Яндекс

SmartData 2024

Каждый байт на вес золота. Опыт построения DMP в рекламе Яндекса

Алексей Стыценко, руководитель группы разработки DMP Logos

Disclaimer

Содержание

Введение

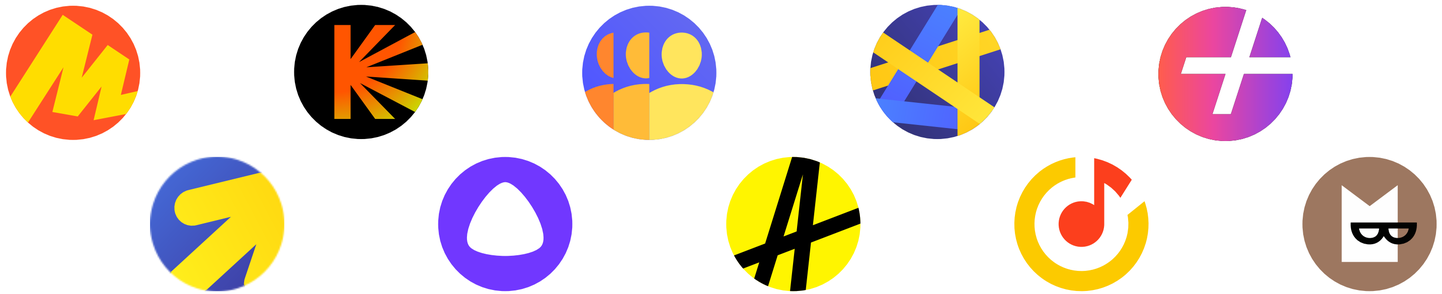
YTsaurus — основная система хранения и обработки больших данных Яндекса, теперь open source

YTsaurus	Аналоги
Cypress with static row- & column-oriented tables & ACID transactions	hdfs, s3 with Parquet, ORC, Avro
Sorted Dynamic Tables	HBase
Ordered Dynamic Tables	Kafka
YQL	Hive
Spark over YT	Spark
Clickhouse over YT	Impala, Presto, Trino, …
Fair Share & Preemption планировщик	YARN или даже Kubernetes

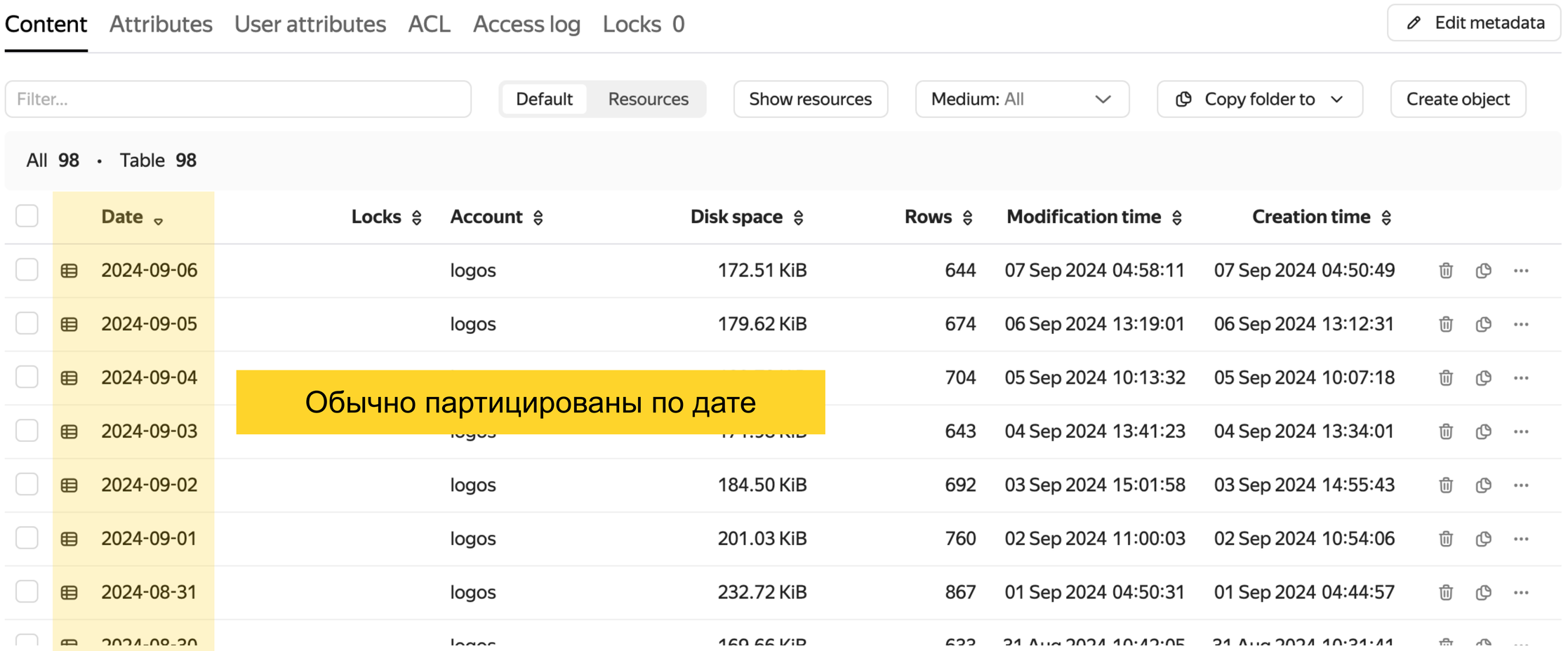
https://ytsaurus.tech https://smartdataconf.ru/archive/2023/talks/a9ac6358d6b6435ab0e0c13d64c01721/

DMP Logos

- Платформа управления данными на YTsaurus
- Была создана для надёжного построения датасетов для ML в рекламе
- Сегодня Logos используют DWH Рекламы, DWH Яндекс Маркета, Алиса, Кинопоиск, Яндекс Музыка, Яндекс Плюс и другие сервисы
- Более 10 000 регулярных процессов



Данные на YTsaurus



Данные на YTsaurus

Content | Attributes | User attributes | ACL | Access log | Locks 0 | Schema

Edit metadata

Настройки хранения

Metadata

Id	4010-5981c0-3fe0191-cb64ca5c
Owner	yql
Account	yql
Creation time	20 Oct 2016 22:54:41
Modification time	20 Oct 2016 22:55:36
Access time	15 Aug 2024 03:16:31

Rows	12
Uncompressed size	1.75 KiB
Compressed size	1.64 KiB
Primary medium	Default
Disk space	7.86 KiB
Default disk space	7.86 KiB
Data weight	–

Table type	static
Optimize for	Scan
Compression ratio	0.93267
Compression codec	lz4
Replication factor	3

Sorted true

« ‹ › » | # No offset | Columns 8/8 | YQL Kit | Jupyter | DataLens | Download | Upload

#	"name"	"last_visit_time"	"ip"	"age"	"last_time_on_site"	"user_agent"
0	"Alena"	1441890000u	"::ffff:193.34.173.188"	5u	22.5	"Mozilla/5.0"
1	"Anya"	1447027200u	"95.106.17.32"	15u	0.5	"Mozilla/5.0"

Хранятся в виде таблиц

Данные на YTsaurus

Content Attributes User attributes ACL Access log Locks 0 **Schema** Edit metadata

Schema mode strong
Strict true
Unique keys false

Help

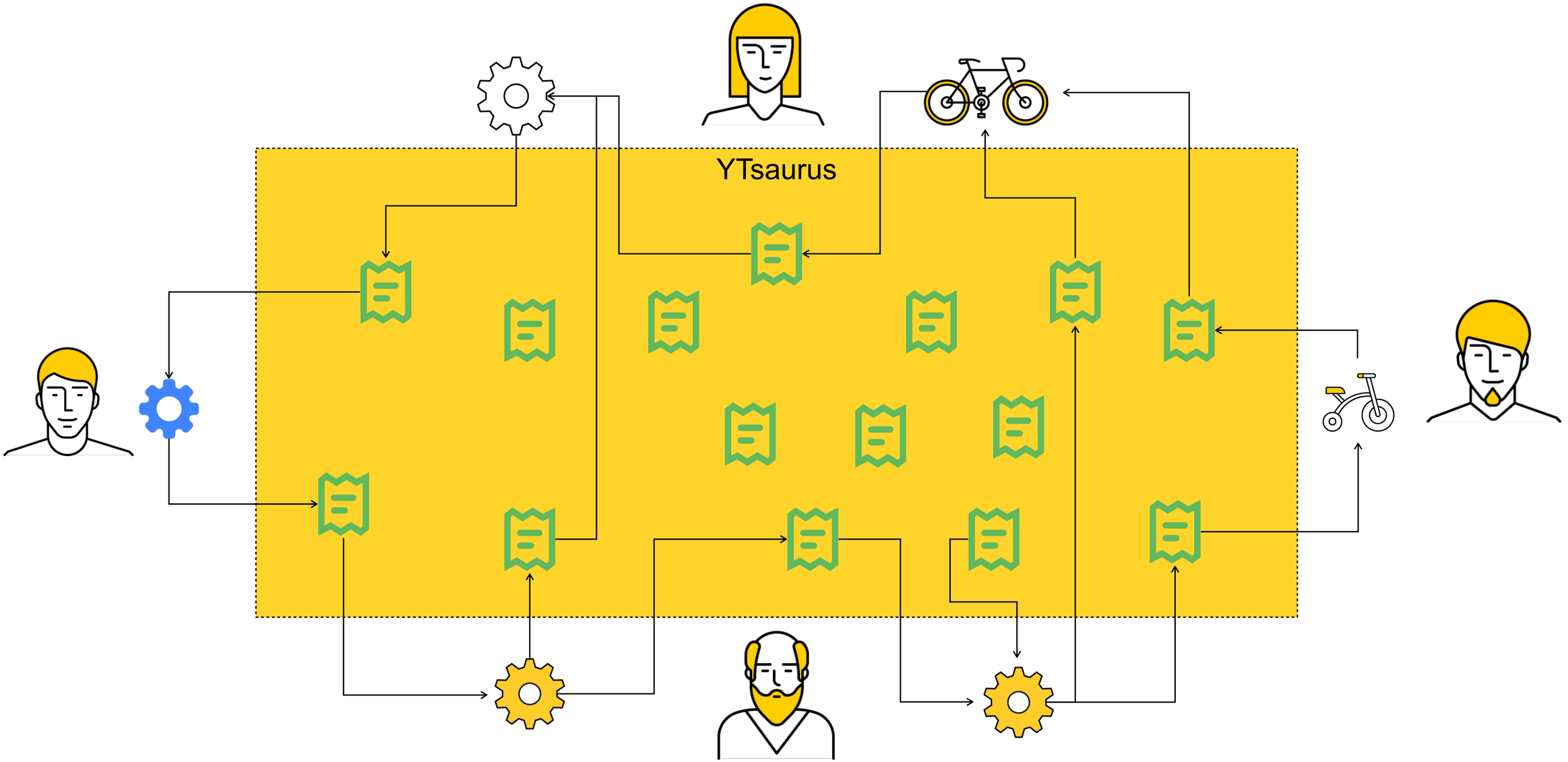
Схема контролируется и валидируется самим YTsaurus

Filter by name...

Интеграция с data catalog

#	Name	Type v3	Datacatalog title (exte...	Datacatalog descriptio...
0	cluster	Utf8	no description	Кластер.
1	compressed_compression_codec	Utf8	no description	compression_codec у ...
2	compressed_disk_space	Int64	no description	disk_space у пережат...
3	compressed_erasure_codec	Utf8	no description	erasure_codec у пере...
4	cpu_max	Double	no description	Максимальное число ...
5	cpu_sum	Double	no description	Суммарное число ср...

6 лет назад в рекламе…



Проблемы

- Сложно понять, где производятся и потребляются данные
- Нет тестов, пайплайн хрупкий
- Непонятно когда запускать процессы, cron+polling
- Большие трудозатраты на восстановление данных при инцидентах
- Данные хранятся дольше, чем нужны, либо вообще больше не нужны
- Часть полей из датасетов нигде не используется
- Нет документации полей таблиц

Инцидент из жизни

```
- IF(find_in_set('gurantee', L.Options), 1, 0)
+ IF(find_in_set('guarantee', L.Options), 1, 0)
```

- Инженер исправил опечатку в коде, что могло пойти не так?
- Из-за этой опечатки поле в датасете всегда было ноликом, а стало не ноликом
- Предсказания ML моделей в production разнесло
- Через несколько дней стала заметна потеря денег в рекламе

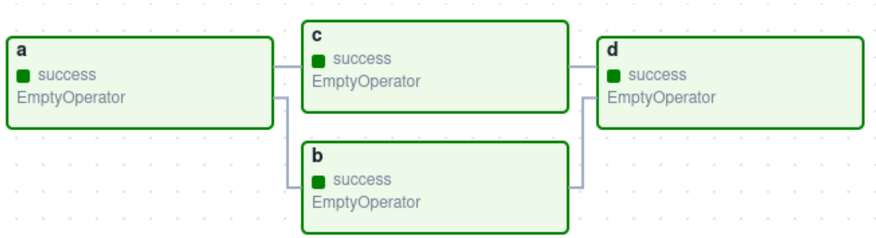
Что хотели получить?

- Собрать все данные и процессы в одном месте
- Понимать какие данные какими процессами потребляются и производятся
- ~~cron~~Запускать процессы при появлении необходимых данных
- Алерты и мониторинги на данные и пайплайн
- Тесты и релизный процесс
- Легко пересчитывать данные с учётом зависимостей
- Документацию данных

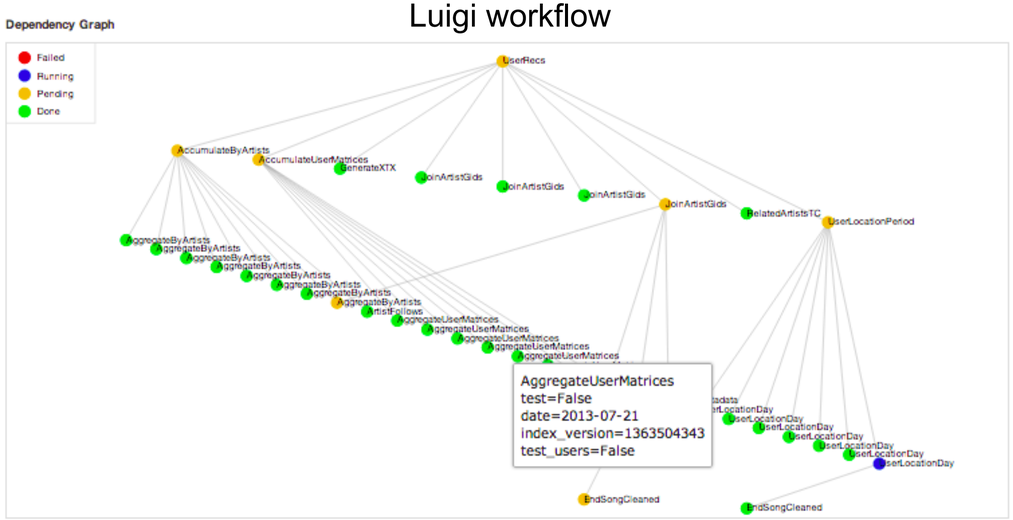
Фреймворк

Дата-пайплайн

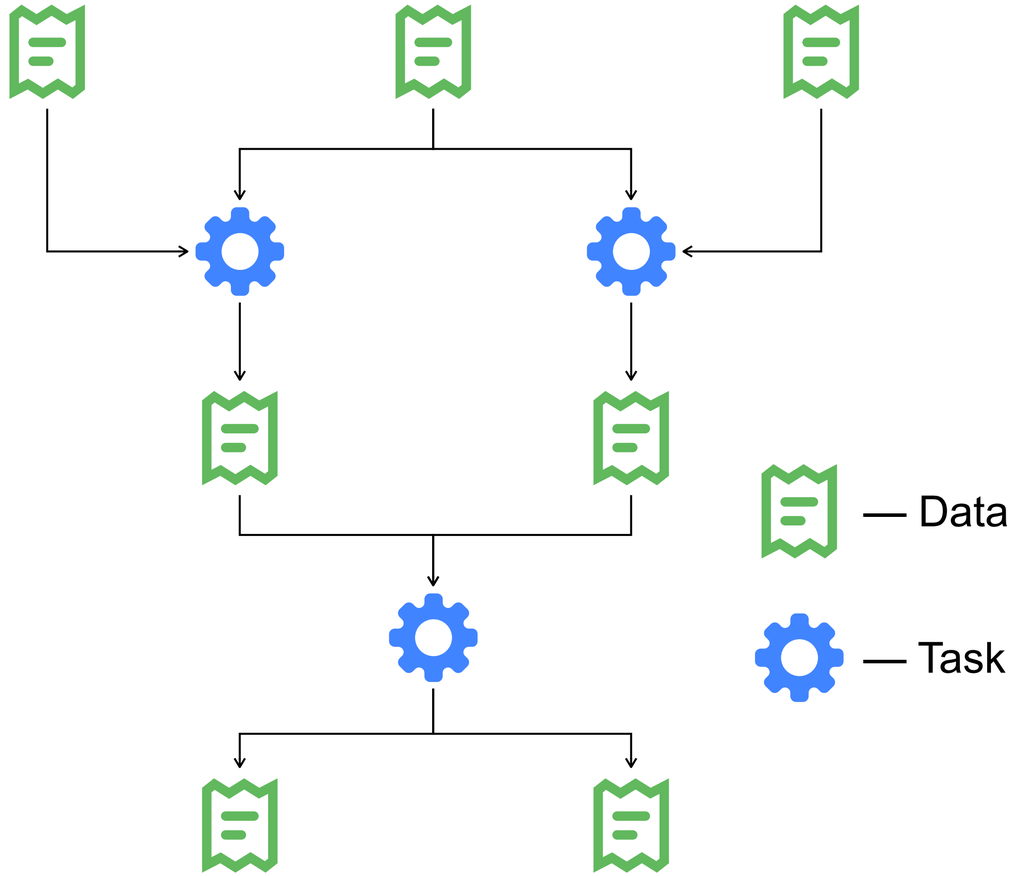
Airflow DAG



Luigi workflow



Logos graph



https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html#dags
https://luigi.readthedocs.io/en/stable/index.html#dependency-graph-example

Описание данных

```
ClickLog1D = lt.Log(
    location=lt.Daily(path="//logs/click-log/1d"),
    responsible="ads-dwh",
    life_time="365d",
    delay="1d30m",
)
```

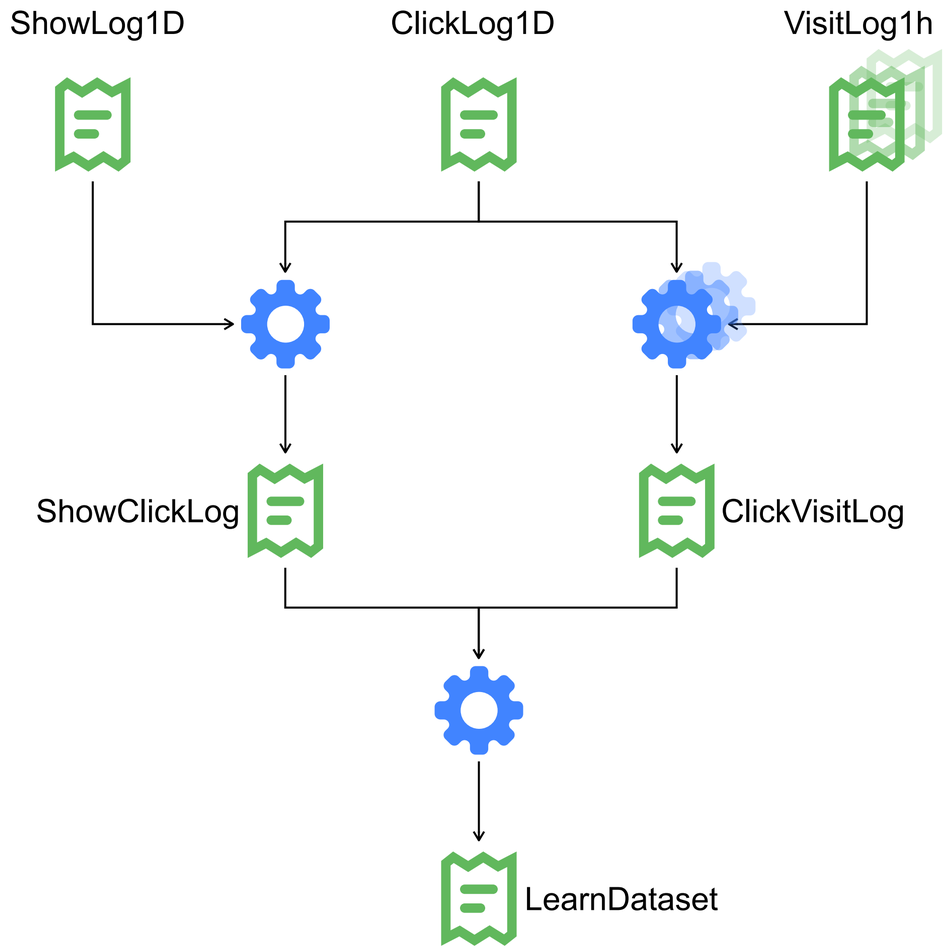
Описание процессов: зависимости

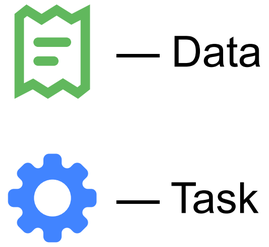
```
class MyNewTask(BaseLogTask):
    def dependencies(self):
        return Dependencies(
            inputs={
                "clicks": SingleTableDependency(logs.ClickLog1d),
                "visits": RangeOfTablesDependency(logs.VisitLog1h, tables_after=23)
            },
            outputs={
                "joined_clicks_visits": OutputDependency(logs.ClickVisitLog1d)
            },
        )

    def run(self, inputs, outputs, services, ctx):
        ...
```

Описание процессов: исполняемый код

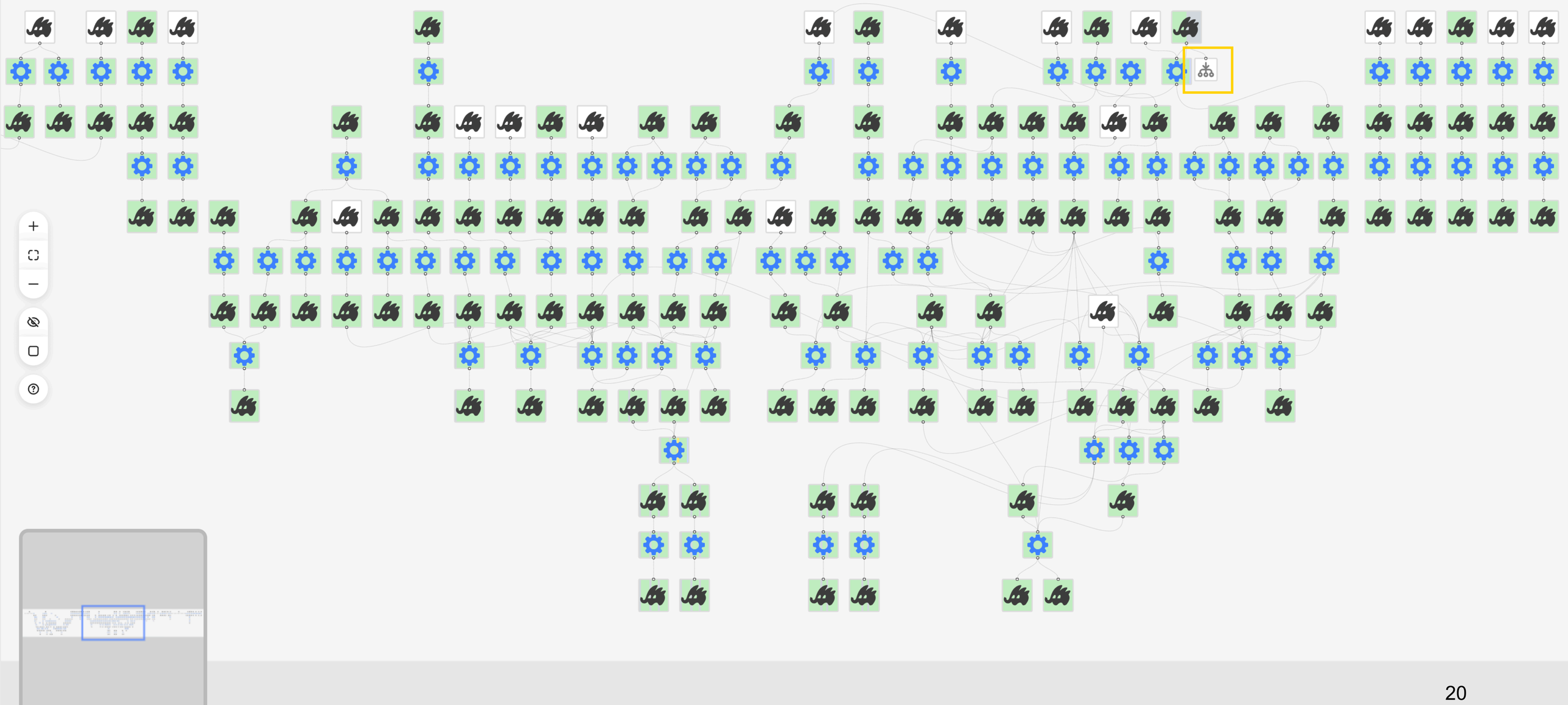
```
class MyNewTask(BaseLogTask):
    def run(self, inputs, outputs, services, ctx):
        yql_client = services.yql.get_client()
        query = """
            INSERT INTO `{result}` WITH TRUNCATE
            SELECT
                clicks.UserID, clicks.EventTime,
                visits.*,
                clicks.Cost * visits.ExchangeRate as CostUSD
            FROM `{clicks}` as clicks
            LEFT JOIN `{visits}` as visits
            USING (eventid, uniqid, showtime)
        """.format(clicks=inputs["clicks"][0].path),
                   visits=inputs["visits"][0].path,
                   result=outputs["joined_clicks_visits][0].path
        yql_client.execute(query)
```

Граф вычислений





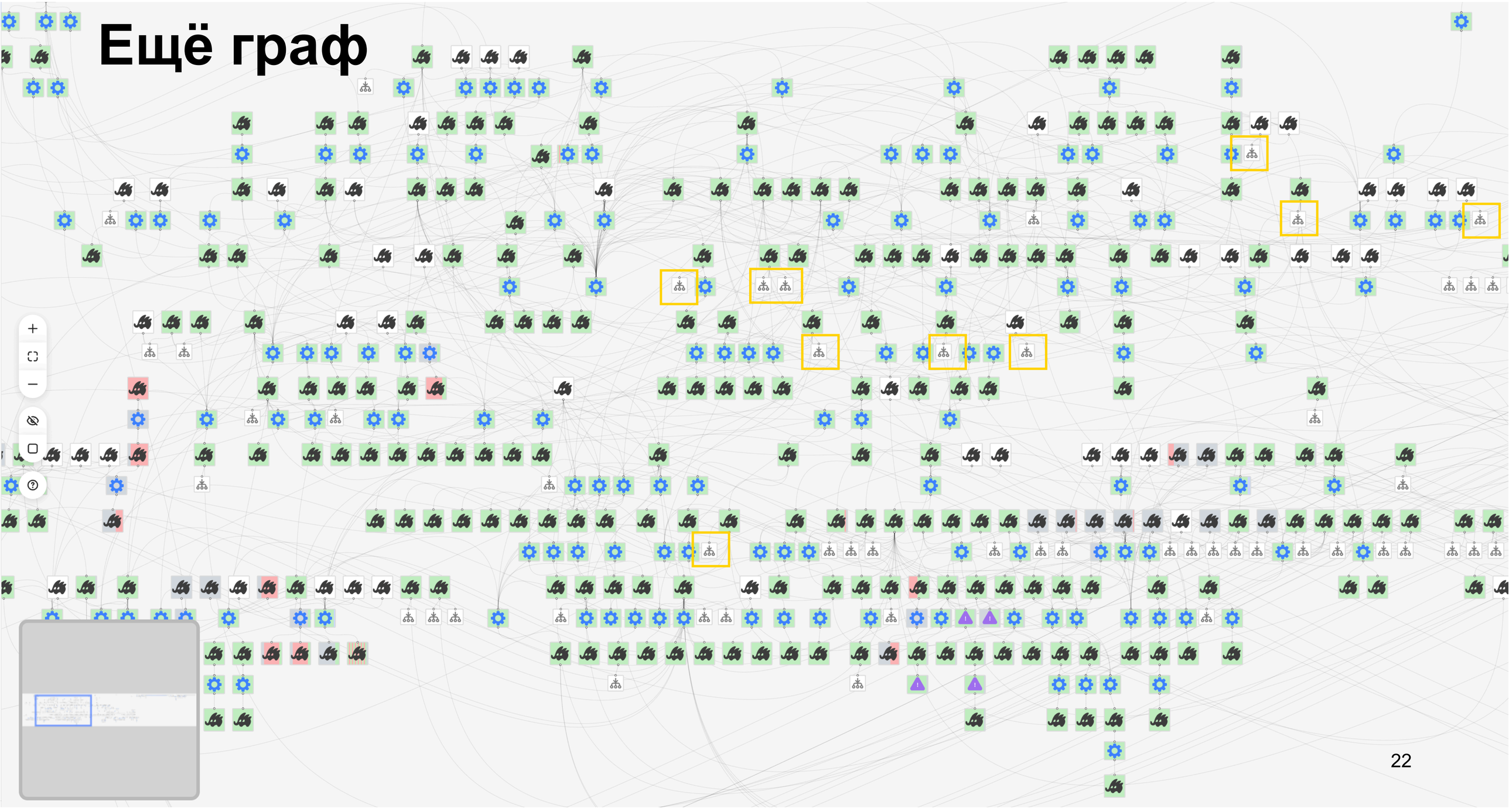
Граф вычислений



Метаграф

- И данные, и процессы — код в монорепозитории
- Графы — домены, зоны ответственности и единицы деплоя
- Данные могут переиспользоваться между графами
- Графы связываются между собой через данные
- Объединение графов образует метаграф

Ещё граф



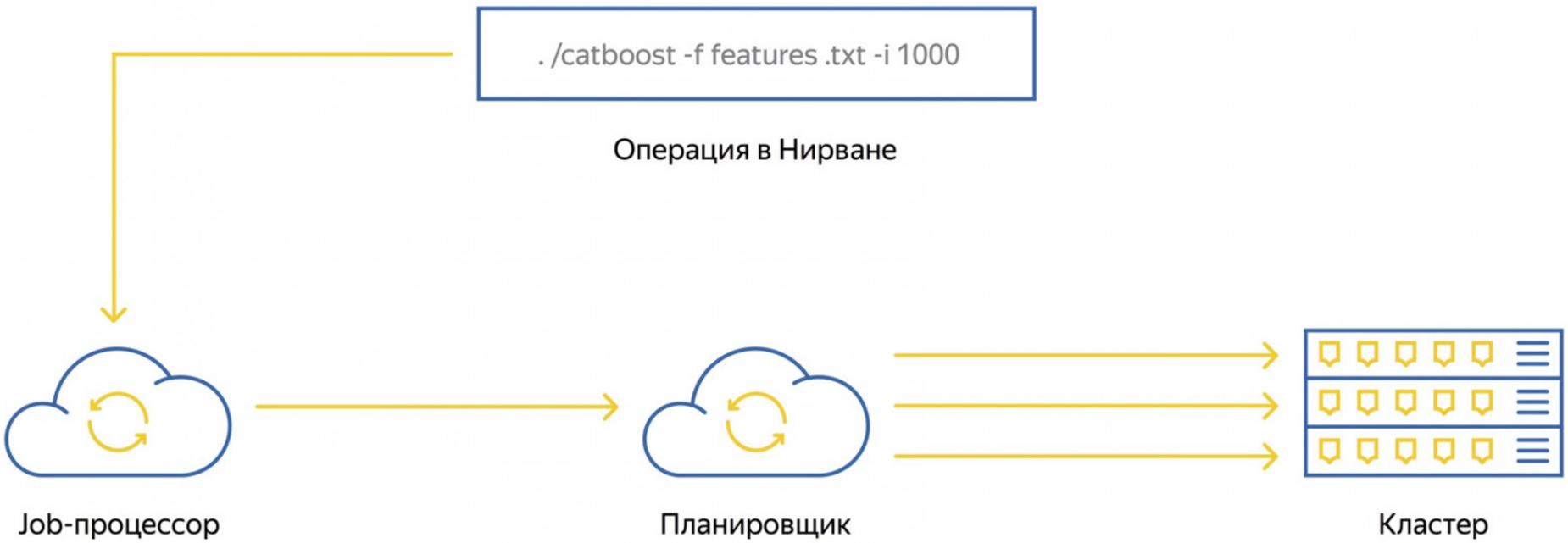
Что получилось

- Собрали все данные и процессы в одном месте
- Понимаем какие данные какими процессами потребляются и производятся
- Запускать процессы при появлении необходимых данных
- Алерты и мониторинги на данные и пайплайн
- Тесты и релизный процесс
- Легко пересчитывать данные в прошлое с учётом зависимостей
- Документацию данных

Рантайм

Внутренняя инфраструктура Яндекса: Nirvana

Облачная платформа для управления вычислительными процессами

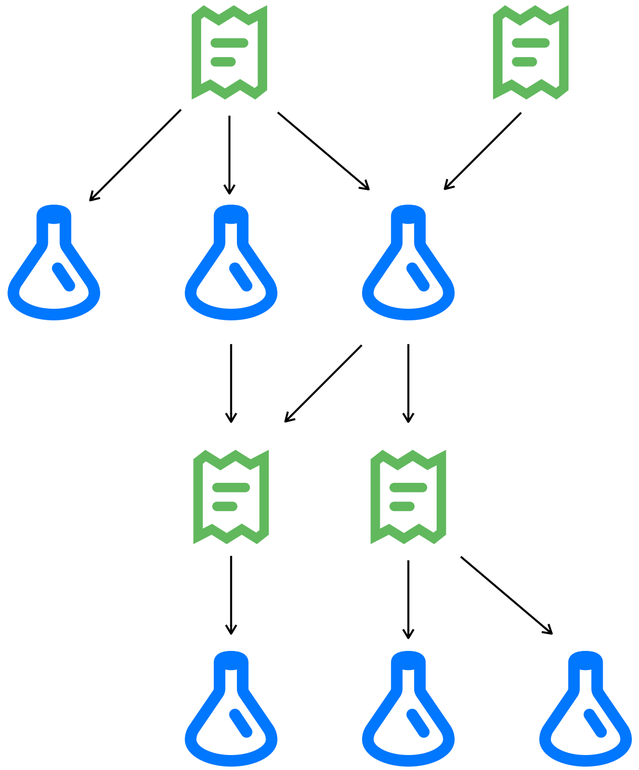


https://habr.com/ru/companies/yandex/articles/351016/

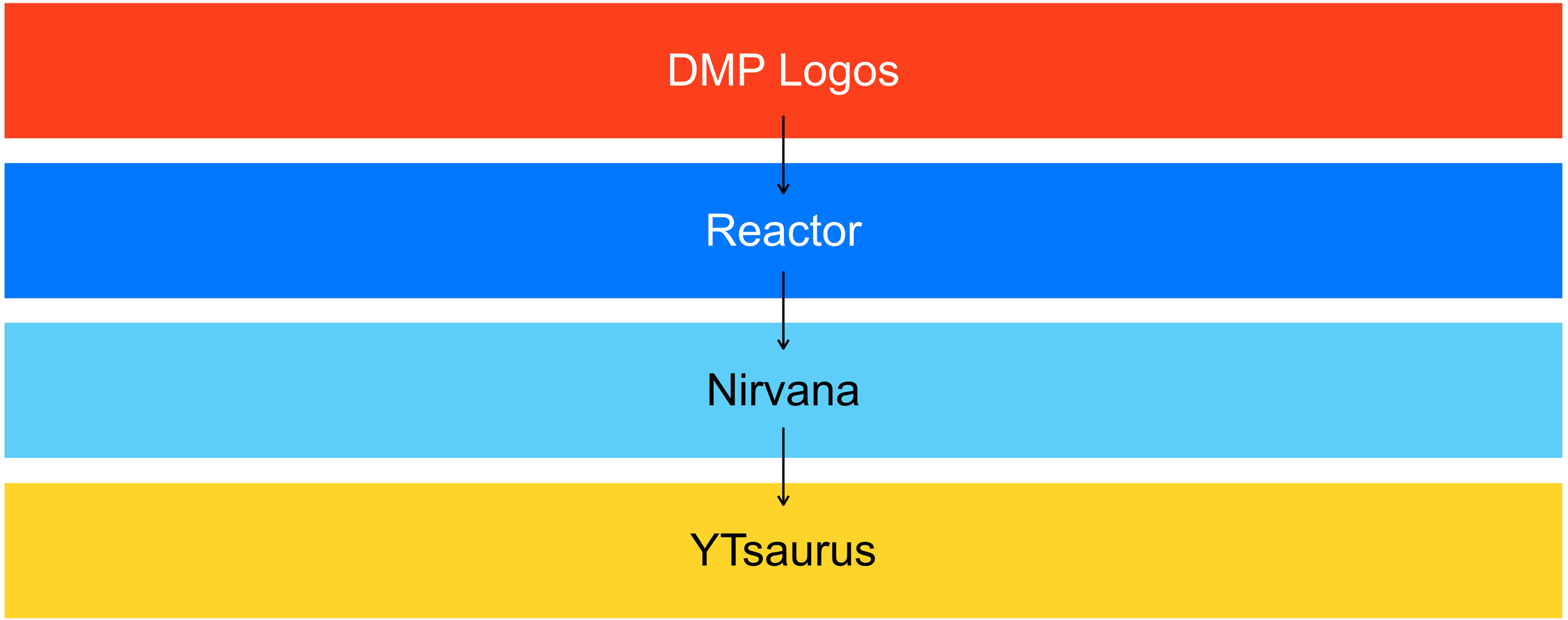
Внутренняя инфраструктура Яндекса: Reactor

Система автоматизированного событийного запуска процессов

- Позволяет публиковать и подписываться на события
- Умеет запускать операции Нирваны
- Гибкая конфигурация условий запуска, вплоть до произвольных starlark-скриптов



Внутренняя инфраструктура Яндекса




Так у меня рантайма-то
как такового и нет

Описание процессов: зависимости

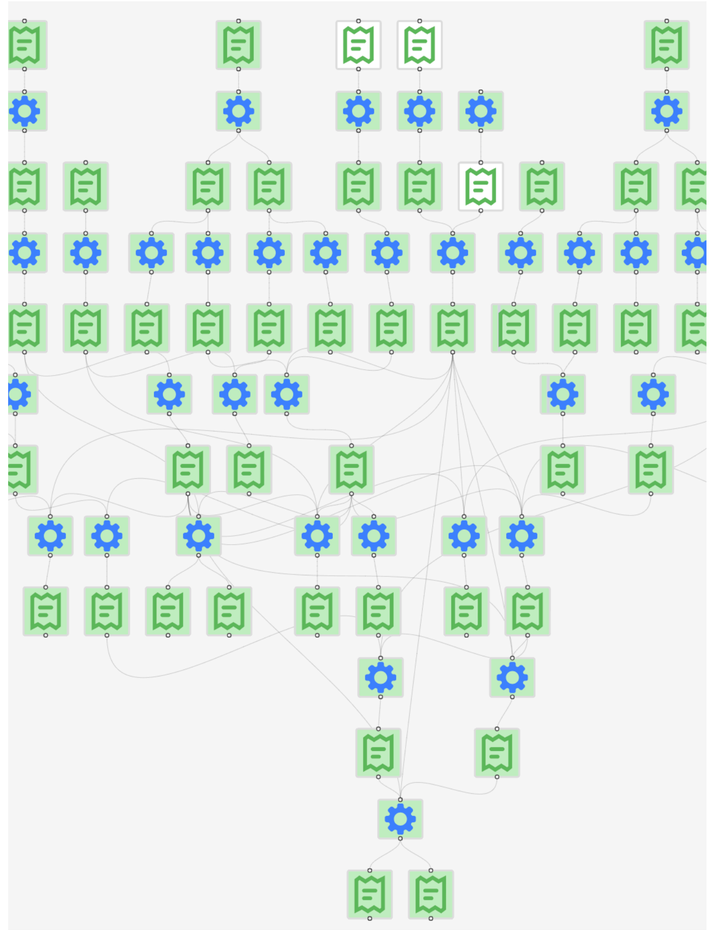
```
class MyNewTask(BaseLogTask):
    def dependencies(self):
        return Dependencies(
            inputs={
                "clicks": SingleTableDependency(logs.ClickLog1D),
                "visits": RangeOfTablesDependency(logs.VisitLog1h, tables_after=23)
            },
            outputs={
                "joined_clicks_visits": OutputDependency(logs.ClickVisitLog)
            },
        )

    def run(self, inputs, outputs, services, ctx):
        ...
```

Как это работает

- У всех данных, зарегистрированных в платформе, есть соответствующее событие в Реакторе
- При деплое для каждого процесса мы настраиваем в Реакторе запуск при появлении соответствующих событий
- Когда события появляются, Реактор запускает операцию в Нирване
- После завершения операции публикуются события для результатов
- На них подписаны следующие узлы графа, в том числе других графов

Граф вычислений

Процессы запускаются реактивно при появлении входных партиций



Что получилось

- Собрали все данные и процессы в одном месте
- Понимаем какие данные какими процессами потребляются и производятся
- Запускаем процессы при появлении необходимых данных
- Алерты и мониторинги на данные и пайплайн
- Тесты и релизный процесс
- Легко пересчитывать данные в прошлое с учётом зависимостей
- Документацию данных

Мониторинг

База

- Мониторим потребление ресурсов: YTsaurus, Nirvana, Reactor
- Мониторим время работы процессов, падения и перезапуски

- Что если процессы и не запускались?
- Потому что входные данные не приехали
- Нужен мониторинг отставания данных!

Описание данных

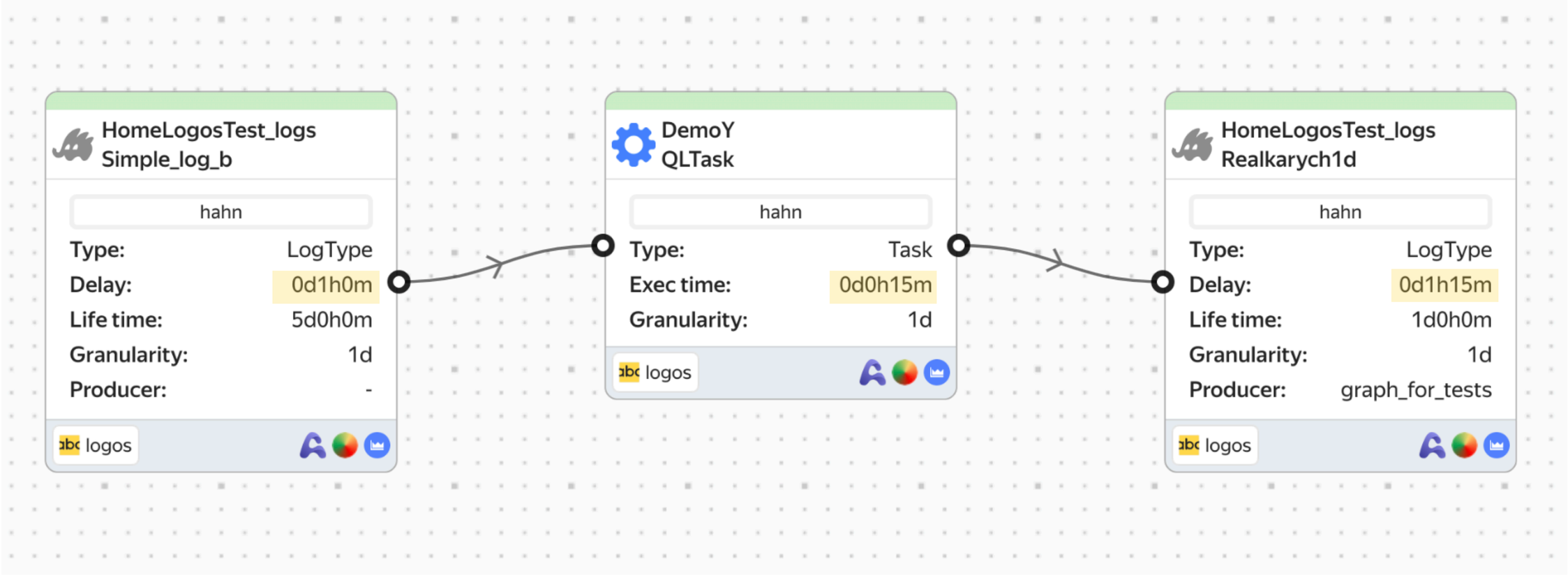
```
# logos/logs/click_log.py

ClickLog1D = lt.Log(
    location=lt.Daily(path="//logs/click-log/1d"),
    life_time="365d",  # чистим старые партиции
    delay="1d30m",  # мониторим появление новых партиций
    responsible="ads-dwh",  # шлём алёрты этим чувакам
)
```

Описание процессов: конфигурация

```
class MyNewTask(BaseLogTask):
    return logos.TaskConfig(
        abc="ads-dwh",  # шлём алёрты этим чувакам
        retry_policy=[...],
        exec_time="2h",  # можем вычислить отставание данных по графу
    )
```

Мониторинг данных с учётом зависимостей

Вычисляем допустимое отставание конкретной партиции по метаграфу



Мониторинг данных с учётом зависимостей

Алёртим только корневую причину задержки, не спамим всем кто от неё зависит

Что получилось

- Собрали все данные и процессы в одном месте
- Понимаем какие данные какими процессами потребляются и производятся
- Запускаем процессы при появлении необходимых данных
- Алерты и мониторинги на данные и пайплайн
- Тесты и релизный процесс
- Легко пересчитывать данные в прошлое с учётом зависимостей
- Документацию данных

https://highload.ru/foundation/2022/abstracts/8295

Тестирование

TL;DR

- Тесты на семплах в CI
- Удобный ручной запуск на production-данных в dev-окружении
- Релизный процесс для каждого графа
- Приёмочные тесты или prestable-контур до релиза в production
- Датачеки в production

Тесты на семплах production-данных

```
class MyNewTask(BaseLogTask):
    def samplers(self):
        return {
            "clicks": logos.FilterTableSampler(
                lambda r: r["eventid"] % (1134 * 10) == 1
            ),
            "visits": logos.FilterTableSampler(
                lambda r: r["eventid"] % (1134 * 10 * 21) == 1
            ),
        }
```

https://pycon.ru/kak-my-testiruem-data-pajplajny-v-reklame-yandeksa

Тесты на семплах production-данных

```
$ ya test —test-param download  # семплируем входные данные
    sbr://123456789 # Table //logs/clicks/2022-02-13  (60.0 MiB, Updated 2024-07-26)
    sbr://987654321 # Table //logs/visits/2022-02-13  (60.0 MiB, Updated 2024-07-26)

$ ya test --canonize-tests  # канонизируем выходные данные
    sbr://123454321 # Table //dwh/joined/2022-02-13  (60.0 MiB, Updated 2024-07-26)

$ ya test  # запускаем тесты
```

> **Запускаем эти тесты на локальной on-demand инсталляции YTsaurus**

https://github.com/yandex/yatool

Проверяем важные инварианты и антипаттерны

- Идемпотентность трансформации
- Идемпотентность семплеров
- Независимость от таймзоны окружения
- Использование только данных, описанных в dependencies
- Операции вне транзакции
- Мусор после операции
- Ошибки конфигурации
- …

Ручной запуск на production-данных

Одной командой можно запустить расчёт на production-данных в dev-окружении

```
$ logos_tool --dev run-tasks MyTask -e 2020-02-02
```

› По результатам оцениваем потребление ресурсов кластера и сравниваем данные dev vs. prod

YT resources spent	Diff	Current x1 tasks	Prod x1 tasks
CPU-days	0.63%	5.06	5.03
Used CPU-days	12.64%	1.40	1.24
Reserved GiB days	-37.78%	4.50	7.24
Used GiB days	0.10%	6.93	6.92
Operations count	0.00%	1	1

Такой отчёт обязательно прилагается ко всем пул-реквестам в DWH Рекламы

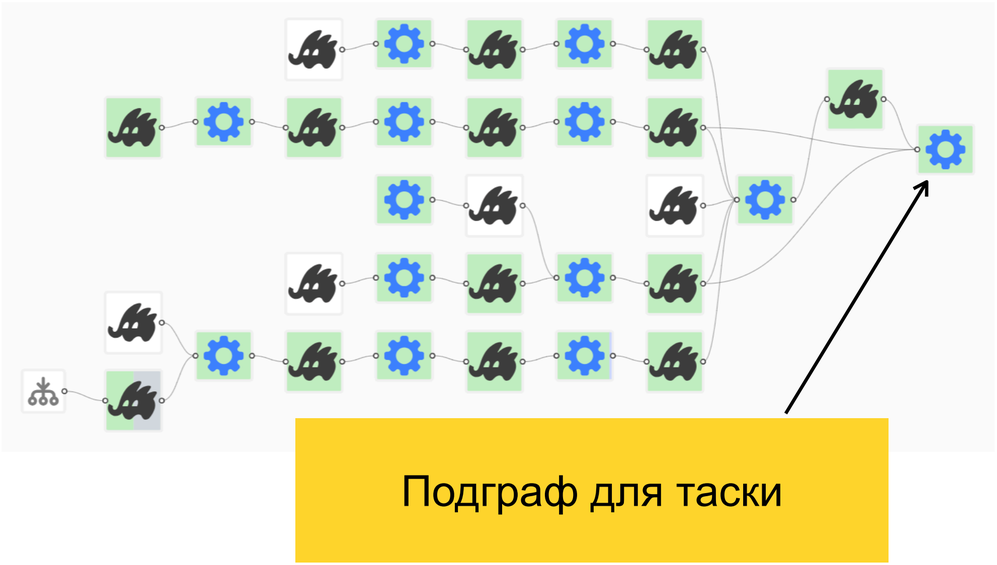
Ручной запуск на production-данных

```
======= RECORDS STATS =======
Given records: 8203
Expected records: 8203
OK records: 0
Changed records: 8203
Changed records columns stat:
    NEW columns stat:
        SomeNewColumn: 8203
Missing records: 0
New records: 0
======= CHANGED RECORDS =======
Changed record for key: {'ShowTime': 1721769365L, 'EventID': 1264982616L}
* NEW     : SomeNewColumn : None => Hello, SmartData!
Changed record for key: {'ShowTime': 1721769239L, 'EventID': 338550515L}
* NEW     : SomeNewColumn : None => Hello, SmartData!
Changed record for key: {'ShowTime': 1721770428L, 'EventID': 1001938076L}
* NEW     : SomeNewColumn : None => Hello, SmartData!
Changed record for key: {'ShowTime': 1721770379L, 'EventID': 1073889016L}
* NEW     : SomeNewColumn : None => Hello, SmartData!
...
```

> **Такой отчёт обязательно прилагается ко всем пул-реквестам в DWH Рекламы**

Приёмочные тесты или prestable-контур

```
$ logos_tool --preprod run-task
             --tasks MyTask AnotherTask
             --expand-to-external
              -e 2020-02-02
```



- **Запускаем произвольный подграф в prestable-окружении, разово или регулярно.**
- **При приёмке релиза формируется такой же отчёт, как для ручного запуска.**

Датачеки в production

- Не все баги удаётся поймать до релиза
- Но можно их найти уже в проде и смягчить последствия для конвейера
- Внедряем проверки прямо в конвейер, блокируем исполнение если что-то не так

Что получилось

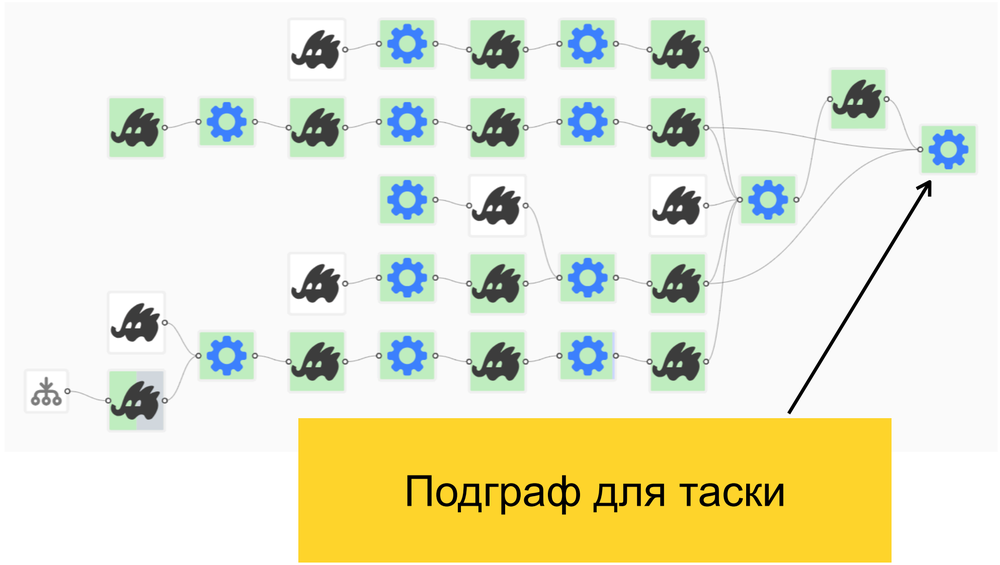
- Собрали все данные и процессы в одном месте
- Понимаем какие данные какими процессами потребляются и производятся
- Запускаем процессы при появлении необходимых данных
- Алерты и мониторинги на данные и пайплайн
- Тесты и релизный процесс
- Легко пересчитывать данные в прошлое с учётом зависимостей
- Документацию данных

https://pycon.ru/kak-my-testiruem-data-pajplajny-v-reklame-yandeksa

Пересчёты

Ручные пересчёты в прошлое (backfill)

```
$ logos_tool --dev run-task
             --tasks MyTask AnotherTask
             --expand-to-external
              -s 2024-01-01
              -e 2024-09-01
             --publish-to-prod
```



Ручной запуск произвольного подграфа, результат публикуем в production

Автоматические пересчёты

```
class MyNewTask(BaseLogTask):
    def dependencies(self):
        return Dependencies(
            inputs={
                "clicks": SingleTableDependency(
                    logs.ClickLog1D,
                    update_policy=logos.UpdatePolicy(
                        attr_name="row_count",
                        rel_threshold=0.01,
                    ),
                ),
            },
            outputs={…}
        )
```

Можно настроить политику автоматического перезапуска тасок при изменении данных

https://smartdataconf.ru/archive/2022/talks/c456b470a593431e9fb535b7a1f9a94e/

Что получилось

- Собрали все данные и процессы в одном месте
- Понимаем какие данные какими процессами потребляются и производятся
- Запускаем процессы при появлении необходимых данных
- Алерты и мониторинги на данные и пайплайн
- Тесты и релизный процесс
- Легко пересчитывать данные в прошлое с учётом зависимостей
- Документацию данных

Документация данных

Описание данных

```
# logos/logs/click_log.py

ClickLog1D = lt.Log(
    location=lt.Daily(path="//logs/click-log/1d"),
    life_time="365d",  # чистим старые партиции
    delay="1d30m",  # мониторим появление новых партиций
    responsible="ads-dwh",  # шлём алёрты этим чувакам
)
```

Документация данных

```
# logos/logs/click_log.md

# ClickLog

Описание лога с кликами

### name | string

Имя юзера

### last_visit_time | uint64

Unixtime последнего визита юзера

### ip | string

Айпишник юзера
...
```

Описание процессов: исполняемый код

```
class MyNewTask(BaseLogTask):
    def run(self, inputs, outputs, services, ctx):
        yql_client = services.yql.get_client()
        query = """
            INSERT INTO `{result}` WITH TRUNCATE
            SELECT
                clicks.UserID, clicks.EventTime,
                visits.*,
                clicks.Cost * visits.ExchangeRate as CostUSD
            FROM `{clicks}` as clicks
            LEFT JOIN `{visits}` as visits
            USING (eventid, uniqid, showtime)
        """.format(clicks=inputs["clicks"][0].path),
                   visits=inputs["visits"][0].path,
                   result=outputs["joined_clicks_visits][0].path
        yql_client.execute(query)
```

Описание процессов: документация вручную

```
class MyNewTask(BaseLogTask):
    def doc(self):
        output_schema = logos.DocySchema(keys=["ClickID"])
        output_schema.copy_fields_from(
            field_names=["UserID", "EventTime"],
            source="clicks",
        )
        output_schema.copy_all_fields_from(source="visits")
        output_schema.map(
            name="CostUSD",
            field_type="float",
            source={
                "clicks": ["Cost"],
                "visits": ["ExchangeRate"]
            }
        )
        return {"joined_clicks_visits": output_schema}
```

Из источника “clicks” копируются два поля

Описание процессов: документация вручную

```
class MyNewTask(BaseLogTask):
    def doc(self):
        output_schema = logos.DocySchema(keys=["ClickID"])
        output_schema.copy_fields_from(
            field_names=["UserID", "EventTime"],
            source="clicks",
        )
        output_schema.copy_all_fields_from(source="visits")
        output_schema.map(
            name="CostUSD",
            field_type="float",
            source={
                "clicks": ["Cost"],
                "visits": ["ExchangeRate"]
            }
        )
        return {"joined_clicks_visits": output_schema}
```

Из источника “visits” копируются все поля

Описание процессов: документация вручную

```
class MyNewTask(BaseLogTask):
    def doc(self):
        output_schema = logos.DocySchema(keys=["ClickID"])
        output_schema.copy_fields_from(
            field_names=["UserID", "EventTime"],
            source="clicks",
        )
        output_schema.copy_all_fields_from(source="visits")
        output_schema.map(
            name="CostUSD",
            field_type="float",
            source={
                "clicks": ["Cost"],
                "visits": ["ExchangeRate"]
            }
        )
        return {"joined_clicks_visits": output_schema}
```

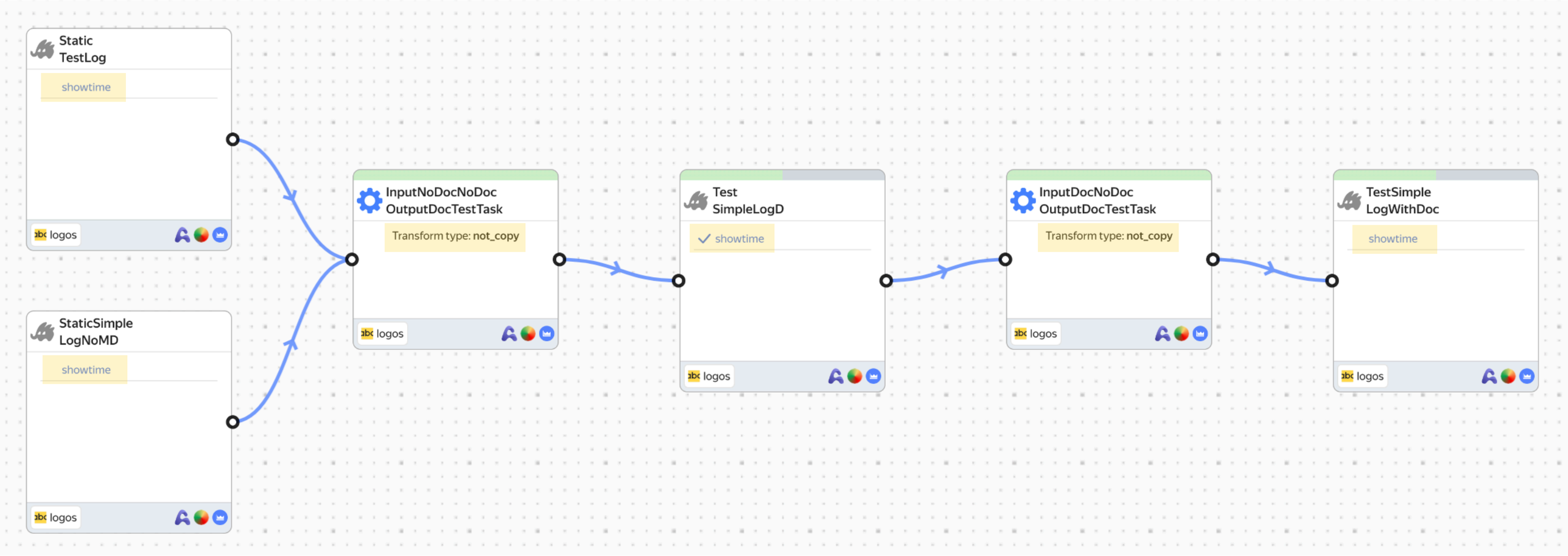
Поле “CostUSD” получено преобразованием над полями из разных источником

Описание процессов: исполняемый код

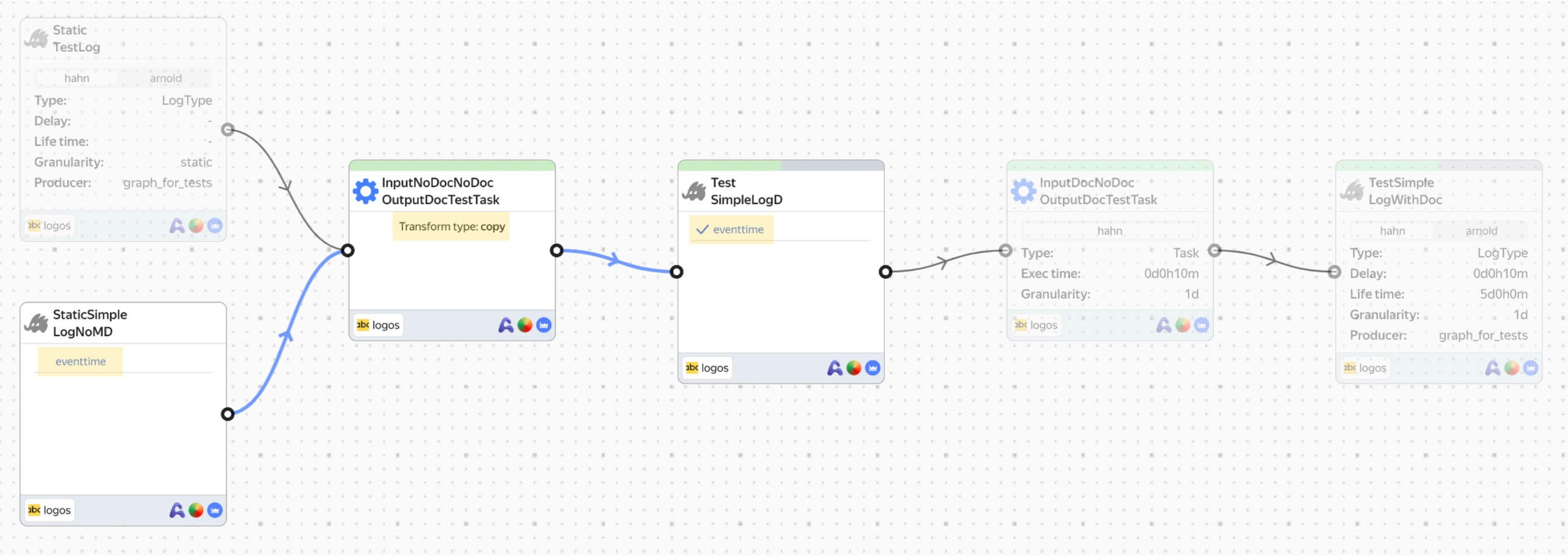
```
class MyNewTask(BaseLogTask):
    def run(self, inputs, outputs, services, ctx):
        yql_client = services.yql.get_client()
        query = """
            INSERT INTO `{result}` WITH TRUNCATE
            SELECT
                clicks.UserID, clicks.EventTime,
                visits.*,
                clicks.Cost * visits.ExchangeRate as CostUSD
            FROM `{clicks}` as clicks
            LEFT JOIN `{visits}` as visits
            USING (eventid, uniqid, showtime)
        """.format(clicks=inputs["clicks"][0].path),
                   visits=inputs["visits"][0].path,
                   result=outputs["joined_clicks_visits"][0].path
        yql_client.execute(query)
```

По YQL-запросу можно построить column-level data lineage автоматически!

Column-level data lineage



Column-level data lineage



Данные на YTsaurus

Content Attributes User attributes ACL Access log Locks 0 **Schema** Edit metadata

Schema mode strong
Strict true
Unique keys false

Help

Filter by name...

Описания полей экспортируются в datacatalog и далее на YTsaurus

#	Name	Type v3	Datacatalog title (exte...	Datacatalog descriptio...
0	cluster	Utf8	no description	Кластер.
1	compressed_compression_codec	Utf8	no description	compression_codec у ...
2	compressed_disk_space	Int64	no description	disk_space у пережат...
3	compressed_erasure_codec	Utf8	no description	erasure_codec у пере...
4	cpu_max	Double	no description	Максимальное число ...
5	cpu_sum	Double	no description	Суммарное число ср...

Что получилось

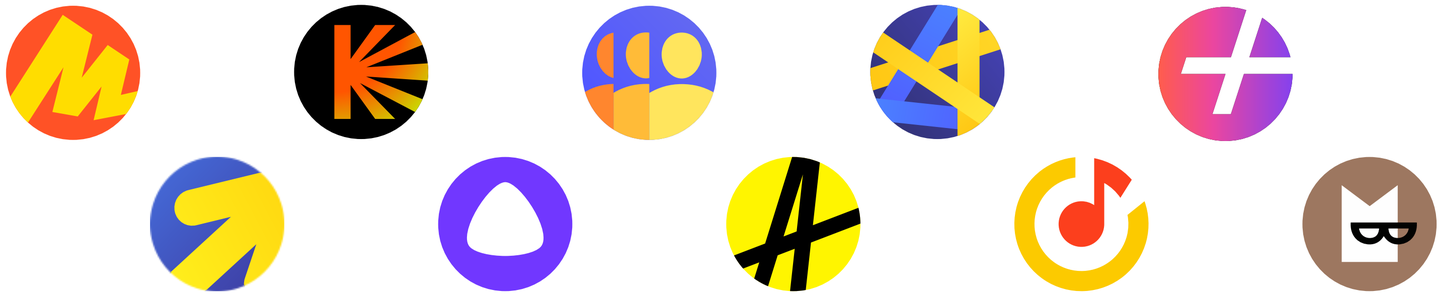
- Собрали все данные и процессы в одном месте
- Понимаем какие данные какими процессами потребляются и производятся
- Запускаем процессы при появлении необходимых данных
- Алерты и мониторинги на данные и пайплайн
- Тесты и релизный процесс
- Легко пересчитывать данные в прошлое с учётом зависимостей
- Документация данных, column-level data lineage

https://pycon.ru/kak-my-testiruem-data-pajplajny-v-reklame-yandeksa

Заключение

Заключение

- Зависимости по данным, а не по таскам — топ
- Тесты, релизный цикл и мониторинги — залог надёжности пайплайна
- Отдельные таски вместо DAG’ов — гибкость
- Монолитный граф — для документации и мониторинга
- Data mesh — для разработки



Яндекс

Спасибо

Алексей Стыценко

Руководитель группы разработки DMP Logos

 astyts@yandex-team.ru

 @stytz