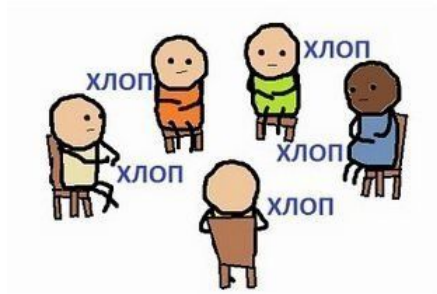


Datapipe — трансформация данных при помощи K8s и S3

Захарченко Сергей - "Эпоха Восемь"

Вступление

Компания “Эпоха Восемь” начала свою деятельность как стартап в 2017 году и специализируется на BI, ML, CV, Recsys решениях для бизнеса под ключ.



Источник: pikabu.ru

Моя карьера в Эпохе началась в 2022 году на позиции аналитика, а с 2023 мне стало интересно погружаться в инженерию данных.

Постановка задачи

Постановка задачи

В 2022 году к нам обратился крупный отечественный ритейл по вопросу импортозамещения мощной зарубежной BI системы Amplitude, сотрудничество с которой прекращалось из-за санкций.

Amplitude представляет собой SaaS под ключ:

- сбор событий от пользователей;
- хранение и обработка данных;
- подготовка отчётов.



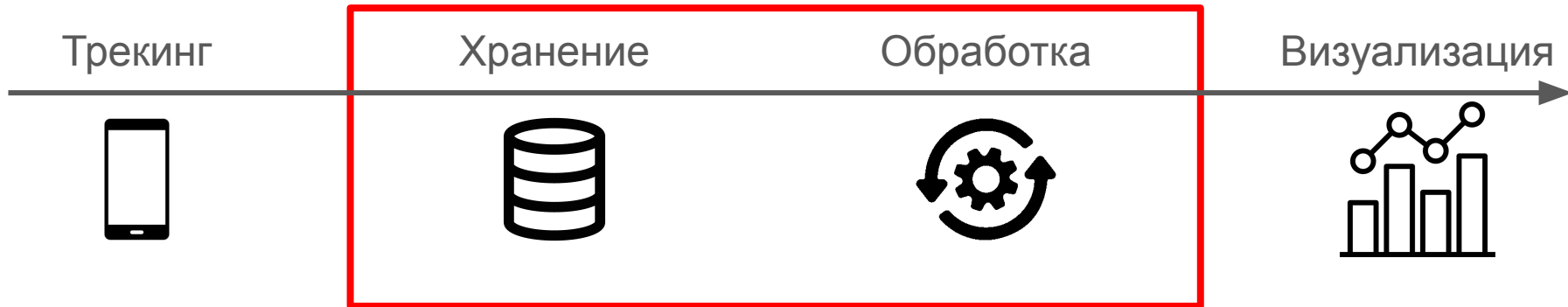
Источник: amplitude.com

Постановка задачи

Решаем задачу в области инженерии данных



Источник: amplitude.com



Постановка задачи

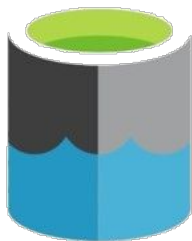
Процессинг данных - тривиальная задача при наличии мощного Data Warehouse.

Нетривиальная - при отсутствии.



Google
Big Query

Источник: cloud.google.com



Azure Data Lake

Источник: azure.microsoft.com



Источник: snowflake.com



amazon
ATHENA

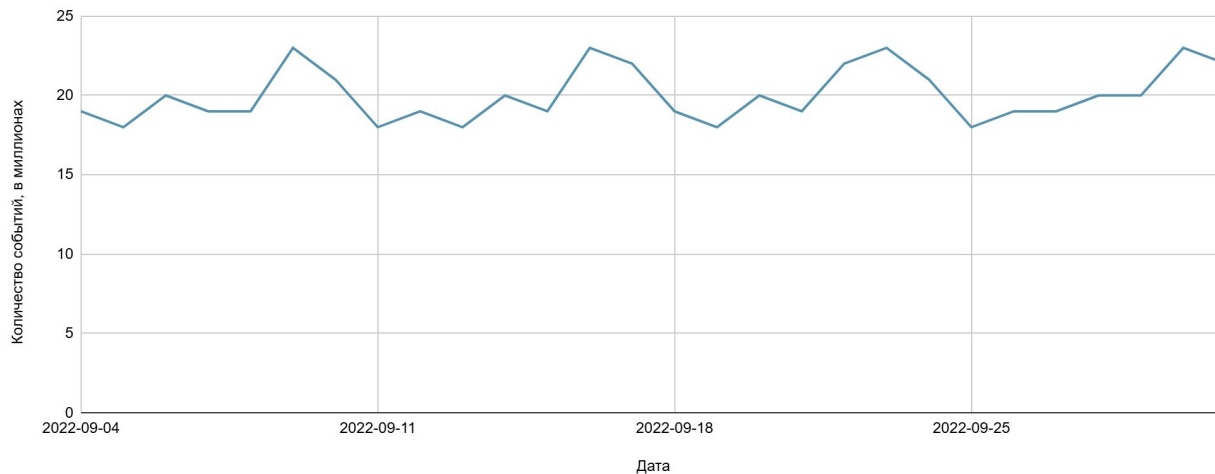
Источник: aws.amazon.com

Постановка задачи

Объём данных - около 20кк событий в сутки и до 50кк событий в пиках.

Терабайты данных в год в виде parquet файлов со сжатием snappy.

Объём данных в сутки



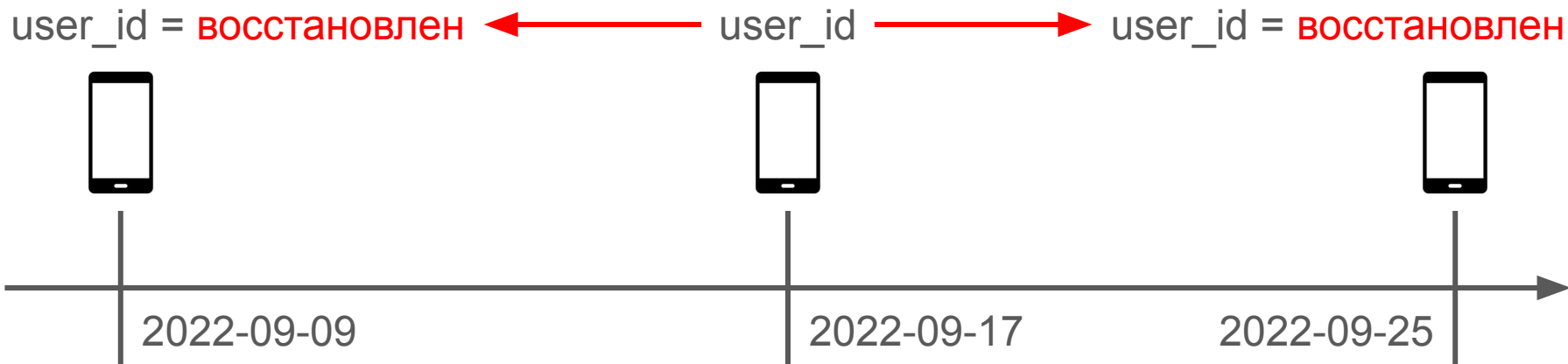
Постановка задачи

Нетривиальные алгоритмы - восстановление пользователя как в Amplitude (user_pseudo_id & user_id), воспроизведение когорт и т.д.



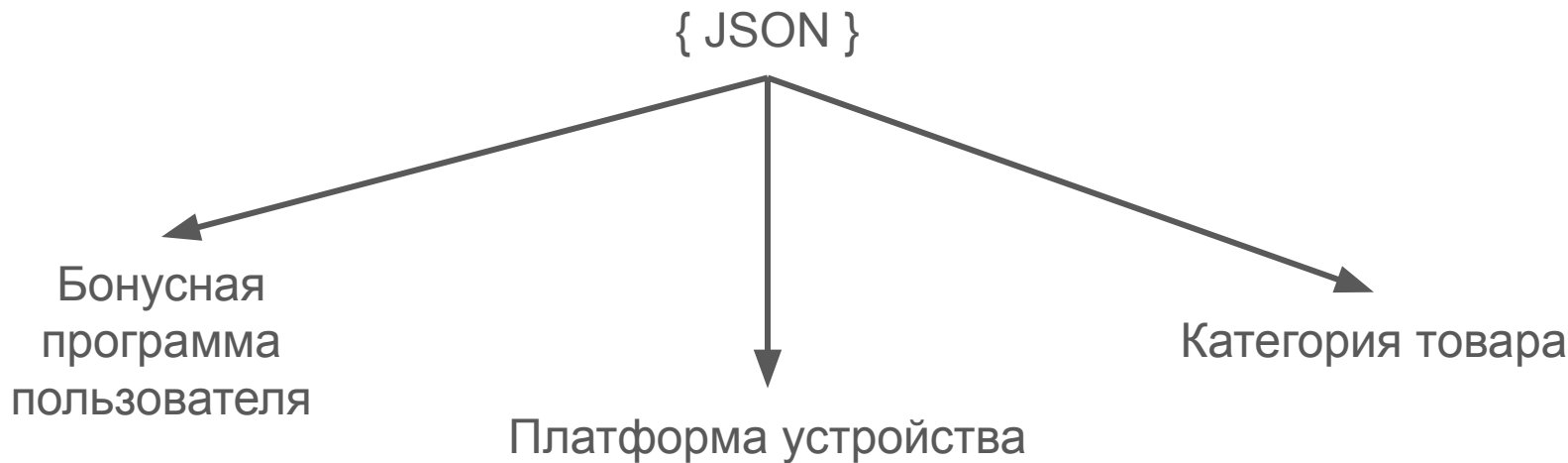
Постановка задачи

Нетривиальные алгоритмы - восстановление пользователя как в Amplitude (user_pseudo_id & user_id), воспроизведение когорт и т.д.



Постановка задачи

Ресурсоёмкие вычисления - парсинг JSON структур с наборами атрибутов пользователя, устройства, товара и т.д.



Постановка задачи

Класс задач не для SQL - трекинг геолокации магазинов по IP устройства, дополнительная категоризация продукции и т.д.

IP адрес



Атрибуты продукции



Дополнительная категоризация

Первый подход

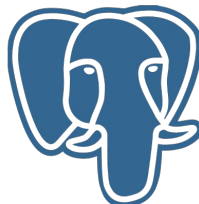
Первый подход

Бизнес выбрал Облако Яндекса из-за наличия managed сервисов и соотношения цены и качества.



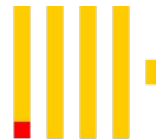
Яндекс Облако

Источник: yandex.cloud



PostgreSQL

Источник: postgresql.org



ClickHouse

Источник: clickhouse.com

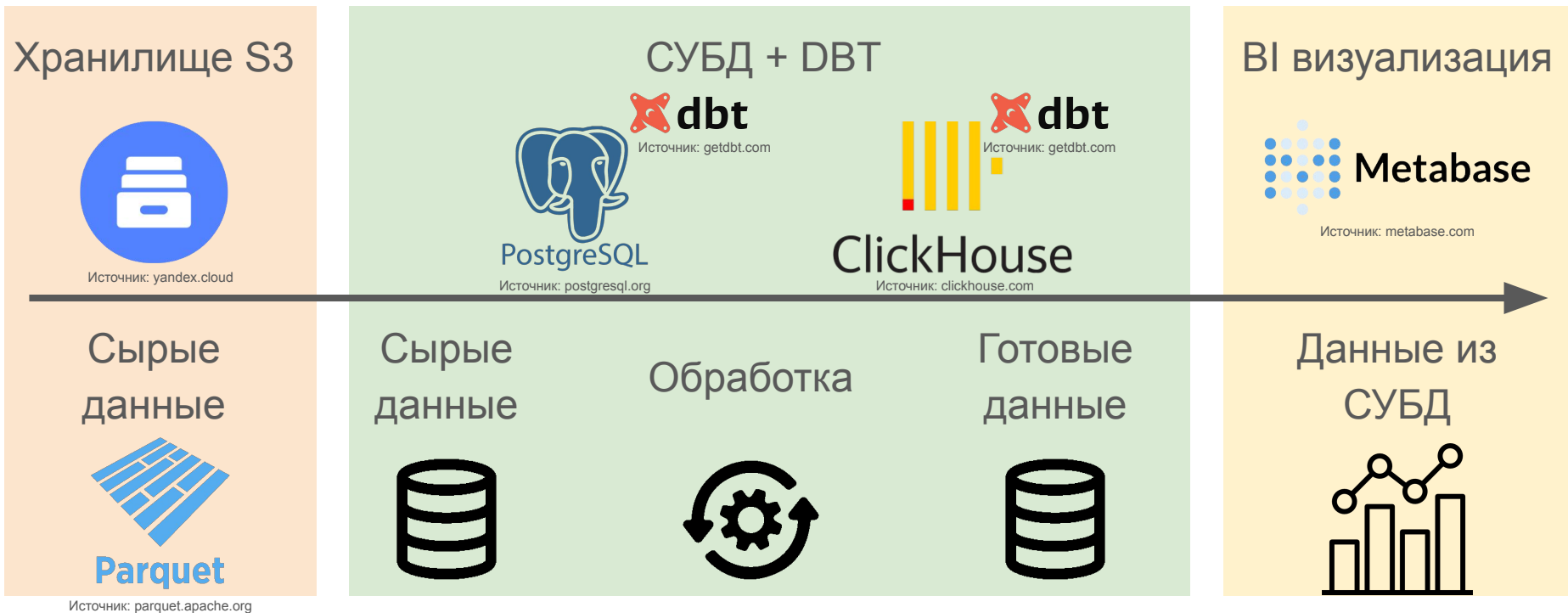


Хранилище S3

Источник: yandex.cloud

Первый подход

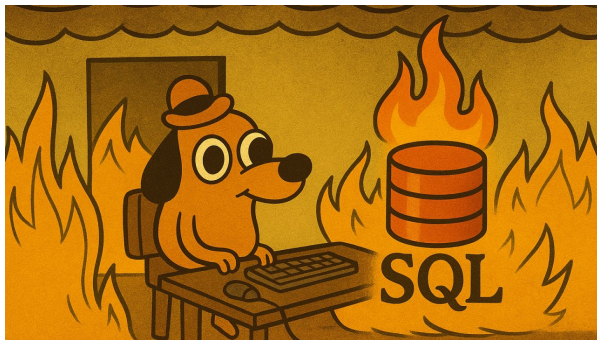
Стараемся использовать SQL для удобства команды заказчика.



Первый подход

Существенный минус - на мощностях СУБД происходит:

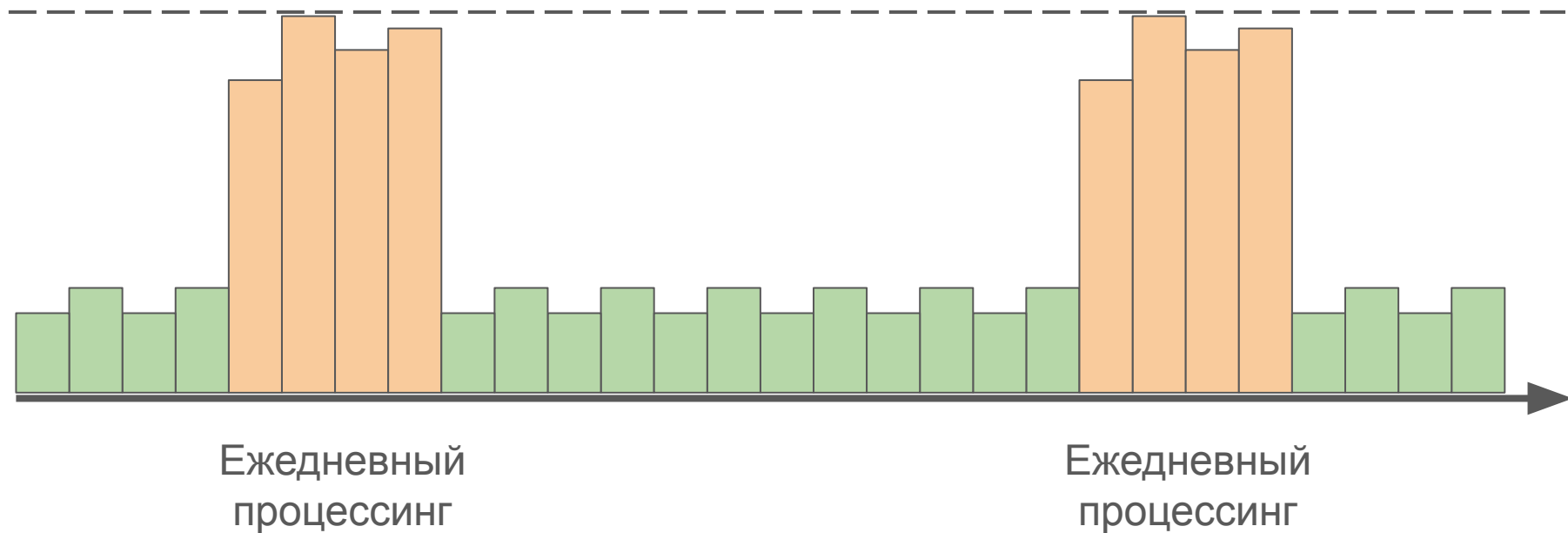
- и ежедневная ресурсоёмкая обработка данных;
- и “ad hoc” запросы;
- и лавинообразные запросы к витринам для визуализации.



Источник: reddit.com

Первый подход

Загрузка СУБД неравномерна - приходится оплачивать мощности под пик загрузки.



Первый подход

В результате:

- для эффективной обработки данных требовался мощный кластер СУБД;
- кластер использовал 100% мощностей только несколько часов в сутки;
- во время простоя необходимо оплачивать аренду мощностей.

Порядок стоимости:
сотни тысяч рублей
в месяц

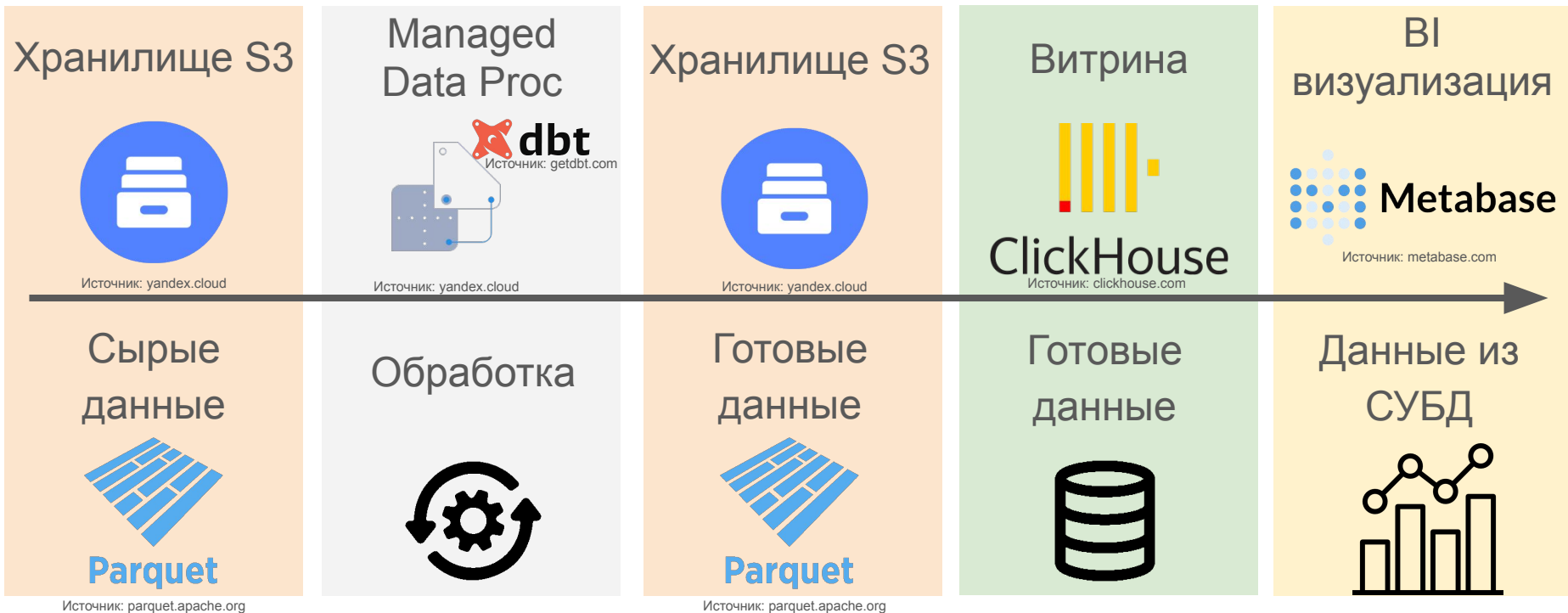


Источник: magnific.com

Второй подход

Второй подход

Всё ещё стараемся использовать SQL для удобства команды заказчика.



Второй подход

Исчезла необходимость оплачивать мощный кластер СУБД на 24 часа в сутки.

Теперь ресурсоёмкие вычисления выполнялись на нодах Data Proc, которые оплачивались по факту использования.

СУБД стала отвечать только на аналитические запросы.

Порядок стоимости:
десятки тысяч рублей в месяц



Второй подход

В результате:

- чек за облако сократился;
- частью настроек Managed Hive управляет поставщик сервиса;
- Hadoop сложен для администрирования;
- не все задачи алгоритмически эффективно решаются при помощи SQL и DBT.



Третий подход

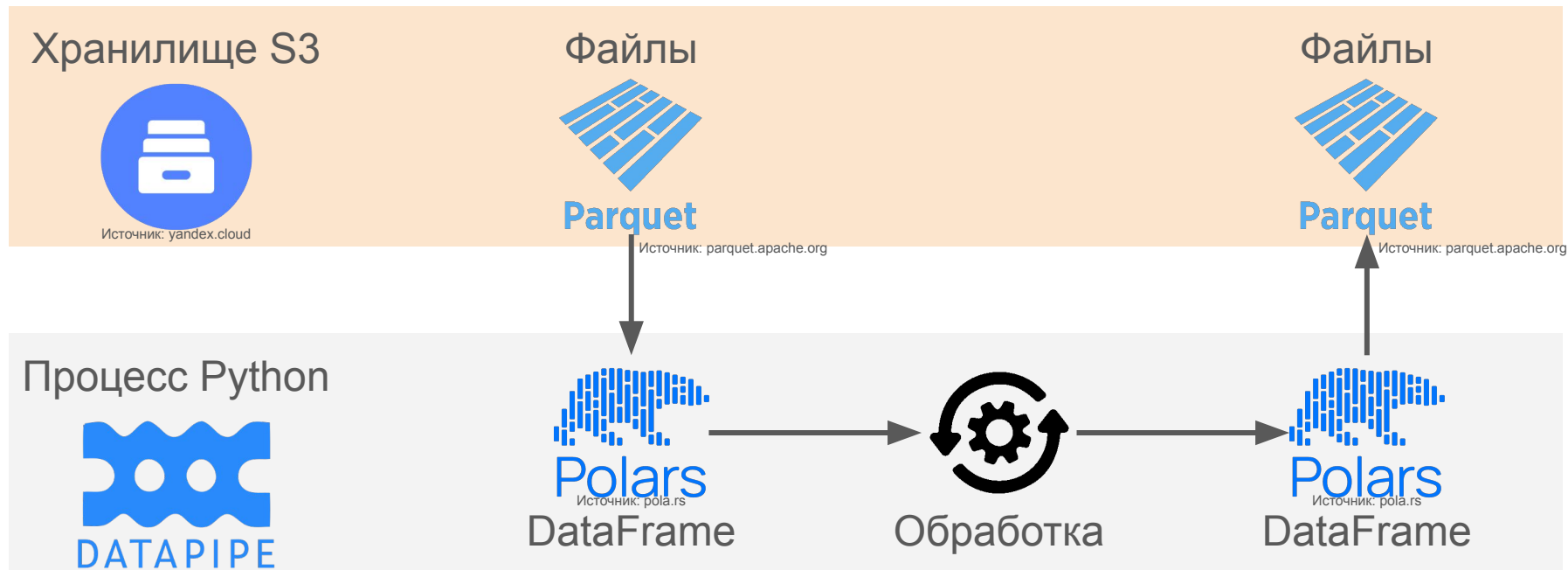
Третий подход

Отказываемся от SQL и пишем процессинг на Datarpipe.



Третий подход

Использовали Python Polars с Apache Arrow под капотом.



Третий подход

Использовали Python Polars с Apache Arrow под капотом.

Чтение данных:

```
21 # Loading
22 df_pl = pl.scan_parquet(
23     source=FILEPATH_INPUT,
24     storage_options=STORAGE_OPTIONS,
25     hive_partitioning=True,
26     retries=5,
27 ).collect()
28
```

Обработка:

```
30 # Processing
31 df_pl = df_pl.select(
32     [
33         pl.col("user_id"),
34         pl.col("user_name"),
35         pl.col("user_phone"),
36     ]
37 ).filter(
38     pl.col("user_phone").str.contains("+7")
39 )
40
```



Запись:

```
42 # Saving
43 with fsspec.open(FILEPATH_OUTPUT, "wb") as f:
44     df_pl.write_parquet(f, compression="snappy")
45
```

Третий подход

Почему не Airflow Assets?

Asset Users

partitions	files
partition_1	filepath_1
partition_2	filepath_2
...	...
partition_n	filepath_n



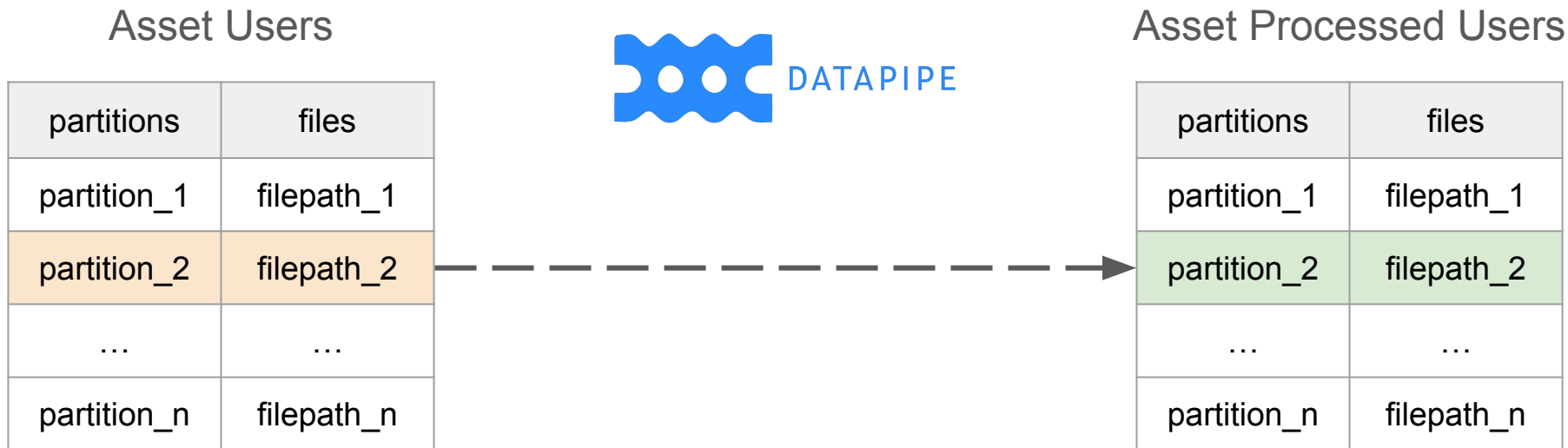
Asset Processed Users

partitions	files
partition_1	filepath_1
partition_2	filepath_2
...	...
partition_n	filepath_n

Третий подход

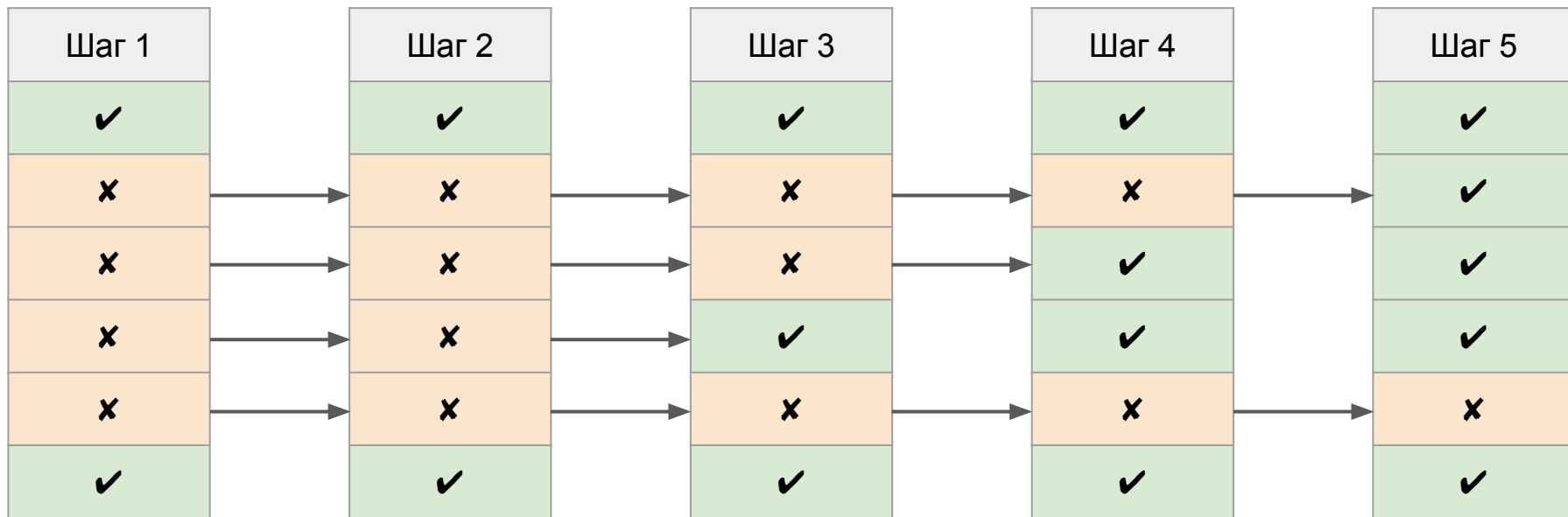
Потому что нужен более эффективный трекинг изменений.

Нет фактических изменений в данных - следующий шаг не выполняется.

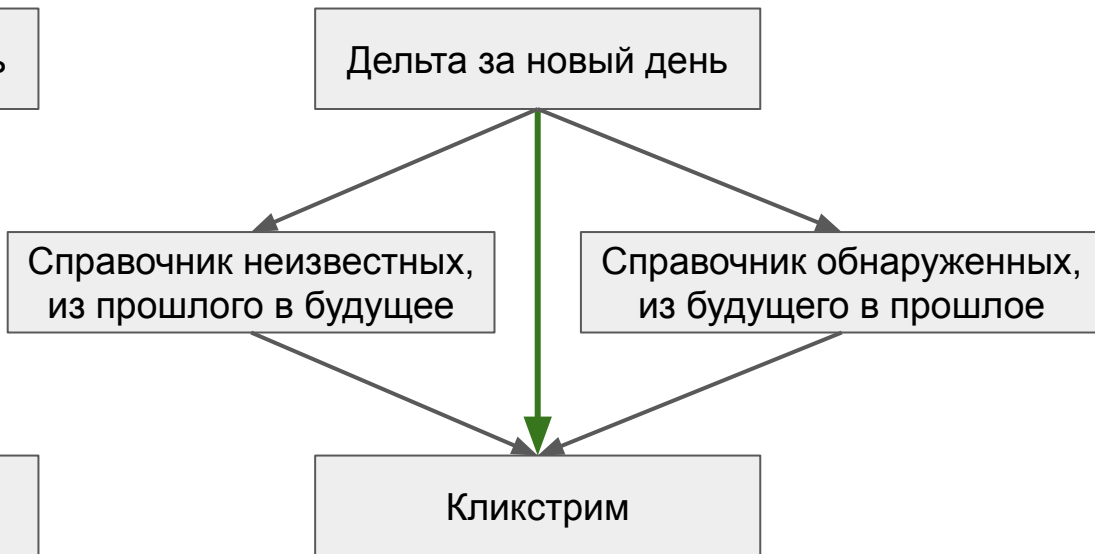
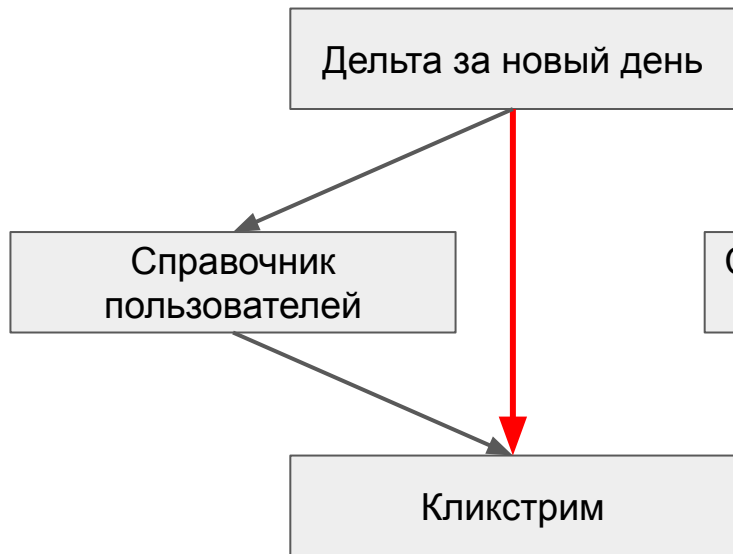


Третий подход

Точечный инкремент и более гибкие графы преобразований Datarpipe способствовали оптимизации.



Третий подход



Третий подход

Ядро процессинга на Python Polars использует Apache Arrow под капотом и имеет сопоставимую с DataProc производительность.

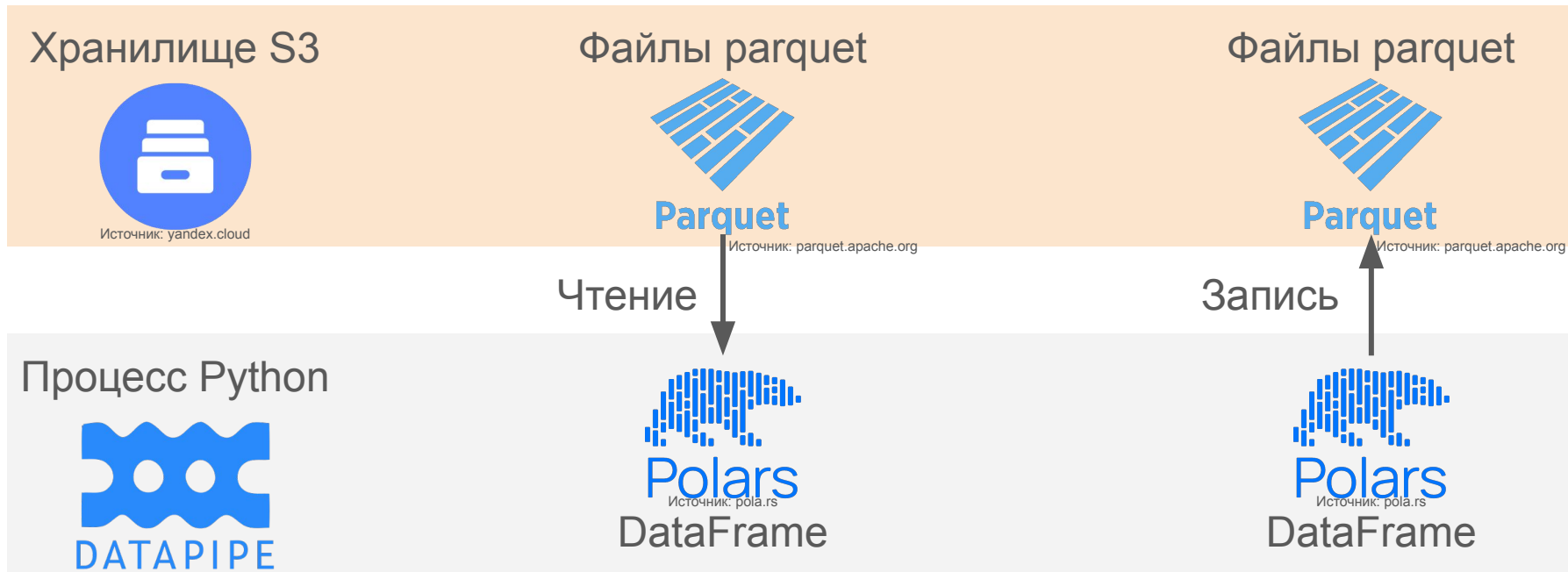


Источник: reddit.com



Третий подход

Чек за облако ещё сократился за счёт управления чтением и записью.



Третий подход

Привлечение сторонних Python библиотек позволило эффективно решать класс задач не для SQL.



Источник: python.org



Источник: reddit.com

Сводка

	СУБД	Dataproc	Datapipeline
Стоимость	Дорого	Дёшево	Дёшево
Компетенции	Легко: SQL	Сложно: Hadoop	Умеренно: SQL, Python, S3

Плюсы: +++++++

Минусы: -----

Подводные камни:

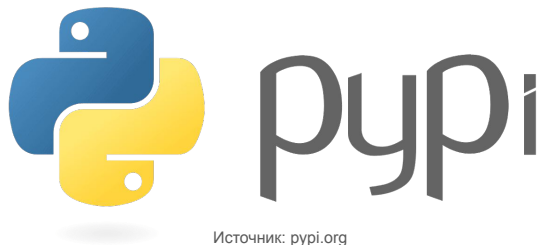


Источник: pikabu.ru

Datapipe

Что такое Datapipe?

Это собственная разработка компании "Эпоха Восемь".



Ссылка на PyPI:

<https://pypi.org/project/datapipe-core/>

Что такое Datarpipe?

Datarpipe - это Python библиотека для построения эффективных конвейеров обработки данных. Пользователю предоставляются возможности для описания каталога данных и трансформаций, взаимосвязи которых формируют вычислительный граф с метой для отслеживания изменений.



VS



VS



Особенности Datarpipe

Открытый исходный код - лицензия BSD 3-Clause (Copyright Epoch8) позволяет свободное распространение, использование, изучение и доработки.



Особенности Datarpipe

Отказоустойчивость - данные обрабатываются батчами, как партии в Airflow Assets, но при этом отслеживается статус актуальности каждого батча. При аварийном завершении актуальные, уже обработанные, батчи не будут обработаны повторно.



DATAPIPE

VS

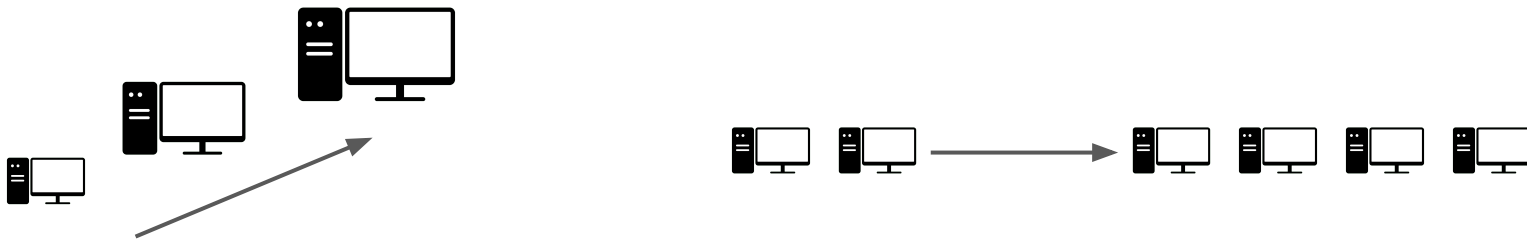


Apache
Airflow

Источник: airflow.apache.org

Особенности Datarpipe

Масштабируемость - возможность вертикального масштабирования мощностью VM или пода K8s и горизонтального масштабирования при помощи Ray Executor или нескольких инстансов приложения*.



* - здесь аккуратней с метой, пожалуйста.



Особенности Datarpipe

Гибкость деплоймента - запуск Python приложения при помощи CLI команд на on-prem VM, как Docker контейнер или как под в K8s идентичен и не требует доработок.



Особенности Datarpipe

Требования к разработчику - для построения пайплайнов достаточно экспертизы Python и библиотек для работы с данными (Pandas, Polars).



DATAPIPE

VS



Особенности Datarpipe

Расширяемость - на основе базовых примитивов **TableStore**, **PipelineStep**, **Executor** пользователю доступно написание собственной логики описания и обработки данных и отслеживания изменений.

TableStore



PipelineStep



Executor



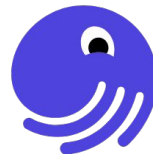
Особенности Datarpipe

Отслеживание изменений - подобно Dagster, критерий актуальности данных рассчитывается при помощи хэшей, но с возможностью построение более сложных и гибких графов обработки.



DATAPIPE

VS



dagster

Источник: dagster.io

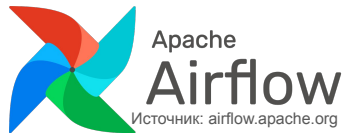
Особенности Datarpipe

Резюмируя, Datarpipe - гибкий Python фреймворк для процессинга данных. Проще, чем Hadoop + PySpark. Мощнее, чем Airflow Assets и Dagster.

Просто



Умеренно



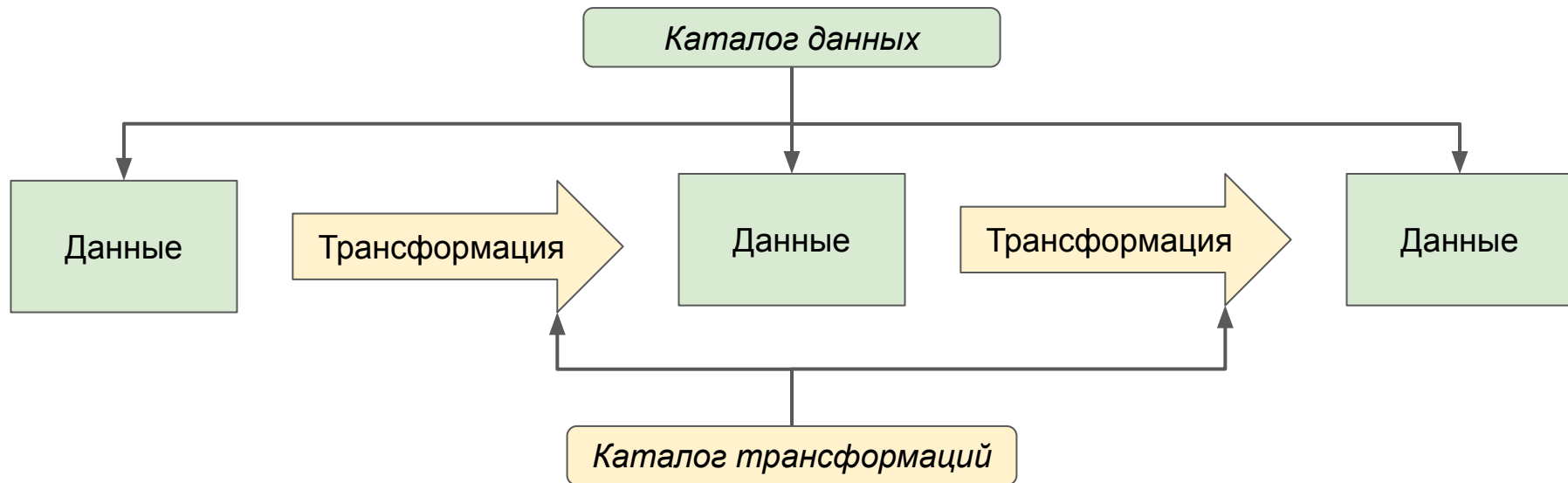
Сложно



Вычислительный граф

Описание вычислительного графа

Datapipe оперирует двумя основными сущностями для построения и запуска графа вычислений - это *каталог данных*, и *каталог трансформаций*, которые определяет пользователь.



Пример вычислительного графа

Пример *каталога данных*:

в коде представлен как “catalog”

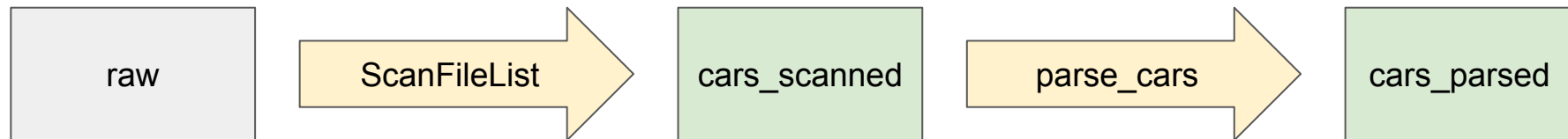


Пример *каталога трансформаций*:

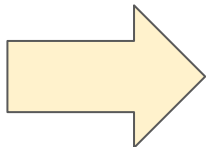
в коде представлен как “pipeline”



Пример вычислительного графа



- некие файлы на входе в систему, “сырые” данные.



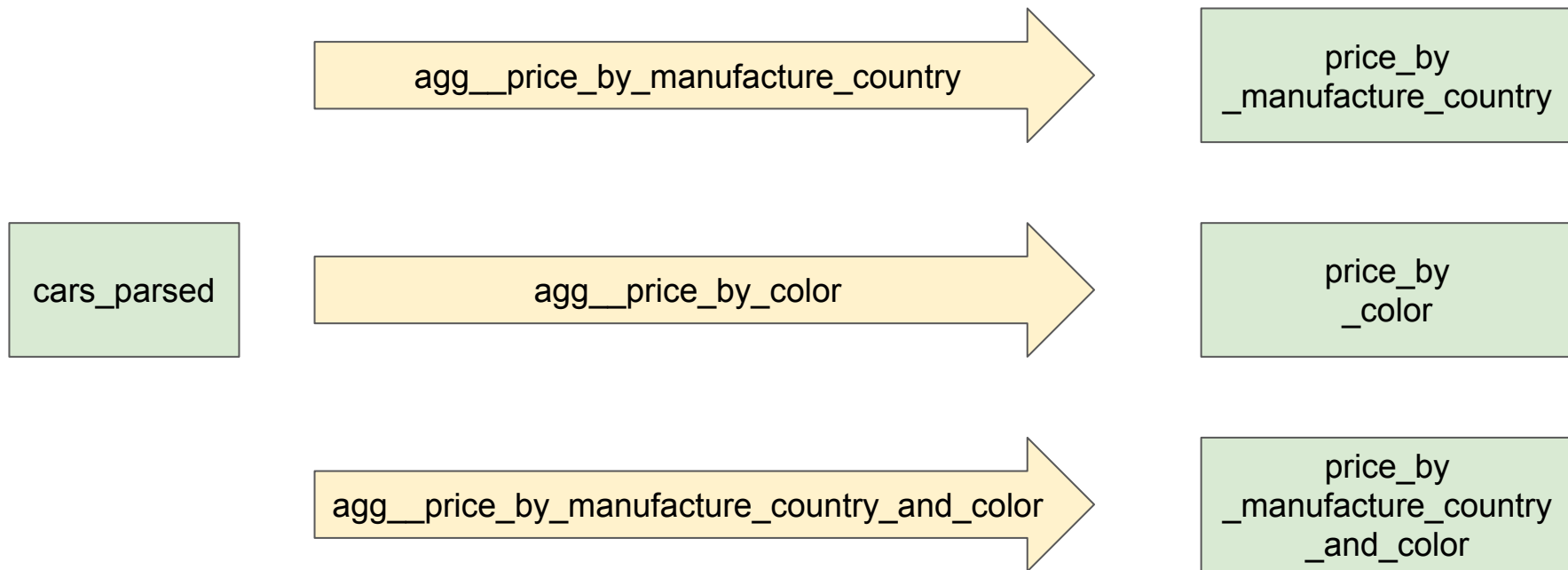
- сканирование файлов и разбор их содержимого.



- промежуточные данные, полученные в результате сканирования и разбора файлов.

Пример вычислительного графа

SUM(price) GROUP BY **manufacture_country** / **color** / **manufacture_country**, **color**



Описание данных

Описание данных

В примере было использовано хранилище “*TableStoreJsonLine*”, которое принимает аргументы “*filename*” для сохранения и “*primary_schema*” для описания данных:

```
45     "cars_parsed": Table(  
46         store=TableStoreJsonLine(  
47             filename=FILEPATH__PROCESSED__CARS.format(data="cars_parsed"),  
48             primary_schema=[  
49                 Column("file_name", String, primary_key=True),  
50                 Column("model_id", Integer, primary_key=True),  
51                 Column("manufacture_country", String, primary_key=True),  
52                 Column("color", String, primary_key=True),  
53             ],  
54         ),  
55     ),
```



Описание данных

Возможно использование других хранилищ:

- TableStoreExcel
- TableStoreDB
- BytesFile & PILFile
- PandasParquetFile
- RedisStore
- QdrantStore
- ElasticStore
- MilvusStore
- Neo4JStore



Источник: logos-world.net



redis

Источник: redis.io



Parquet

Источник: parquet.apache.org



SQLAlchemy

Источник: habr.com



pillow

Источник: pillow.readthedocs.io



elastic

Источник: elastic.co



Qdrant

Источник: qdrant.tech

Описание данных

Резюмируя, хранилища позволяют разработчику абстрагироваться от взаимодействий со средой хранения и сконцентрироваться на содержательной части процессинга данных.



Источник: logos-world.net



Parquet

Источник: parquet.apache.org



Источник: redis.io

Замена хранилищ бесшовна и не требует доработок*

* - ну почти не требует

Описание данных

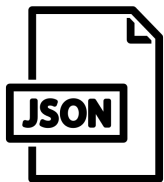
Пример меты таблицы из каталога данных:

file_name VARCHAR NOT NULL PRIMARY KEY	hash INTEGER	create_ts FLOAT	update_ts FLOAT	process_ts FLOAT	delete_ts FLOAT
record_1	1761214248	1787669420.8	1787669436.3	1787669522.2	NULL
record_2	2129005775	1787669420.8	1787669436.3	1787669436.2	NULL
record_3	1053874086	1787669420.8	1787669436.3	1787669522.2	NULL
record_4	-794066035	1787669420.8	1787669436.3	1787669436.2	NULL

Описание данных

Уникальные комбинации первичных ключей однозначно определяют неделимые единицы данных - батчи данных.

Гранулярностью батчей управляет разработчик.



Источник: parquet.apache.org

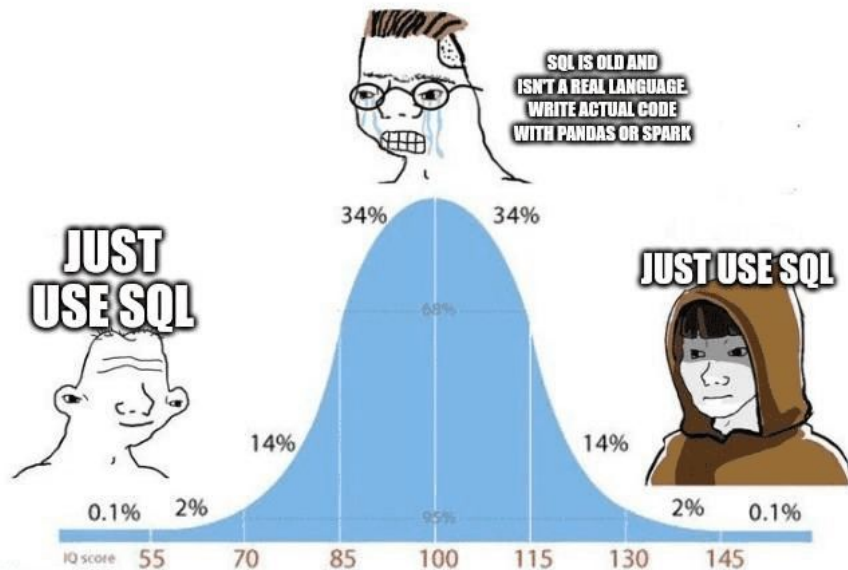


Источник: reddit.com

Описание данных

В настоящий момент мета живёт в реляционных БД, но вы всегда можете реализовать свою собственную мету, используя примитив MetaPlane.

MetaPlane



Описание данных

Каталог данных обладает следующими свойствами:

- описывает способ хранения данных и реализует методы взаимодействия с ними;
- при помощи первичных ключей определяет атомарные записи в данных - батчи данных;
- хранит хэш и метки времени создания, обновления, удаления записей для отслеживания изменений в хранилище меты - Postgres, SQLite или ClickHouse*.

* - в разработке.



Трансформация данных

Трансформация данных

Каталог трансформаций представляет из себя список шагов трансформации.

В примере используется “BatchTransform” - трансформация, при которой сопоставляются входные и выходные батчи по первичным ключам.

```
129 BatchTransform(  
130     agg__price_by_color,  
131     inputs=["cars_parsed"],  
132     outputs=["price_by_color"],  
133     kwargs={},  
134     chunk_size=1,  
135     executor_config=ExecutorConfig(parallelism=1),  
136     transform_keys=[  
137         "color",  
138     ],  
139     labels=[  
140         ("entity", "cars"),  
141         ("layer", "agg"),  
142         ("environment", "prod"),  
143     ],  
144 ),
```



Трансформация данных

Шаг трансформации “BatchTransform” реализован на основе примитива “PipelineStep” и является основным линейным шагом для построения вычислительного графа.

BatchTransform



PipelineStep



Трансформация данных

Аргументы шага трансформации “BatchTransform” следующие:

- функция трансформации “func” - содержательная часть операций над данными;
- “inputs” и “outputs” - наборы входных и выходных данных, описанные в каталоге данных;
- “transform_keys” - набор ключей трансформации, при помощи которых сопоставляются входные и выходные данные.

```
129 BatchTransform(  
130     agg_price_by_color,  
131     inputs=["cars_parsed"],  
132     outputs=["price_by_color"],  
133     kwargs={},  
134     chunk_size=1,  
135     executor_config=ExecutorConfig(parallelism=1),  
136     transform_keys=[  
137         "color",  
138     ],  
139     labels=[  
140         ("entity", "cars"),  
141         ("layer", "agg"),  
142         ("environment", "prod"),  
143     ],  
144 ),
```

Трансформация данных

Аргументы шага трансформации “BatchTransform” следующие:

- “kwargs” - опциональные дополнительные аргументы для функции трансформации;
- “chunk_size” - количество батчей данных в одной итерации запуска;
- “executor_config” - настройки экзекутора;
- “labels” - метки для разграничения запуска отдельных шагов вычислительного графа.

```
129 BatchTransform(  
130     agg_price_by_color,  
131     inputs=["cars_parsed"],  
132     outputs=["price_by_color"],  
133     kwargs={},  
134     chunk_size=1,  
135     executor_config=ExecutorConfig(parallelism=1),  
136     transform_keys=[  
137         "color",  
138     ],  
139     labels=[  
140         ("entity", "cars"),  
141         ("layer", "agg"),  
142         ("environment", "prod"),  
143     ],  
144 ),
```

Трансформация данных

Функция трансформации принимает Pandas DataFrame и возвращает его же после обработки.

Необходимые операции внутри функции определяет разработчик.

```
62 def agg_price_by_color(  
63     df_input_cars_parsed: pd.DataFrame,  
64 ) -> pd.DataFrame:  
65     df_output_price_by_color = df_input_cars_parsed[["color", "price"]]  
66     df_output_price_by_color = (  
67         df_output_price_by_color  
68         .groupby("color", as_index=False)  
69         .sum()  
70     )  
71  
72     return df_output_price_by_color  
73
```



Трансформация данных

Пример данных на входе в функцию трансформации.

Необходимые к обработке данные определяются и передаются автоматически.



Pandas DataFrame:

file_name	model_id	manufacture_country	color	price	year	new
record_2	1	Germany	Blue	20000	2026	True
record_3	3	Germany	Blue	25000	2026	True
record_5	5	Japan	Blue	15000	2025	True

Трансформация данных

```
20 def parse_cars(df_input_cars_scanned: pd.DataFrame) -> pd.DataFrame:
21     list_output_dfs = []
22
23     for _, row in df_input_cars_scanned.iterrows():
24         with open(row["filepath"], "r") as file:
25             json_record = json.load(file)
26
27             df_record_parsed = pd.DataFrame(
28                 {
29                     "file_name": row["file_name"],
30                     "model_id": json_record["model_id"],
31                     "manufacture_country": json_record["manufacture_country"],
32                     "color": json_record["color"],
33                     "price": json_record["price"],
34                     "year": json_record["year"],
35                     "new": json_record["new"],
36                 },
37                 index=[0]
38             )
39
40             list_output_dfs.append(df_record_parsed)
41
42     df_output_cars_parsed = pd.concat(list_output_dfs)
43
44     df_output_cars_parsed["new"] = df_output_cars_parsed["new"].astype(bool)
45
46     return df_output_cars_parsed
47
```

Либо вы можете управлять чтением данных самостоятельно.

Так появляются новые хранилища.



Трансформация данных

Пример данных на входе в функцию трансформации, где чтением управляет разработчик.

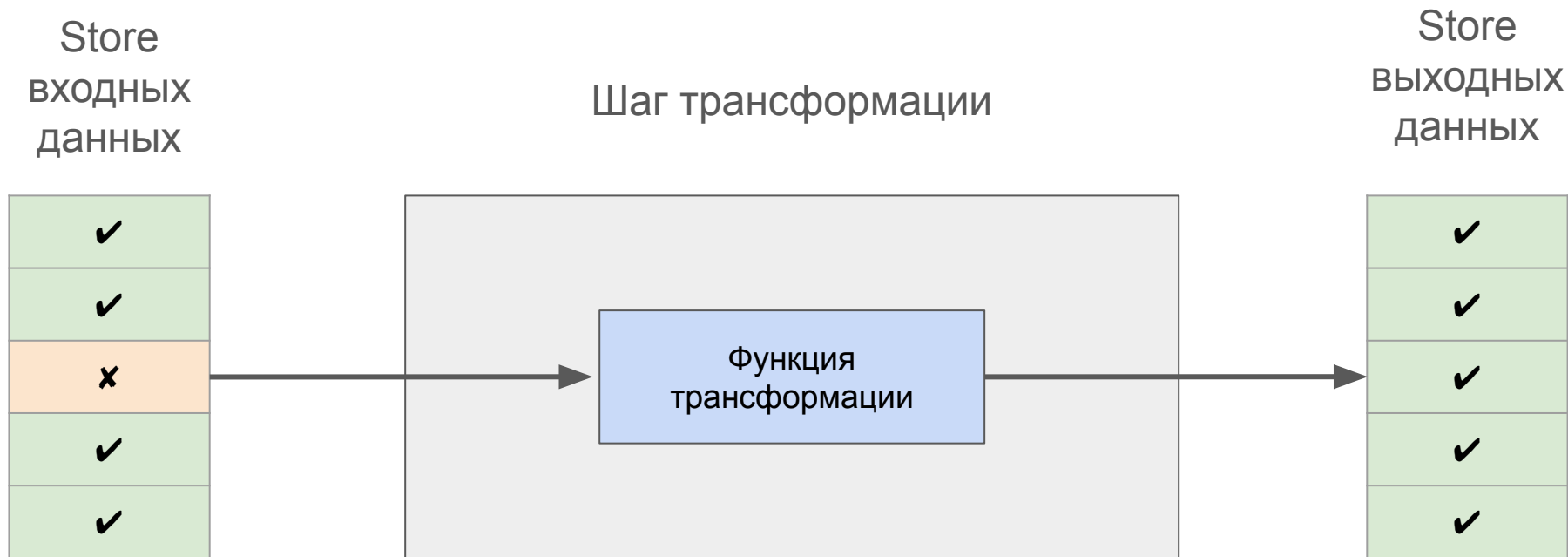


Pandas DataFrame:

file_name	filepath	size_bytes
record_1	examples/catalogs/data/raw/cars/record_1.json	136
record_2	examples/catalogs/data/raw/cars/record_2.json	136
record_3	examples/catalogs/data/raw/cars/record_3.json	136
record_4	examples/catalogs/data/raw/cars/record_4.json	134
record_5	examples/catalogs/data/raw/cars/record_5.json	134

Трансформация данных

Схема взаимодействия данных и трансформаций.



Трансформация данных

Схема взаимодействия с S3 и parquet файлами.



Трансформация данных

Пример меты таблицы из каталога трансформаций:

file_name VARCHAR NOT NULL PRIMARY KEY	process_ts FLOAT	is_success BOOLEAN	priority INTEGER	error VARCHAR
record_1	1787669464.1	True	0	NULL
record_2	1787669464.1	True	0	NULL
record_3	1787669464.2	False	0	“OOM killed”
record_4	1787669464.2	True	0	NULL

Трансформация данных

Каталог трансформаций обладает следующими свойствами:

- использует входные и выходные данные для построения графа вычислений;
- при помощи *ключей трансформации* определяет взаимодействие входных и выходных данных;
- хранит время и статус последнего запуска для отслеживания изменений в хранилище меты - Postgres, SQLite или ClickHouse*.

* - в разработке.



Отслеживание изменений

Отслеживание изменений

У меты данных и у меты трансформаций есть время последней обработки батча - поле *“process_ts”*. Однако обработка батча не означает его изменение. При обработке данных происходит вычисление хэша *“hash”*. Если существующий и новый хэши совпали, то в данных не было обнаружено изменений. В противном случае батч считается обновлённым и для него актуализируется время фактического обновления *“update_ts”*.



Отслеживание изменений

Время фактического обновления батча “*update_ts*” позволяет определить актуальность всех последующих данных и трансформаций: если трансформация последний раз отработала раньше обновления батча, то она устарела и должна быть запущена.

Входные данные

file_name	update_ts
record_1	10
record_2	10

Мета трансформации

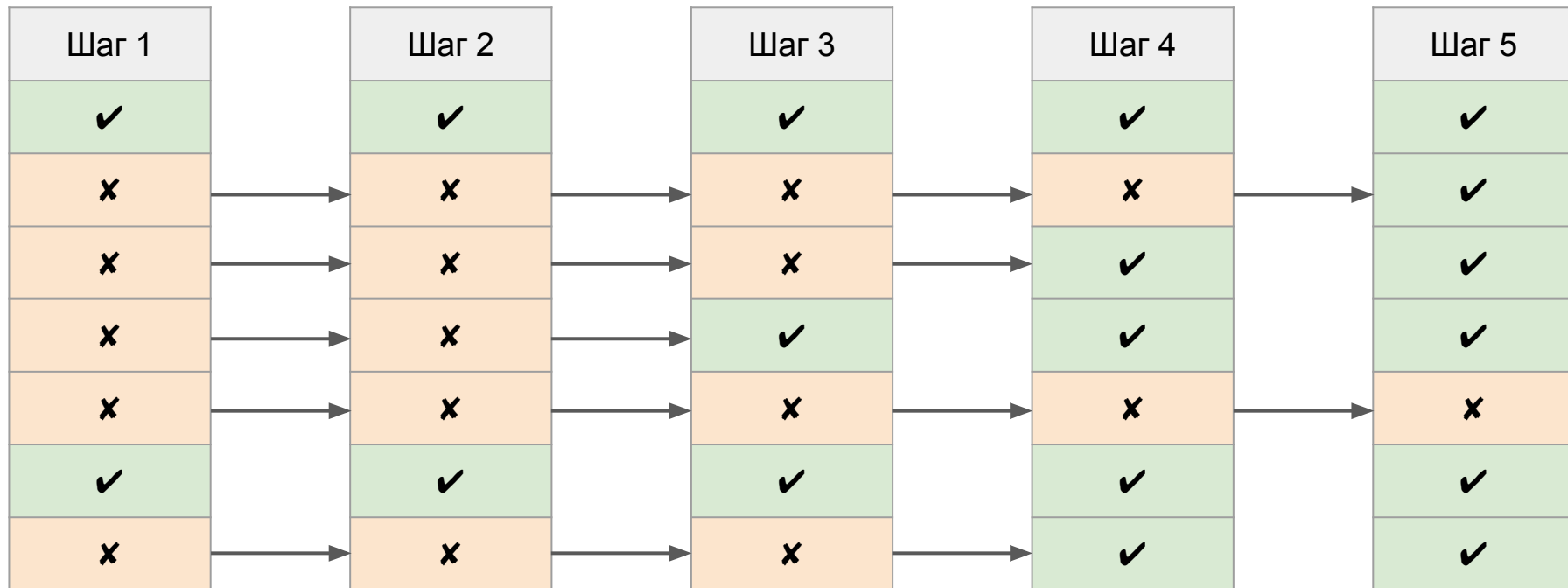
file_name	process_ts
record_1	12
record_2	6

Выходные данные

file_name	update_ts
record_1	12
record_2	6

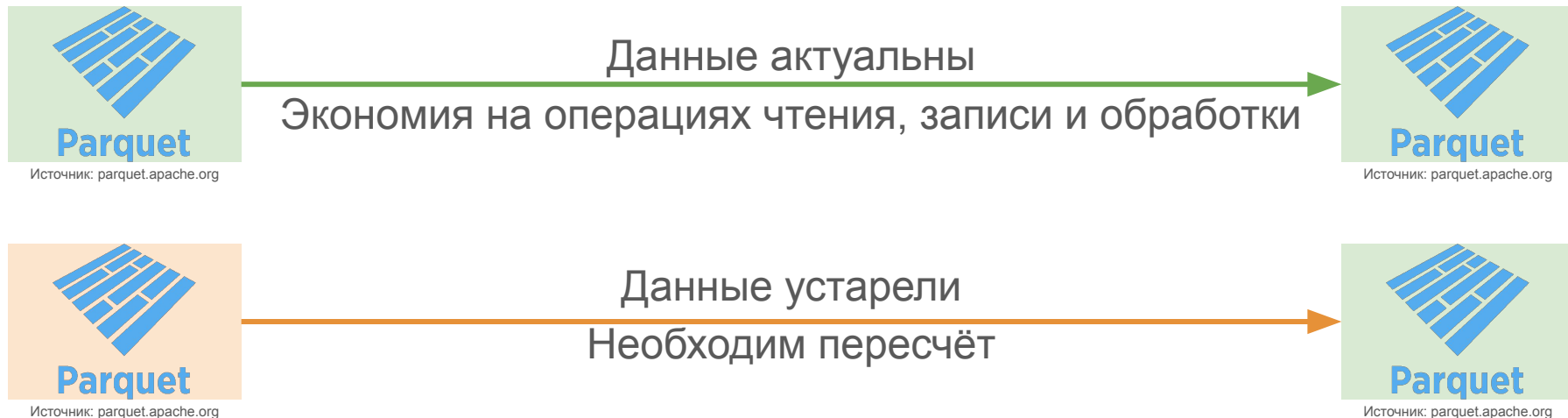
Отслеживание изменений

Мета позволяет точно досчитывать только изменившиеся батчи.



Отслеживание изменений

Возможность грануляции батчей с отслеживанием изменений позволила реализовать эффективный и быстрый процессинг данных на S3 за счёт минимизации количества избыточных операций.



Ключи и индекс трансформации

Ключи и индекс трансформации

Набор ключей трансформации определяет взаимодействие входных и выходных батчей данных. Какой поднабор из набора первичных ключей будет использован.

Входные данные
“cars_scanned”

file_name
record_1
record_2
record_3
record_4

Трансформация по “file_name”

Выходные данные
“cars_parsed”

file_name	model_id
record_1	1
record_2	1
record_3	3
record_4	4

Ключи и индекс трансформации

В обработанных данных могут появляться новые ключи и записи.



Входные данные
“cars_parsed”

file_name
record_1
record_2
record_3
record_4

Трансформация по “file_name”

Выходные данные
“cars_docs”

file_name	doc_name
record_1	passport
record_1	license
record_2	passport
record_2	license

Ключи и индекс трансформации

Возможно сокращение набора первичных ключей.
Например, как при агрегации “GROUP BY”.



Входные данные
“cars_parsed”

Трансформация по “color”

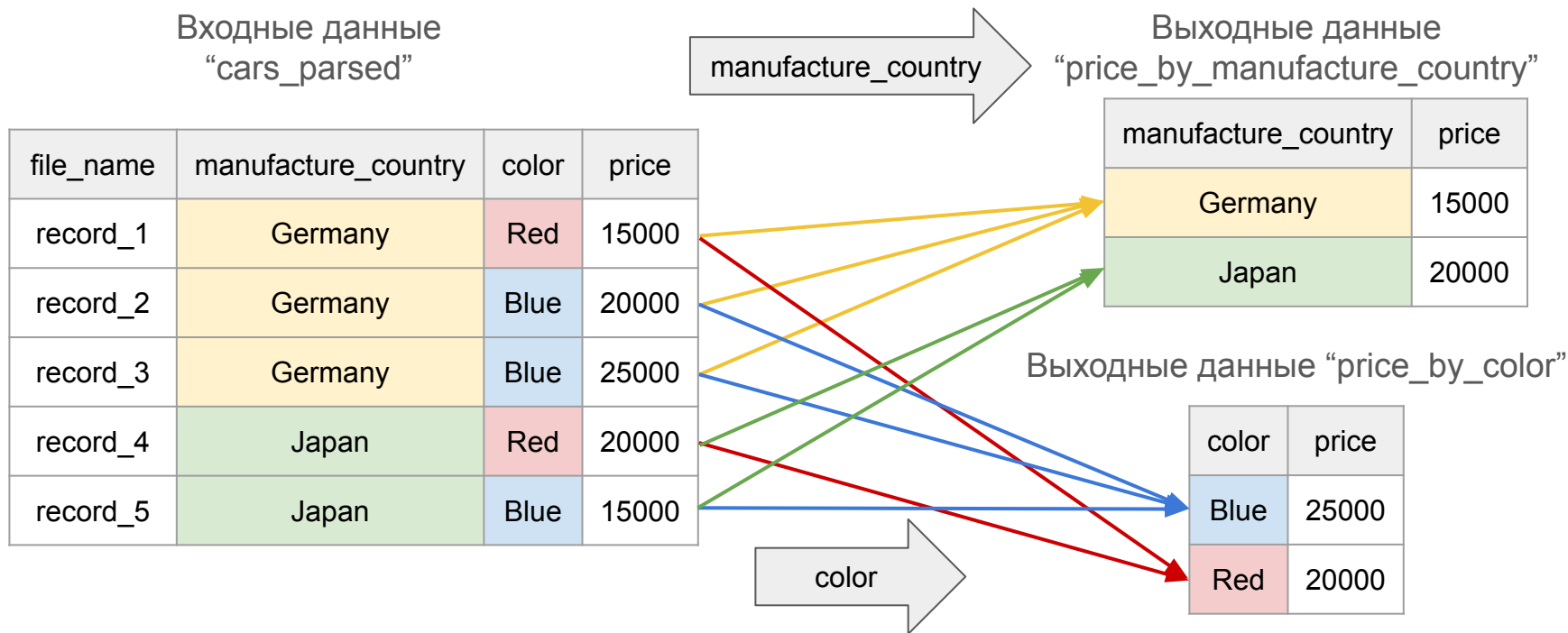
Выходные данные
“price_by_color”

file_name	color	price
record_1	Red	15000
record_4	Red	20000
record_2	Blue	20000
record_3	Blue	25000

color	price
Red	35000
Blue	45000

Ключи и индекс трансформации

На основе одних данных могут быть посчитаны несколько выходных данных.



Ключи и индекс трансформации

Также на вход могут быть поданы несколько данных.



Входные данные
“cars_parsed” и “countries_parsed”

file_name	manufacture_country	price
record_1	Germany	15000
record_4	Japan	20000
record_5	Japan	15000

file_name	manufacture_country	vat
record_1	Germany	0.22
record_2	Japan	0.18

Трансформация по
“manufacture_country”

Выходные данные
“cars_price_with_vat”

file_name	manufacture_country	price	price_with_vat
record_1	Germany	15000	18300
record_4	Japan	20000	23600
record_5	Japan	15000	17700

Ключи и индекс трансформации

ChainTransform - цепочка трансформаций.

Входные данные

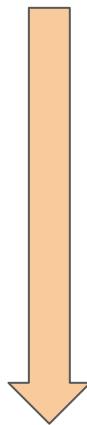
date	orders
2026-09-20	0
2026-09-21	0 → 1
2026-09-22	0
2026-09-23	1
2026-09-24	1

До

date	sum of orders
2026-09-20	0
2026-09-21	0
2026-09-22	0
2026-09-23	1
2026-09-24	2

После

sum of orders
0
1
1
2
3



Ключи и индекс трансформации

ChainTransform - позволяет описать упорядоченную последовательность зависимостей, когда батч данных зависит не только от входных данных, но и от предыдущего состояния согласно критерию ранжирования.

Аналог оконных функций из SQL.

Разработчику доступна настройка критерия ранжирования, окна функции и размер чанка.

До

sum of orders
0
0 → 1
0
1
2

После

sum of orders
0
1
1
2
3

Ключи и индекс трансформации

Индекс трансформации - список батчей для пересчёта:

1. Сопоставление входных и выходных батчей по набору первичных ключей согласно ключам трансформации.
2. Определение актуальности выходных батчей.
3. Устаревшие батчи попадают в индекс трансформации, согласно которому на вход в функцию трансформации передаются соответствующие данные.



Функция трансформации знает, какие данные она обрабатывает!

Ключи и индекс трансформации

Для *record_1* сравнение времени фактического обновления *update_ts* и времени последнего запуска трансформации *process_ts* не приведёт к попаданию в индекс трансформации, так как он был обработан после обновления.

Для *record_2* ситуация обратная - он попадёт в индекс трансформации.

Входные данные

file_name	update_ts
record_1	10
record_2	10

Мета трансформации

file_name	process_ts
record_1	12
record_2	6

Выходные данные

file_name	update_ts
record_1	12
record_2	6

Ключи и индекс трансформации

Индекс трансформации - список батчей,
который формируется автоматически на основе меты.

file_name
record_1
record_2
record_3
record_4

file_name
record_1
record_2
record_3
record_4

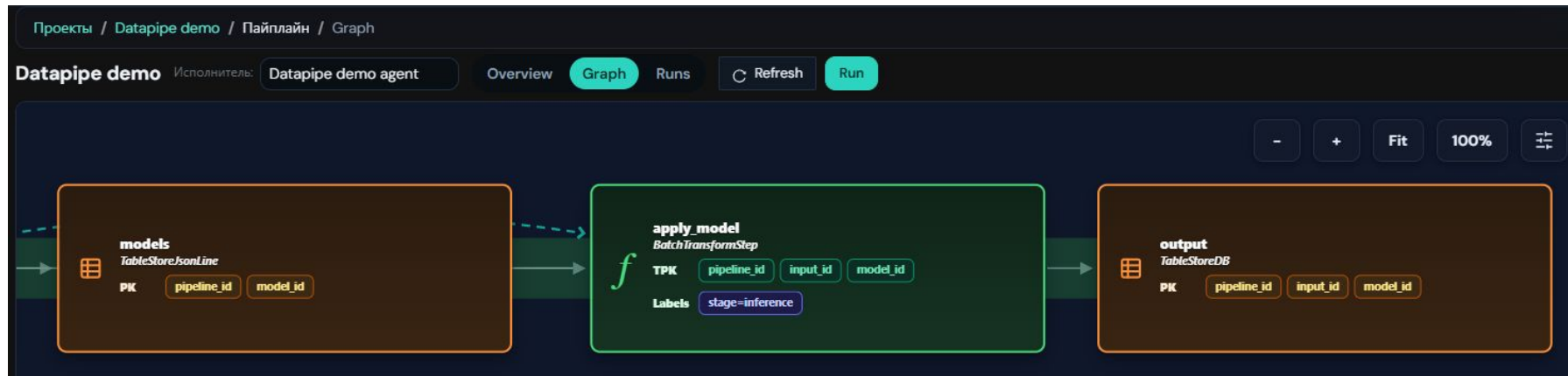
file_name
record_1
record_2
record_3
record_4

Разработчик может управлять размером батчей
для итерации и параллелизмом.

Дальнейшее развитие проекта

Дальнейшее развитие проекта

Пользовательский интерфейс - визуализация вычислительного графа с возможностью просмотра прогресса обработки данных.



Дальнейшее развитие проекта

Облачное решение - единый сервис с авторизацией, разграничением доступов, мониторингом, логированием и алёртингом запусков.

Проекты

sergeyz06@epoch8.co

+ Новый проект

Каталог проектов; конфиг в Git-репозитории каждого проекта.

ID	НАЗВАНИЕ	GIT HEAD	ОБНОВЛЁН	ДЕЙСТВИЕ
datapipe-demo	Datapipe demo	—	15 сентября 2026 г. 19:18	Открыть →
██████-label	██████ Scan	e1fb51b796a9	17 сентября 2026 г. 13:29	Открыть →

Дальнейшее развитие проекта

Мультиагентная система - возможность дистанционного наблюдения и управления несколькими инстансами приложения из единой системы.

Проекты / Агенты

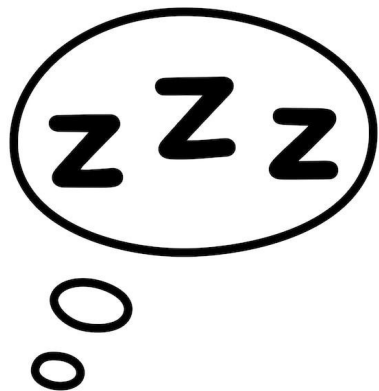
Агенты

ПРОЕКТ

Datapipe demo

ID	НАЗВАНИЕ	КОД	ДЕЙСТВИЕ
datapipe-demo	Datapipe demo agent	datapipe-demo	Открыть →

Благодарю за внимание!



Автор доклада

Захарченко Сергей - "Эпоха Восемь"

t.me/SergeyZ06

SergeyZ06@yandex.ru

epoch8.ru

www.linkedin.com/company/epoch8