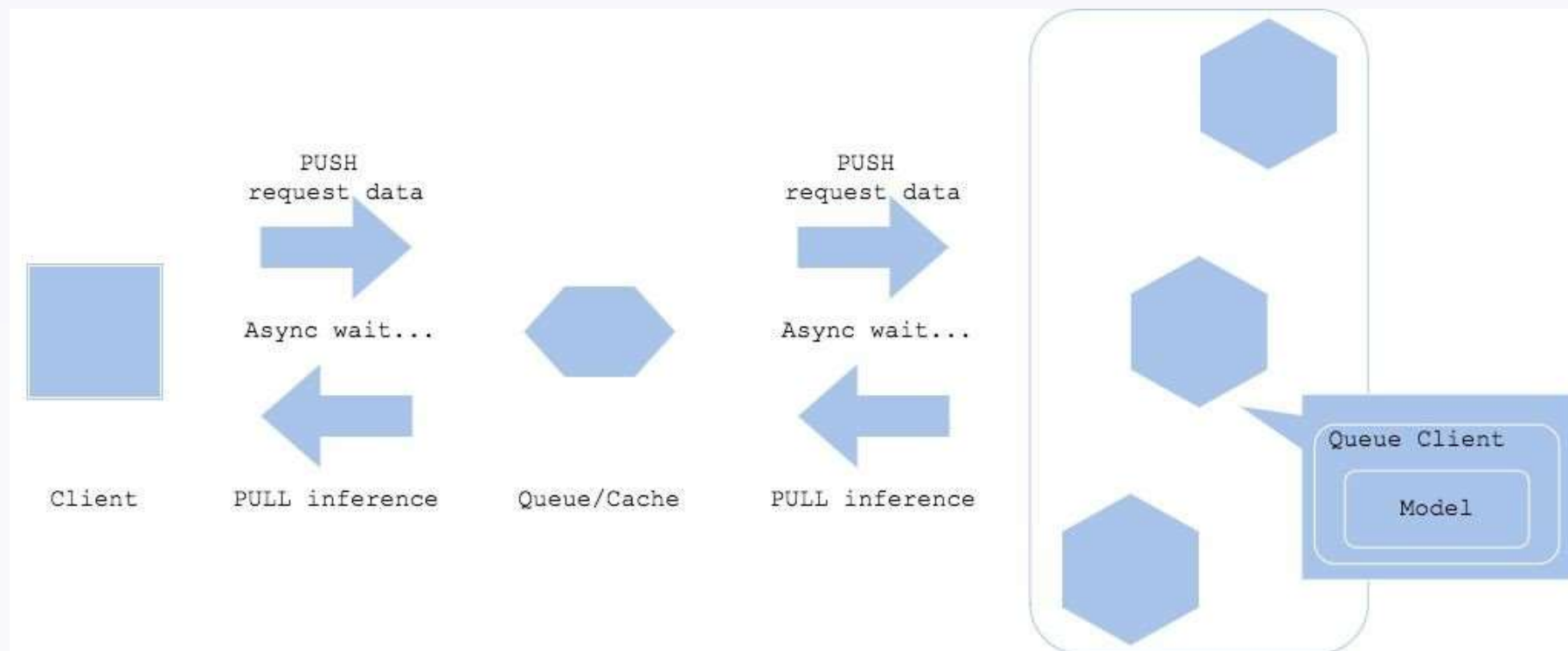
Яндекс Нирвана





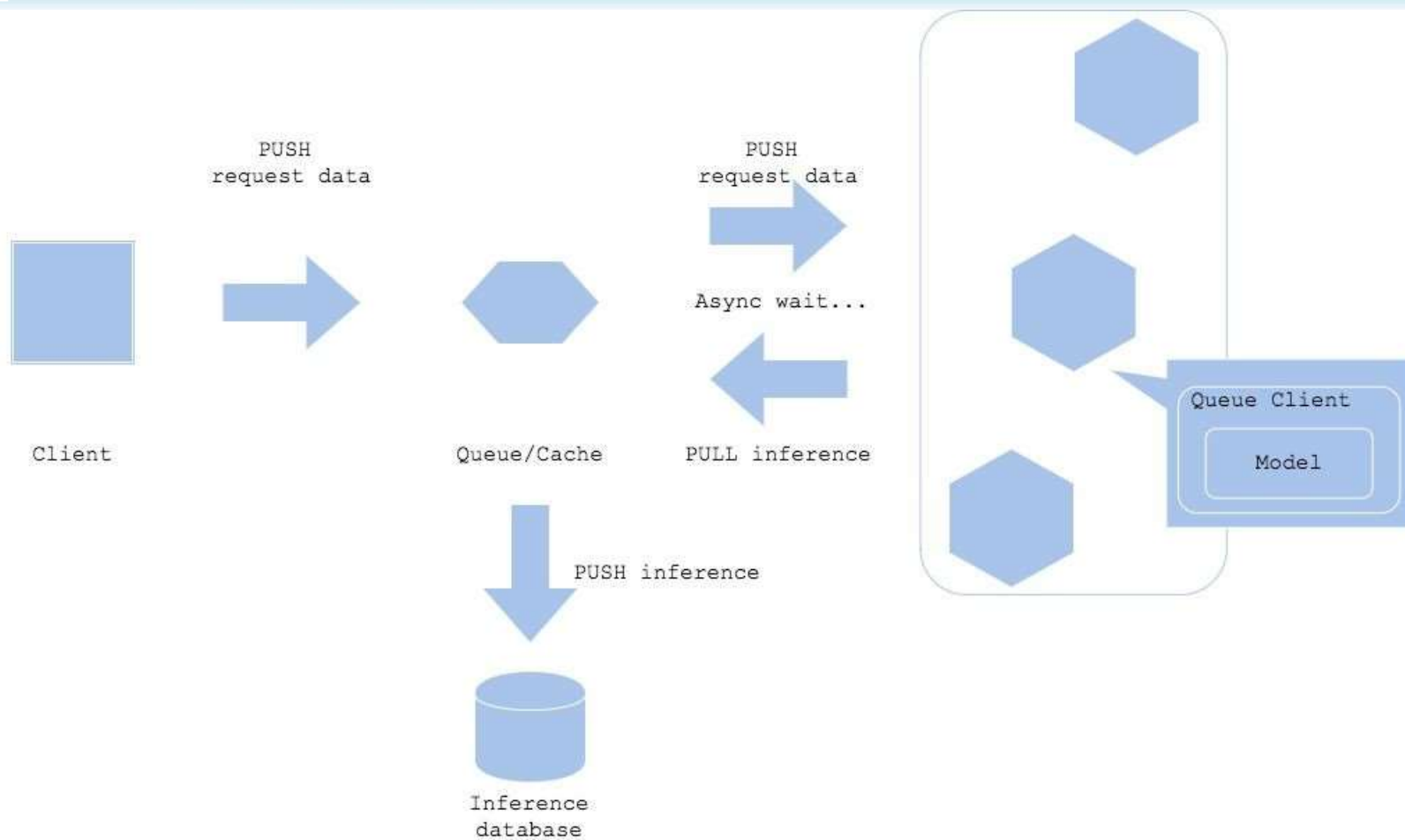
Асинхронная обработка

Асинхронный паттерн



https://github.com/mercari/ml-system-design-pattern/blob/master/Serving-patterns/Asynchronous-pattern/design_en.md

Асинхронный паттерн



Асинхронный паттерн - когда нужно использовать?

Если не нужно обрабатывать прямо сейчас

ПЛЮСЫ	МИНУСЫ
Можно отделить бизнес логику и логику предсказания	Как правило не подходит для обработки в реальном времени
Более высокая(синхронным) пропускная способность	Нужны очереди/кеши, etc
Вы не заблокированы временем предсказания	

https://github.com/mercari/ml-system-design-pattern/blob/master/Serving-patterns/Asynchronous-pattern/design_en.md

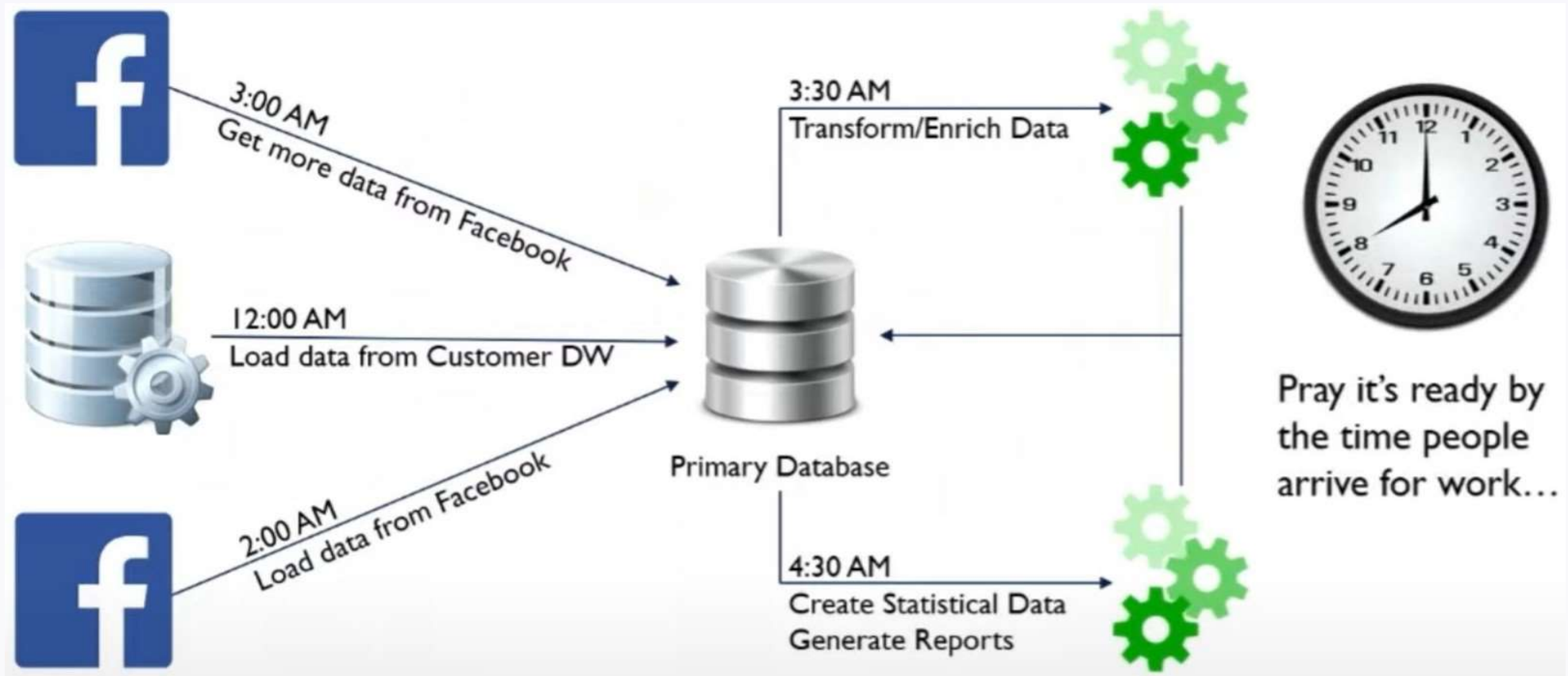
The background of the slide is a high-angle, blue-tinted aerial photograph of a city skyline, likely New York City, showing numerous skyscrapers and buildings. A semi-transparent blue band with a white network diagram of interconnected nodes and lines runs horizontally across the middle of the image, serving as a backdrop for the title text.

Потоковая обработка и брокеры сообщений

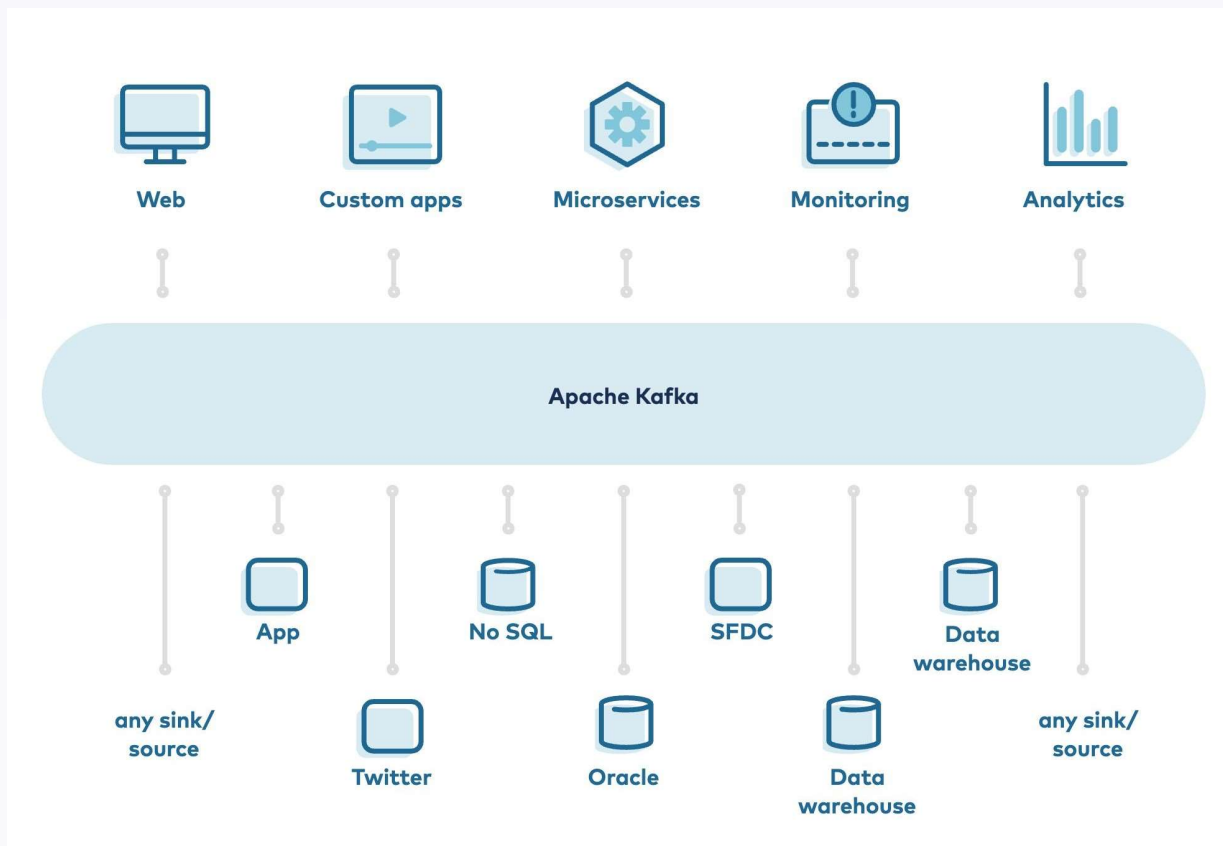
Quick recap

- 1) Данные собираем в очередь. Наиболее распространенный брокер - Kafka.
- 2) Из очереди обрабатываем потоково, либо батчами.
- 3) Поточковых фреймворков много, нет серебряной пули.
- 4) Результаты пишем либо в очередь, либо складываем в хранилища.

Standard ETL



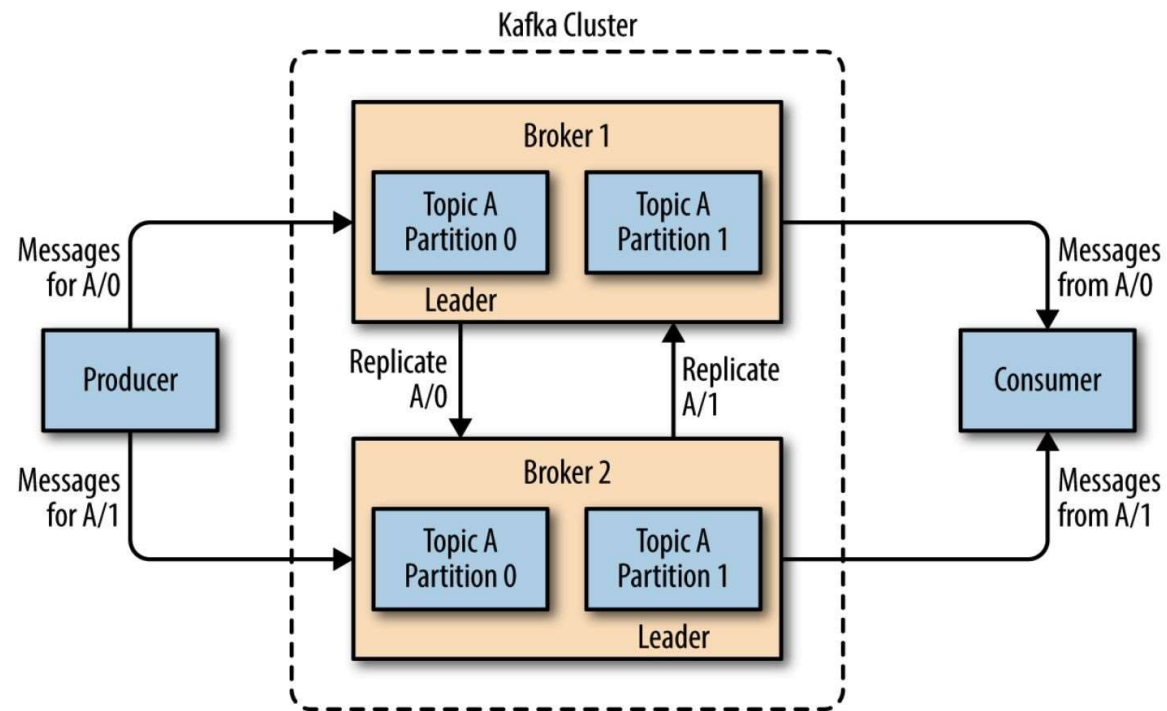
Quick recap. Kafka



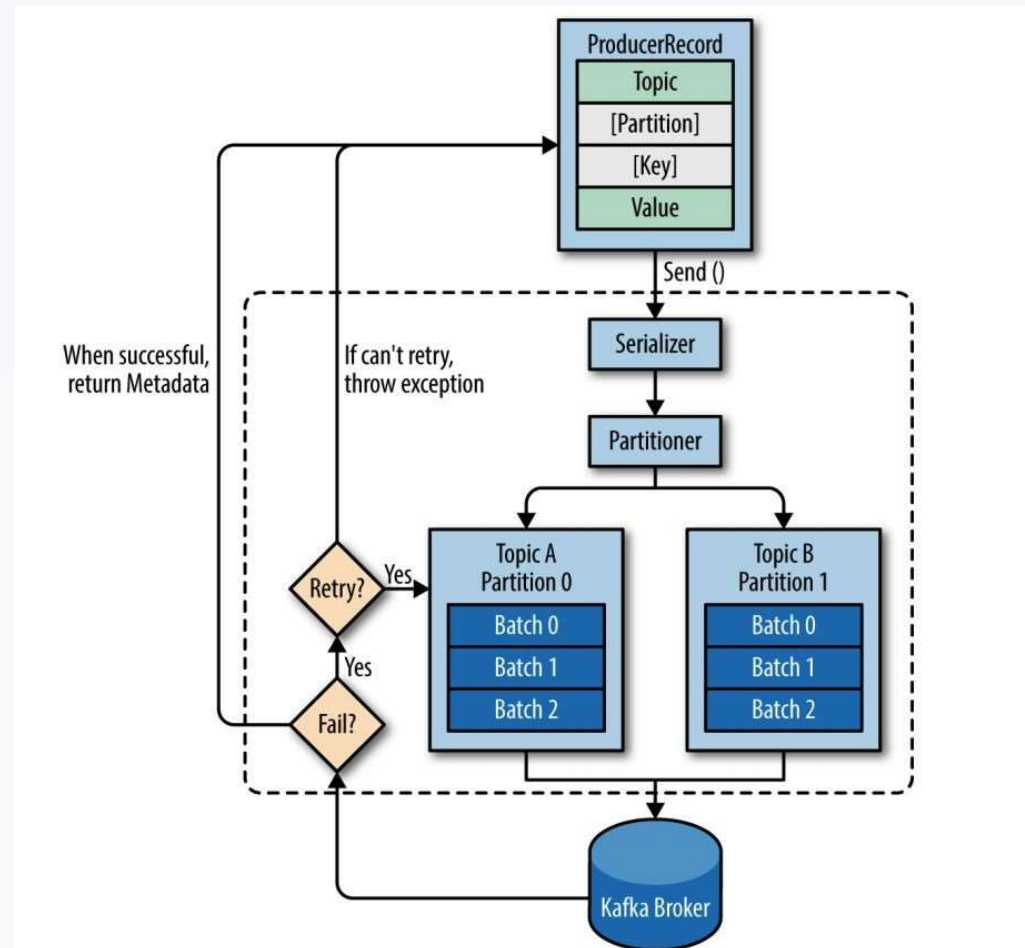
Ключевые возможности:

1. publish + subscribe
2. хранение
3. обработка

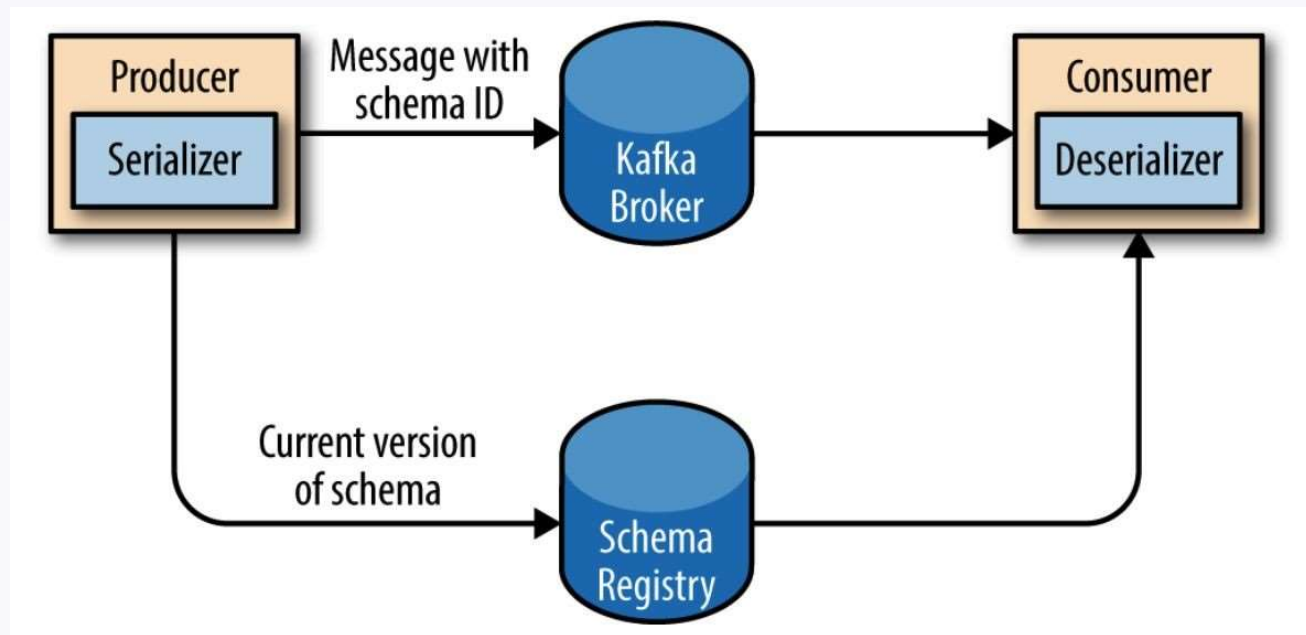
Quick recap. Kafka



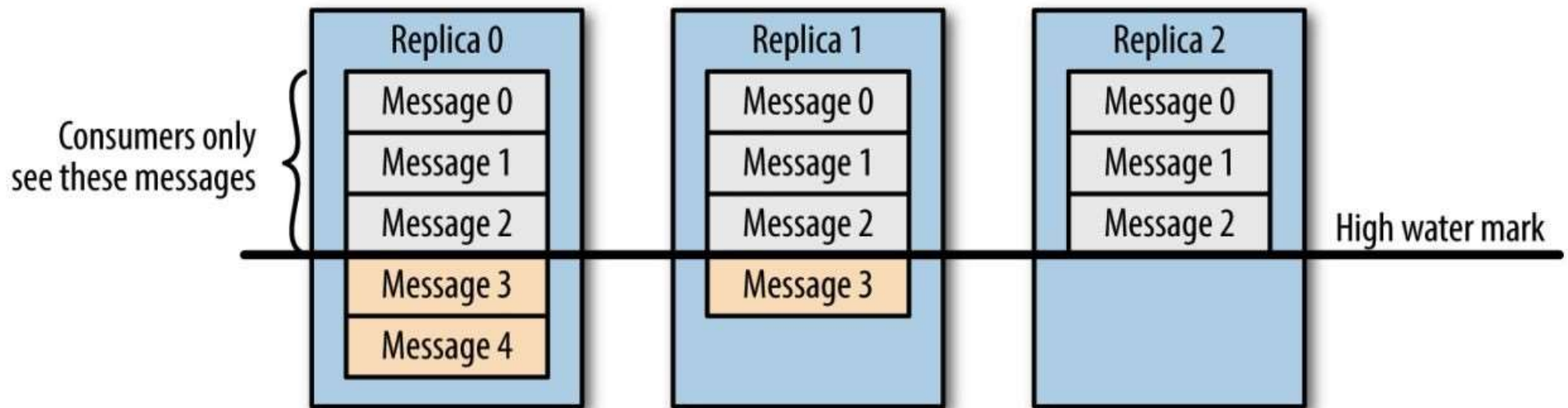
Quick recap. Kafka



Quick recap. Kafka



Quick recap. Kafka



Cluster Manager for Apache Kafka

[← test1](#)

Topic Summary

Replication	1
Number of Partitions	4
Sum of partition offsets	71918
Total number of Brokers	1
Number of Brokers for Topic	1
Preferred Replicas %	100
Brokers Skewed %	0
Brokers Spread %	100
Under-replicated %	0

Metrics

Rate	Mean	1 min	5 min	15 min
Messages in /sec	0.27	10.00	9.99	8.96
Bytes in /sec	0.2k	10k	10k	9.4k
Bytes out /sec	2k	10k	10k	19k
Bytes rejected /sec	0.00	0.00	0.00	0.00
Failed fetch request /sec	0.00	0.00	0.00	0.00
Failed produce request /sec	0.00	0.00	0.00	0.00

Operations

Delete TopicReassign PartitionsGenerate Partition Assignments

Add PartitionsUpdate ConfigManual Partition Assignments

Partitions by Broker

Broker	# of Partitions	Partitions	Skewed?
0	4	(0,1,2,3)	false

Consumers consuming from this topic

perf-consumer-14274	ZK
perf-consumer-95553	ZK
perf-consumer-93654	KF

- Manage multiple clusters
- Easy inspection of cluster state (topics, consumers, offsets, brokers, replica distribution, partition distribution)
- Run preferred replica election
- Generate partition assignments with option to select brokers to use
- Run reassignment of partition
- Create a topic with optional topic configs
- Delete topic
- Topic list now indicates topics marked for deletion
- Batch generate partition assignments for multiple topics with option to select brokers to use
- Batch run reassignment of partition for multiple topics
- Add partitions to existing topic
- Update config for existing topic

Python libs for Apache Kafka

no. msg	1K	10K	100K	1M
confluent-producer-async	0.02s	0.06s	0.53s	4.34s
confluent-producer-sync	3.15s	31.12s	309.29s	x
confluent-consumer	3.07s	3.08s	3.32s	5.66s
pykafka-producer-async	10.04s	15.38s	12.41s	45.69s
pykafka-producer-sync	6.93s	24.33s	188.89	x
pykafka-consumer	0.26s	0.84s	6.91s	60.69s
kafka-python-producer-async	0.08s	0.67s	6.72s	68.26s
kafka-python-producer-sync	3.27s	30.81s	318.29	x
kafka-python-consumer	0.06s	0.78s	3.15s	33.94s

<https://towardsdatascience.com/3-libraries-you-should-know-to-master-a-pache-kafka-in-python-c95fdf8700f2>

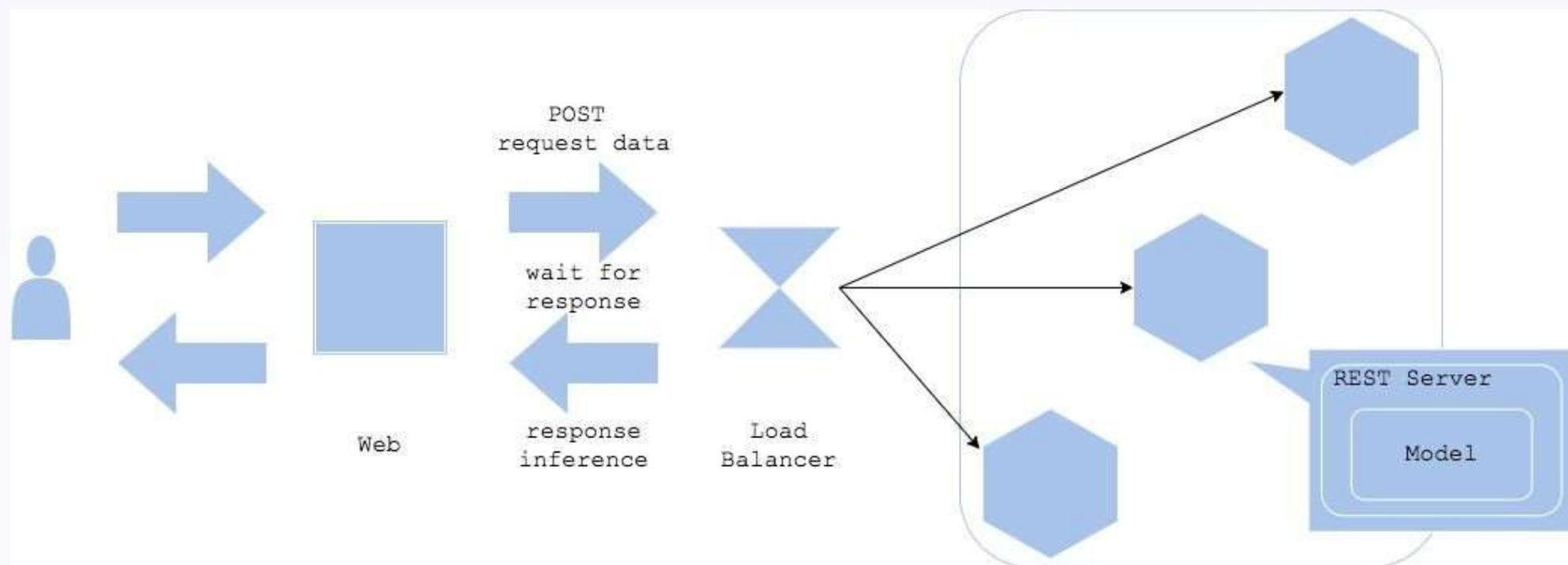


Синхронная обработка

Нейронная сеть - выделяет объекты на изображении



Синхронный паттерн



https://github.com/mercari/ml-system-design-pattern/blob/master/Serving-patterns/Synchronous-pattern/design_en.md

Синхронный паттерн - когда нужно использовать?

Когда нужен результат предсказания, чтобы сделать следующий шаг

ПЛЮСЫ	МИНУСЫ
Очень просто реализовать	Скорость предсказания будет бутылочным горлышком производительности системы
Как правило, низкое latency	Если предсказание долгое, то нужно, чтобы пользователь этого не замечал

https://github.com/mercari/ml-system-design-pattern/blob/master/Serving-patterns/Synchronous-pattern/design_en.md



REST

Representational State Transfer

Центральная идея: концепция распределенного приложения, в которой каждый запрос клиента (REST-запрос) содержит в себе исчерпывающую информацию о желаемом ответе сервера, и сервер не обязан сохранять информацию о состоянии клиента (клиентскую сессию).

Сам термин REST впервые упоминается в диссертации Роя Филдинга, одного из создателей протокола HTTP.

Пример RESTless: FTP

220 Hello World!

USER anonymous

331 Anonymous login ok, send your complete email address as your password

PASS *****

230 Logged in anonymously.

PASV

227 Entering Passive Mode (192,168,254,253,233,92) //Клиент должен открыть соединение на переданный IP

LIST

150 Here comes the directory listing.

226 Directory send OK.

QUIT

221 Goodbye.

Пример RESTfull: HTTP

GET / HTTP/1.1

Host: example.com

Authorization: Bearer <token>

HTTP/1.1 200 OK

Date: Sat, 09 Oct 2010 14:28:02 GMT

Server: Apache

Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT

Content-Length: 29769

Content-Type: text/html

<!DOCTYPE html> (остальные байты запрошенного ресурса)

Практическое использование REST

Самые частые реализации REST архитектуры используют:

- протокол - HTTP/1.1
- шифрование - TLS
- формат сериализации - JSON
- аутентификация - JWT

REST API на python



aiohttp



FastAPI



Sebastián Ramírez @tiangolo · 11 июл. 2020 г.



I saw a job post the other day. 📄

It required 4+ years of experience in FastAPI. 🧑

I couldn't apply as I only have 1.5+ years of experience since I created that thing. 😅

Maybe it's time to re-evaluate that "years of experience = skill level". ♻️



1 тыс.



47,7 тыс.



176,7 тыс.



FastAPI

- высокая производительность (благодаря Starlette и pydantic)
- быстрая скорость разработки
- позволяет делать меньше багов
- интуитивный интерфейс
- легкий в изучении
- небольшой объем кода
- production ready с автоматическим документированием
- основан на стандартах OpenAPI (Swagger) и JSON Schema

Пример: FastAPI

demo.py:

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_healthcheck():
    return {"status": "Green"}
```

Запуск:

```
$uvicorn main.app --reload
```

Итоги - тезисы

1 Рассмотрели каждый паттерн

2 Рассмотрели пример на spark

3 Узнали про альтернативные оркестраторы

4 Рассмотрели синхронный и асинхронный паттерны

без картинок

0-9

