Яндекс Go

Рецепт платформы потоковой обработки данных на Apache Flink

Данил Сабиров

Руководитель группы

О себе

- Руковожу группой развития потоковой обработки данных в DMP Yandex Go
- Разрабатывал NRT процессинг в Belka Games

1

Apache Flink в Yandex Go

Apache Flink в Yandex Go в 2023 году

1

~10 кластеров

2

1 кластер = 1 сервис

3

1 кластер = 1 поставка

4

Ручной управление поставкой

5

Ручной контроль состояния в S3

6

Высокий порог входа

Apache Flink в Yandex Go в 2023 году

1

~10 кластеров

2

1 кластер = 1 сервис

3

1 кластер = 1 поставка

4

Ручной управление поставкой

5

Ручной контроль состояния в S3

6

Высокий порог входа

Apache Flink в Yandex Go в 2023 году

1

~10 кластеров

2

1 кластер = 1 сервис

3

1 кластер = 1 поставка

4

Ручной управление поставкой

5

Ручной контроль состояния в S3

6

Высокий порог входа

Apache Flink в Yandex Go в 2023 году

1

~10 кластеров

2

1 кластер = 1 сервис

3

1 кластер = 1 поставка

4

Ручной управление поставкой

5

Ручной контроль состояния в S3

6

Высокий порог входа

Apache Flink в Yandex Go в 2023 году

1

~10 кластеров

2

1 кластер = 1 сервис

3

1 кластер = 1 поставка

4

Ручной управление поставкой

5

Ручной контроль состояния в S3

6

Высокий порог входа

Apache Flink в Yandex Go в 2023 году

1

~10 кластеров

2

1 кластер = 1 сервис

3

1 кластер = 1 поставка

4

Ручной управление поставкой

5

Ручной контроль состояния в S3

6

Высокий порог входа

Apache Flink в Yandex Go в 2023 году

1

~10 кластеров

2

1 кластер = 1 сервис

3

1 кластер = 1 поставка

4

Ручной управление поставкой

5

Ручной контроль состояния в S3

6

Высокий порог входа

Требования DWH

- Стриминг для рядового Data Engineer
- 100+ потенциально новых поставок
- Apache Flink as Service

Требования DWH

- Стриминг для рядового Data Engineer
- 100+ потенциально новых поставок
- Apache Flink as Service
- 1000+ существующих микробатчинговых поставок

Решать задачу нужно с двух сторон

Причины высокого порога входа

```
    DataStream<Tuple2<String, Integer>> dataStream = env
            .socketTextStream("localhost", 9999)
            .flatMap(new Splitter())
            .keyBy(value -> value.f0)
            .window(TumblingProcessingTimeWindows.of(Time.seconds(5)))
            .sum(1);
    env.execute("Window WordCount");
}

public static class Splitter implements FlatMapFunction<String, Tuple2<String, Integer>> {
    @Override
    public void flatMap(String sentence, Collector<Tuple2<String, Integer>> out) throws Exception {
        for (String word: sentence.split(" ")) {
            out.collect(new Tuple2<String, Integer>(word, 1));
        }
    }
}
```

Сложный DSL

Stream API на Java оперирует сообщениями как POJO и предоставляет прямой доступ к управлению состоянием

```
./bin/flink run -pyExec /usr/bin/python3.7 \
    -pyClientExec /usr/bin/python3.7 \
    -pyfs /opt/flink/resources/test/1.23 \
    -s s3://yandex-go-test/flink-savepoint/savepoint-426afc-4179383e300c \
    --python jobs/my_job/job.py some_props
```

Ручное управление поставками

Data Engineer должен самостоятельно запускать/останавливать/обновлять свои поставки и следить за состоянием в S3

Решать задачу нужно с двух сторон

Причины высокого порога входа

```
    DataStream<Tuple2<String, Integer>> dataStream = env
            .socketTextStream("localhost", 9999)
            .flatMap(new Splitter())
            .keyBy(value -> value.f0)
            .window(TumblingProcessingTimeWindows.of(Time.seconds(5)))
            .sum(1);
    env.execute("Window WordCount");
}

public static class Splitter implements FlatMapFunction<String, Tuple2<String, Integer>> {
    @Override
    public void flatMap(String sentence, Collector<Tuple2<String, Integer>> out) throws Exception {
        for (String word: sentence.split(" ")) {
            out.collect(new Tuple2<String, Integer>(word, 1));
        }
    }
}
```

Сложный DSL

Stream API на Java оперирует сообщениями как POJO и предоставляет прямой доступ к управлению состоянием

```
./bin/flink run -pyExec /usr/bin/python3.7 \
    -pyClientExec /usr/bin/python3.7 \
    -pyfs /opt/flink/resources/test/1.23 \
    -s s3://yandex-go-test/flink-savepoint/savepoint-426afc-4179383e300c \
    --python jobs/my_job/job.py some_props
```

Ручное управление поставками

Data Engineer должен самостоятельно запускать/останавливать/обновлять свои поставки и следить за состоянием в S3

Решать задачу нужно с двух сторон

Причины высокого порога входа

```
    DataStream<Tuple2<String, Integer>> dataStream = env
            .socketTextStream("localhost", 9999)
            .flatMap(new Splitter())
            .keyBy(value -> value.f0)
            .window(TumblingProcessingTimeWindows.of(Time.seconds(5)))
            .sum(1);
    env.execute("Window WordCount");
}

public static class Splitter implements FlatMapFunction<String, Tuple2<String, Integer>> {
    @Override
    public void flatMap(String sentence, Collector<Tuple2<String, Integer>> out) throws Exception {
        for (String word: sentence.split(" ")) {
            out.collect(new Tuple2<String, Integer>(word, 1));
        }
    }
}
```

Сложный DSL

Stream API на Java оперирует сообщениями как POJO и предоставляет прямой доступ к управлению состоянием

```
./bin/flink run -pyExec /usr/bin/python3.7 \
    -pyClientExec /usr/bin/python3.7 \
    -pyfs /opt/flink/resources/test/1.23 \
    -s s3://yandex-go-test/flink-savepoint/savepoint-426afc-4179383e300c \
    --python jobs/my_job/job.py some_props
```

Ручное управление поставками

Data Engineer должен самостоятельно запускать/останавливать/обновлять свои поставки и следить за состоянием в S3

2

Выбираем API для описания поставок

API для построения преобразований

Stream API

Низкоуровневый API для описания преобразований в императивном стиле с прямым доступом к состоянию

Table API

Sources, sinks и промежуточные преобразования представляются в виде таблиц, а управление стейтом скрыто

SQL

Аналогичный с Table API подход, но для описания поставок используется синтаксис SQL

API для построения преобразований

Stream API

Низкоуровневый API для описания преобразований в императивном стиле с прямым доступом к состоянию

Table API

Sources, sinks и промежуточные преобразования представляются в виде таблиц, а управление стейтом скрыто

SQL

Аналогичный с Table API подход, но для описания поставок используется синтаксис SQL

API для построения преобразований

Stream API

Низкоуровневый API для описания преобразований в императивном стиле с прямым доступом к состоянию

Table API

Sources, sinks и промежуточные преобразования представляются в виде таблиц, а управление стейтом скрыто

SQL

Аналогичный с Table API подход, но для описания поставок используется синтаксис SQL

API для построения преобразований

Stream API

Низкоуровневый API для описания преобразований в императивном стиле с прямым доступом к состоянию

Table API

Sources, sinks и промежуточные преобразования представляются в виде таблиц, а управление стейтом скрыто

SQL

Аналогичный с Table API подход, но для описания поставок используется синтаксис SQL

API для построения преобразований

Stream API

Низкоуровневый API для описания преобразований в императивном стиле с прямым доступом к состоянию

Table API

Sources, sinks и промежуточные преобразования представляются в виде таблиц, а управление стейтом скрыто

SQL

Аналогичный с Table API подход, но для описания поставок используется синтаксис SQL

Преимущества Table API

```
Table orders = tEnv.from(
        TableDescriptor.forConnector("kafka")
                .format("json")
                .schema(schema)
                .build());

Table validOrders = orders
        .select(
                col("id"),
                col("name").lowerCase().as("name"),
                col("cost"),
                col("rowtime"))
        .where(col("cost").isNotNull());

Table result = validOrders
        .window(Tumble.over(lit(1).hours()).on(col("rowtime"))
                .as("hourlyWindow"))
        .groupBy(col("hourlyWindow"), col("name"))
        .select(
                col("name"),
                col("hourlyWindow").end().as("hour"),
                col("cost").avg().as("avgBillingAmount"));

result.insertInto("SinkTable").execute();
```

Понятный API для пользователей знакомых с SQL

Универсальный способ описания данных

Базовые трансформации из коробки

Набор базовых функций

Преимущества Table API

```
Table orders = tEnv.from(
        TableDescriptor.forConnector("kafka")
                .format("json")
                .schema(schema)
                .build());

Table validOrders = orders
        .select(
                col("id"),
                col("name").lowerCase().as("name"),
                col("cost"),
                col("rowtime"))
        .where(col("cost").isNotNull());

Table result = validOrders
        .window(Tumble.over(lit(1).hours()).on(col("rowtime"))
                .as("hourlyWindow"))
        .groupBy(col("hourlyWindow"), col("name"))
        .select(
                col("name"),
                col("hourlyWindow").end().as("hour"),
                col("cost").avg().as("avgBillingAmount"));

result.insertInto("SinkTable").execute();
```

Понятный API для пользователей знакомых с SQL

Универсальный способ описания данных

Базовые трансформации из коробки

Набор базовых функций

Преимущества Table API

```
Table orders = tEnv.from(
        TableDescriptor.forConnector("kafka")
                .format("json")
                .schema(schema)
                .build());

Table validOrders = orders
        .select(
                col("id"),
                col("name").lowerCase().as("name"),
                col("cost"),
                col("rowtime"))
        .where(col("cost").isNotNull());

Table result = validOrders
        .window(Tumble.over(lit(1).hours()).on(col("rowtime"))
                .as("hourlyWindow"))
        .groupBy(col("hourlyWindow"), col("name"))
        .select(
                col("name"),
                col("hourlyWindow").end().as("hour"),
                col("cost").avg().as("avgBillingAmount"));

result.insertInto("SinkTable").execute();
```

Понятный API для пользователей знакомых с SQL

Универсальный способ описания данных

Базовые трансформации из коробки

Набор базовых функций

Преимущества Table API

```
Table orders = tEnv.from(
        TableDescriptor.forConnector("kafka")
                .format("json")
                .schema(schema)
                .build());

Table validOrders = orders
        .select(
                col("id"),
                col("name").lowerCase().as("name"),
                col("cost"),
                col("rowtime"))
        .where(col("cost").isNotNull());

Table result = validOrders
        .window(Tumble.over(lit(1).hours()).on(col("rowtime"))
                .as("hourlyWindow"))
        .groupBy(col("hourlyWindow"), col("name"))
        .select(
                col("name"),
                col("hourlyWindow").end().as("hour"),
                col("cost").avg().as("avgBillingAmount"));

result.insertInto("SinkTable").execute();
```

Понятный API для пользователей знакомых с SQL

Универсальный способ описания данных

Базовые трансформации из коробки

Набор базовых функций

Преимущества Table API

```
Schema schema = Schema.newBuilder()
        .column("id", DataTypes.STRING())
        .column("name", DataTypes.STRING())
        .column("cost", DataTypes.DOUBLE())
        .column("doc", DataTypes.ROW(
                DataTypes.FIELD("param_1", DataTypes.STRING()),
                DataTypes.FIELD("param_2", DataTypes.INT()),
                DataTypes.FIELD("doc_in_doc", DataTypes.ROW(
                        DataTypes.FIELD("param_3", DataTypes.BOOLEAN()),
                        DataTypes.FIELD("param_4", DataTypes.ARRAY(DataTypes.INT()))
                ))))
        .column("rowtime", DataTypes.TIMESTAMP())
        .watermark("rowtime", "rowtime - INTERVAL '30 min'")
        .build();
```

Понятный API для пользователей знакомых с SQL

Универсальный способ описания данных

Базовые трансформации из коробки

Набор базовых функций

Преимущества Table API

```
Table validOrders = orders
        .select(
                col("id"),
                col("name").lowerCase().as("name"),
                col("cost"),
                col("rowtime"))
        .where(col("cost").isNotNull())
        .leftOuterJoin(transactions, col("id").isEqual(col("orderId")))
        .unionAll(mobileOrders)
        .addColumns(currentTimestamp().as("processTimestamp"))
        .addOrReplaceColumns(currentDate().as("processDate"))
        .dropColumns(col("processDate"))
        .distinct()
        .intersect(acceptedOrders)
        .window(Tumble.over(lit(1).hours()).on(col("rowtime"))
                .as("hourlyWindow"))
        .groupBy(col("hourlyWindow"), col("name"))
        ...
```

Понятный API для пользователей знакомых с SQL

Универсальный способ описания данных

Базовые трансформации из коробки

Набор базовых функций

Преимущества Table API

```
Table validOrders = orders
        .select(// string
                col("id").upperCase(),
                col("name").lowerCase(),
                concat(col("id"), col("name")),
                col("id").substr(1, 2),
                // numeric
                col("cost").round(2),
                col("cost").abs(),
                uuid(),
                // temporal
                currentTimestamp(),
                currentTime(),
                col("event_dttm").toTimestamp(),
                // collections
                col("array").at(1),
                col("array").arrayContains("test"),
                map("red", "FF0000",
                      "blue", "0000FF",
                      "green", "00FF00"),
                // json
                col("json").jsonValue("$.name"),
                jsonObject(JsonOnNull.NULL, "doc",
                      jsonObject(JsonOnNull.NULL, "key", "value")))
```

- Понятный API для пользователей знакомых с SQL
- Универсальный способ описания данных
- Базовые трансформации из коробки
- Набор базовых функций

Преимущества Table API

- Поддержка популярных форматов
- Автоматический контроль состояния
- SQL вставки
- Stream API вставки

CSV
JSON
Apache Avro
Protobuf
Debezium CDC
Raw
...

YSON
TSKV
...

Преимущества Table API

- **Поддержка популярных форматов**
- Автоматический контроль состояния
- SQL вставки
- Stream API вставки

CSV
JSON
Apache Avro
Protobuf
Debezium CDC
Raw
...

YSON
TSKV
...

Преимущества Table API

- Поддержка популярных форматов
- Автоматический контроль состояния
- SQL вставки
- Stream API вставки

~~ProcessFunction~~
~~KeyedProcessFunction~~
~~CoProcessFunction~~
~~KeyedCoProcessFunction~~
~~BroadcastProcessFunction~~
~~KeyedBroadcastProcessFunction~~

Преимущества Table API

- Поддержка популярных форматов
- Автоматический контроль состояния
- SQL вставки
- Stream API вставки

```
Table sqlInColumn = orders
        .select(
                col("id"),
                callSql("LOWER(name) AS name"),
                col("cost"),
                col("rowtime"),
                callSql("PROCTIME() AS proctime"));

tEnv.createTemporaryView("orders", sqlInColumn);

Table lookupJoin = tEnv.sqlQuery(
        "SELECT * FROM orders as l" +
        " JOIN lookup_table FOR SYSTEM_TIME AS OF l.proctime as r" +
        " ON l.id = r.id"
);
```

Преимущества Table API

- Поддержка популярных форматов
- Автоматический контроль состояния
- SQL вставки
- Stream API вставки

```
Table sqlInColumn = orders
        .select(
                col("id"),
                callSql("LOWER(name) AS name"),
                col("cost"),
                col("rowtime"),
                callSql("PROCTIME() AS proctime"));

tEnv.createTemporaryView("orders", sqlInColumn);

Table lookupJoin = tEnv.sqlQuery(
        "SELECT * FROM orders as l" +
        " JOIN lookup_table FOR SYSTEM_TIME AS OF l.proctime as r" +
        " ON l.id = r.id"
);
```

Преимущества Table API

- Поддержка популярных форматов
- Автоматический контроль состояния
- SQL вставки
- Stream API вставки

```
Table sqlInColumn = orders
        .select(
                col("id"),
                callSql("LOWER(name) AS name"),
                col("cost"),
                col("rowtime"),
                callSql("PROCTIME() AS proctime"));

tEnv.createTemporaryView("orders", sqlInColumn);

Table lookupJoin = tEnv.sqlQuery(
        "SELECT * FROM orders as l" +
        " JOIN lookup_table FOR SYSTEM_TIME AS OF l.proctime as r" +
        " ON l.id = r.id"
);
```

Преимущества Table API

- Поддержка популярных форматов
- Автоматический контроль состояния
- SQL вставки
- Stream API вставки

```
Table tableOrders = orders
        .select(
                col("id"),
                col("name"),
                col("cost"),
                col("rowtime"));
Table tableTransaction = tEnv.from("transactions");

DataStream<Order> orderStream = tEnv.toDataStream(tableOrders, Order.class);
DataStream<Transaction> transactionStream
        = tEnv.toDataStream(tableTransaction, Transaction.class);

var orderWithTransaction = orderStream
        .map(order -> order.setName(order.getName().toLowerCase()))
        .keyBy(Order::getId)
        .connect(transactionStream.keyBy(Transaction::getOrderId))
        .process(new OrderTransactionProcessFunction());

Table ordersWithTransactions = tEnv.fromDataStream(orderWithTransaction,
        Schema.newBuilder()
                .column("id", DataTypes.STRING())
                .column("name", DataTypes.STRING())
                .column("cost", DataTypes.DOUBLE())
                .column("transactionStatus", DataTypes.STRING())
                .build());
```

Преимущества Table API

- Поддержка популярных форматов
- Автоматический контроль состояния
- SQL вставки
- Stream API вставки

```
Table tableOrders = orders
        .select(
                col("id"),
                col("name"),
                col("cost"),
                col("rowtime"));
Table tableTransaction = tEnv.from("transactions");

DataStream<Order> orderStream = tEnv.toDataStream(tableOrders, Order.class);
DataStream<Transaction> transactionStream
        = tEnv.toDataStream(tableTransaction, Transaction.class);

var orderWithTransaction = orderStream
        .map(order -> order.setName(order.getName().toLowerCase()))
        .keyBy(Order::getId)
        .connect(transactionStream.keyBy(Transaction::getOrderId))
        .process(new OrderTransactionProcessFunction());

Table ordersWithTransactions = tEnv.fromDataStream(orderWithTransaction,
        Schema.newBuilder()
                .column("id", DataTypes.STRING())
                .column("name", DataTypes.STRING())
                .column("cost", DataTypes.DOUBLE())
                .column("transactionStatus", DataTypes.STRING())
                .build());
```

Преимущества Table API

- Поддержка популярных форматов
- Автоматический контроль состояния
- SQL вставки
- Stream API вставки

```
Table tableOrders = orders
        .select(
                col("id"),
                col("name"),
                col("cost"),
                col("rowtime"));
Table tableTransaction = tEnv.from("transactions");

DataStream<Order> orderStream = tEnv.toDataStream(tableOrders, Order.class);
DataStream<Transaction> transactionStream
        = tEnv.toDataStream(tableTransaction, Transaction.class);

var orderWithTransaction = orderStream
        .map(order -> order.setName(order.getName().toLowerCase()))
        .keyBy(Order::getId)
        .connect(transactionStream.keyBy(Transaction::getOrderId))
        .process(new OrderTransactionProcessFunction());

Table ordersWithTransactions = tEnv.fromDataStream(orderWithTransaction,
        Schema.newBuilder()
                .column("id", DataTypes.STRING())
                .column("name", DataTypes.STRING())
                .column("cost", DataTypes.DOUBLE())
                .column("transactionStatus", DataTypes.STRING())
                .build());
```

Преимущества Table API

- Поддержка популярных форматов
- Автоматический контроль состояния
- SQL вставки
- Stream API вставки

```
Table tableOrders = orders
        .select(
                col("id"),
                col("name"),
                col("cost"),
                col("rowtime"));
Table tableTransaction = tEnv.from("transactions");

DataStream<Order> orderStream = tEnv.toDataStream(tableOrders, Order.class);
DataStream<Transaction> transactionStream
        = tEnv.toDataStream(tableTransaction, Transaction.class);

var orderWithTransaction = orderStream
        .map(order -> order.setName(order.getName().toLowerCase()))
        .keyBy(Order::getId)
        .connect(transactionStream.keyBy(Transaction::getOrderId))
        .process(new OrderTransactionProcessFunction());

Table ordersWithTransactions = tEnv.fromDataStream(orderWithTransaction,
        Schema.newBuilder()
                .column("id", DataTypes.STRING())
                .column("name", DataTypes.STRING())
                .column("cost", DataTypes.DOUBLE())
                .column("transactionStatus", DataTypes.STRING())
                .build());
```

Преимущества Table API

- Поддержка популярных форматов
- Автоматический контроль состояния
- SQL вставки
- Stream API вставки

```
Table tableOrders = orders
        .select(
                col("id"),
                col("name"),
                col("cost"),
                col("rowtime"));
Table tableTransaction = tEnv.from("transactions");

DataStream<Order> orderStream = tEnv.toDataStream(tableOrders, Order.class);
DataStream<Transaction> transactionStream
        = tEnv.toDataStream(tableTransaction, Transaction.class);

var orderWithTransaction = orderStream
        .map(order -> order.setName(order.getName().toLowerCase()))
        .keyBy(Order::getId)
        .connect(transactionStream.keyBy(Transaction::getOrderId))
        .process(new OrderTransactionProcessFunction());

Table ordersWithTransactions = tEnv.fromDataStream(orderWithTransaction,
        Schema.newBuilder()
                .column("id", DataTypes.STRING())
                .column("name", DataTypes.STRING())
                .column("cost", DataTypes.DOUBLE())
                .column("transactionStatus", DataTypes.STRING())
                .build());
```

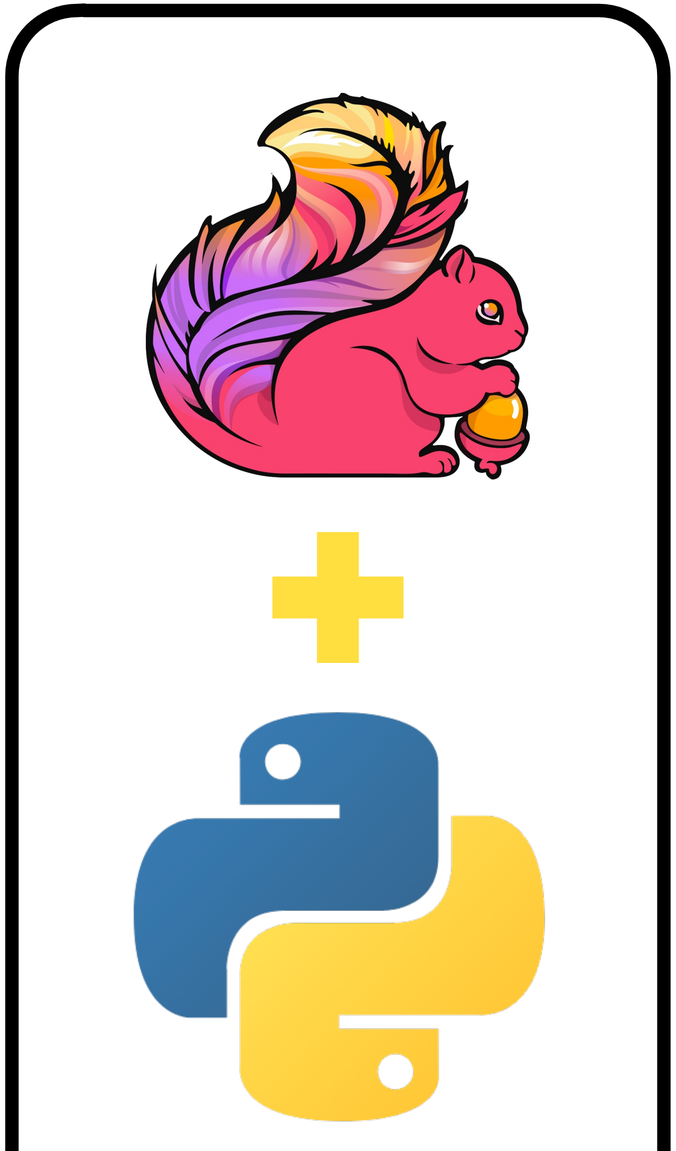
Преимущества Table API на PyFlink

Работает на Java

Без Python UDF, всё преобразование работает на Java

Возможность писать UDF на Python

Ускоряет разработку и делает инструмент более дружелюбным



UDF на Java

Набор стандартных функций, можно расширить собственными на Java

Кастомизация

На Python можно реализовать удобный DSL, а Py4J позволит расширить имеющийся функционал

3

Готовим инфраструктуру

Требования к инфраструктуре

- Автоматическое управление Flink Job
- Несколько Apache Flink кластеров в Session Mode
- Внутри кластера работает несколько поставок
- Контроль состояния в S3

Требования к инфраструктуре

- Автоматическое управление Flink Job
- Несколько Apache Flink кластеров в Session Mode
- Внутри кластера работает несколько поставок
- Контроль состояния в S3

Требования к инфраструктуре

- Автоматическое управление Flink Job
- Несколько Apache Flink кластеров в Session Mode
- Внутри кластера работает несколько поставок
- Контроль состояния в S3

Требования к инфраструктуре

- Автоматическое управление Flink Job
- Несколько Apache Flink кластеров в Session Mode
- Внутри кластера работает несколько поставок
- Контроль состояния в S3

Требования к инфраструктуре

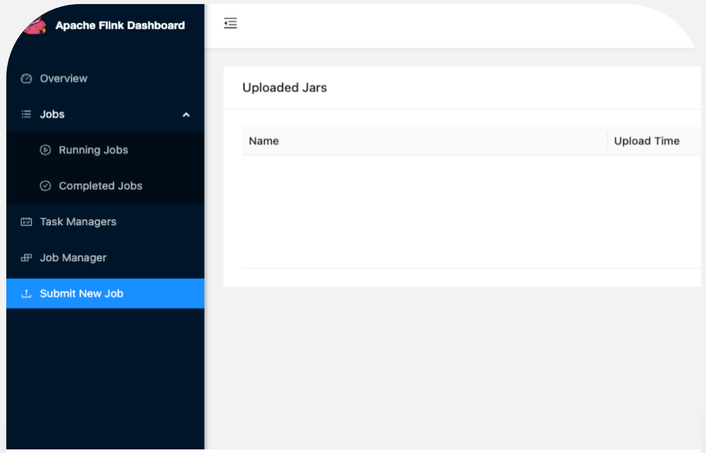
- Автоматическое управление Flink Job
- Несколько Apache Flink кластеров в Session Mode
- Внутри кластера работает несколько поставок
- Контроль состояния в S3

Управление Apache Flink поставками

```
./bin/flink run -pyExec /usr/bin/python3.7 \
   -pyClientExec /usr/bin/python3.7 \
   -pyfs /opt/flink/resources/test/1.23 \
   -s s3://yandex-go-test/flink-savepoint/savepoint... \
   --python jobs/my_job/job.py some_props
```

Flink CLI

Управление поставками производится через flink-client на Job Manager



Flink UI

Дает возможность отправить и запустить jar архив с поставкой

```
curl -X 'POST' \
  'https://apache-flink-cluster.net/jars/upload' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/x-java-archive' \
  --data-binary '@flink.jar'
```

Rest API

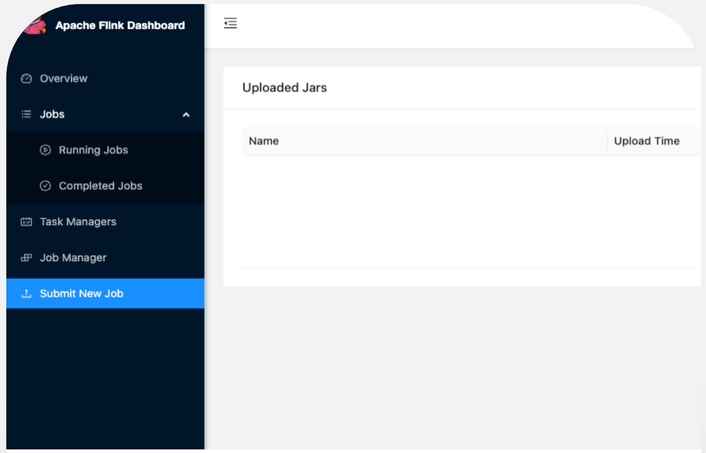
Ограничение на отправку только jar архивов

Управление Apache Flink поставками

```
./bin/flink run -pyExec /usr/bin/python3.7 \
   -pyClientExec /usr/bin/python3.7 \
   -pyfs /opt/flink/resources/test/1.23 \
   -s s3://yandex-go-test/flink-savepoint/savepoint... \
   --python jobs/my_job/job.py some_props
```

Flink CLI

Управление поставками производится через flink-client на Job Manager



Flink UI

Дает возможность отправить и запустить jar архив с поставкой

```
curl -X 'POST' \
  'https://apache-flink-cluster.net/jars/upload' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/x-java-archive' \
  --data-binary '@flink.jar'
```

Rest API

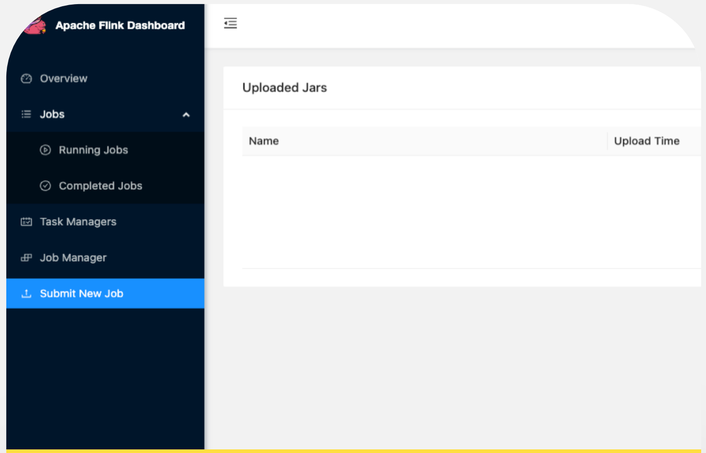
Ограничение на отправку только jar архивов

Управление Apache Flink поставками

```
./bin/flink run -pyExec /usr/bin/python3.7 \
   -pyClientExec /usr/bin/python3.7 \
   -pyfs /opt/flink/resources/test/1.23 \
   -s s3://yandex-go-test/flink-savepoint/savepoint... \
   --python jobs/my_job/job.py some_props
```

Flink CLI

Управление поставками производится через flink-client на Job Manager



Flink UI

Дает возможность отправить и запустить jar архив с поставкой

```
curl -X 'POST' \
  'https://apache-flink-cluster.net/jars/upload' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/x-java-archive' \
  --data-binary '@flink.jar'
```

Rest API

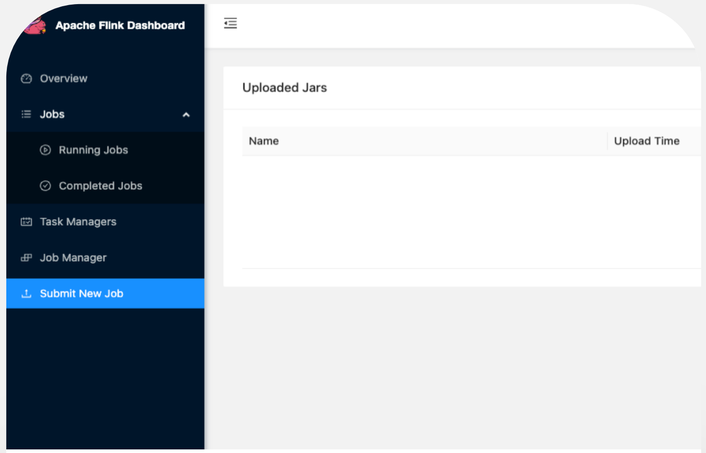
Ограничение на отправку только jar архивов

Управление Apache Flink поставками

```
./bin/flink run -pyExec /usr/bin/python3.7 \
   -pyClientExec /usr/bin/python3.7 \
   -pyfs /opt/flink/resources/test/1.23 \
   -s s3://yandex-go-test/flink-savepoint/savepoint... \
   --python jobs/my_job/job.py some_props
```

Flink CLI

Управление поставками производится через flink-client на Job Manager



Flink UI

Дает возможность отправить и запустить jar архив с поставкой

```
curl -X 'POST' \
  'https://apache-flink-cluster.net/jars/upload' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/x-java-archive' \
  --data-binary '@flink.jar'
```

Rest API

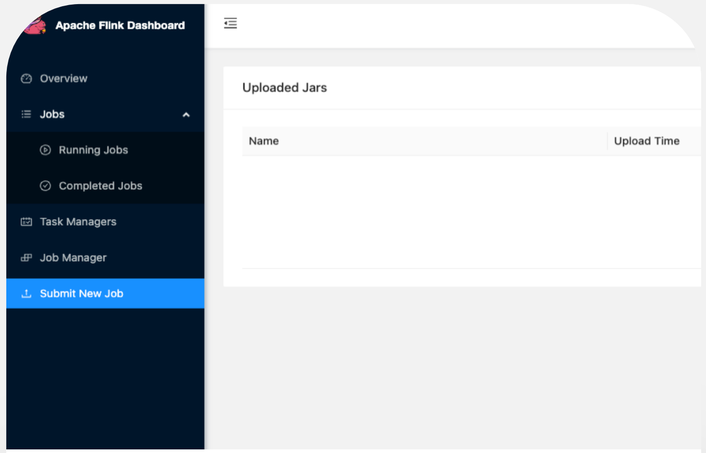
Ограничение на отправку только jar архивов

Управление Apache Flink поставками

```
./bin/flink run -pyExec /usr/bin/python3.7 \
   -pyClientExec /usr/bin/python3.7 \
   -pyfs /opt/flink/resources/test/1.23 \
   -s s3://yandex-go-test/flink-savepoint/savepoint... \
   --python jobs/my_job/job.py some_props
```

Flink CLI

Управление поставками производится через flink-client на Job Manager



Flink UI

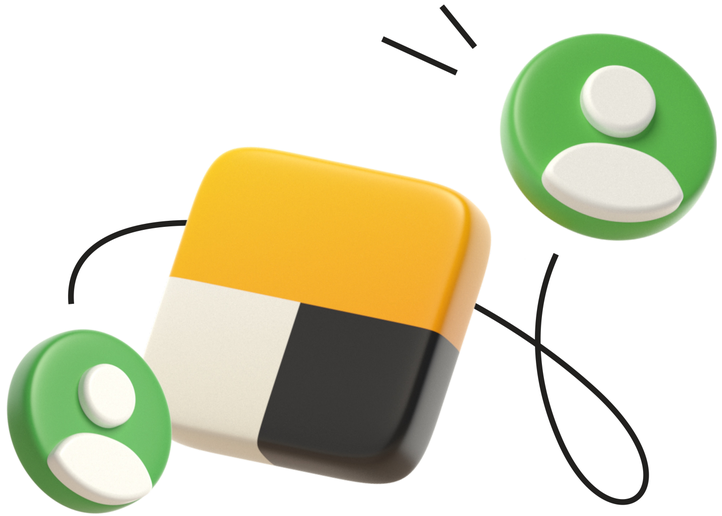
Дает возможность отправить и запустить jar архив с поставкой

```
curl -X 'POST' \
  'https://apache-flink-cluster.net/jars/upload' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/x-java-archive' \
  --data-binary '@flink.jar'
```

Rest API

Ограничение на отправку только jar архивов

Job Manager Sidecar



Job Manager Sidecar

API для управления Flink Job

Генерирует команды для flink-client

API для загрузки Flink Jobs

Поставки и другие ресурсы
загружаются из доверенных источников

API для очистки состояния в S3

Защищает от случайного удаления
актуального состояния

Job Manager Sidecar

API для управления Flink Job

Генерирует команды для flink-client

API для загрузки Flink Jobs

Поставки и другие ресурсы
загружаются из доверенных источников

API для очистки состояния в S3

Защищает от случайного удаления
актуального состояния

Job Manager Sidecar

API для управления Flink Job

Генерирует команды для flink-client

API для загрузки Flink Jobs

Поставки и другие ресурсы
загружаются из доверенных источников

API для очистки состояния в S3

Защищает от случайного удаления
актуального состояния

Job Manager Sidecar

API для управления Flink Job

Генерирует команды для flink-client

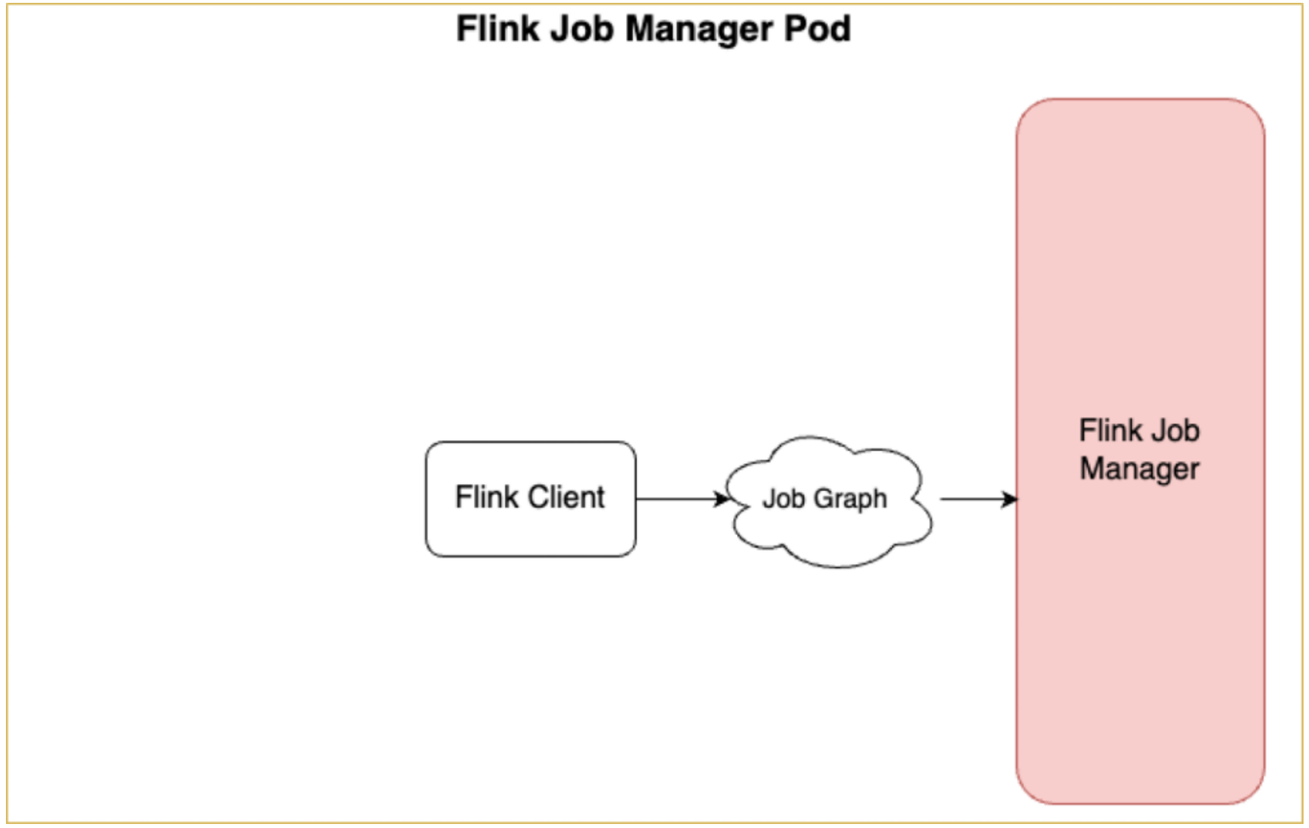
API для загрузки Flink Jobs

Поставки и другие ресурсы
загружаются из доверенных источников

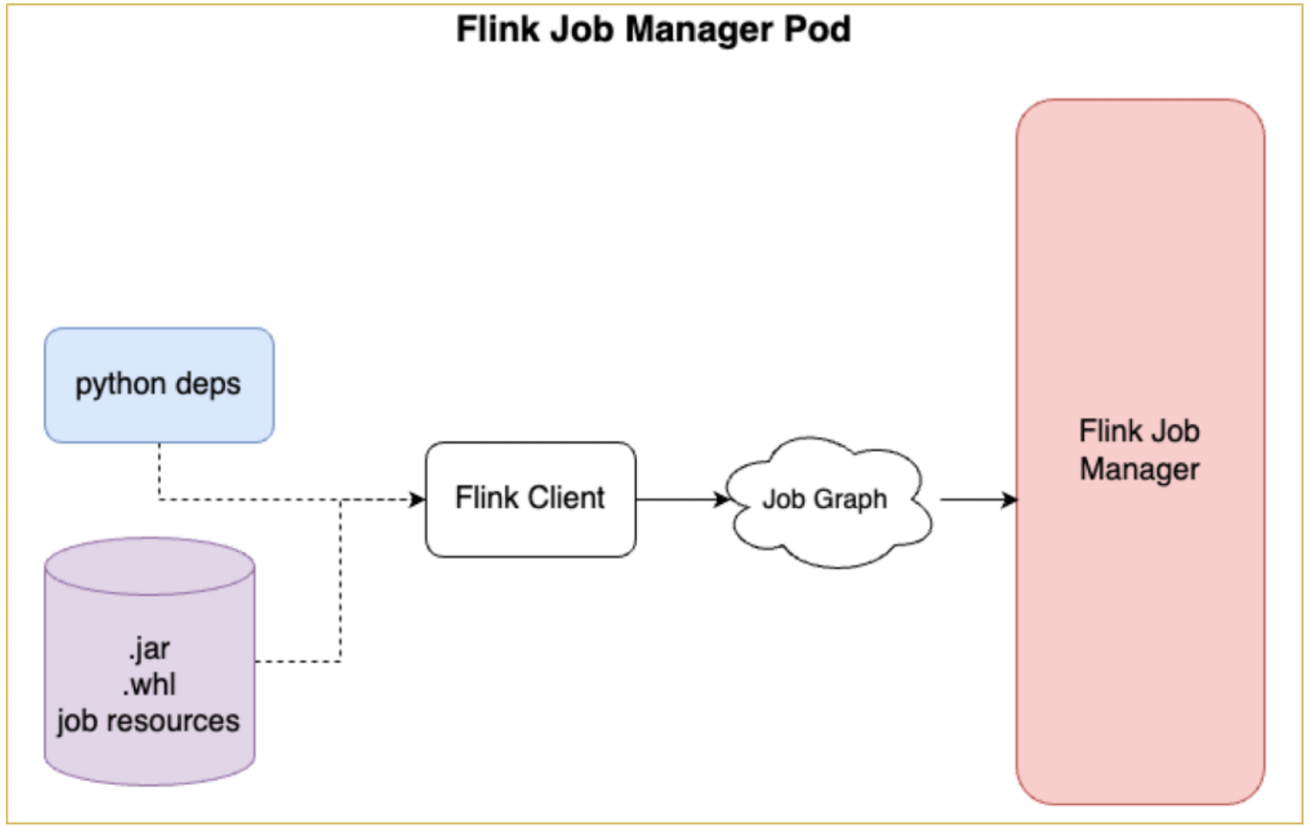
API для очистки состояния в S3

Защищает от случайного удаления
актуального состояния

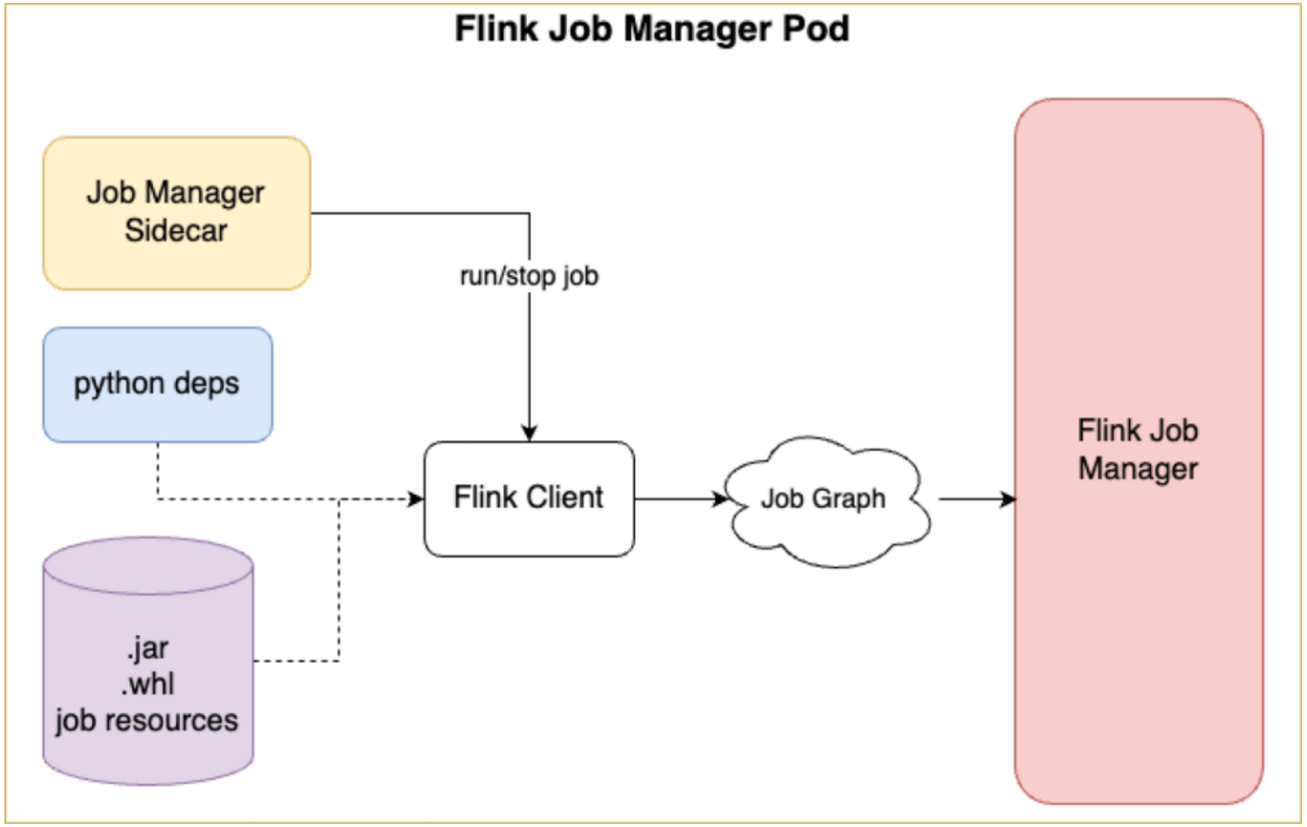
Job Manager Sidecar



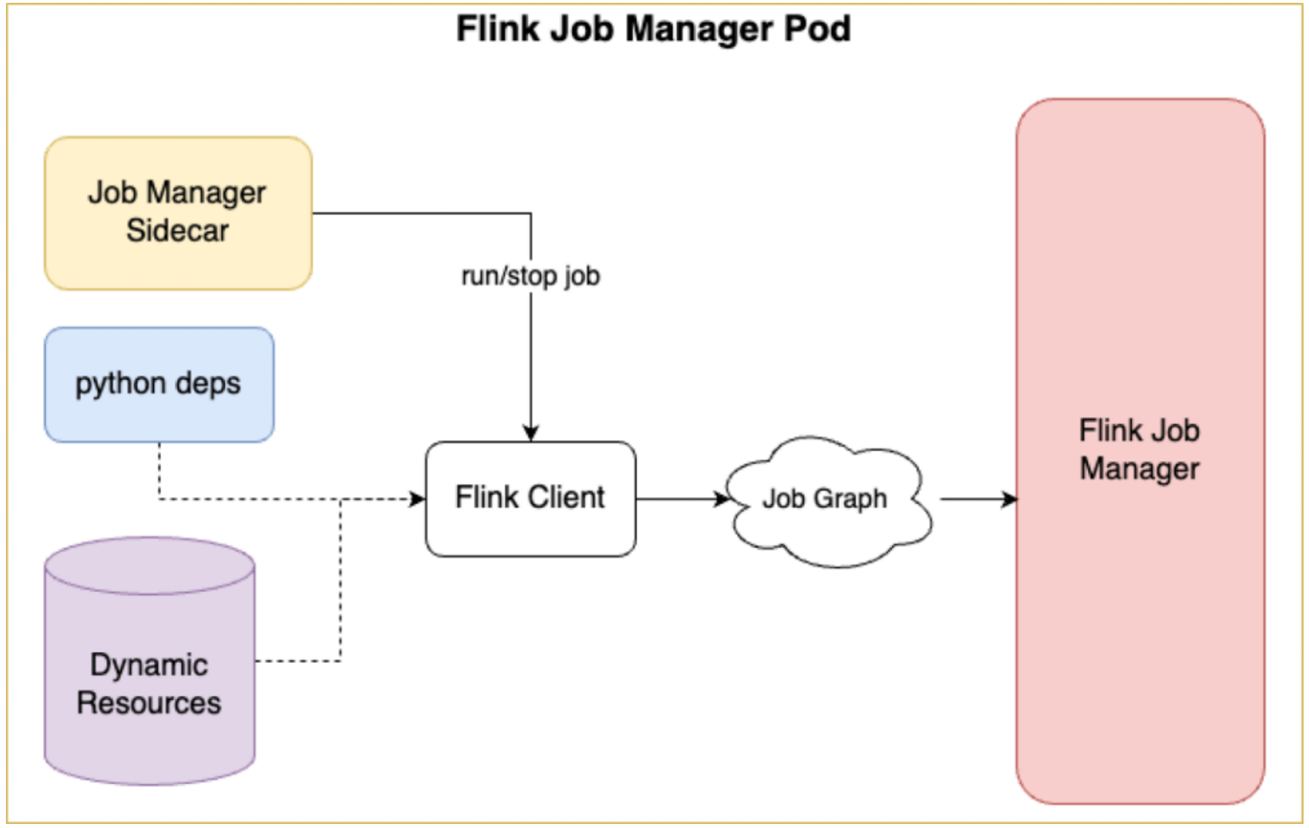
Job Manager Sidecar



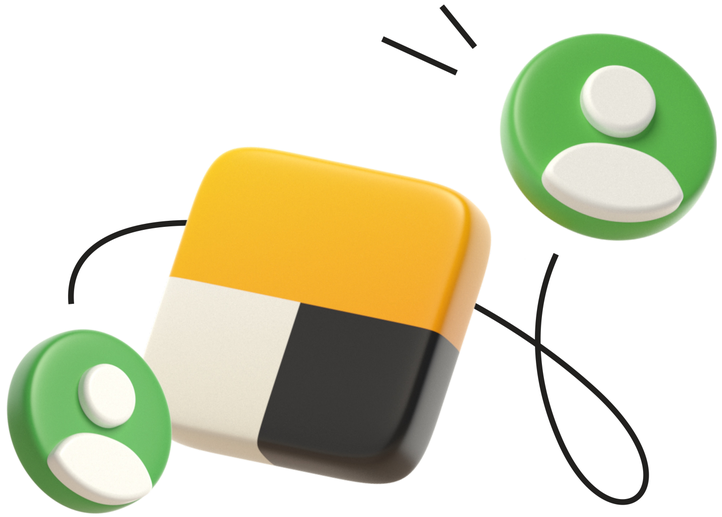
Job Manager Sidecar



Job Manager Sidecar



Flink Jobs Supervisor



Flink Jobs Supervisor

- Знает про все Flink кластеры
- Знает в каком состоянии должна находить каждая поставка
- Приводит реальное состояние поставки к требуемому
- Контроль состояния в S3

Flink Jobs Supervisor

- Знает про все Flink кластеры
- Знает в каком состоянии должна находить каждая поставка
- Приводит реальное состояние поставки к требуемому
- Контроль состояния в S3

Flink Jobs Supervisor

Знает про все Flink кластеры

Знает в каком состоянии должна находить каждая поставка

Приводит реальное состояние поставки к требуемому

Контроль состояния в S3

Flink Jobs Supervisor

Знает про все Flink кластеры

Знает в каком состоянии должна находить каждая поставка

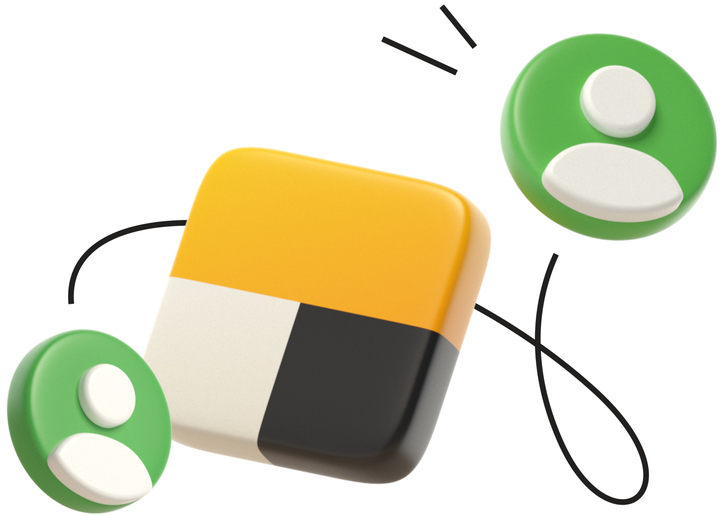
Приводит реальное состояние поставки к требуемому

Контроль состояния в S3

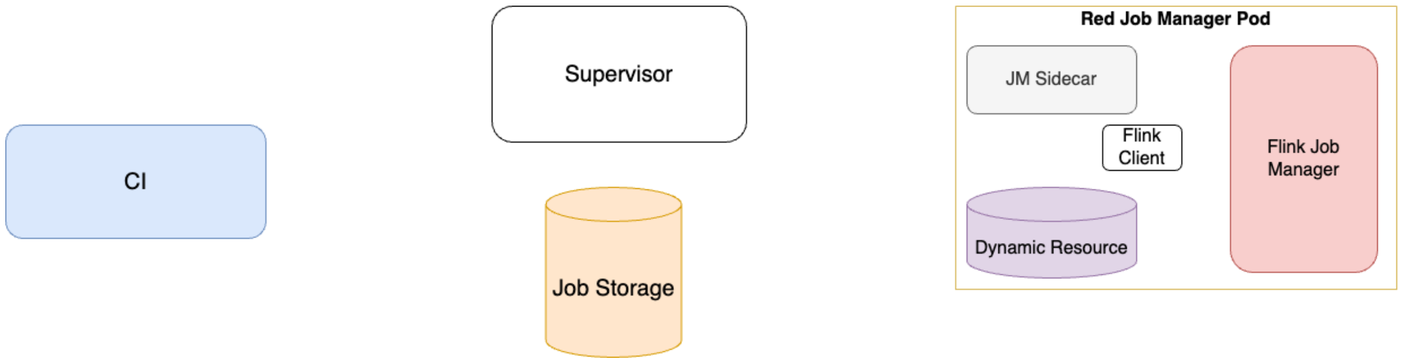
Flink Jobs Supervisor

- Знает про все Flink кластеры
- Знает в каком состоянии должна находить каждая поставка
- Приводит реальное состояние поставки к требуемому
- Контроль состояния в S3

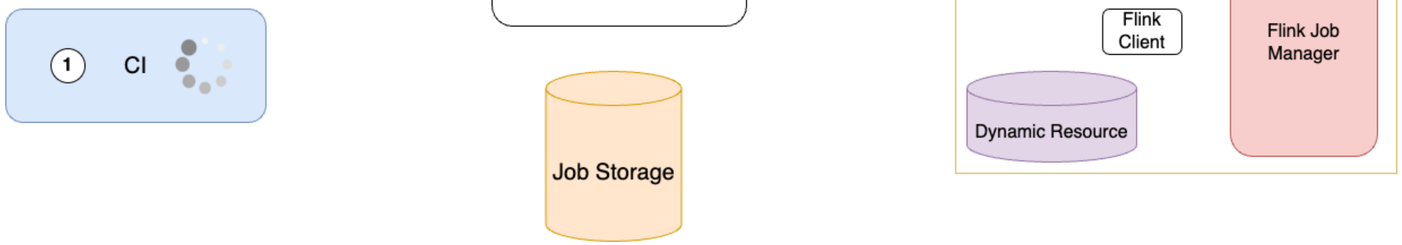
Релизный процесс



Релизный процесс

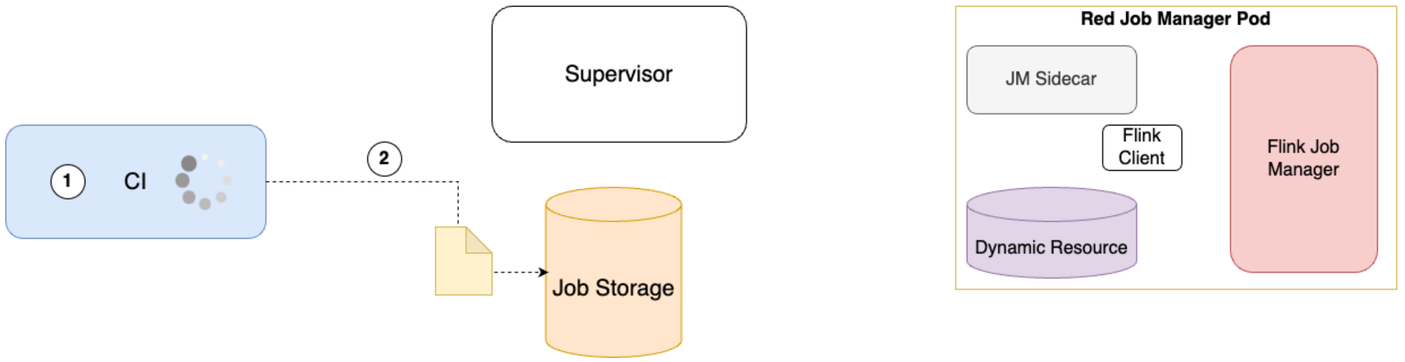


Релизный процесс



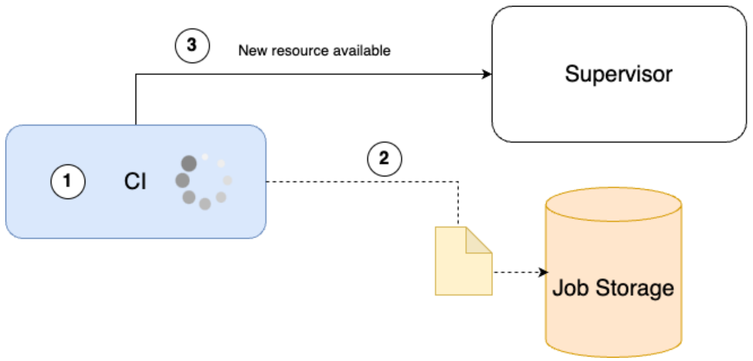
1 Сборка поставок

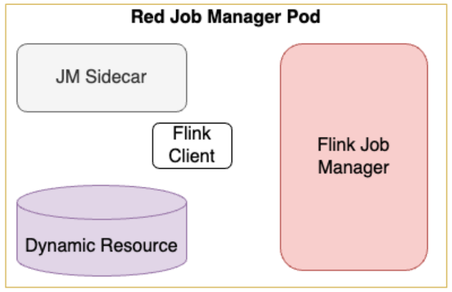
Релизный процесс



1. Сборка поставок
2. Отправка в хранилище

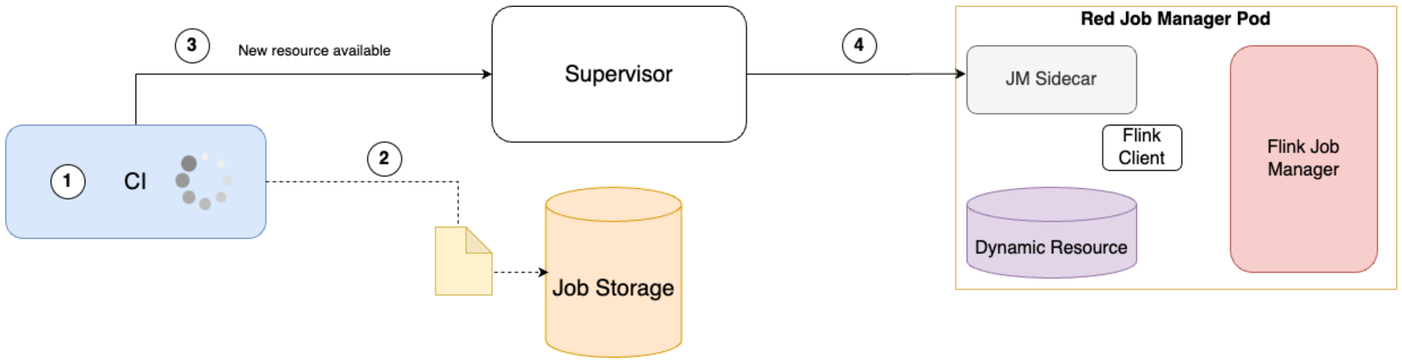
Релизный процесс





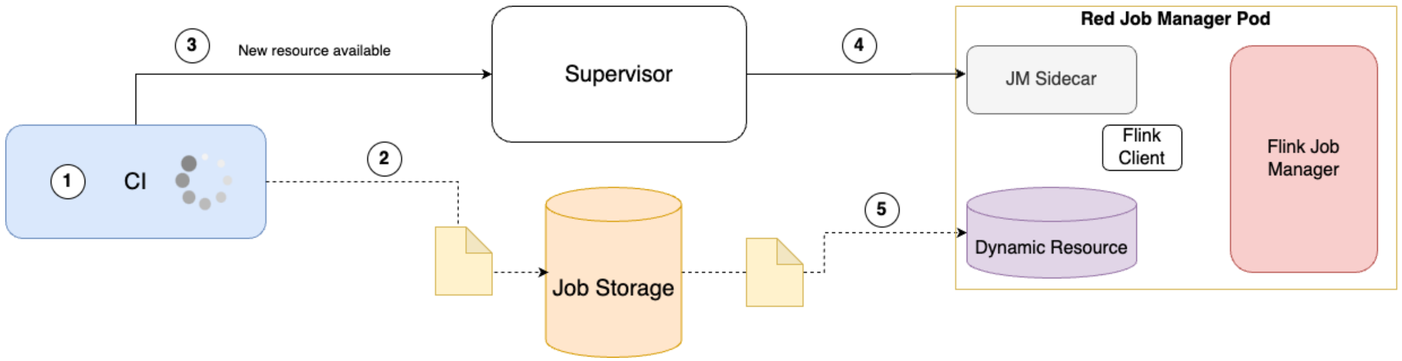
1. Сборка поставок
2. Отправка в хранилище
3. Обновление состояния

Релизный процесс



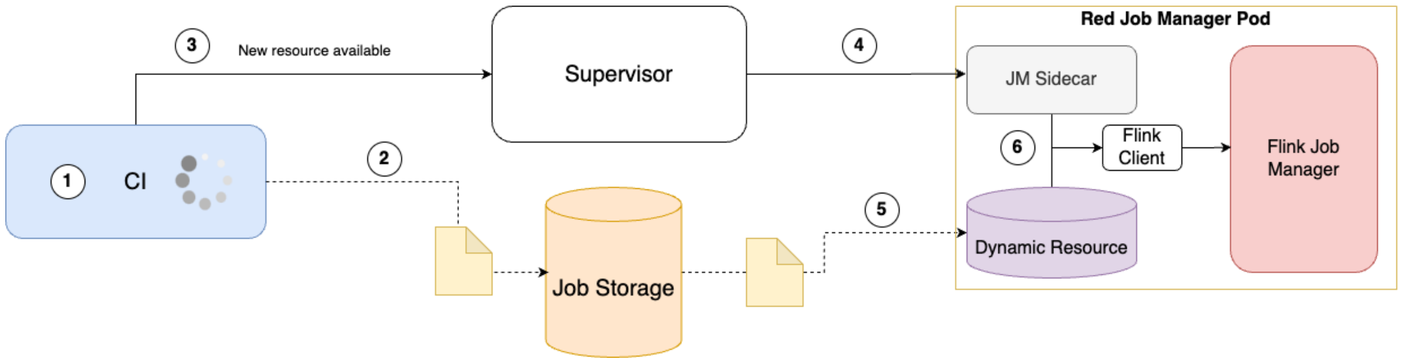
1. Сборка поставок
2. Отправка в хранилище
3. Обновление состояния
4. Обновление поставки

Релизный процесс



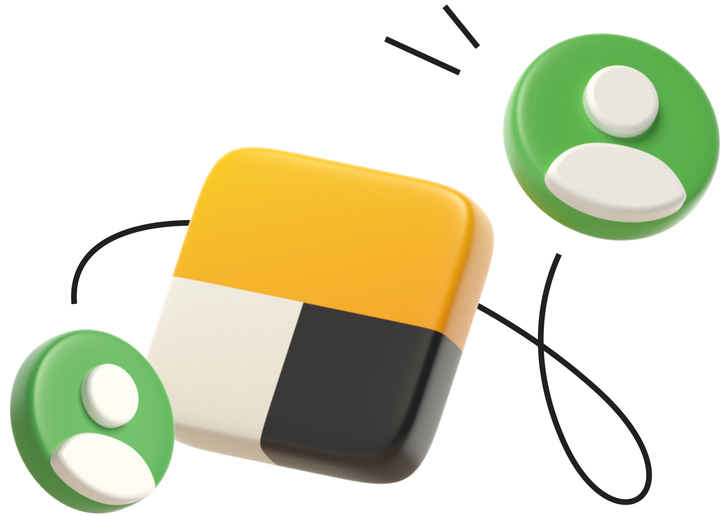
1. Сборка поставок
2. Отправка в хранилище
3. Обновление состояния
4. Обновление поставки
5. Загрузка новой версии

Релизный процесс

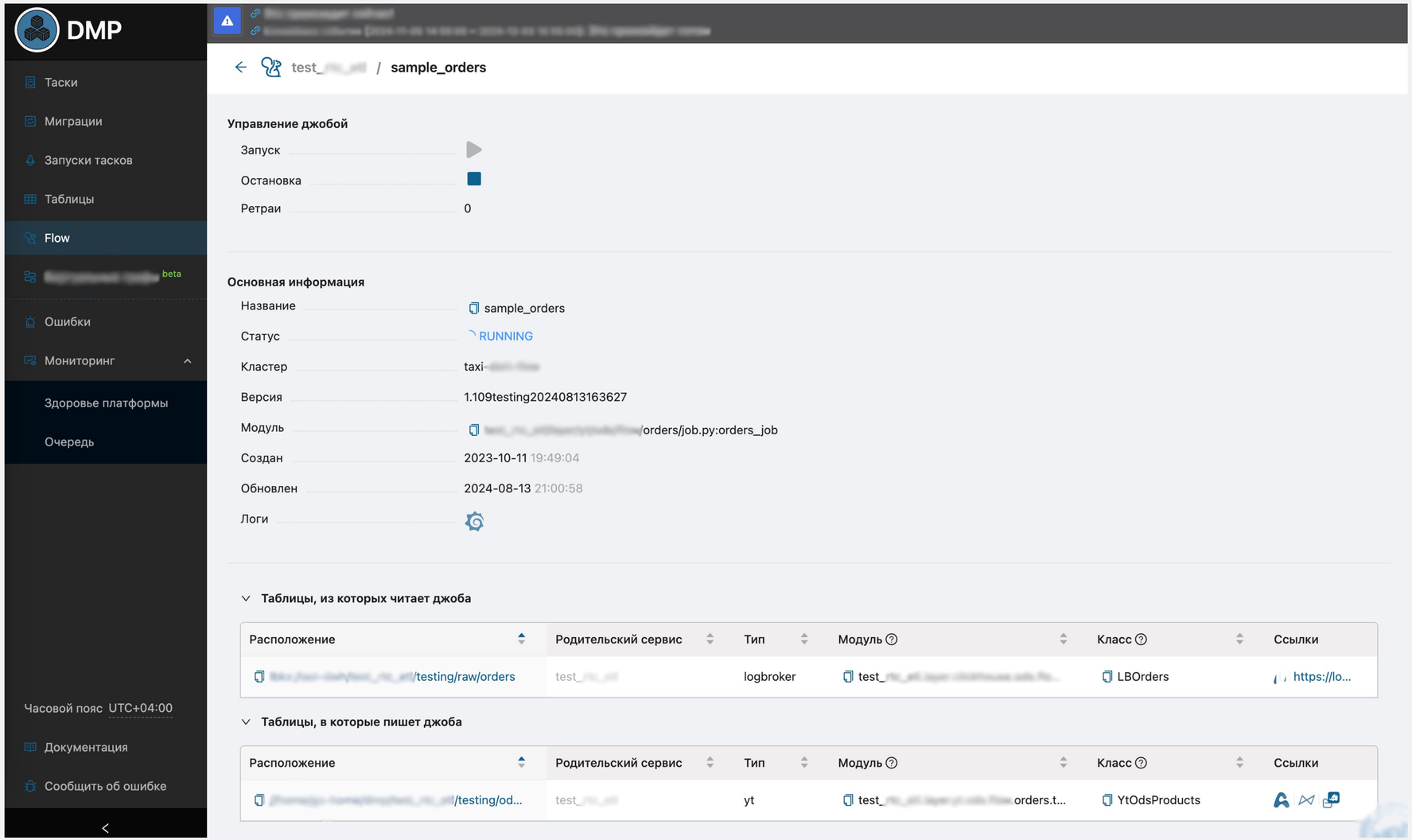


1. Сборка поставок
2. Отправка в хранилище
3. Обновление состояния
4. Обновление поставки
5. Загрузка новой версии
6. Остановка/запуск поставки

Интегрируем в DMP UI



Интегрируем в DMP UI



4

Результаты

Результаты

Apache Flink в Yandex Go в 2024

1. 1 кластер = N поставок
2. Supervisor управляет поставкой
3. Supervisor очищает неактуальное состояние в S3
4. Python DSL основанный на Table API

Apache Flink в Yandex Go в 2023

1. 1 кластер = 1 поставка
2. Ручное управление поставкой
3. Ручной контроль состояния в S3
4. Высокий порог входа

5

Выводы

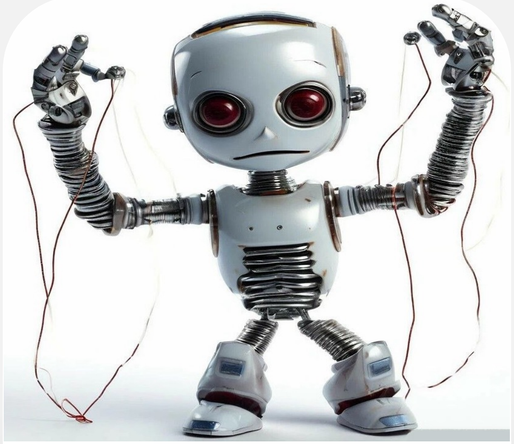
Выводы

Единицы поставок

Все хорошо и базовых возможностей хватает для управления поставками

Десятки поставок

Поставки по прежнему можно контролировать, но уже нужен помощник, который упростит этот процесс



Сотни поставок

Требуется supervisor, который не потеряет поставки и не создаст дубли, а также уследит за стейтом

Доступный API

Мало интегрировать инструмент, нужно предоставить удобный API, чтобы сделать его доступным

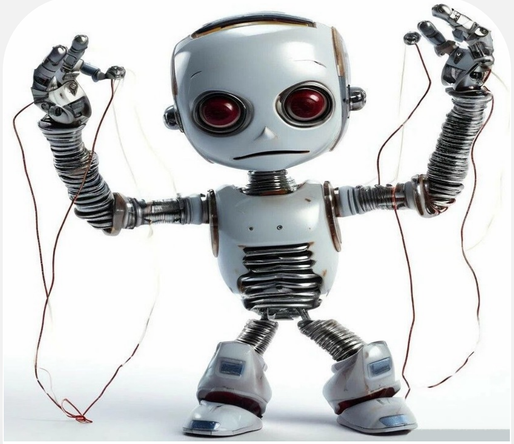
Выводы

Единицы поставок

Все хорошо и базовых возможностей хватает для управления поставками

Десятки поставок

Поставки по прежнему можно контролировать, но уже нужен помощник, который упростит этот процесс



Сотни поставок

Требуется supervisor, который не потеряет поставки и не создаст дубли, а также уследит за стейтом

Доступный API

Мало интегрировать инструмент, нужно предоставить удобный API, чтобы сделать его доступным

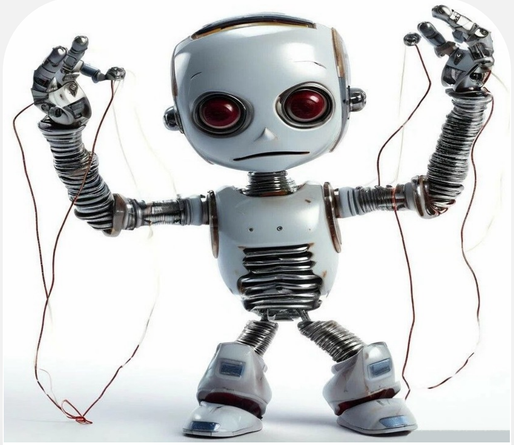
Выводы

Единицы поставок

Все хорошо и базовых возможностей хватает для управления поставками

Десятки поставок

Поставки по прежнему можно контролировать, но уже нужен помощник, который упростит этот процесс



Сотни поставок

Требуется supervisor, который не потеряет поставки и не создаст дубли, а также уследит за стейтом

Доступный API

Мало интегрировать инструмент, нужно предоставить удобный API, чтобы сделать его доступным

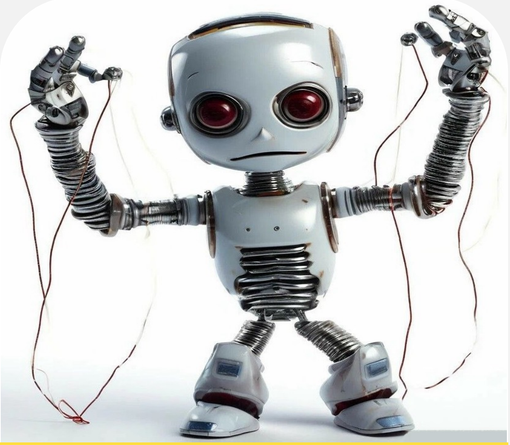
Выводы

Единицы поставок

Все хорошо и базовых возможностей хватает для управления поставками

Десятки поставок

Поставки по прежнему можно контролировать, но уже нужен помощник, который упростит этот процесс



Сотни поставок

Требуется supervisor, который не потеряет поставки и не создаст дубли, а также уследит за стейтом

Доступный API

Мало интегрировать инструмент, нужно предоставить удобный API, чтобы сделать его доступным

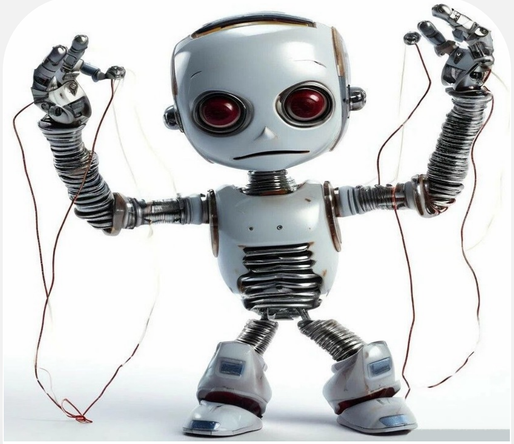
Выводы

Единицы поставок

Все хорошо и базовых возможностей хватает для управления поставками

Десятки поставок

Поставки по прежнему можно контролировать, но уже нужен помощник, который упростит этот процесс



Сотни поставок

Требуется supervisor, который не потеряет поставки и не создаст дубли, а также уследит за стейтом

Доступный API

Мало интегрировать инструмент, нужно предоставить удобный API, чтобы сделать его доступным

Может быть интересно

YTsaurus

Flink Backfill

Пересчет потоковых данных на кластере YTSaurus

DSL на базе PyFlink

Как кастомизировать и привнести новый функционал в PyFlink

Разработка коннекторов

Мы имеем опыт в написании коннекторов/форматеров для Apache Flink

Яндекс Go

Вопросы

Данил Сабиров

Руководитель группы