~~Любовь~~ и ненависть к Prefect 2.0 после Apache Airflow

SmartData
2022

~~Любовь~~ и ненависть к Prefect 2.0 после Apache Airflow

SmartData 2022

Или история про хреновый маркетинг

Спикер

Iuliia Volkova
xnuinside

Developer https://twitter.com/xnuinside |
https://www.linkedin.com/in/xnuinside

Edit profile

95 followers · **9** following

xnuinside@gmail.com
https://medium.com/@xnuinside
@xnuinside

ex-GridDynamics, ex-EPAM,
недавно в Avito DWH

https://github.com/xnuinside

Материалы

https://github.com/xnuinside/smartdata_2022

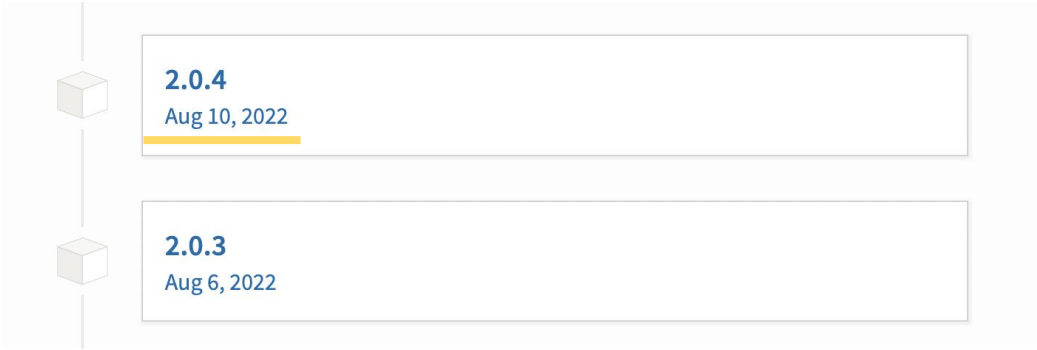
Причем тут Apache Airflow

https://docs-v1.prefect.io/core/about_prefect/why-not-airflow

Релизный цикл

8 Августа я решила

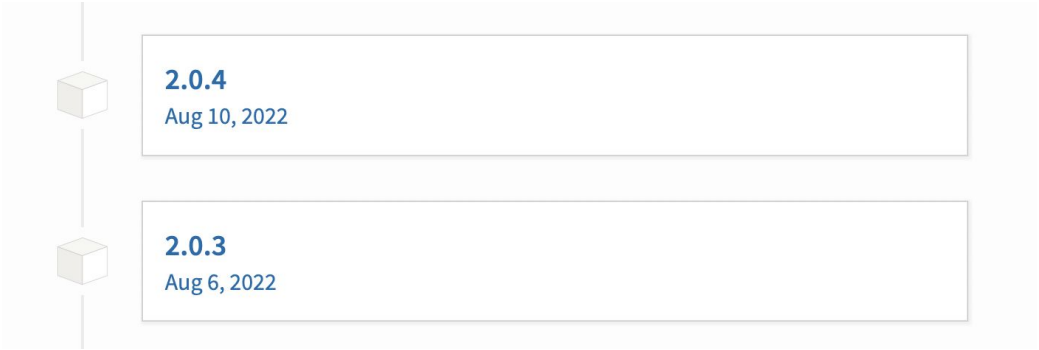
подать доклад на SmartData



Релизный цикл

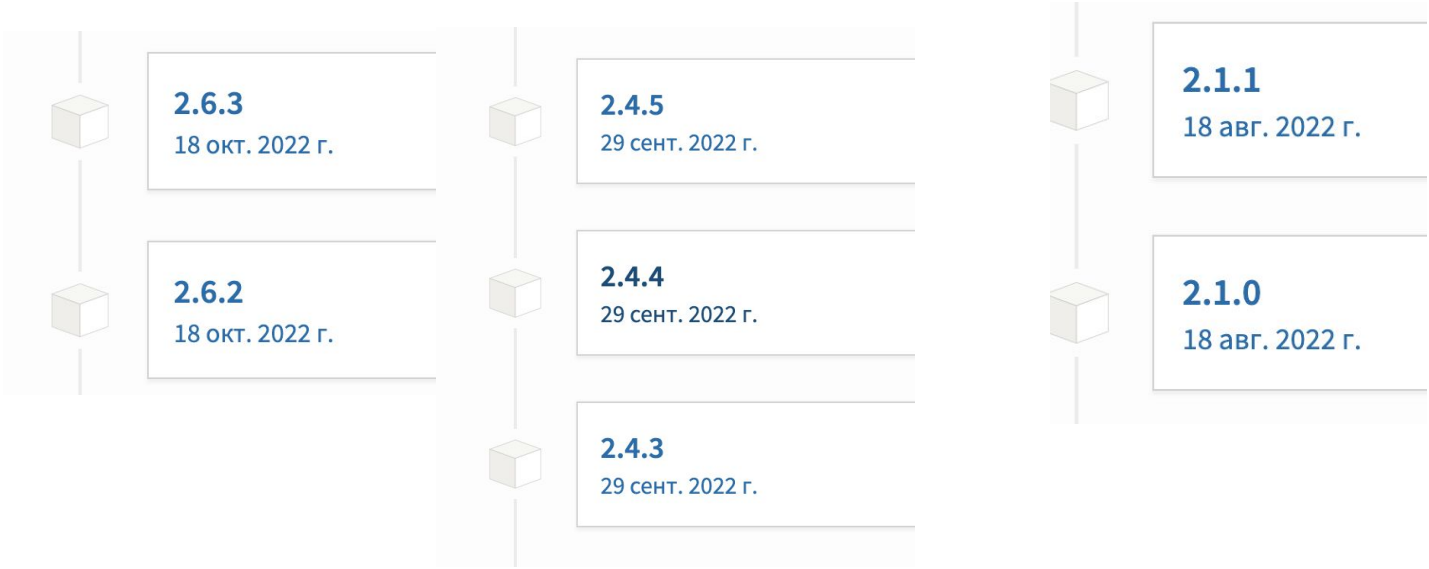
8 Августа - решила

подать доклад на SmartData



Сегодня

Релизный “цикл”??



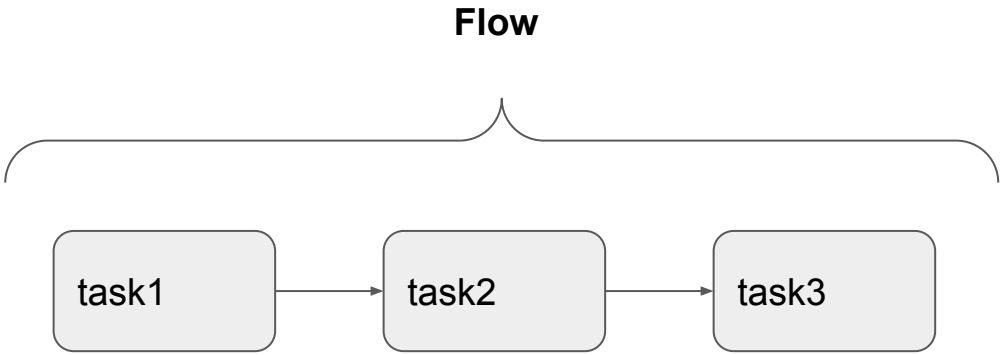
Где релизный цикл?

Агенда

- Что такое Prefect
- Попробуем перенести DAG с Apache Airflow на Prefect
- Посмотрим базовые концепты Prefect
- Посмотрим на передачу данных между задачами
- Поговорим про Block, Deployment, Infrastructure, Storage и тд
- Много нытья в течении доклада

Что такое Prefect?

*Инструмент для создания **каких-то дата пайплайнов***



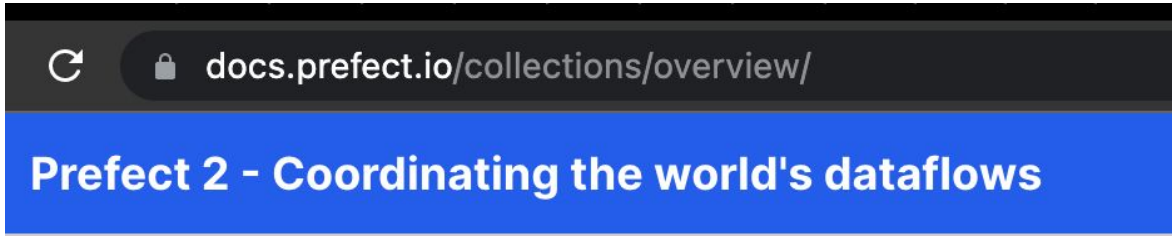
Могла бы и бранчу нарисовать..

Цитаты

Prefect is a global coordination plane for **any data stack**. It allows you to design, build, schedule, and monitor your dataflows, **covering the full spectrum from observability to orchestration.**

https://medium.com/towards-data-science/prefect-aws-ecs-fargate-github-actions-make-serverless-dataflows-as-easy-as-py-f6025335effc

Цитаты



https://www.prefect.io ▾ Перевести эту страницу

Prefect - The New Standard in Dataflow Automation - Prefect

Prefect Cloud is **a trusted tool that allows users to create powerful data pipelines**. Let prefect take care of scheduling, infrastructure, error handling, ...

Те что-то вроде

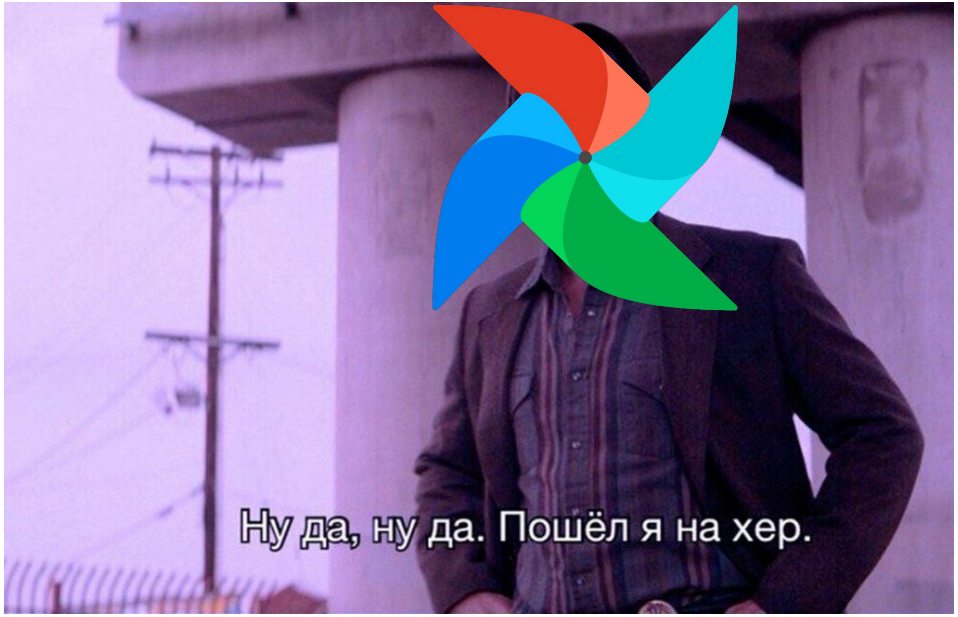
Зная о постоянном сравнении с Apache Airflow

Цитаты

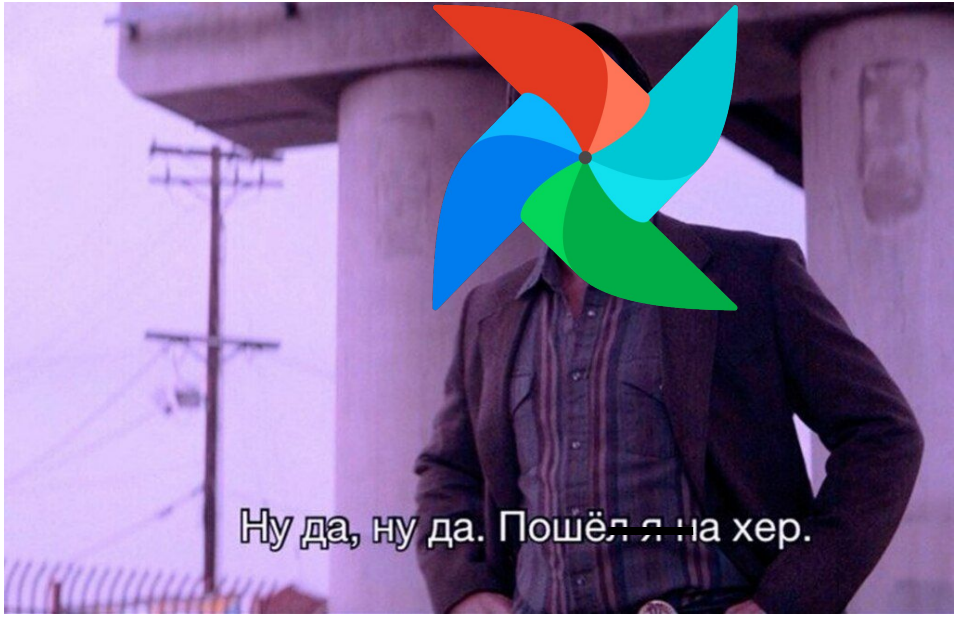
https://www.prefect.io ▾ Перевести эту страницу

Prefect - The New Standard in Dataflow Automation - Prefect

Prefect Cloud is **a trusted tool that allows users to create powerful data pipelines**. Let prefect take care of scheduling, infrastructure, error handling, ...


Ну да, ну да. Пошёл я на хер.

Но после знакомства с Prefect



Или не я ?

Что вообще такое data pipeline?

Data Pipelines & ETL | Apache Flink

One very common use case for Apache **Flink** is to implement ETL (extract, transform, load) **pipelines** that take **data** from one or more sources, perform some ...

Stateless Transformations · Stateful Transformations · Why is Flink Involved in...

https://hevodata.com › learn ▾ Перевести эту страницу

Building Apache Spark Data Pipeline?|Made Easy 101 - Learn

21 апр. 2022 г. — A **Data Pipeline** is a series of steps that ingest raw data from various sources and transport it to a storage and analysis location. The data is ...

What is Data Pipeline? · What is Apache Spark? · How to Build Apache Spark...

Founder

Prefect was founded in 2018 by Jeremiah Lowin, who took his learnings as a PMC member of Apache Airflow

Founder

Jeremiah has a
Finance/Risk Management background.

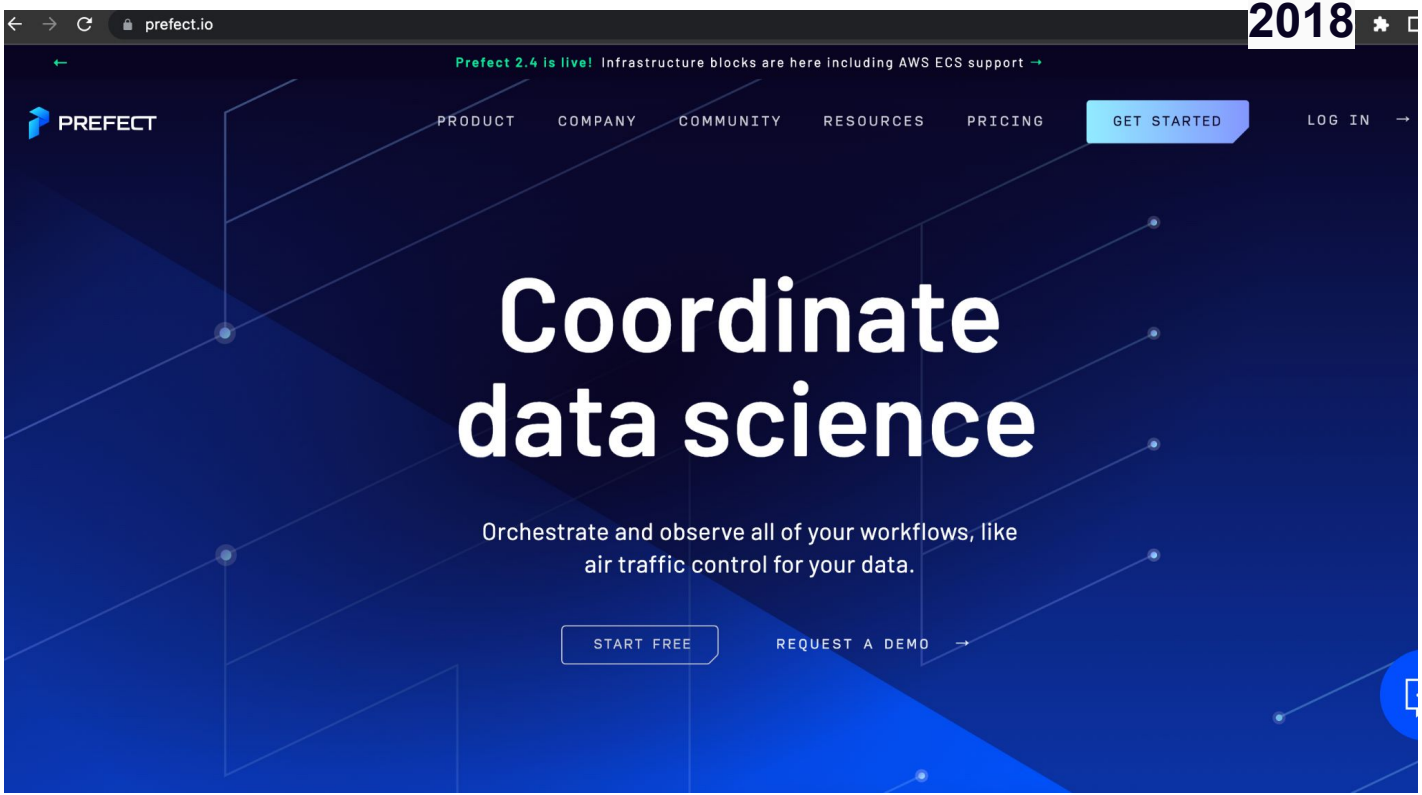
While **his father is a value investor and entrepreneur**, Jeremiah likes to dabble in side projects that catch his interest, but having started a business a decade ago, being a founder again wasn't something he was looking for.

https://peteweishaupt.medium.com/founder-in-focus-jeremiah-lowin-of-prefect-ab36b0dc472f

Prefect - start up & product

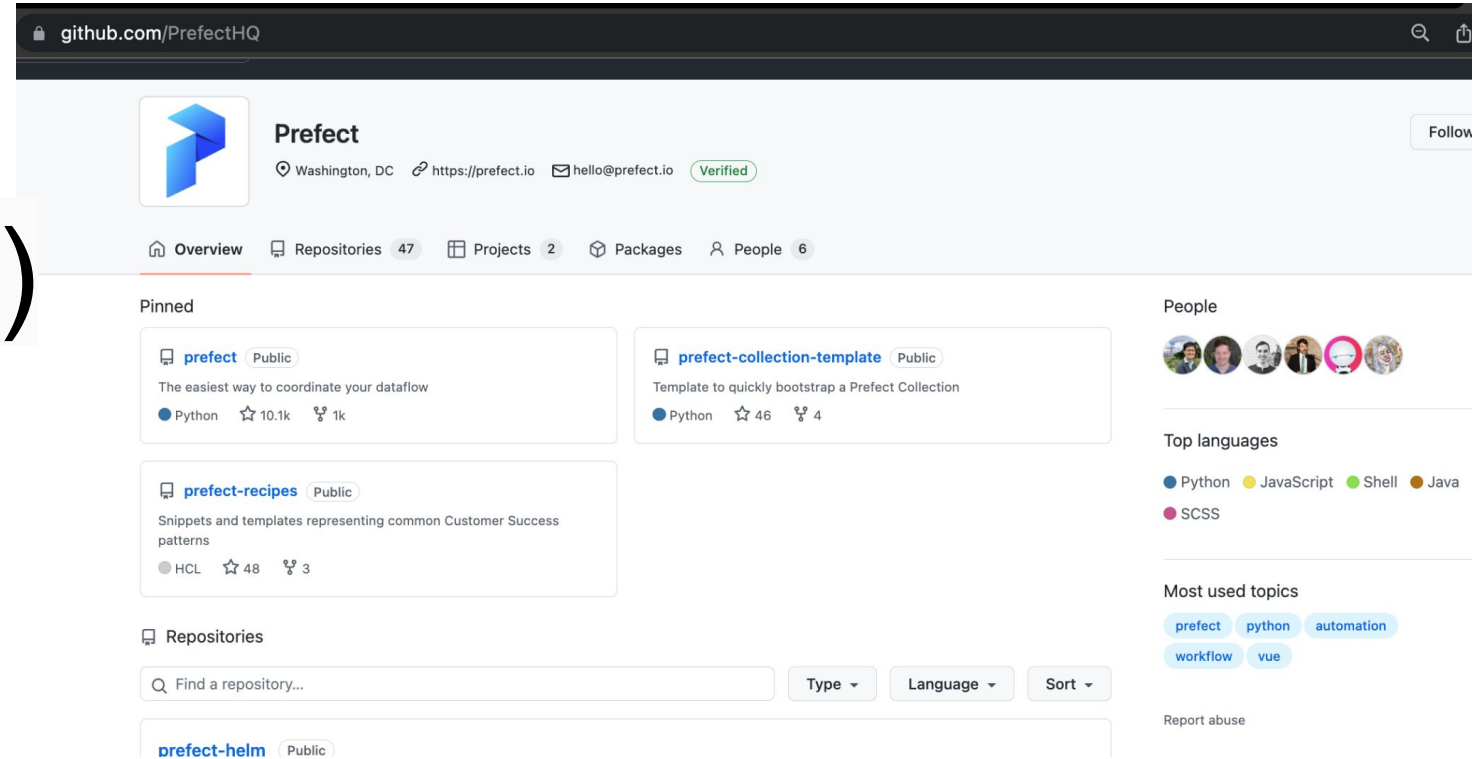
Запущен в 2018

1)



Prefect - open source

2)

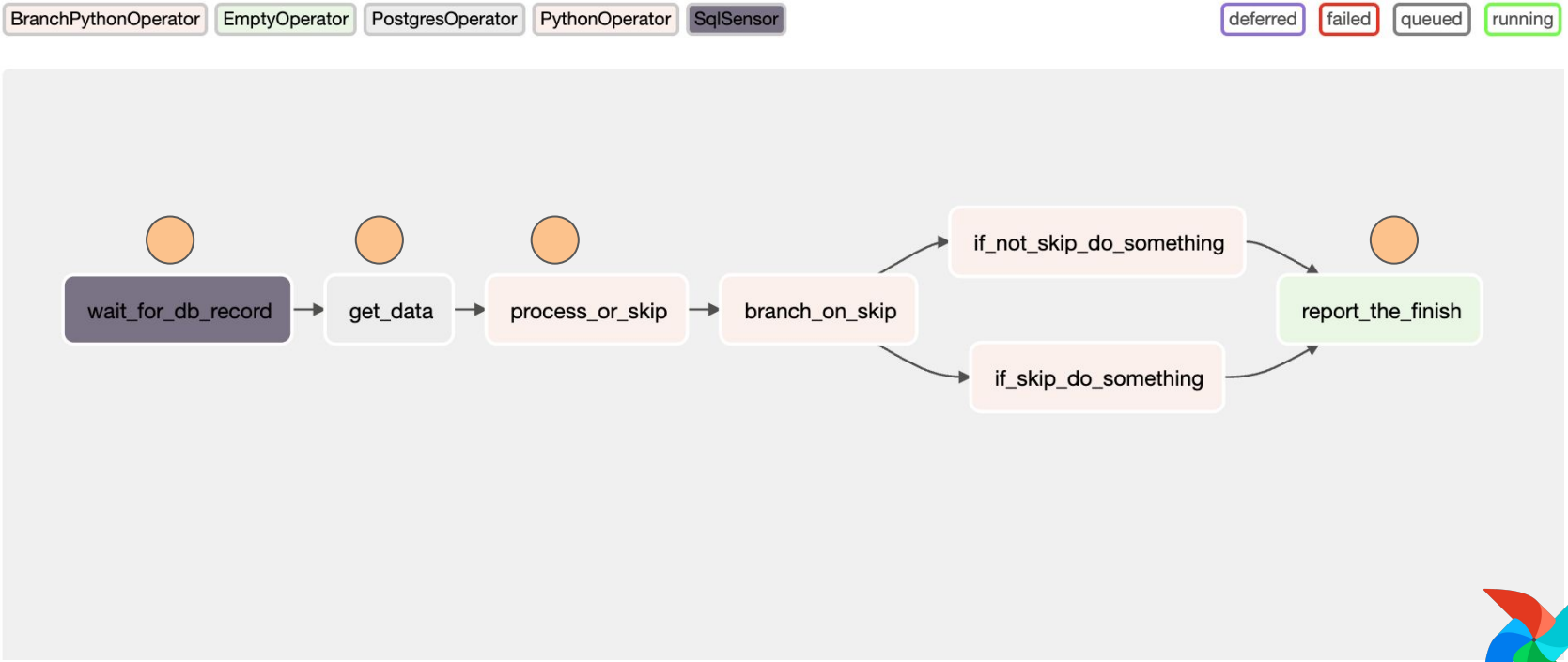


Слайд с претензиями к Apache Airflow

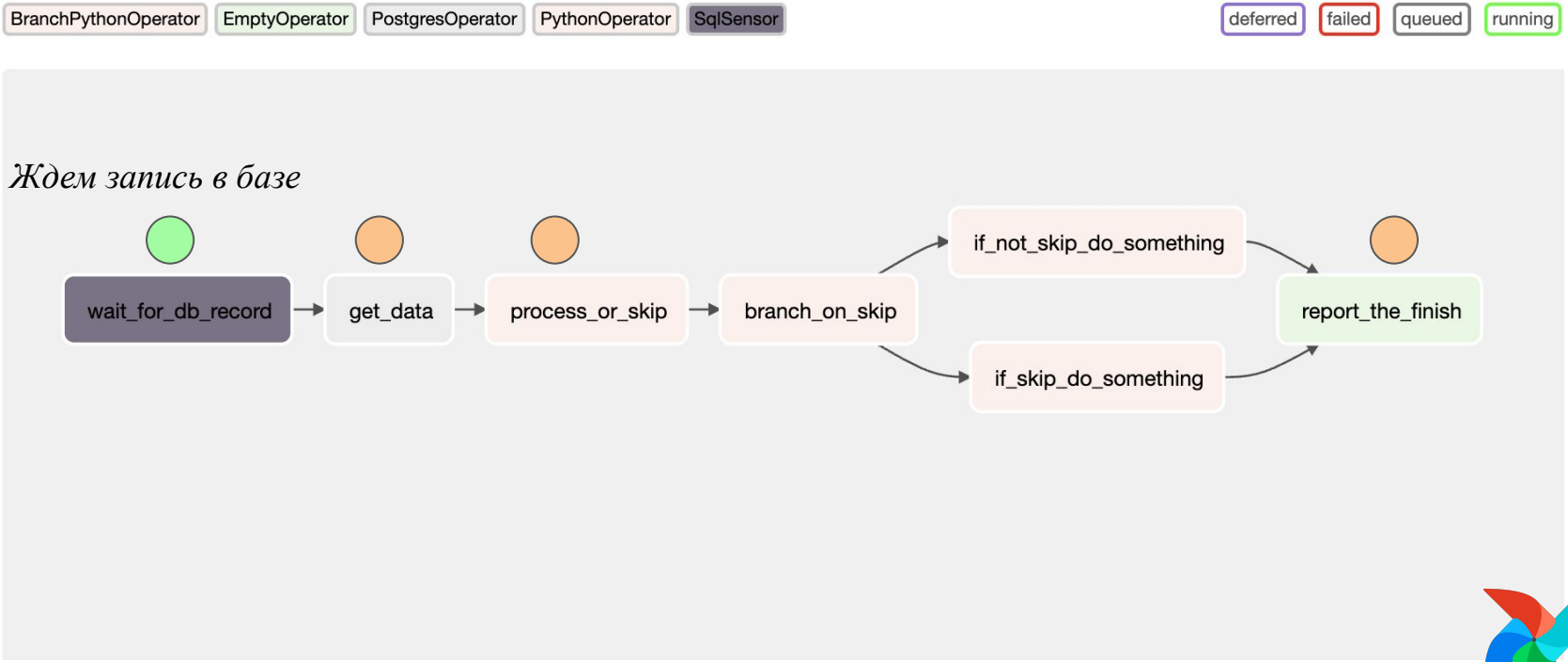
Список претензий к Apache Airflow, так часто менялся/меняется во времени, что я решила сфокусироваться только на нижеприведенных

- Невозможность передавать N-ые объемы данных между задачами (Task),
  потому что XCom пишет в базу
- Невозможность удобно передавать данные между задачами (Task),
  потому что надо делать xcom_pull/push
- ~~Scheduler - плохой, узкое горлышко~~
- ~~SubDags плохие и неудобные~~
- ~~Параметры нормально не передать в DAG~~
- Высокий порог вхождения / много надо сделать для запуска первых DAGs
- Не может удовлетворить запросы/потребности Аналитиков и DS

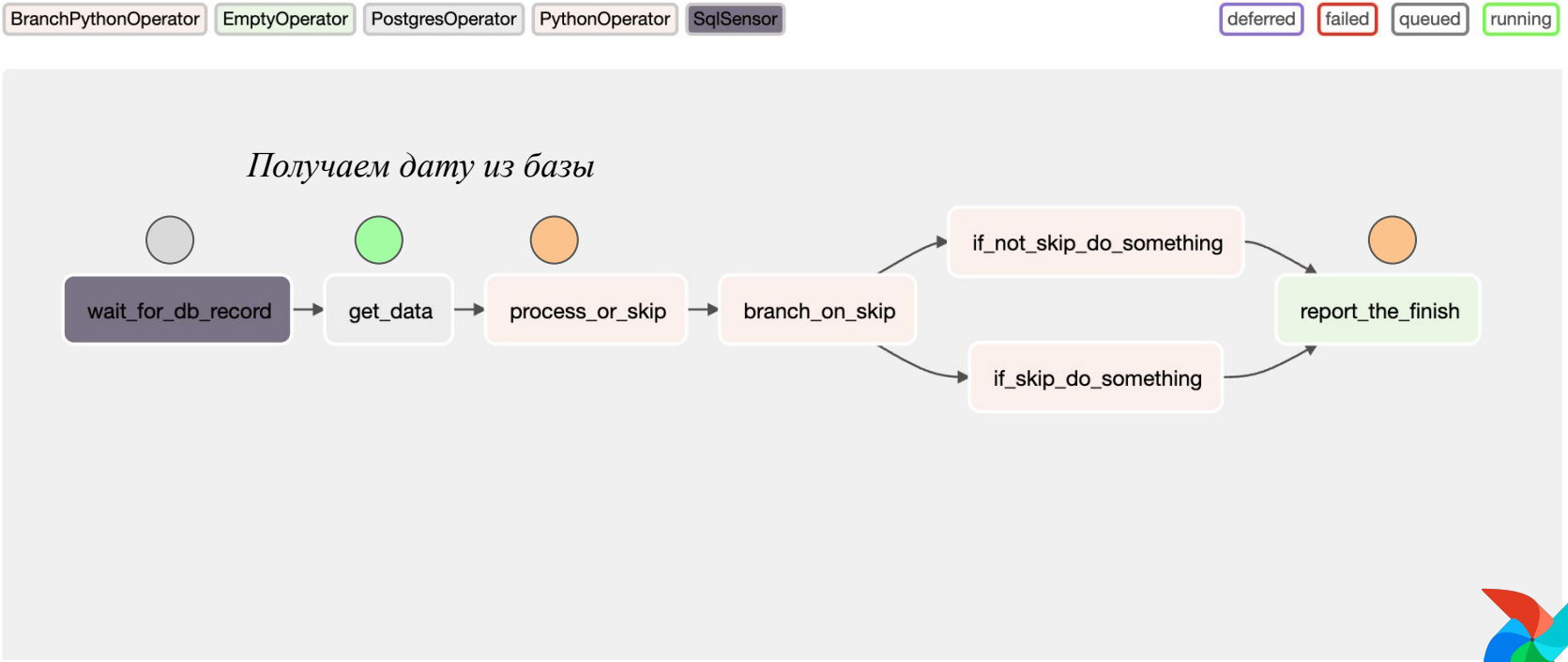
Возьмем DAG


BranchPythonOperator
EmptyOperator
PostgresOperator
PythonOperator
SqlSensor
deferred
failed
queued
running
wait_for_db_record
get_data
process_or_skip
branch_on_skip
if_not_skip_do_something
if_skip_do_something
report_the_finish

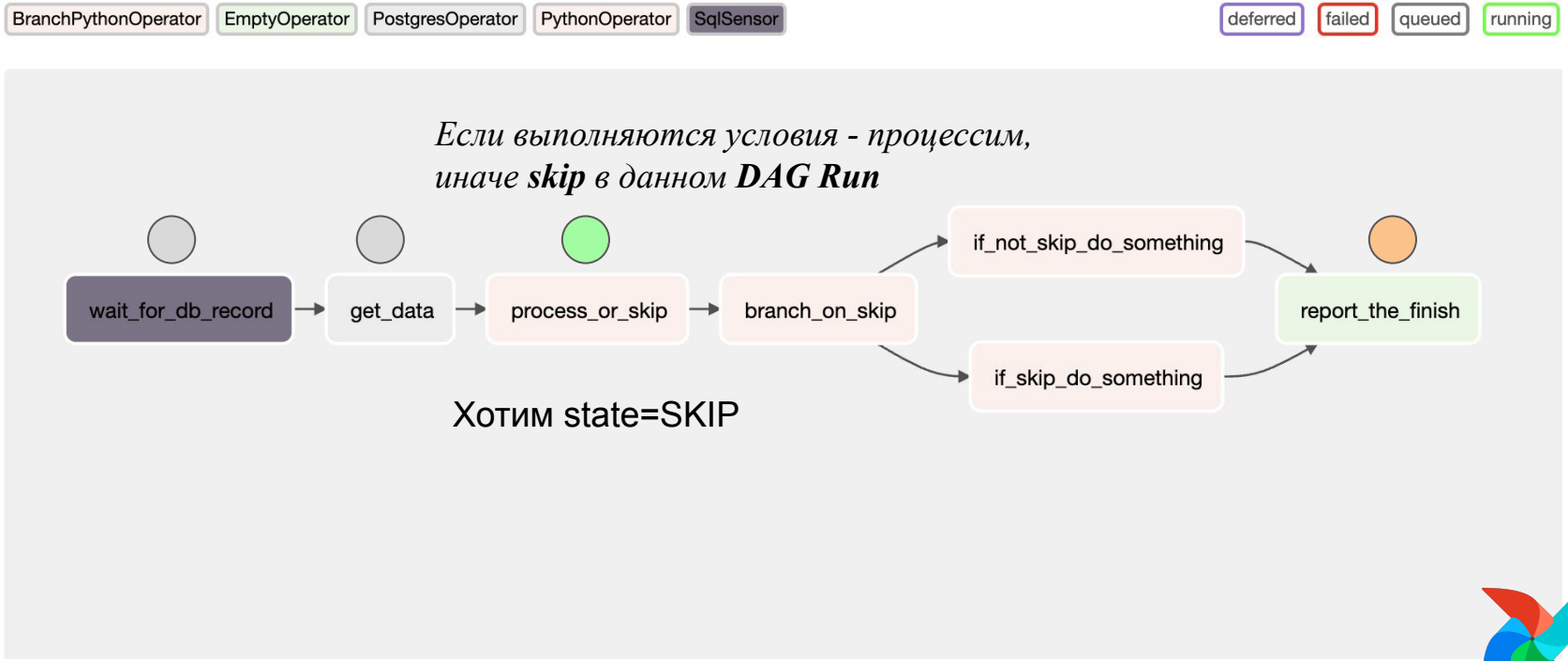
Возьмем DAG


BranchPythonOperator
EmptyOperator
PostgresOperator
PythonOperator
SqlSensor
deferred
failed
queued
running
Ждем запись в базе
wait_for_db_record
get_data
process_or_skip
branch_on_skip
if_not_skip_do_something
if_skip_do_something
report_the_finish

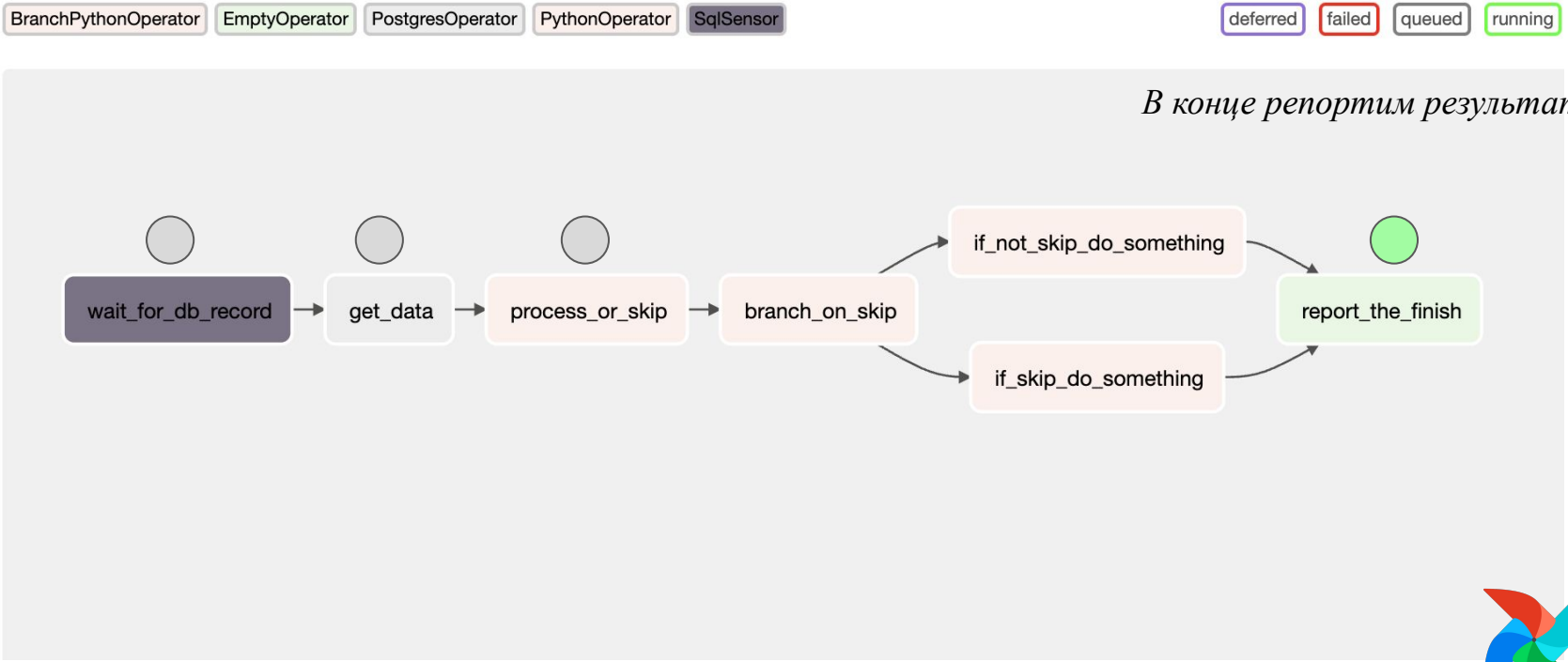
Возьмем DAG


BranchPythonOperator
EmptyOperator
PostgresOperator
PythonOperator
SqlSensor
deferred
failed
queued
running
Получаем дату из базы
wait_for_db_record
get_data
process_or_skip
branch_on_skip
if_not_skip_do_something
if_skip_do_something
report_the_finish

Возьмем DAG


BranchPythonOperator
EmptyOperator
PostgresOperator
PythonOperator
SqlSensor
deferred
failed
queued
running
Если выполняются условия - процессим,
иначе skip в данном DAG Run
wait_for_db_record
get_data
process_or_skip
branch_on_skip
if_not_skip_do_something
if_skip_do_something
report_the_finish
Хотим state=SKIP

Возьмем DAG


BranchPythonOperator
EmptyOperator
PostgresOperator
PythonOperator
SqlSensor
deferred
failed
queued
running
В конце репортим результат
wait_for_db_record
get_data
process_or_skip
branch_on_skip
if_not_skip_do_something
if_skip_do_something
report_the_finish

Давайте из DAG
сделаем Prefect-flow

Как читать слайды

Вот тут ссылки на все файлы с кодом

main_flow_empty.py

Вот так будут выглядить поинты, которые Prefect позиционирует как преимущества перед Airflow

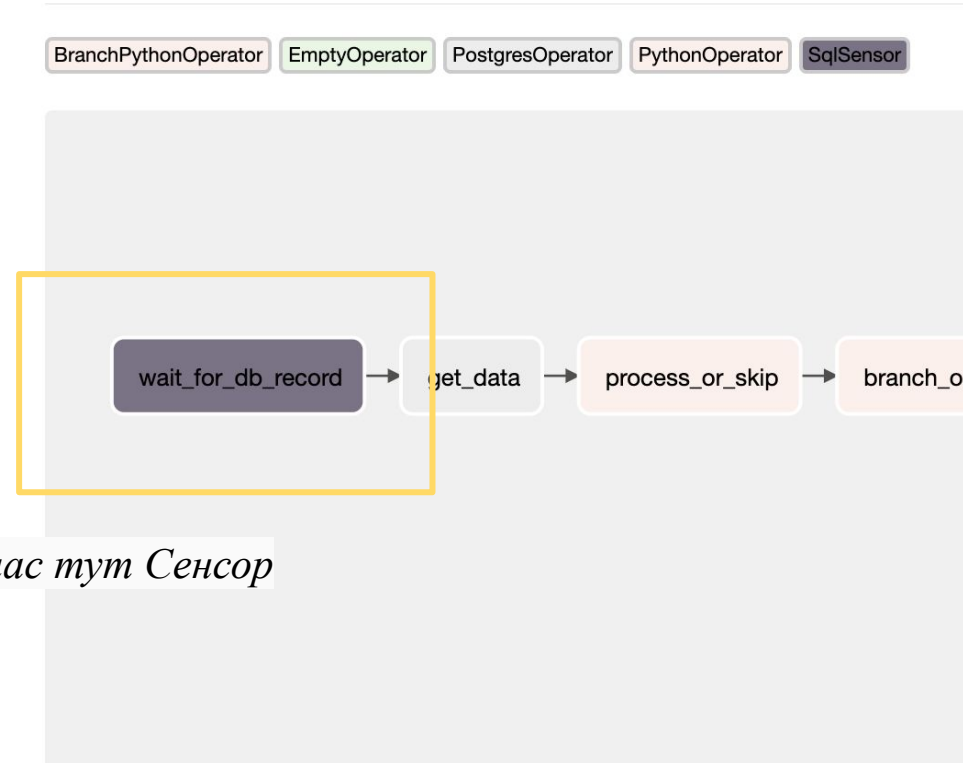
Что-то из Apache Airflow

LIVE DEMO

Слайды вместо которых планируется лайв демо

Начнем со спойлеров!

Присмотримся к DAGу



Господа, у нас тут Сенсор

В Prefect нет Сенсоров

А еще нет никаких Операторов и Хуков - сюрприз

В Prefect нет Сенсоров

Интеграции все-таки есть

LIVE DEMO

Prefect Collections Catalog

Below you can find a list of all available Prefect Collections.

prefect-airbyte

Maintained by Prefect

prefect-aws

Maintained by Prefect

prefect-azure

Maintained by Prefect

Prefect Collections are groupings of pre-built tasks and flows used to quickly build dataflows with Prefect.

https://docs.prefect.io/collections/catalog/

Попробуем найти Collection для общения с базой

https://github.com/PrefectHQ/prefect-postgres

Welcome!

Prefect integrations for interacting with prefect-postgres.

Installation

Install `prefect-postgres` with `pip`:

```
pip install prefect-postgres
```

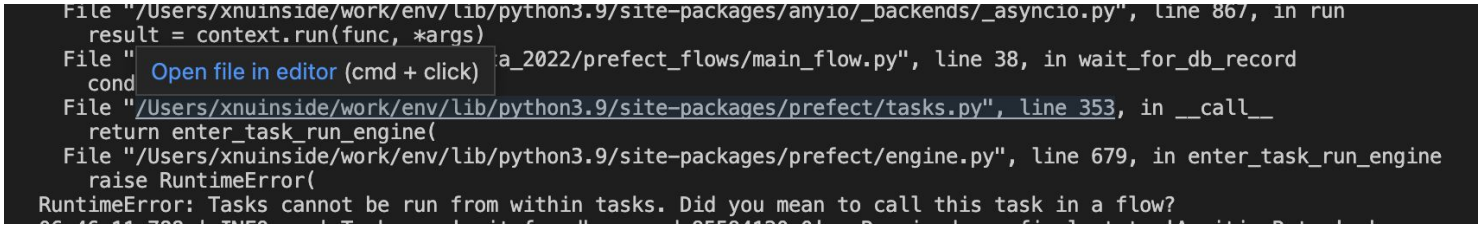
Попробуем найти Collection для общения с базой

```
(env) xnuinside@Iuliias-MacBook-Pro prefect_flows % pip install prefect-postgres
ERROR: Could not find a version that satisfies the requirement prefect-postgres (from versions: none)
ERROR: No matching distribution found for prefect-postgres
WARNING: There was an error checking the latest version of pip.
```

Все нормально - есть SQLAlchemy collection

https://prefecthq.github.io/prefect-sqlalchemy/

И оно тоже не будет работать и будет падать с ошибкой



В общем как обычно - все руками

Пример: https://discourse.prefect.io/t/connection-to-database-in-task/983

Что там еще в DAG

BranchPythonOperator | EmptyOperator | PostgresOperator | PythonOperator | SqlSensc

*Если выполняются условия - процессим, иначе **skip** в данном **DAG Run***

Хотим state=SKIP

И **skip** тоже нет

Есть все остальные +

Crashed	Yes	The run did not complete because of an infrastructure issue.

Late	No	The run's scheduled start time has passed, but it has not transitioned to PENDING (5 seconds by default).

https://docs.prefect.io/concepts/states/

Зададим структуру Flow в Prefect

main_flow_empty.py

```
from prefect import task
```

```
@task
def if_not_skip_do_something_func():
    pass
```

Зададим структуру Flow в Prefect

main_flow_empty.py

```
from prefect import flow
```

```
@flow
def db_data_process_flow():
    wait_for_db_record()
    get_data()
    process_or_skip_func()
```

Зададим структуру Flow в Prefect

main_flow_empty.py

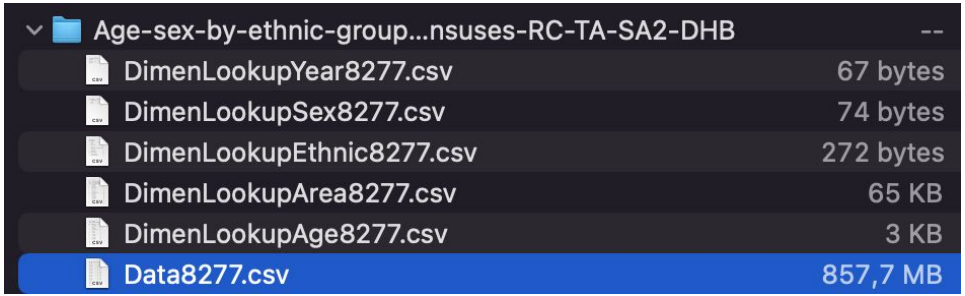
```python
@task
def if_not_skip_do_something_func():
    pass


@task
def report_the_finish():
    pass
```

Передача результатов сразу в таску

```python
@flow
def db_data_process_flow():
    wait_for_db_record()
    data = get_data()
    status = process_or_skip_func(data)
    if status == 'skip':
        val = if_skip_do_something()
        report_the_finish(val)
    else:
        val = if_not_skip_do_something_func()
        report_the_finish(val)
```

Отсутствие необходимости определять порядок тасок

Сколько можно передавать между тасками




```
['Year', 'Age', 'Ethnic', 'Sex', 'Area', 'count']
```

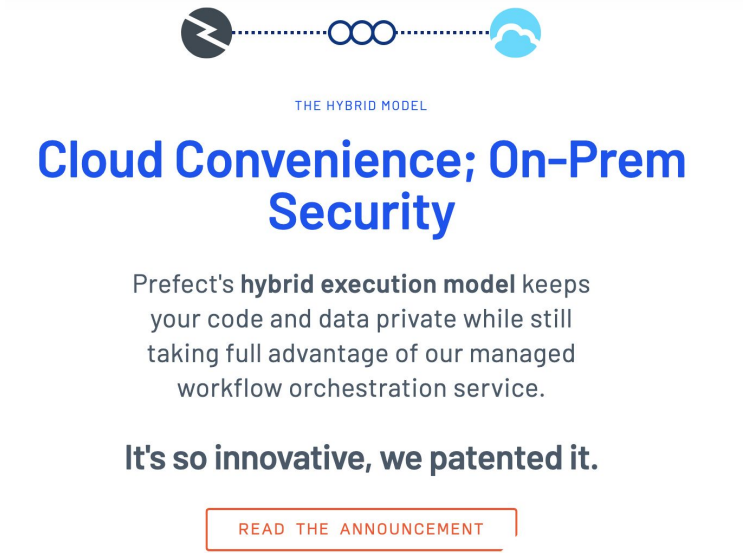
Age and sex by ethnic group (grouped total responses), for census usually resident population counts, 2006, 2013, and 2018 Censuses (RC, TA, SA2, DHB)

Сколько можно передавать между тасками

dataframe_flow.py

```python
@task
def count_wrong_values(df: pd.DataFrame):
    return df.loc[lambda x: x['Age'] > 120]['Age'].count()


@flow
def DataFrame_processing(path_to_csv: str):
    df = read_the_DataFrame(path_to_csv)
    avr_age = get_average_age(df)
    if avr_age > 120:
        # too much immortal peoples
        wrong_rows = count_wrong_values(df)
        get_run_logger().error(f'Wrong rows in the dataset based on the Age column: {wrong_rows}')
    # or do something else
```

Как реализован механизм передачи в Prefect



https://www.prefect.io/why-prefect/hybrid-model/

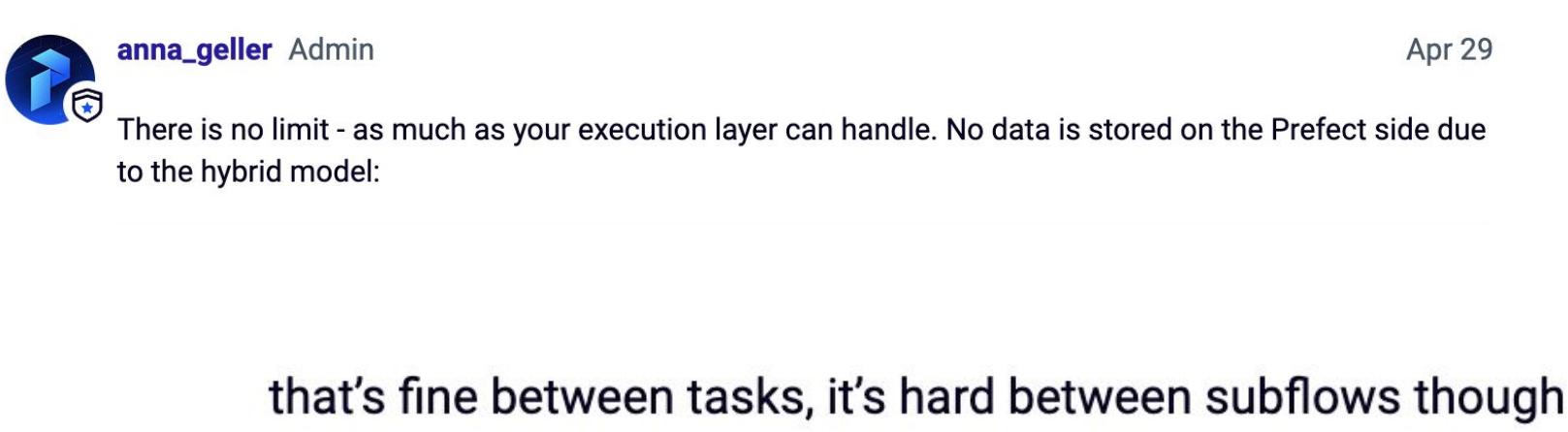
Когда я попыталась почитать статью

It was the winter of 2017, and I was sitting in the basement of one of the largest hedge funds in the world. Its CTO, who is now one of our technical advisors, was gently explaining why he would never use a managed workflow

https://medium.com/the-prefect-blog/the-prefect-hybrid-model-1b70c7fd296


За что ты так со мной?
meme-arsenal.ru

По существу



https://discourse.prefect.io/t/is-there-a-limit-to-how-much-data-can-be-passed-between-tasks-in-prefect/857

Сравнение с Airflow

```python
@task
def if_not_skip_do_something_func():
    pass


@task
def report_the_finish():
    pass
```

*Более Pythonic-way
аля “декораторы и
функции”*

vs

```python
@flow
def db_data_process_flow():
    wait_for_db_record()
    data = get_data()
    status = process_or_skip_func(data)
    if status == 'skip':
        val = if_skip_do_something()
        report_the_finish(val)
    else:
        val = if_not_skip_do_something_func()
        report_the_finish(val)
```

Сравнение с Airflow

```python
@task
def if_not_skip_do_something_func():
    pass


@task
def report_the_finish():
    pass
```

Более Pythonic-way аля “декораторы и функции”

```python
@flow
def db_data_process_flow():
    wait_for_db_record()
    data = get_data()
    status = process_or_skip_func(data)
    if status == 'skip':
        val = if_skip_do_something()
        report_the_finish(val)
    else:
        val = if_not_skip_do_something_func()
        report_the_finish(val)
```

vs

```python
with DAG(dag_id="airflow_to_prefect_dag_dummy",
         start_date=datetime(2019, 11, 28),
         schedule_interval="@daily") as dag:
         wait_for_db_record = DummyOperator(
            task_id = "wait_for_db_record"
         )
         get_data = DummyOperator(
            task_id = "get_data"
            )
         process_or_skip = DummyOperator(
            task_id = "process_or_skip"
         )
         branch_on_skip = DummyOperator(
            task_id = "branch_on_skip"
         )
         if_skip_do_something = DummyOperator(
            task_id = "if_skip_do_something"
         )
         if_not_skip_do_something = DummyOperator(
            task_id = "if_not_skip_do_something"
         )
```

НО в Airflow 2.* есть TaskFlow

```python
@dag(
    schedule=None,
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
    catchup=False,
)
def airflow_to_prefect_dag_task_flow():
    @task()
    def wait_for_db_record():
        pass
    @task()
    def get_data():
        pass
    @task()
    def process_or_skip():
        pass
```

```python
@task
def report_the_finish():
    pass

result = wait_for_db_record()
data = get_data(result)
state = process_or_skip_func(data)
branch_on_skip(state) >> [if_not_skip_do_something_func(),
if_skip_do_something()] >> report_the_finish()
```

В теории, можно перенести TaskFlow на Prefect и обратно практически безболезненно

Бранч

Branch == case в Prefect 1

```
with case(status, 'skip'):
    val = if_skip_do_something()
    report_the_finish(val)

with case(status, 'process'):
    val =if_not_skip_do_something_func()
    report_the_finish(val)
```

Бранч

Branch == case в Prefect 1

```
with case(status, 'skip'):
    val = if_skip_do_something()
    report_the_finish(val)

with case(status, 'process'):
    val =if_not_skip_do_something_func()
    report_the_finish(val)
```

Prefect 2

```
@flow
def db_data_process_flow():
    wait_for_db_record()
    data = get_data()
    status = process_or_skip_func(data)
    if status == 'skip':
        val = if_skip_do_something()
        report_the_finish(val)
    else:
        val = if_not_skip_do_something_func()
        report_the_finish(val)
```

Попробуем запустить Flow

main_flow_empty.py

```
>> python main_flow_empty.py
```

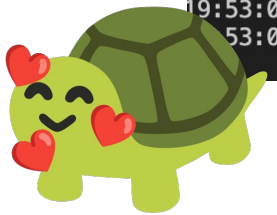
LIVE DEMO

Посмотрели логи после запуска

LIVE DEMO

```
19:53:08.961 | INFO    | Flow run 'inventive-gaur' - Executing 'process_or_ski
iately...
19:53:08.975 | INFO    | Task run 'process_or_skip_func-b4c543e9-0' - Task pro
19:53:08.983 | INFO    | Task run 'process_or_skip_func-b4c543e9-0' - Finished
19:53:08.992 | INFO    | Flow run 'inventive-gaur' - Created task run 'if_not_
cca1307-0' for task 'if_not_skip_do_something_func'
19:53:08.992 | INFO    | Flow run 'inventive-gaur' - Executing 'if_not_skip_do_something_func-4cca1307
-0' immediately...
19:53:09.004 | INFO    | Task run 'if_not_skip_do_something_func-4cca1307-0' - Task if_not_skip_do_som
ething_func
19:53:09.012 | INFO    | Task run 'if_not_skip_do_something_func-4cca1307-0' - Finished in state Compl
eted()
19:53:09.021 | INFO    | Flow run 'inventive-gaur' - Created task run 'report_the_finish-eeb267e8-0' f
or task 'report_the_finish'
19:53:09.021 | INFO    | Flow run 'inventive-gaur' - Executing 'report_the_finish-eeb267e8-0' immediat
ely...
19:53:09.034 | INFO    | Task run 'report_the_finish-eeb267e8-0' - Task report_the_finish
19:53:09.042 | INFO    | Task run 'report_the_finish-eeb267e8-0' - Finished in state Completed()
53:09.052 | INFO    | Flow run 'inventive-gaur' - Finished in state Completed('All states completed
```

Можно просто immed
не поднимая сервер
прогнать полностью ***Flow***



Можно что-нибудь сломать

LIVE DEMO

```python
@task
def if_not_skip_do_something_func():
    get_run_logger().info(f"Task {if_not_skip_do_something_func.__name__}")
    raise
```

Добавим raise - увидим ошибку

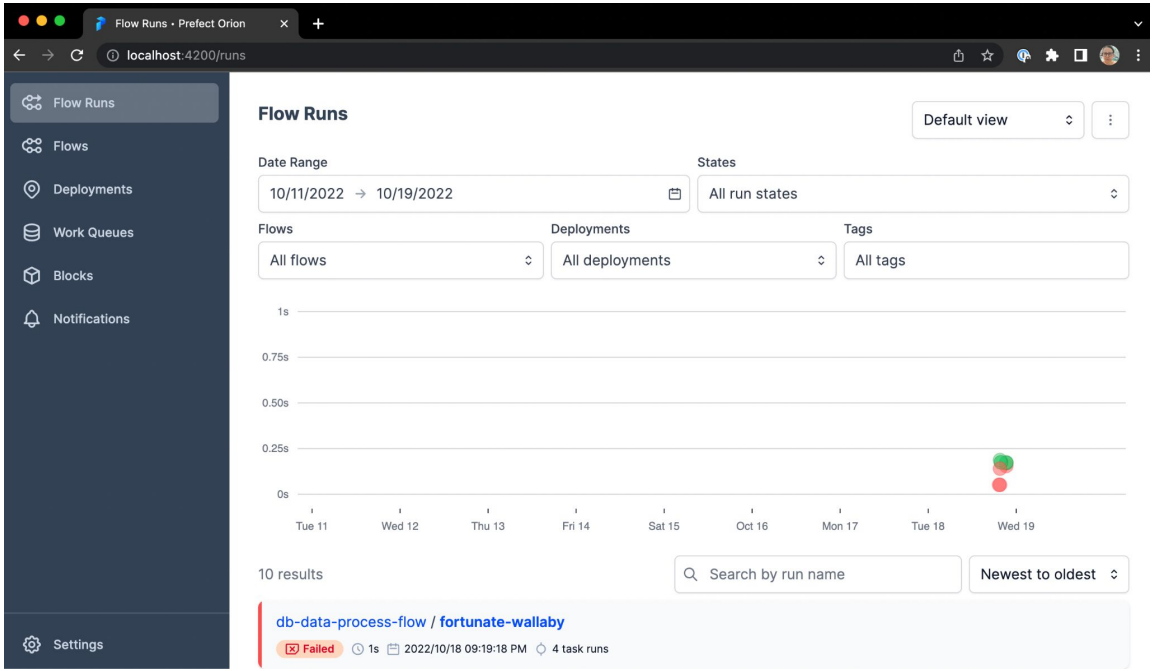
```
  File "/Users/ynvolkova/airflow_in_docker_compose/env/lib/python3.10/site-packages/anyio/_backends/_asyncio.py", line 937, in run_sync_in_worker_thread
    return await future
  File "/Users/ynvolkova/airflow_in_docker_compose/env/lib/python3.10/site-packages/anyio/_backends/_asyncio.py", line 867, in run
    result = context.run(func, *args)
  File "/Users/ynvolkova/smartdata_2022/prefect_flows/main_flow_empty.py", line 26, in if_not_skip_do_something_func
    raise
RuntimeError: No active exception to reraise
```

А где же UI?

LIVE DEMO

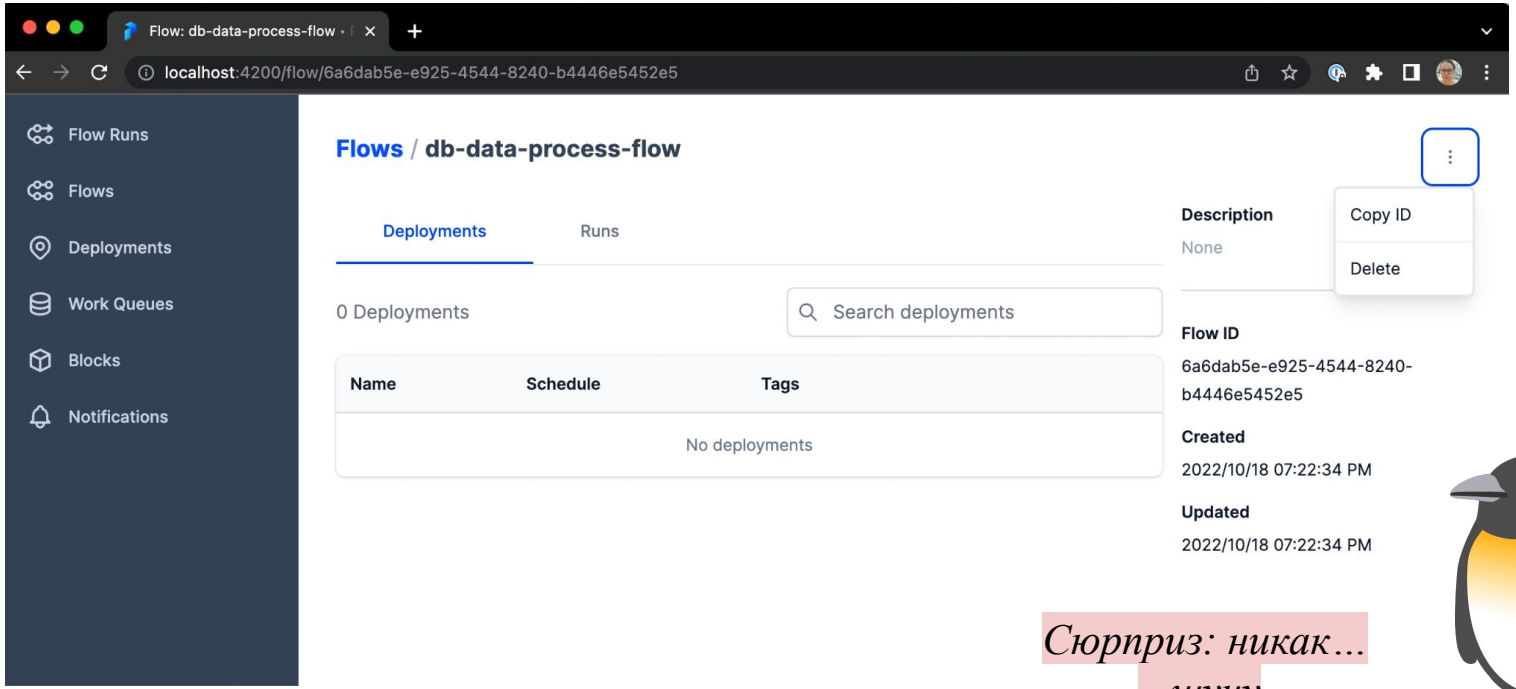
```
>> prefect orion start
```

http://localhost:4200



А как из UI стартануть Flow?

LIVE DEMO



Сюрприз: никак… .. шучу

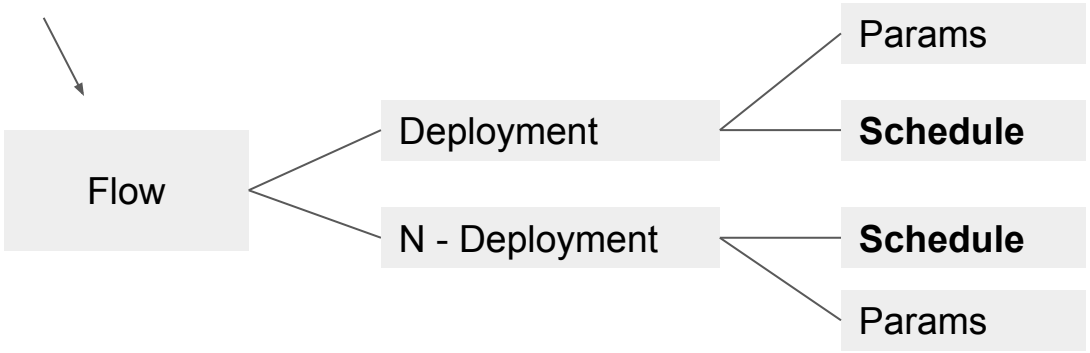
Deployments

1. Flow
2. Deployment

DAG 

Deployments

По факту просто сущность-тег, хранение истории запусков, связки с FlowRun



Deployment

```
...
  "flow_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "schedule": {
    "interval": 0,
    "anchor_date": "2022-09-27T08:06:58.807Z",
    "timezone": "America/New_York"
  },
  "is_schedule_active": true,
  "infra_overrides": {},
  "parameters": {},
  "tags": ["tag-1", "tag-2"],
  "work_queue_name": "string",
  "parameter_openapi_schema": {},
  "path": "string",
  "entrypoint": "string",
  "manifest_path": "string",
  "storage_document_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "infrastructure_document_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
```

Как задеплоить?

LIVE DEMO

Способ 1: простой “как в документации”

Это мы только сделаем build - создастся db_data_process_flow-deployment.yaml

```
>> prefect deployment build
prefect_flows/main_flow_empty.py:db_data_process_flow --name first
```

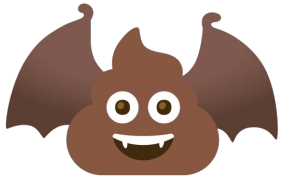
Путь к py-модулю с flow

Функция самого flow

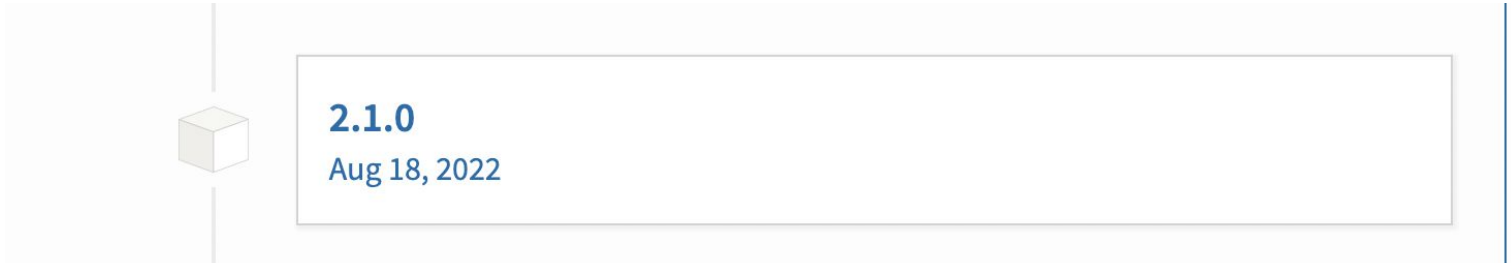
Имя для Deployment (не для Flow)

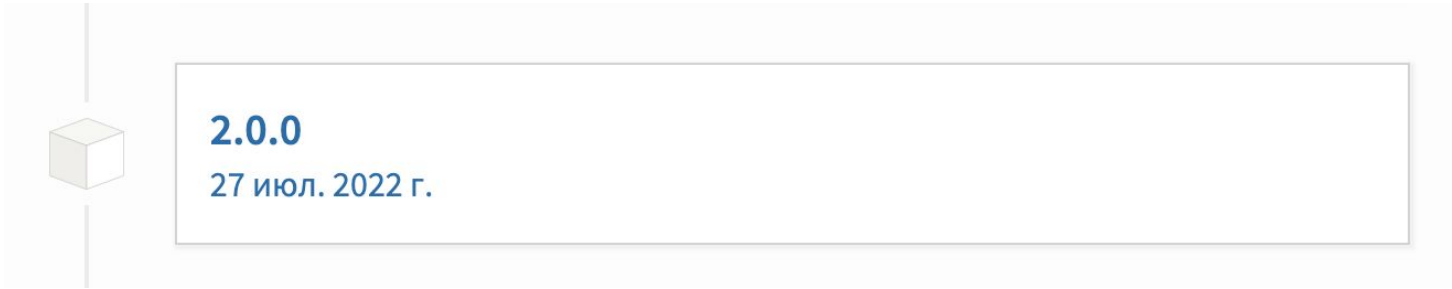
Manifest-файл

*В **2.0.0** ещё был **manifest-файл**,*
*а потом в версии **2.1.0***
уже сделали его необязательным



Г - стабильность





Всё ещё деплоим

LIVE DEMO

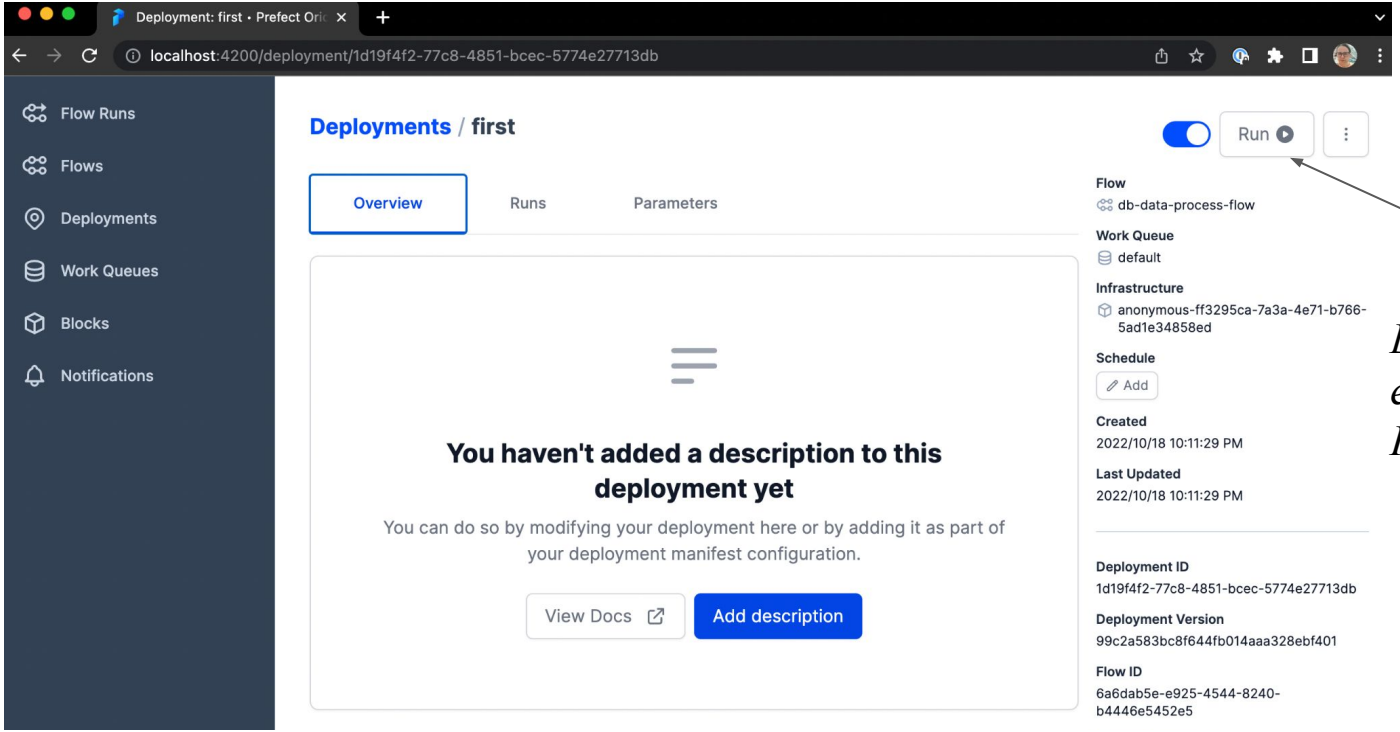
db_data_process_flow-deployment.yaml у нас уже есть

```
>> prefect deployment apply db_data_process_flow-deployment.yaml
```

Имя нашего deployment-файла для flow

Смотрим UI

LIVE DEMO



Вот наша кнопка, но если мы нажмем, то Flow не пройдет

Посмотрим детальнее на Deployment.yaml

Тот самый manifest

```
25  ###
26  flow_name: db-data-process-flow
27  manifest_path: null
28  storage: null
29  path: /Users/ynvolkova/smartdata_2022
```

```
infrastructure:
  type: process
  env: {}
  labels: {}
  name: null
  command: null
  stream_output: true
  block_type_slug: process
```

Посмотрим детальнее на Deployment

```
###
flow_name: db-data-process-flow
manifest_path: null
storage: null
path: /Users/ynvolkova/smartdata_2022
```

```
infrastructure:
  type: process
  env: {}
  labels: {}
  name: null
  command: null
  stream_output: true
block_type_slug: process
```

Под капотом нет волшебного копирования - это именно ***путь до файла с Flow***

Посмотрим детальнее на Deployment

```
###
flow_name: db-data-process-flow
manifest_path: null
storage: null
path: /Users/ynvolkova/smartdata_2022
```

```
infrastructure:
  type: process
  env: {}
  labels: {}
  name: null
  command: null
  stream_output: true
  block_type_slug: process
```

Что это за Infrastructure?
Type: process?

Посмотрим детальнее на Deployment

```
###
flow_name: db-data-process-flow
manifest_path: null
storage: null
path: /Users/ynvolkova/smartdata_2022
```

Block?

```
infrastructure:
  type: process
  env: {}
  labels: {}
  name: null
  command: null
  stream_output: true
  block_type_slug: process
```

Посмотрим детальнее на Deployment

```
25  ###
26  flow_name: db-data-process-flow
27  manifest_path: null
28  storage: null
29  path: /Users/ynvolkova/smartdata_2022
```

Block?

```
infrastructure:
  type: process
  env: {}
  labels: {}
  name: null
  command: null
  stream_output: true
  block_type_slug: process
```

Посмотрим детальнее на Deployment

```
25  ###
26  flow_name: db-data-process-flow
27  manifest_path: null
28  storage: null
29  path: /Users/ynvolkova/smartdata_2022
```

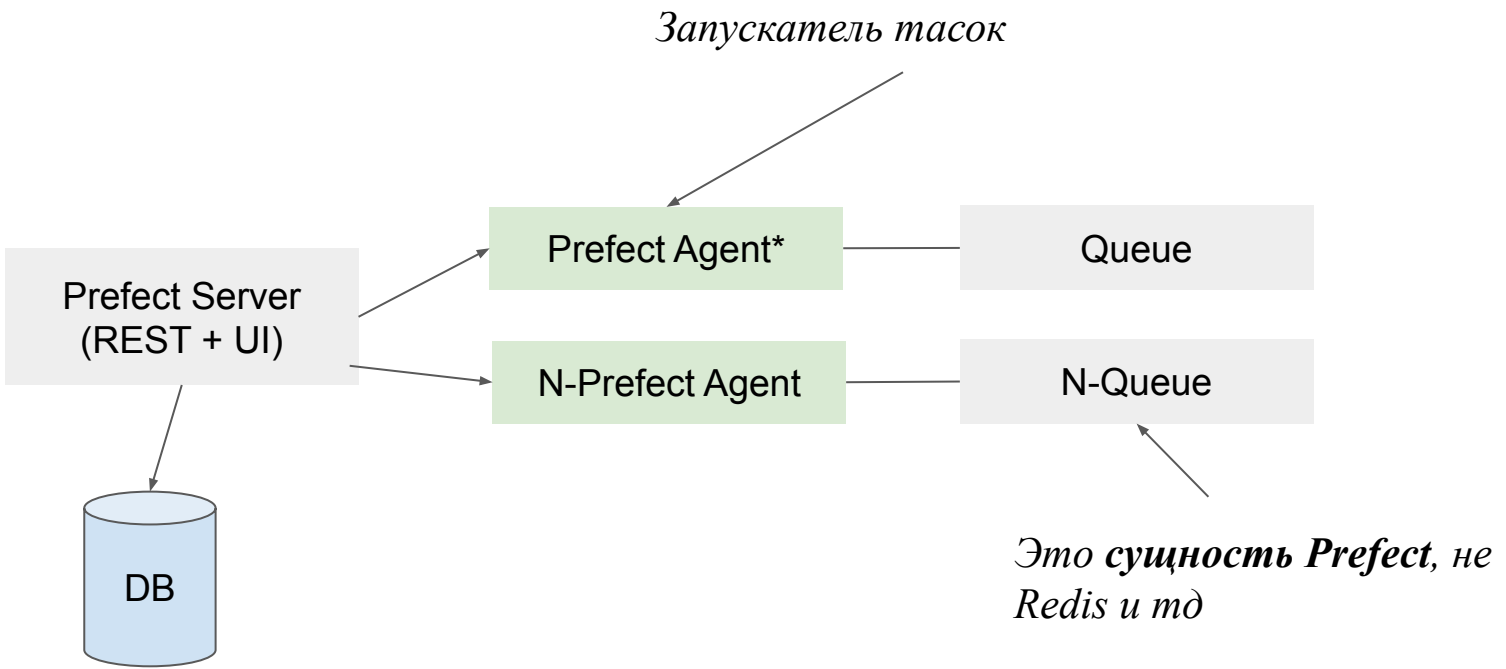
“Низкий порог вхождения” - говорили они

Block?

```
infrastructure:
  type: process
  env: {}
  labels: {}
  name: null
  command: null
  stream_output: true
  block_type_slug: process
```

Компоненты

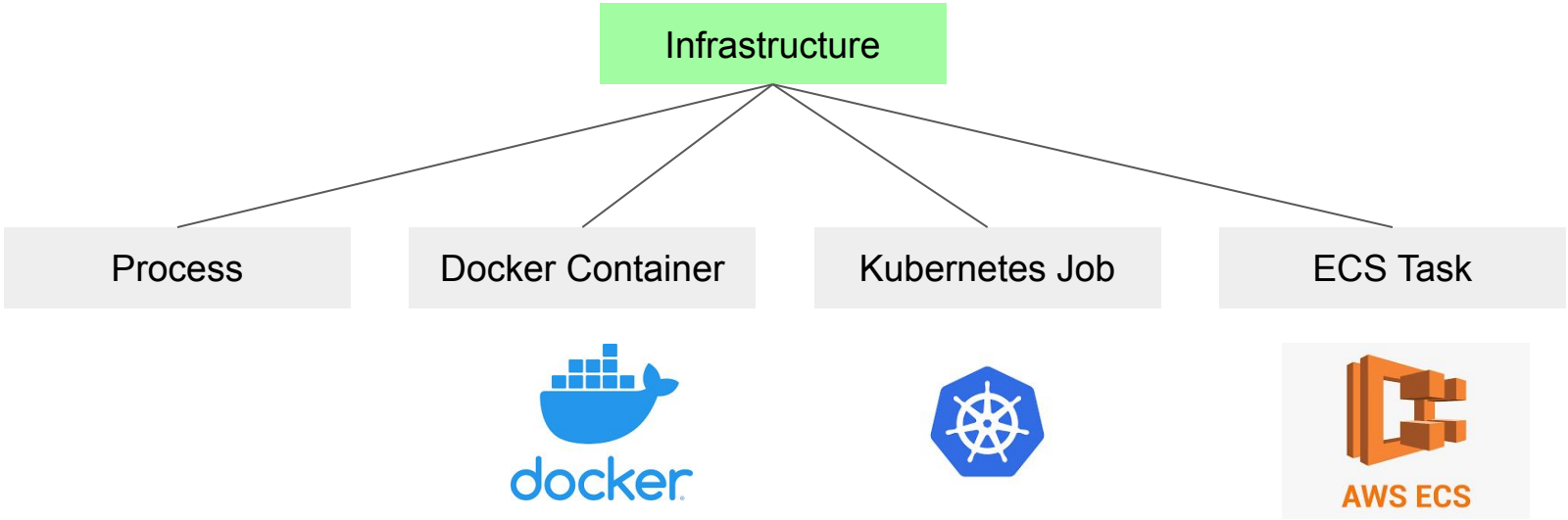
docker-compose-prefect.yml



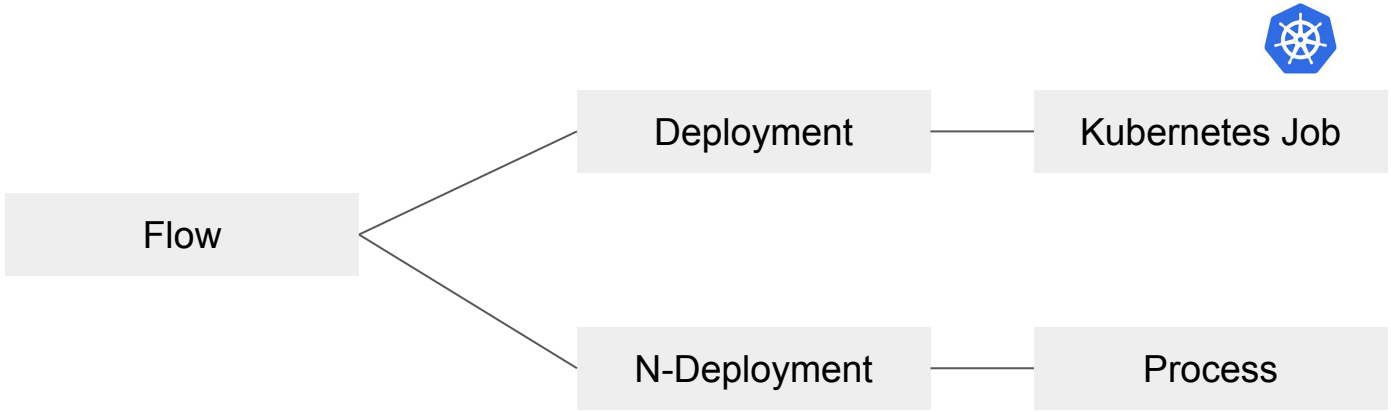
*Несколько агентов могут работать с одной очередью, но 1 агент - 1 очередь

Infrastructure

То, где запускаем flow

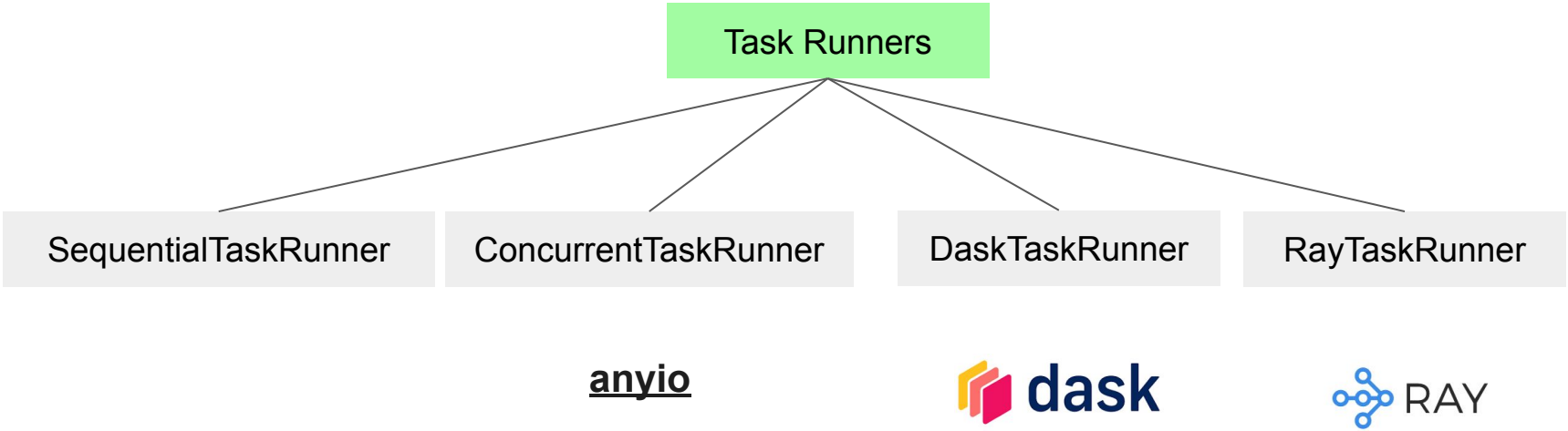


Infrastructure



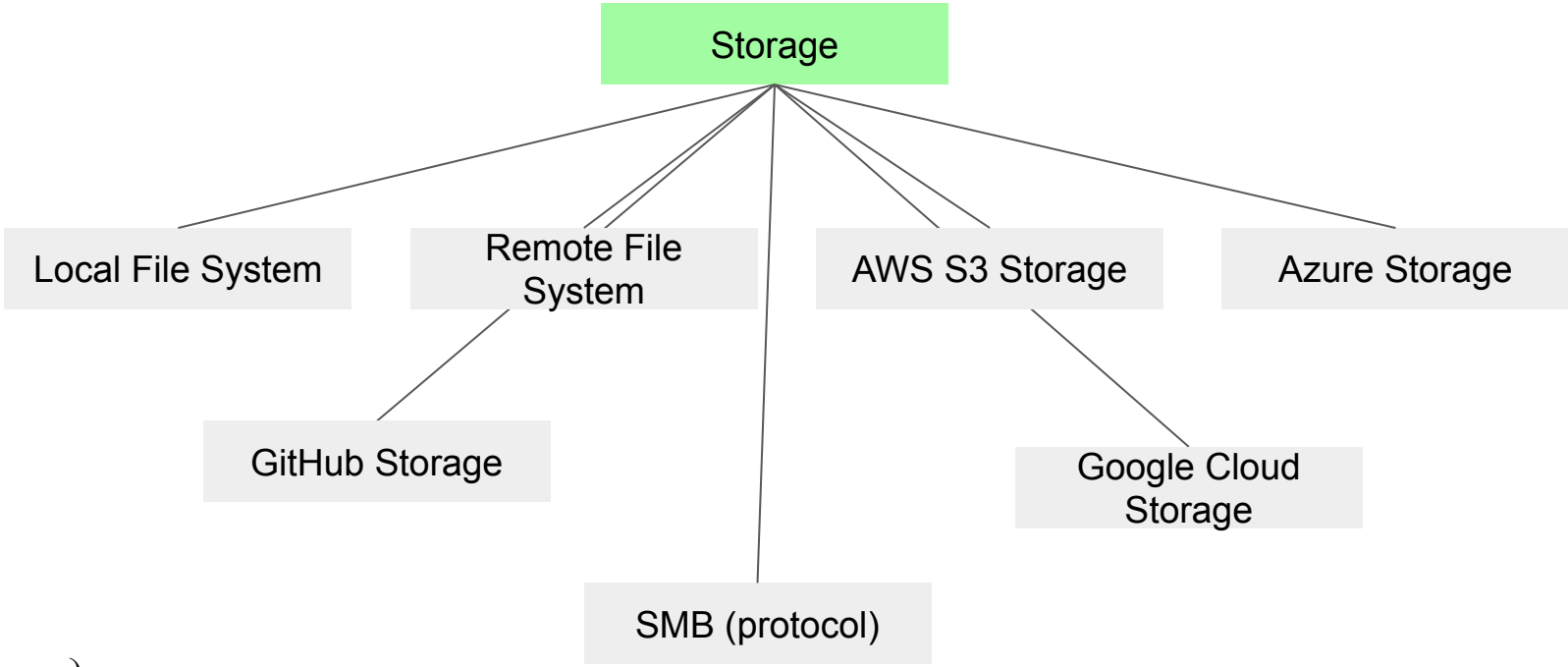
Task Runners

То, как мы запускаем наши таски на выбранной инфраструктуре



Storage

То, где можно хранить Flow



fsspec под капотом

Итого

1. Flow может иметь множество Deployment
2. Deployment определяет то на какой инфраструктуре вы будете запускать Flow
3. TaskRunner определяет как в данном Flow будут запускаться таски вне зависимости от инфраструктуры
4. TaskRunner прописывается в код flow - не в конфиг Deployment

Flow Settings

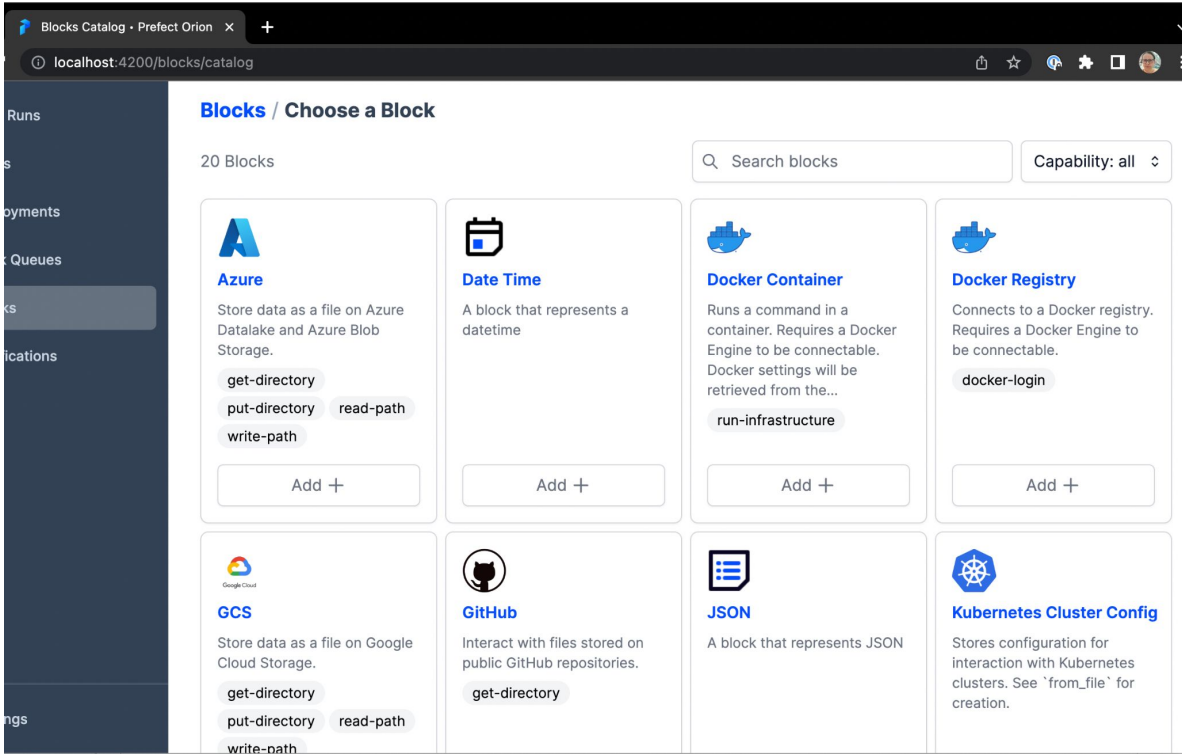
Из интересного - ретраи на Flow

```python
@flow(task_runner=ConcurrentTaskRunner())
def elevator():
    for floor in range(10, 0, -1):
        stop_at_floor.submit(floor)
```

https://docs.prefect.io/concepts/flows/#flow-settings

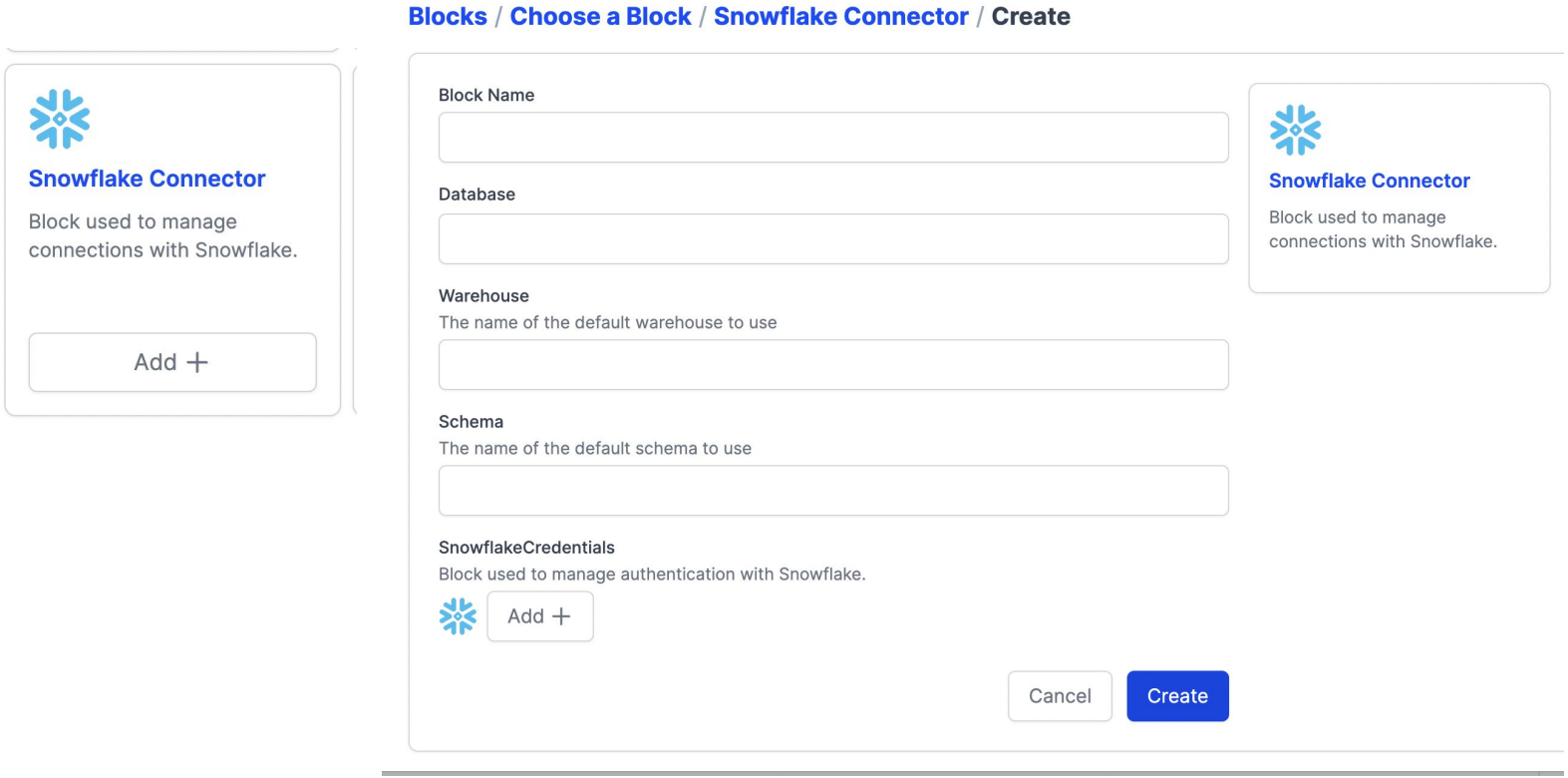
Block

LIVE DEMO



Block ~ Connections в Airflow

LIVE DEMO



Обещанный Сенсор

task_sensor.py

```python
@task(retries=100, retry_delay_seconds=5)
def sensor():
    global run_count
    run_count += 1
    get_run_logger().info(f"Run number {run_count}")
    # here must be your sensor logic:
    #  query to DB, check the file, etc

    condition_met = False
    if not condition_met:
        raise Exception("Oh no! Need to continue wait!")
    return
```

Доп. материалы

1. *Способ 2 для деплоя через REST API можно посмотреть пример кода тут: https://github.com/geobeyond/fastflows/blob/develop/fastflows/core/deployment.py#L22*
2. *Как запускать в Docker-Compose с Minio (локальный s3 для деплоя Flow):https://github.com/rpeden/prefect-docker-compose + настройка и запуска Minio, если ранее не работали с ним https://xnuinside.medium.com/object-storage-minio-quick-start-for-newbies-ccdabf427e7*
3. *Prefect на Kube cluster (тестируется локально с minikube конфиг и readme.md там же в папке, единственное, что там версии более старые): https://github.com/geobeyond/fastflows/tree/develop/docker/kube-infra*

Примеры нормальных use cases для Prefect

1. Процессинг csv, excel и тд, когда 1 запуск - 1 Flow Run - 1 файлик
2. Подготовка изображений - флоу где надо сделать какой-то набор шагов с каждым img
3. И тд per file operations (но речь не про avro на 15 гигов)
4. Flow с забором данных из разных API и агрегацией их в 1 выходном документе (есть поддержка async)

Async в Prefect

```python
import asyncio

from prefect import task, flow

@task
async def print_values(values):
    for value in values:
        await asyncio.sleep(1) # yield
        print(value, end=" ")

@flow
async def async_flow():
    await print_values([1, 2])  # runs immediately
    coros = [print_values("abcd"), print_values("6789")]

    # asynchronously gather the tasks
    await asyncio.gather(*coros)

asyncio.run(async_flow())
```

https://docs.prefect.io/tutorials/execution/#asynchronous-execution

Если все таки хочется Сову на глобус

1. Flow & Deployment - это DAG (в некотором смысле)
2. Вместо Sensors - таски с ретраями
3. Hooks, Operators - готовые tasks в Collections
4. Вместо Executors - Infrastructure
5. Agent вместо Scheduler? (не совсем верное сравнение)
6. Blocks вместо Connections
7. Storage - как место хранения вашего Deployment
8. Skip как статуса таски нет
9. Branch видны не будут

Выводы

1. Не надо корову называть лошадью и пытаться доказать, что она лучше чем лошадь потому, что дает молоко

2. На самом деле Prefect может жить рядом с Apache Airflow и запускаться из Apache Airflow

3. Вероятно, чтобы сделать счастливыми юзеров надо будет написать “ручки” к БД, внутренним сервисам и тд

Всем спасибо

https://github.com/xnuinside/smartdata_2022