

КУРС «Практикум по ETL»

Лекция 6. Создание шаблонов задач с использованием контекста Airflow.

Создание шаблонов задач с использованием контекста Airflow.



Темы

- Отображение с помощью шаблонов переменных во время выполнения рабочего процесса.
- Реализация шаблонов переменных с помощью PythonOperator в сравнении с другими операторами.
- Формирование шаблонных переменных во время отладки.
- Выполнение операций над внешними системами(хранилищами).

Бизнес-кейс «StockSense»

https://github.com/BosenkoTM/workshop-on-ETL/tree/main/business_case_stocksense#%D0%B1%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BA%D0%B5%D0%B9%D1%81-stocksense

Проверка данных для обработки с помощью Airflow



- Рассмотрим инструмент прогнозирования фондового рынка, применяющего анализ настроений, который назовем **StockSense**.
- Применим аксиому о том, что увеличение количества просмотров страниц компании указывает на позитивные настроения, и акции компании, вероятно, вырастут.
- С другой стороны, уменьшение количества просмотров страниц говорит нам о потере интереса, и цена акций, скорее всего, снизится.

Определение способа загрузки дополнительных данных



- Фонд Викимедиа (организация, стоящая за Википедией) предоставляет все просмотры страниц с 2015 года в машиночитаемом формате.
- Просмотры страниц можно загрузить в формате gzip.
- Размер каждого почасового дампа составляет около 50 МБ в сжатых текстовых файлах и где-то от 200 до 250 МБ в разархивированном виде.
- Загрузим один часовой дамп и проверим данные вручную. Чтобы разработать конвейер данных, мы должны понимать, как его поэтапно загружать и как работать с данными

Скачивание и проверка данных о просмотрах страниц «Викимедиа»

❶ Формат URL-адреса «Викимедиа» следует этой структуре

```
https://dumps.wikimedia.org/other/pageviews/{year}/  
{year}-{month}/pageviews-{year}{month}{day}-{hour}0000.gz
```

❷ Дата и время в имени файла относятся к концу периода, например 210000 относится к 20:00:00–21:00:00

```
$ wget https://dumps.wikimedia.org/other/pageviews/  
2019/2019-07/pageviews-20190701-010000.gz  
$ gunzip pageviews-20190701-010000.gz  
$ head pageviews-20190701-010000
```

```
aa Main_Page 1 0  
aa Special:GlobalUsers/sysadmin 1 0  
aa User_talk:Qoan 1 0  
aa Wikipedia:Community_Portal 1 0  
aa.d Main_Page 2 0  
aa.m Main_Page 1 0  
ab 1005 1 0  
ab 105 2 0  
ab 1099 1 0  
ab 1150 1 0
```

❸ Заархивированный файл содержит единственный текстовый файл с тем же именем, что и у архива

❹ Содержимое файла содержит следующие элементы, разделенные пробелами:

- 1) код домена;
- 2) заголовок страницы;
- 3) количество просмотров;
- 4) размер ответа в байтах.

Так, например, «en.m American Bobtail 6 0» значит шесть просмотров страницы https://en.m.wikipedia.org/wiki/American_Bobtail (представитель семейства кошачьих) за определенный час

❺ Данные о просмотрах страниц обычно публикуются примерно через 45 минут после окончания интервала; однако иногда выпуск может занять до 3–4 часов

Определение способа загрузки дополнительных данных



- Видно, что URL-адреса следуют фиксированному шаблону, который можно использовать при пакетной загрузке данных.
- В качестве мысленного эксперимента и для проверки данных проверим, какие коды доменов наиболее часто используются 7 июля с 10:00 до 11:00

```
wget https://dumps.wikimedia.org/other/pageviews/2019/2019-07/pageviews-20190707-110000.gz
```

```
gunzip pageviews-20190707-110000.gz
```

```
awk -F ' ' '{print $1}' pageviews-20190707-110000 | sort | uniq -c | sort -nr | head
```

```
1061202 en
995600 en.m
300753 ja.m
286381 de.m
257751 de
226334 ru
201930 ja
198182 fr.m
193331 ru.m
171510 it.m
```

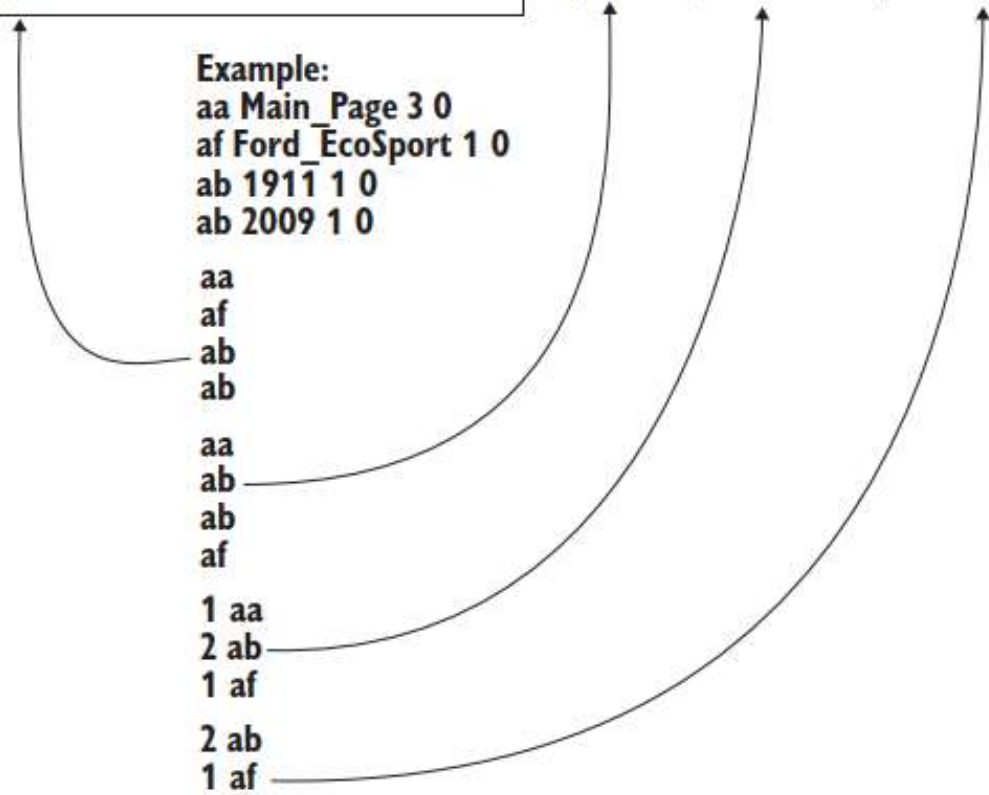

Первый простой анализ данных о просмотрах страниц Викимедиа

```
$ wget https://dumps.wikimedia.org/other/pageviews/ 2019/2019-07/pageviews-20190707-110000.gz
$ gunzip pageviews-20190707-110000.gz
$ awk -F' ' '{print $1}' pageviews-20190707-110000 | sort | uniq -c | sort -nr | head
```

```
1061202 en
995600 en.m
300753 ja.m
286381 de.m
257751 de
226334 ru
201930 ja
198182 fr.m
193331 ru.m
171510 it.m
```

Example:
aa Main Page 3 0
af Ford EcoSport 1 0
ab 1911 1 0
ab 2009 1 0

```
aa
af
ab
ab
aa
ab
ab
af
1 aa
2 ab
1 af
2 ab
1 af
1 aa
```



Первый простой анализ данных о просмотрах страниц Викимедиа



- **1061202 en** и **995600 en.m** - наиболее просматриваемыми доменами в период с 10:00 до 11:00 7 июля являются «en» и «en.m» (мобильная версия .en), английский является наиболее используемым языком в мире.
- Подтверждается отсутствие неопределенных символов или смещения столбцов, а это означает, что не требуется выполнять дополнительную предобработку для очистки данных.

1061202	en
995600	en.m
300753	ja.m
286381	de.m
257751	de
226334	ru
201930	ja
198182	fr.m
193331	ru.m
171510	it.m

Определение способа загрузки дополнительных данных

23 марта 2024



```
wget https://dumps.wikimedia.org/other/pageviews/2024/2024-03/pageviews-20240322-220000.gz
```

HTTP request sent, awaiting response... 404 Not Found

Обновление каждые 45 мин -1 час, но во время теракта в Кроксе Викимедиа прекратило индексацию.
3 часа индексации отключены.

```
wget https://dumps.wikimedia.org/other/pageviews/2024/2024-03/pageviews-20240322-200000.gz
```

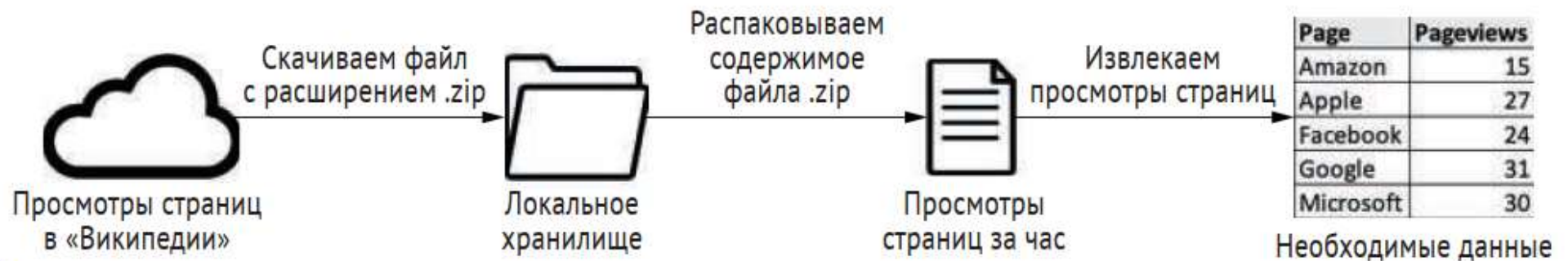
```
gunzip pageviews-20240322-200000.gz
```

```
awk -F ' ' '{print $1}' pageviews-20240322-200000 | sort | uniq -c | sort -nr | head
```

```
1290891 en.m
1247927 en
 301621 de.m
 265680 es.m
 250060 ru.m
 244587 fr.m
 198261 it.m
 188956 de
 165083 es
 149428 ru
```

Определение способа загрузки дополнительных данных

- Создадим первую версию DAG, который будет учитывать количество просмотров страниц Википедии.
- Загрузить, извлечь и прочесть данные.
- Выберем пять компаний (Amazon, Apple, Facebook, Google и Microsoft) для первоначального отслеживания и проверки гипотезы.



Определение способа загрузки дополнительных данных

- Первым шагом является загрузка ZIP-файла для каждого интервала. URL-адрес состоит из различных компонентов даты и времени:

```
https://dumps.wikimedia.org/other/pageviews/  
{year}/{year}-{month}/pageviews-{year}{month}{day}-{hour}0000.gz
```

- Для каждого интервала нужно будет вставить дату и время для этого конкретного интервала в URL-адрес.
- Есть много способов загрузить данные о просмотрах страниц. **BashOperator** и **PythonOperator**. Метод вставки переменных во время выполнения в эти операторы можно распространить на все другие типы операторов.

Загрузка просмотров страниц «Википедии» с помощью BashOperator

- Для начала скачать просмотры страниц «Википедии» с помощью оператора **BashOperator**, принимающего аргумент **bash_command**, которому предоставляем команду **Bash** для выполнения — в начале и в конце каждого компонента URL-адреса, куда нам нужно вставить переменную, стоят двойные фигурные скобки.

```
import airflow.utils.dates
from airflow import DAG
from airflow.operators.bash import BashOperator

dag = DAG(
    dag_id="chapter4_stocksense_bashoperator",
    start_date=airflow.utils.dates.days_ago(3),
    schedule_interval="@hourly",
)

get_data = BashOperator(
    task_id="get_data",
    bash_command=(
        "curl -o /tmp/wikipageviews.gz "
        "https://dumps.wikimedia.org/other/pageviews/"
        "{{ execution_date.year }}/ "
        "{{ execution_date.year }}-"
        "{{ '{:02}'.format(execution_date.month) }}/"
        "pageviews-{{ execution_date.year }}"
        "{{ '{:02}'.format(execution_date.month) }}"
        "{{ '{:02}'.format(execution_date.day) }}-"
        "{{ '{:02}'.format(execution_date.hour) }}0000.gz"
    ),
    dag=dag,
)
```

Двойные фигурные скобки обозначают переменную, вставленную во время выполнения

Может быть указана любая переменная или выражение Python

Определение способа загрузки дополнительных данных

- **Execution_date** — это одна из переменных, которая «волшебным образом» доступна во время выполнения задачи. Двойные фигурные скобки обозначают строку шаблона Jinja.
- **Jinja** — это шаблонизатор, который заменяет переменные и/или выражения в шаблонной строке во время выполнения. Шаблонизация используется, когда вы, как программист, не знаете ценность чего-либо на момент написания, но знаете ценность чего-то во время выполнения. Пример: у вас есть форма, в которую вы можете вставить свое имя, и код печатает вставленное имя:

Вставьте имя здесь

```
print("Hello {{ name }}!")
```

Двойные фигурные скобки сообщают **Jinja**, что внутри есть переменная или выражение, которое нужно оценить.

Определение способа загрузки дополнительных данных

- Значение имени неизвестно при программировании, поскольку пользователь вводит свое имя в форму во время выполнения. Что мы действительно знаем, так это то, что вставленное значение присваивается переменной с именем `name`, и затем мы можем предоставить шаблонную строку «Hello {{ name }}!» для визуализации и вставки значения `name` во время выполнения.

Вставьте имя здесь

```
print("Hello {{ name }}!")
```

Двойные фигурные скобки сообщают **Jinja**, что внутри есть переменная или выражение, которое нужно оценить.

Определение способа загрузки дополнительных данных

- В **Airflow** есть ряд переменных, доступных во время выполнения из контекста задачи.
- Одна из этих переменных — **Execute_date**.
- **Airflow** использует библиотеку **Pendulum** (<https://pendulum.eustace.io>) для даты и времени, и **Execution_date** является таким объектом даты и времени Pendulum.
- Это замена встроенного **Python datetime**, поэтому все методы, которые можно применить к Python, также можно применить и к **Pendulum**. Точно так же, как используем **datetime.now().year**, получим тот же результат с помощью **pendulum.now().year**.

```
>>> from datetime import datetime
>>> import pendulum
>>> datetime.now().year
2024
>>> pendulum.now().year
2024
```

Что доступно для создания шаблонов? Вывод контекста задачи

- С помощью **PythonOperator** можно вывести полный контекст задачи и изучить его.

```
import airflow.utils.dates
from airflow import DAG
from airflow.operators.python import PythonOperator

dag = DAG(
    dag_id="chapter4_print_context",
    start_date=airflow.utils.dates.days_ago(3),
    schedule_interval="@daily",
)

def _print_context(**kwargs):
    print(kwargs)

print_context = PythonOperator(
    task_id="print_context",
    python_callable=_print_context,
    dag=dag,
)
```

Что доступно для создания шаблонов? Вывод контекста задачи

- При выполнении этой задачи выводится словарь всех доступных
- переменных в контексте задачи.

```
{  
    'dag': <DAG: print_context>,  
    'ds': '2019-07-04',  
    'next_ds': '2019-07-04',  
    'next_ds_nodash': '20190704',  
    'prev_ds': '2019-07-03',  
    'prev_ds_nodash': '20190703',  
    ...  
}
```

- Все переменные записываются в **** kwargs** и передаются функции `print()`. Все эти переменные доступны в среде выполнения.

Все переменные контекста задачи

Ключ	Описание	Пример
conf	Предоставляет доступ к конфигурации Airflow	
dag	Текущий объект DAG	Объект DAG
dag_run	Текущий объект DagRun	Объект DagRun
ds	execution_date в формате %Г-%М-%д	"2019-01-01"
ds_nodash	execution_date в формате %Г%M%д	"20190101"
execution_date	Дата начала и время интервала задачи	Объект pendulum.datetime
inlets	Сокращение для task.inlets, функции для отслеживания источников входных данных для происхождения данных	[]
macros	Модуль airflow.macros	Модуль macros
next_ds	execution_date следующего интервала (= конец текущего интервала) в формате %Г-% М-%д	"2019-01-02"
next_ds_nodash	execution_date следующего интервала (= конец текущего интервала) в формате %Г%M%д	"20190102"
next_execution_date	Дата начала и время следующего интервала задачи (= конец текущего интервала)	Объект pendulum.datetime .DateTime
outlets	Сокращение для task.outlets, функции для отслеживания источников выходных данных для происхождения данных	[]
params	Пользовательские переменные для контекста задачи	{}

Все переменные контекста задачи

prev_ds	execution_date предыдущего интервала в формате %Г-%м-%д	"2018-12-31"
prev_ds_nodash	execution_date предыдущего интервала в формате %Г%m%д	"20181231"
prev_execution_date	Дата начала и время предыдущего интервала задачи	Объект pendulum.datetime .DateTime
prev_execution_date_success	Дата начала и время последнего успешно завершенного запуска одной той же задачи (только в прошлом)	Объект pendulum.datetime .DateTime
prev_start_date_success	Дата и время начала последнего успешного запуска одной и той же задачи (только в прошлом)	Объект pendulum.datetime .DateTime
run_id	run_id объекта DagRun (обычно ключ состоит из префикса + datetime)	"manual_2019-01-01T00:00:00+00:00"
task	Текущий оператор	Объект PythonOperator
task_instance	Текущий объект TaskInstance	Объект TaskInstance
task_instance_key_str	Уникальный идентификатор текущего объекта TaskInstance ({dag_id}__{task_id}__{ds_nodash})	"dag_id_task_id_20190101"
templates_dict	Пользовательские переменные для контекста задачи	{}
test_mode	Работает ли Airflow в тестовом режиме (свойство конфигурации)	False
ti	Текущий объект TaskInstance, то же, что и task_instance	Объект TaskInstance

Все переменные контекста задачи

Ключ	Описание	Пример
tomorrow_ds	ds плюс один день	"2019-01-02"
tomorrow_ds_nodash	ds_nodash плюс один день	"20190102"
ts	execution_date, отформатированная в соответствии с форматом ISO8601	"2019-01-01T00:00:00+00:00"
ts_nodash	execution_date в формате %Г%M%дВ%Ч%M%С	"20190101T000000"
ts_nodash_with_tz	ts_nodash с информацией о часовом поясе	"20190101T000000+0000"
var	Объекты-помощники для работы с переменными Airflow	{}
yesterday_ds	ds минус один день	"2018-12-31"
yesterday_ds_nodash	ds_nodash минус один день	"20181231"

Создание шаблона для PythonOperator

- **PythonOperator** не принимает аргументы, которые можно шаблонизировать, используя контекст среды выполнения. Вместо этого он принимает аргумент **python_callable**, в котором можно применить данный контекст.
- Проверим код для скачивания просмотров страниц «Википедии», реализованного с использованием **PythonOperator**.

Создание шаблона для PythonOperator

```
from urllib import request

import airflow
from airflow import DAG
from airflow.operators.python import PythonOperator

dag = DAG(
    dag_id="stocksense",
    start_date=airflow.utils.dates.days_ago(1),
    schedule_interval="@hourly",
)

def _get_data(execution_date):
    year, month, day, hour, *_ = execution_date.timetuple()
    url = (
        "https://dumps.wikimedia.org/other/pageviews/"
        f"{year}/{year}{month:0>2}/"
        f"pageviews{year}{month:0>2}{day:0>2}{hour:0>2}0000.gz"
    )
    output_path = "/tmp/wikipageviews.gz"
    request.urlretrieve(url, output_path)

get_data = PythonOperator(
    task_id="get_data",
    python_callable=_get_data,
    dag=dag,
```

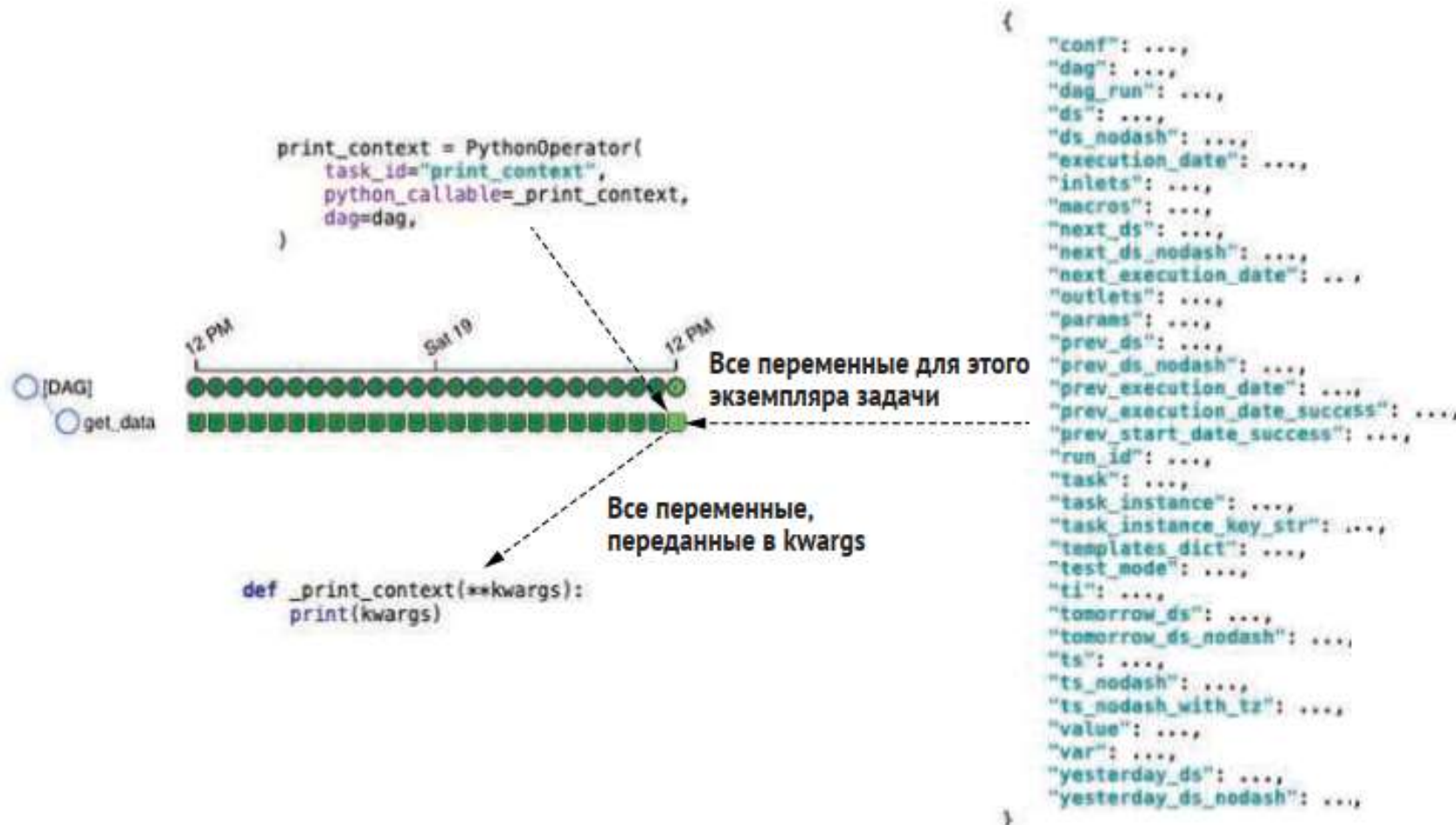
PythonOperator
принимает
функцию
Python,
тогда как
BashOperator
принимает
команду Bash
в качестве
строки для
выполнения

Создание шаблона для PythonOperator

Функции – это объекты первого класса в Python, и мы предоставляем вызываемый объект (функция – это вызываемый объект) аргументу **python_callable** оператора **PythonOperator**.

При выполнении **PythonOperator** выполняет предоставленный вызываемый объект, которым может быть любая функция. Поскольку это функция, а не строка, как у всех других операторов, код внутри функции нельзя шаблонизировать автоматически.

Предоставление контекста задачи с помощью PythonOperator



Оператор PythonOperator, скачивающий ежечасные просмотры страниц «Википедии»

PythonOperator принимает функцию вместо строковых аргументов, и, следовательно, в данном случае нельзя использовать шаблонизатор **Jinja**. В этой вызываемой функции мы извлекаем компоненты datetime из **execution_date** для динамического создания URL-адреса

```
def _get_data(**context):
    year, month, day, hour, *_ = context["execution_date"].timetuple()
    url = (
        "https://dumps.wikimedia.org/other/pageviews/"
        f"{year}/{year}-{month:0>2}/pageviews-{year}{month:0>2}{day:0>2}-{hour:0>2}0000.gz"
    )
    output_path = "/tmp/wikipageviews.gz"
    request.urlretrieve(url, output_path)
```

Переменные контекста задачи

Извлекаем компоненты datetime из execution_date

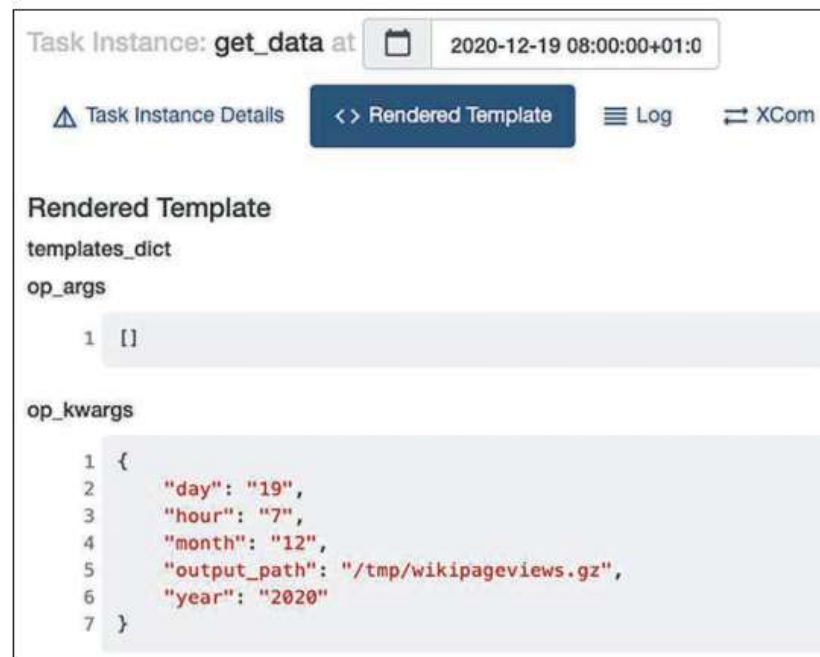
Форматируем URL-адрес с компонентами datetime

Получаем данные

Изучение шаблонных аргументов

Полезный инструмент для отладки проблем с шаблонными аргументами – пользовательский интерфейс **Airflow**.

Вы можете проверить значения шаблонных аргументов после запуска задачи, выбрав ее на диаграмме или в дереве и щелкнув по кнопке **Rendered Template** (Визуализированный шаблон)



The screenshot shows the 'Rendered Template' view in the Airflow web interface. At the top, it displays 'Task Instance: get_data at' followed by a calendar icon and the timestamp '2020-12-19 08:00:00+01:00'. Below this is a navigation bar with 'Task Instance Details', a highlighted '< > Rendered Template' button, and links for 'Log' and 'XCom'. The main content area is titled 'Rendered Template' and shows the rendered values for 'templates_dict', 'op_args', and 'op_kwargs'. 'op_args' is shown as a list containing an empty list. 'op_kwargs' is shown as a dictionary with keys for 'day', 'hour', 'month', 'output_path', and 'year'.

```
Task Instance: get_data at 2020-12-19 08:00:00+01:00

Task Instance Details < > Rendered Template Log XCom

Rendered Template
templates_dict
op_args
1 []

op_kwargs
1 {
2   "day": "19",
3   "hour": "7",
4   "month": "12",
5   "output_path": "/tmp/wikipageviews.gz",
6   "year": "2020"
7 }
```

Подключение других систем. Чтение просмотров страниц для заданных имен страниц

- Обработка ежечасных просмотров страниц Википедии. Следующие два оператора будут извлекать архив и обрабатывать извлеченный файл, просматривая его и выбирая количество просмотров страниц для заданных имен страниц. Результат затем печатается в журналах.

```
extract_gz = BashOperator(
    task_id="extract_gz",
    bash_command="gunzip --force /tmp/wikipageviews.gz",
    dag=dag,
)

def _fetch_pageviews(pagenames):
    result = dict.fromkeys(pagenames, 0)
    with open(f"/tmp/wikipageviews", "r") as f:
        for line in f:
            domain_code, page_title, view_counts, _ = line.split(" ")
            if domain_code == "en" and page_title in pagenames:
                result[page_title] = view_counts
    print(result)
    # Выводит, например, '{"Facebook': '778', 'Apple': '20', 'Google': '451',
    # 'Amazon': '9', 'Microsoft': '119'}"
```

Открываем файл, написанный в предыдущем задании

Извлекаем элементы на строке

Фильтруем только домен "en"

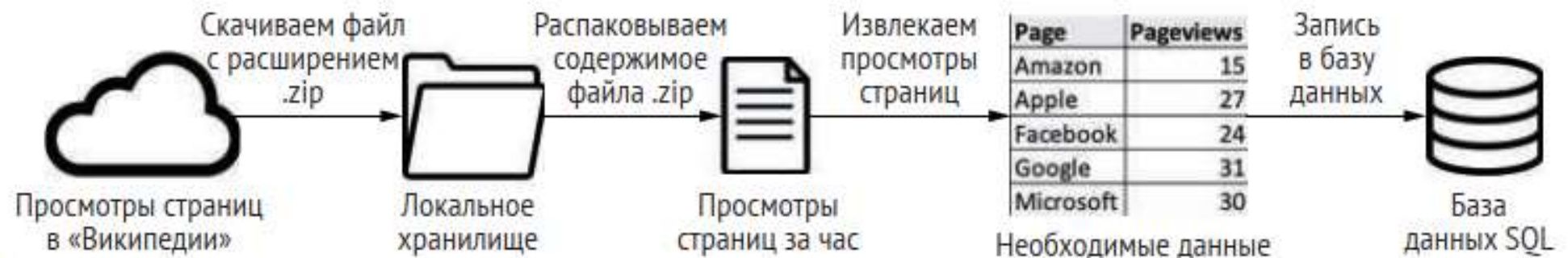
Проверяем, находится ли page_title в pagenames

```
fetch_pageviews = PythonOperator(
    task_id="fetch_pageviews",
    python_callable=_fetch_pageviews,
    op_kwargs={
        "pagenames": {
            "Google",
            "Amazon",
            "Apple",
            "Microsoft",
            "Facebook",
        }
    },
    dag=dag,
)
```

ПЕРВАЯ МОДИФИКАЦИЯ

- В качестве первой модификации требуется записать данные в базу данных, что позволит запрашивать ее с помощью SQL и выполнять запросы:

«Каково среднее количество просмотров страниц в час на странице Google в Википедии?»



Концептуальное представление рабочего процесса. После извлечения просмотров страниц запишите количество просмотров в базу данных SQL

Оператор CREATE TABLE для хранения вывода

```
CREATE TABLE pageview_counts (  
    pagename VARCHAR(50) NOT NULL,  
    pageviewcount INT NOT NULL,  
    datetime TIMESTAMP NOT NULL  
);
```

Операция INSERT, сохраняющая вывод в таблице
pageview_counts

```
INSERT INTO pageview_counts  
VALUES ('Google', 333, '20190717T00:00:00');
```


Написание инструкций INSERT для передачи в PostgresOperator

```
def _fetch_pageviews(pagenames, execution_date, **_):
    result = dict.fromkeys(pagenames, 0)
    with open("/tmp/wikipageviews", "r") as f:
        for line in f:
            domain_code, page_title, view_counts, _ = line.split(" ")
            if domain_code == "en" and page_title in pagenames:
                result[page_title] = view_counts
    with open("/tmp/postgres_query.sql", "w") as f:
        for pagename, pageviewcount in result.items():
            f.write(
                "INSERT INTO pageview_counts VALUES ("
                f"'{pagename}', {pageviewcount}, '{execution_date}'"
                ");\n"
            )

fetch_pageviews = PythonOperator(
    task_id="fetch_pageviews",
    python_callable=_fetch_pageviews,
    op_kwargs={"pagenames": {"Google", "Amazon", "Apple", "Microsoft",
                              "Facebook"}},
    dag=dag,
)
```

Инициализируем результат для всех просмотров страниц, используя 0

Счетчик просмотров страниц

Для каждого результата пишем SQL-запрос

При выполнении этой задачи будет создан файл (/tmp/postgres_query.sql) для заданного интервала, содержащий все SQL-запросы, которые будут выполняться PostgresOperator.

ПЕРВАЯ МОДИФИКАЦИЯ

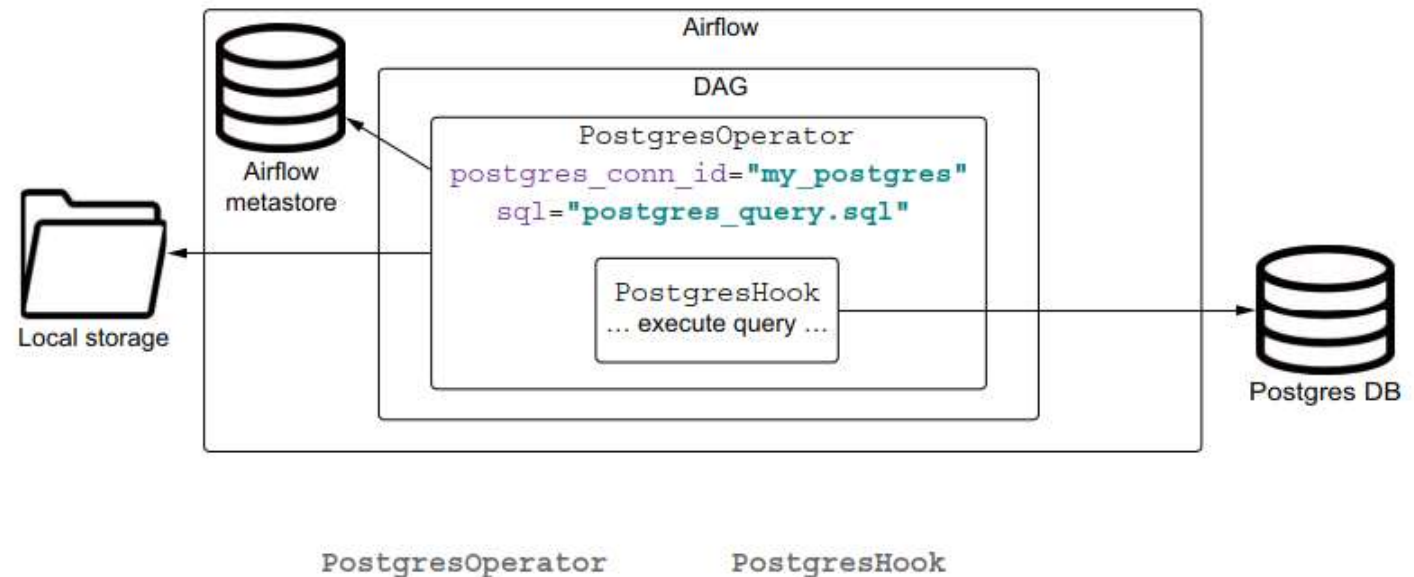
- Соответствующее представление графа будет выглядеть:

BashOperator PostgresOperator PythonOperator



ПЕРВАЯ МОДИФИКАЦИЯ

- PostgresOperator создает т. н. **хук** для обмена данными с **Postgres**.
- **Хук** занимается созданием подключения, отправкой запросов в **Postgres** и последующим закрытием подключения. В данной ситуации оператор просто передает запрос от пользователя точке подключения.



SQL-запрос, спрашивающий, в какое время каждая из страниц пользуется наибольшей популярностью



```
SELECT x.pagename, x.hr AS "hour", x.average AS "average pageviews"
FROM (
  SELECT
    pagename,
    date_part('hour', datetime) AS hr,
    AVG(pageviewcount) AS average,
    ROW_NUMBER() OVER (PARTITION BY pagename ORDER BY AVG(pageviewcount)
DESC)
  FROM pageview_counts
  GROUP BY pagename, hr
) AS x
WHERE row_number=1;
```

SQL-запрос, спрашивающий, в какое время каждая из страниц пользуется наибольшей популярностью

Результаты запроса, показывающие, какое время является наиболее популярным для каждой страницы

Название страницы	Время	Среднее количество просмотров
Amazon	18	20
Apple	16	66
Facebook	16	500
Google	20	761
Microsoft	21	181

СПАСИБО ЗА ВНИМАНИЕ