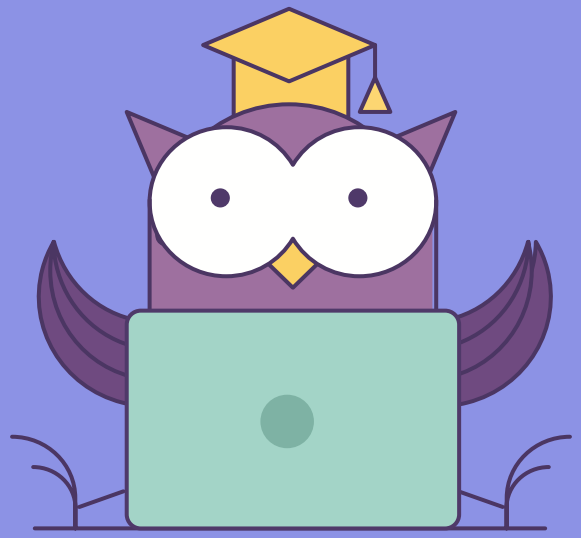
ОНЛАЙН-ОБРАЗОВАНИЕ

Меня хорошо слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

OTUS

Оркестраторы

Oozie & Airflow

Егор Матешук

Цели сегодняшнего занятия

OTUS

После этого занятия вы будете знать

- Какие задачи решают оркестраторы
- Что такое Oozie и Airflow

После этого занятия вы сможете

- Написать свой пайплайн в Airflow

01

Когда cron’а недостаточно

Что такое CRON и для чего он нужен?

cron — классический демон (компьютерная программа в системах класса UNIX), использующийся для периодического выполнения заданий в определённое время. Регулярные действия описываются инструкциями, помещенными в файлы crontab и в специальные каталоги.

Таким образом, cron помогает

- автоматизировать рутинные задачи
- выполнять задачи периодически без ручных запусков

Когда CRON’а недостаточно

OTUS

Проблемы применения cron’а в больших проектах

- трудно мониторить
- непонятно, где лежит конфигурация
- нет перезапусков
- нельзя задавать связи между задачами

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

Чем полезны оркестраторы

- Визуальное представление задач с управлением через GUI
- Единая система для всех задач во всех сервисах на всех серверах
- Обеспечение зависимостей между задачами
- Перезапуск задач со сложной логикой
- Алертинг

Какие есть оркестраторы

- Apache Oozie
- Apache Airflow
- Luigi
- Azkaban

02

Обзор Apache Oozie

Что такое Apache Oozie?

- Оркестратор, рассчитанный на работу с задачами кластера Hadoop
- Может выполнять задачи на основе
  - Spark
  - Hive
  - Pig
  - Sqoop
  - Shell

Что такое Apache Oozie?

- Язык описания задач - XML
- Есть несколько верхнеуровневых объектов
  - **Workflow** - граф задач
  - **Coordinator** - Workflow + расписание
  - **Bundle** - набор первых и вторых

Пример Workflow

```
<workflow-app xmlns = "uri:oozie:workflow:0.4" name = "test-workflow">
  <start to = "run_hive_script" />
  <action name = "run_hive_script">
    <hive xmlns = "uri:oozie:hive-action:0.4">
      <job-tracker>xyz.com:8088</job-tracker>
      <name-node>hdfs://rootname</name-node>
      <script>hdfs_path_of_script/external.hive</script>
    </hive>
    <ok to = "end" />
    <error to = "kill_job" />
  </action>
  <kill name = "kill_job">
    <message>Job failed</message>
  </kill>
  <end name = "end" />
</workflow-app>
```

Запуск Workflow

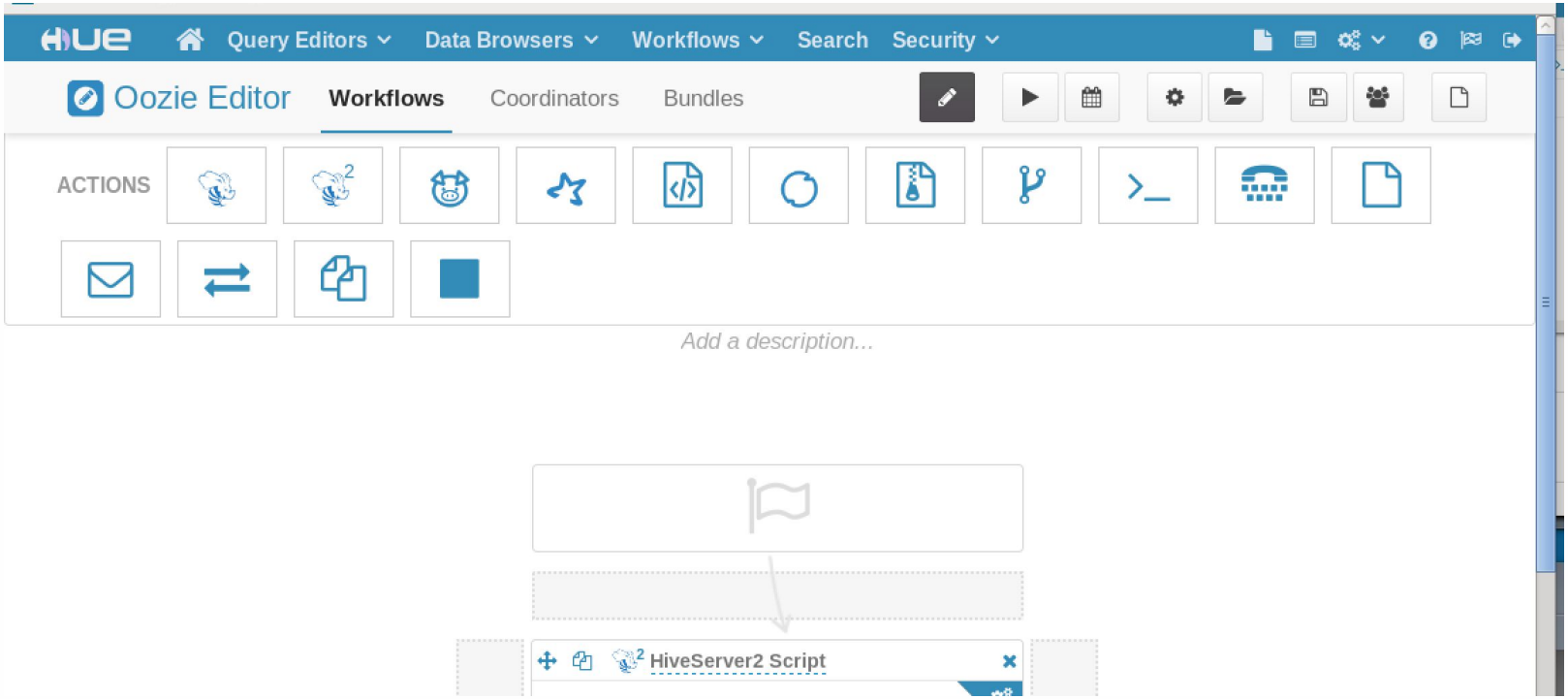
Запуск wf из командной строки, где *host_name:8080* - адрес веб-морды oozie

```
oozie job --oozie http://host_name:8080/oozie -D

oozie.wf.application.path=hdfs:///path/to/workflow.xml
```

Как создавать задачи?

- В реальном мире чаще всего не XML, а HUE



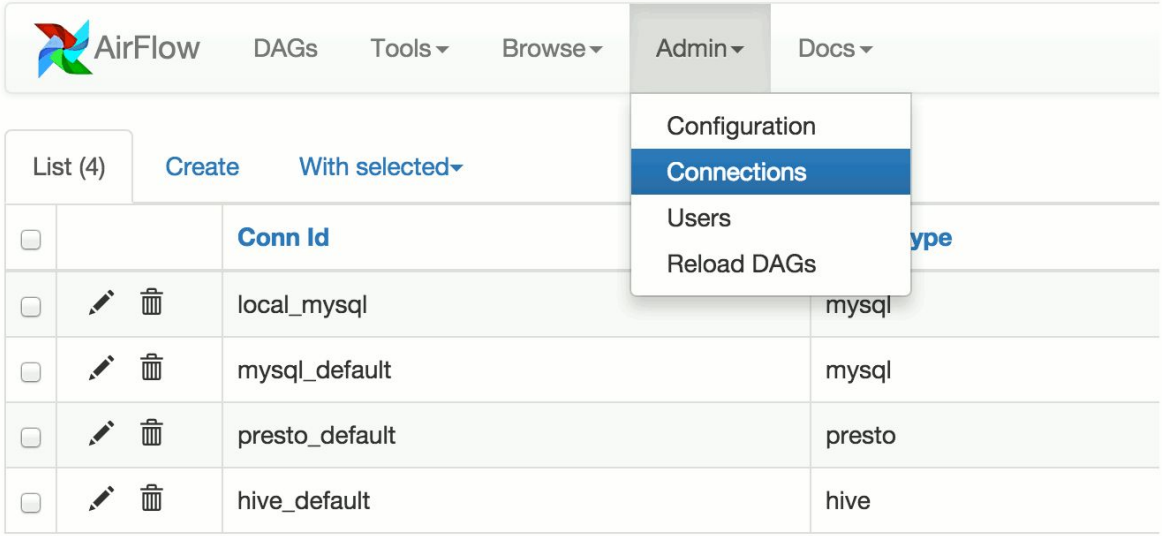
03

Обзор Apache Airflow

Что такое Apache Airflow?

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

Изначально был разработан в AirBNB.



Чем хорош Airflow?

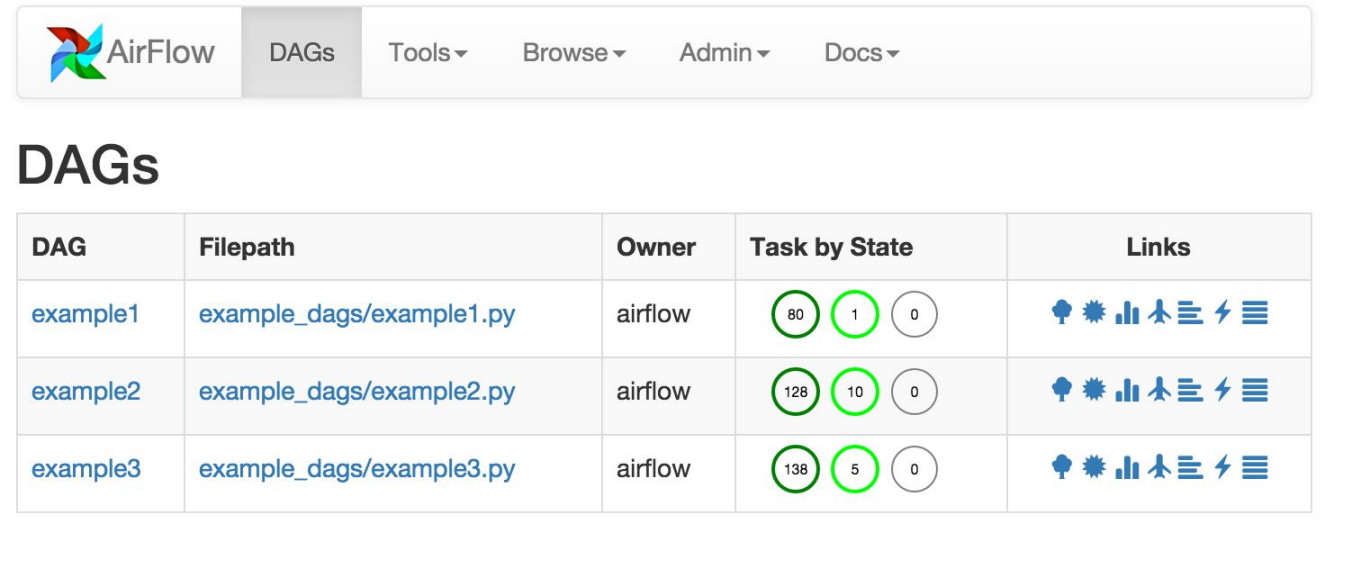
- задачи описываются на Python
- поддерживает сложные процессы из нескольких задач с зависимостями, ветвлением, условиями
- удобный интерфейс
- может масштабироваться с использованием Celery
- легко расширяется под конкретные задачи

Основные понятия

- DAG - граф вычислений
- Operator - тип задачи
- Task - задача, граф состоит из задач
- DAG run - запуск графа для конкретного периода
- Task instance - запуск задачи для конкретного периода

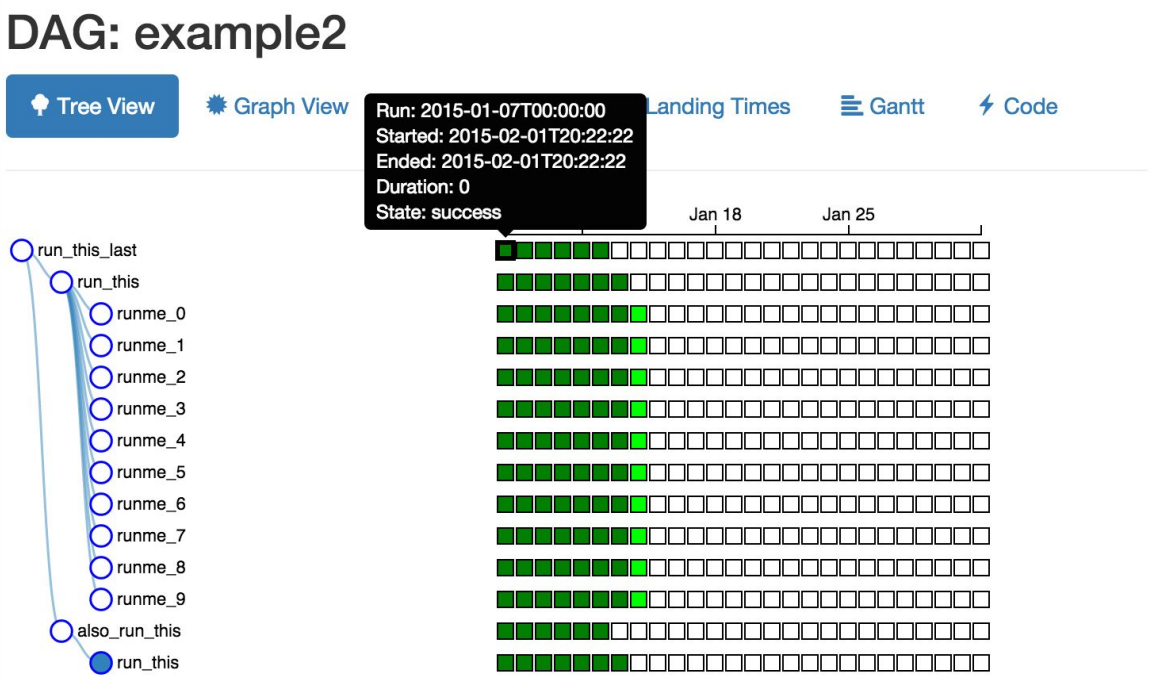
Что такое DAG?

В Airflow задачи собираются в длинные цепочки, называемые DAG.



Что такое DAG?

DAG (Directed Acyclic Graph) – это набор задач, которые вы хотите запускать, организованный таким образом, чтобы отражать их связи и зависимости.



Пример кода DAG

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime, timedelta


default_args = {
    # ...
}

dag = DAG('tutorial', default_args=default_args,
schedule_interval=timedelta(days=1))

t1 = BashOperator(task_id='print_date', bash_command='date', dag=dag)

t2 = BashOperator(
    task_id='sleep',
    bash_command='sleep 5',
    retries=3,
    dag=dag)

t1 >> t2
```

Default Arguments

```
default_args = {
    'start_date': datetime(2016, 1, 1),
    'owner': 'Airflow'
}

dag = DAG('my_dag', default_args=default_args)
op = DummyOperator(task_id='dummy', dag=dag)
print(op.owner) # Airflow
```

Способы запуска DAG

- По расписанию
- Через веб-интерфейс
- Через API
- По сенсору

Примеры операторов

- BashOperator
- PythonOperator
- SimpleHttpOperator
- PostgresOperator
- HiveOperator
- DockerOperator
- SSHOperator
- SlackAPIOperator

Sensors

Сенсоры позволяют выполнять задачи по наступлению определенного события:

- Появление файла на FTP
- Выполнение SQL-запроса
- Успешный ответ по HTTP

и т. д.

Operator chaining

OTUS

Одна из главных полезных фишек - зависимость между задачами внутри DAG

```
dag = DAG('my_dag', start_date=datetime(2016, 1, 1))

op1 = DummyOperator(task_id='op1', dag=dag)

op2 = DummyOperator(task_id='op2', dag=dag)
op1.set_upstream(op2)
```

или

```
op1 >> op2
```

Hooks и operators выглядят похожими объектами, но

- hook - отвечает за техническую часть (запрос к базе, выполнение команды, установка соединения)
- operator - за бизнес-логику (использование результатов запроса, составление команды)

Операторы используют хуки под капотом. Один оператор может использовать несколько хуков.

Пример

Пулы позволяют ограничить число исполняющихся задач одного типа. Где это может быть полезно:

- Использование общего ресурса
- Препятствие “накоплению” задач
- Управление приоритетами

Connections

Этот механизм позволяет безопасно хранить и переиспользовать общие настройки для подключения к сервисам (Main > Admin > Connections)

```
task1 = SSHOperator(
    task_id="ssh1",
    command="echo -n \"airflow works!\"",
    ssh_conn_id="ssh_connection1",
    dag=dag)
```

Variables

OTUS

На уровне сервиса есть возможность задавать “глобальные” переменные (Admin -> Variables)

```
from airflow.models import Variable

foo = Variable.get("foo")

bar = Variable.get("bar", deserialize_json=True)

baz = Variable.get("baz", default_var=None)
```

Для общения между тасками в рамках одного DAG run есть механизм

```
# inside a PythonOperator called 'pushing_task'
def push_function():
    return value


# inside another PythonOperator where provide_context=True
def pull_function(**context):
    value =
context['task_instance'].xcom_pull(task_ids='pushing_task')
```

Backfill

Для заполнения “истории” есть специальная команда **backfill**, которая вызывается консольной командой

```
airflow backfill -s 2019-08-01 -e 2019-08-10 example_dag_id
```

Её можно использовать для того, чтобы очистить и запустить заново выполненные таски

04

Подведем итоги

Цели сегодняшнего занятия

После этого занятия вы будете знать

- Какие задачи решают оркестраторы
- Что такое Oozie и Airflow

После этого занятия вы сможете

- Написать свой пайплайн в Airflow

Заполните, пожалуйста, опрос в личном кабинете!

Полезные ссылки

- Документация Airflow
- Docker-образ для быстрого запуска
- Сравнение оркестраторов

Егор Матешук

egor@mateshuk.com

Спасибо
за внимание!