

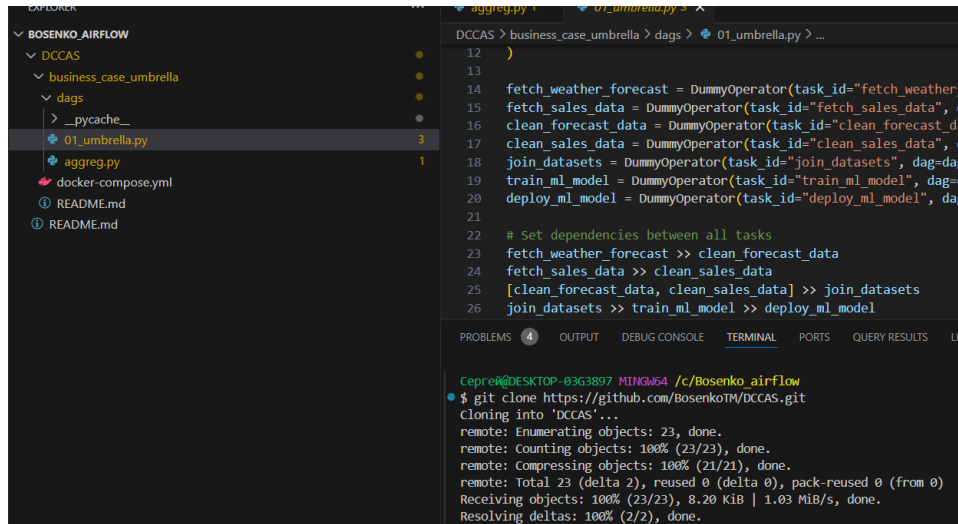
Кузьменко Сергей БД-231. Отчет ЛР-3. Вариант 18

1. Развернуть ВМ [ubuntu_mgpu.ova](#) в [VirtualBox](#).

Выполнял на локальной машине

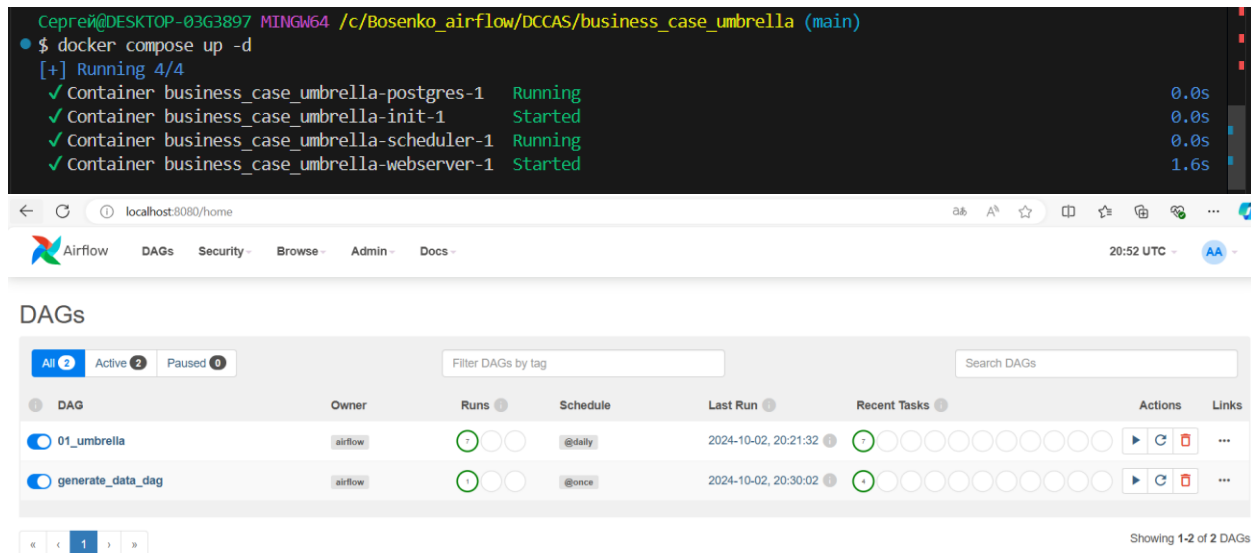
2. Клонировать на ПК задание Бизнес кейс Umbrella в домашний каталог ВМ.

git clone <https://github.com/BosenkoTM/DCCAS>



The screenshot shows a code editor with two panes. The left pane displays the file structure of the 'BOSENKO_AIRFLOW' project, including 'DCCAS', 'business_case_umbrella', 'dags', and '01_umbrella.py'. The right pane shows the code in '01_umbrella.py', which defines several dummy operators for tasks like 'fetch_weather_forecast', 'fetch_sales_data', 'clean_forecast_data', 'clean_sales_data', 'join_datasets', 'train_ml_model', and 'deploy_ml_model'. The code uses 'DummyOperator' from Airflow and sets dependencies between tasks.

3. Запустить контейнер с кейсом, изучить и описать основные элементы интерфейса Apache Airflow.



The top part of the screenshot shows a terminal window where the command 'docker compose up -d' was executed, resulting in four containers being started: 'business_case_umbrella-postgres-1', 'business_case_umbrella-init-1', 'business_case_umbrella-scheduler-1', and 'business_case_umbrella-webserver-1'. The bottom part shows the Apache Airflow web interface at 'localhost:8080/home'. The interface displays a list of DAGs (Directed Acyclic Graphs) with columns for DAG name, Owner, Runs, Schedule, Last Run, Recent Tasks, Actions, and Links. Two DAGs are listed: '01_umbrella' and 'generate_data_dag', both owned by 'airflow'.

Описание основных элементов:

"""01_umbrella.py

""""DAG demonstrating the umbrella use case with dummy operators.""""

#Импорт библиотек

import airflow.utils.dates

from airflow import DAG

from airflow.operators.dummy import DummyOperator

#Далее задается DAG- направленный ациклический граф. Задано его имя, описание и #периодичность выполнения – ежедневно. Стартовая дата задана как 5 дней назад от #текущей

```
dag = DAG(
dag_id="01_umbrella",
description="Umbrella example with DummyOperators.",
start_date=airflow.utils.dates.days_ago(5),
schedule_interval="@daily",
)
```

#В следующем блоке каждая строчка задает задачу (задачу). Используется dummyoperator. #Этот оператор пустой. Он ничего не делает, нужен как в данном примере для #иллюстрации графа и отладки. Всего получается 7 задач. В каждом задается название #задачи и даг к которому эта задача относится.

```
fetch_weather_forecast = DummyOperator(task_id="fetch_weather_forecast", dag=dag)
fetch_sales_data = DummyOperator(task_id="fetch_sales_data", dag=dag)
clean_forecast_data = DummyOperator(task_id="clean_forecast_data", dag=dag)
clean_sales_data = DummyOperator(task_id="clean_sales_data", dag=dag)
join_datasets = DummyOperator(task_id="join_datasets", dag=dag)
train_ml_model = DummyOperator(task_id="train_ml_model", dag=dag)
deploy_ml_model = DummyOperator(task_id="deploy_ml_model", dag=dag)
# Set dependencies between all tasks
```

#Следующий блок задает зависимости. fetch_weather_forecast >> clean_forecast_data
#Означает, что clean_forecast_data начнется после завершения fetch_weather_forecast

#А например [clean_forecast_data, clean_sales_data] >> join_datasets

#Означается, что join_datasets начнется только после выполнения обоих задач

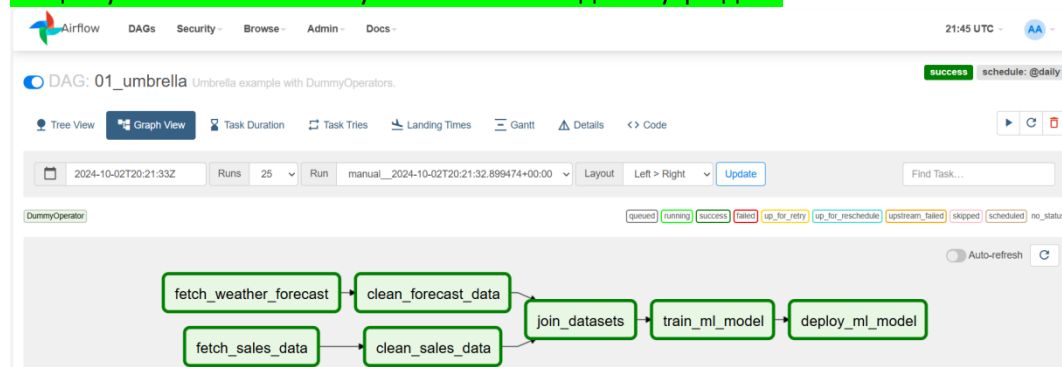
```
fetch_weather_forecast >> clean_forecast_data
```

```
fetch_sales_data >> clean_sales_data
```

```
[clean_forecast_data, clean_sales_data] >> join_datasets
```

```
join_datasets >> train_ml_model >> deploy_ml_model
```

По итогу получается даг, который можно увидеть в виде графа. Запускается кнопкой пуск и по цвету можно понять статус выполнения задач внутри дага.



Аналогично для **agggreg.py**

```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator # type: ignore
from datetime import datetime
import pandas as pd # type: ignore
```

```
import random
```

```
def generate_data_1():
    data_1 = {
        'ID': [i for i in range(1, 101)],
        'Сумма': [random.randint(100, 500) for _ in range(100)],
        'Категория': [random.choice(['Электроника', 'Одежда', 'Продукты']) for _ in range(100)],
    }
    df_1 = pd.DataFrame(data_1)
    df_1.to_csv(f'sample_data_1.csv', index=False, encoding='utf-8')
```

```
# Функция для генерации данных и сохранения их в файл CSV 2
```

```
def generate_data_2():
    data_2 = {
        'ID': [i for i in range(51, 151)],
        'Количество': [random.randint(1, 20) for _ in range(100)],
        'Категория': [random.choice(['Электроника', 'Косметика', 'Одежда']) for _ in range(100)],
    }
    df_2 = pd.DataFrame(data_2)
    df_2.to_csv(f'sample_data_2.csv', index=False, encoding='utf-8')
```

```
# Функция для генерации данных и сохранения их в файл CSV 3
```

```
def generate_data_3():
    data_3 = {
        'ID': [i for i in range(26, 126)],
        'Цена': [random.uniform(10.0, 500.0) for _ in range(100)],
        'Категория': [random.choice(['Одежда', 'Косметика', 'Электроника']) for _ in range(100)],
    }
    df_3 = pd.DataFrame(data_3)
    df_3.to_csv(f'sample_data_3.csv', index=False, encoding='utf-8')
```

```
# Функция для агрегации данных из всех трех файлов
```

```
def aggregate_data():
    df_1 = pd.read_csv(f'sample_data_1.csv', encoding='utf-8')
    df_2 = pd.read_csv(f'sample_data_2.csv', encoding='utf-8')
    df_3 = pd.read_csv(f'sample_data_3.csv', encoding='utf-8')

    # Объединение данных по общему полю 'ID'
    merged_df = pd.merge(df_1, df_2, on='ID', how='outer')
    merged_df = pd.merge(merged_df, df_3, on='ID', how='outer')

    # Проведение агрегации
    aggregated_df = merged_df.groupby('Категория').agg({'Сумма': 'sum', 'Количество': 'sum',
    'Цена': 'sum'}).reset_index()
    aggregated_df.to_csv(f'aggregated_data.csv', index=False, encoding='utf-8')
```

```
# Определение DAG
```

```
dag = DAG('generate_data_dag', description='Generate and aggregate sample data',
```

```
schedule_interval='@once', start_date=datetime(2023, 10, 1), catchup=False)
```

Определение задач

#Здесь уже используется новый оператор при создании таска PythonOperator. Указывается
#название таска и функция на которую он ссылается. Используются функции
#generate_data_1, generate_data_2, generate_data_3 для генерации данных и
#aggregate_data для консолидации. Все сгенерированные файлы и итоговые сохраняются в
#нашем контейнере (см. скриншот)

```
task_generate_data_1 = PythonOperator(task_id='generate_data_1',  
python_callable=generate_data_1, dag=dag)  
task_generate_data_2 = PythonOperator(task_id='generate_data_2',  
python_callable=generate_data_2, dag=dag)  
task_generate_data_3 = PythonOperator(task_id='generate_data_3',  
python_callable=generate_data_3, dag=dag)  
task_aggregate_data = PythonOperator(task_id='aggregate_data',  
python_callable=aggregate_data, dag=dag)
```

Установка зависимостей между задачами

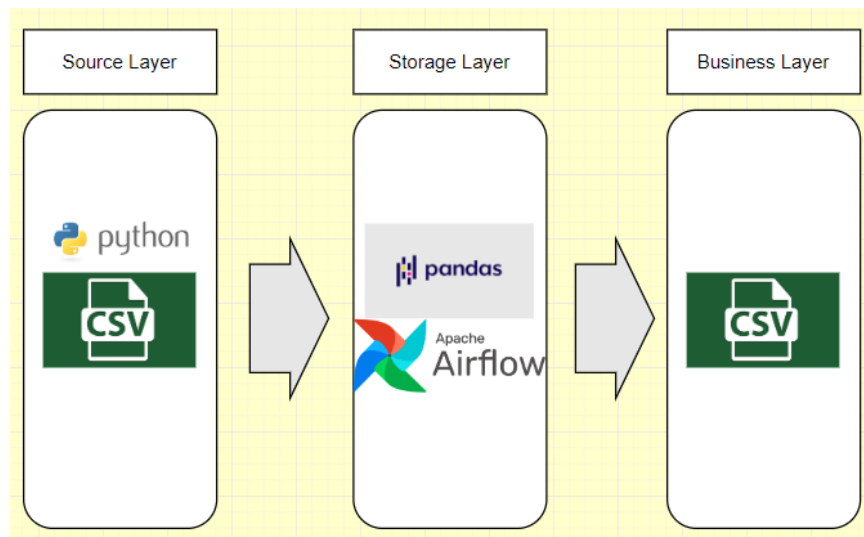
```
task_generate_data_1 >> task_generate_data_2 >> task_generate_data_3 >>  
task_aggregate_data
```



```
Cepрей@DESKTOP-03G3897 MINGW64 /c/Bosenko_airflow/DCCAS/business_case_umbrella (main)  
$ docker exec -it e4bdb4b553aa sh  
$ ls  
aggregated_data.csv  dags  sample_data_1.csv  sample_data_3.csv  webserver_config.py  
airflow.cfg         logs  sample_data_2.csv  unittests.cfg  
$
```

4. Спроектировать верхнеуровневую архитектуру аналитического решения задания **Бизнес кейс Umbrella** в draw.io. Необходимо использовать:

- Source Layer - слой источников данных.
- Storage Layer - слой хранения данных.
- Business Layer - слой для доступа к данным бизнес пользователей.



5. Выполнить индивидуальное задание. Результаты работы представить в виде файла ФИО.pdf.

Создадим новый даг, который в загружает в пандас сгенерированные файлы из Практической работы 3 и проводит консолидацию.

""" data_consolidation_dag.py

```
from airflow import DAG

from airflow.operators.python_operator import PythonOperator # type: ignore
from airflow.operators.dummy import DummyOperator

from datetime import datetime

import pandas as pd # type: ignore
import numpy as np

# Определение DAG
dag = DAG(
    'individual_exercise',
    description='Var. 18. Consolidate data from CSV, Excel, JSON and save to Excel',
    schedule_interval='@once',
    start_date=datetime(2024, 1, 1)
)

def extract_and_transform():
    # Пути к файлам данных

    csv_file = '/opt/airflow/dags/sales_2022.csv'

    excel_file = '/opt/airflow/dags/sales_2023.xlsx'

    json_file = '/opt/airflow/dags/products.json'

    # Шаг 1. Извлечение данных
```

```

df_2022 = pd.read_csv(csv_file)
df_2023 = pd.read_excel(excel_file)
df_products = pd.read_json(json_file)

#Шаг 2. Обработка пропущенных значений

df_2022 = df_2022.dropna(subset=['date', 'product_id', 'quantity'])
df_2022['sales'] = df_2022['sales'].fillna(df_2022['sales'].mean())
df_2023 = df_2023.dropna(subset=['date', 'product_id', 'quantity'])
df_2023['sales'] = df_2023['sales'].fillna(df_2023['sales'].mean())

#Шаг 3. Приведение названия столбцов к единому формату

df_2022.columns = df_2022.columns.str.lower()
df_2023.columns = df_2023.columns.str.lower()
df_products.columns = df_products.columns.str.lower()

#Шаг 4. Объединение продаж за 2022 и 2023 год

df_sales = pd.concat([df_2022, df_2023], ignore_index=True)

#Шаг 5. Добавление информации о товарах

df_consolidated = pd.merge(df_sales, df_products, on='product_id', how='left')

#Шаг 6. Агрегация и анализ. Продажи по категориям и по годам

df_consolidated['year'] = pd.to_datetime(df_consolidated['date']).dt.year
sales_by_category = df_consolidated.groupby('category')['sales'].sum().sort_values(ascending=False)
avg_sales_by_year = df_consolidated.groupby('year')['sales'].mean()

#Шаг 7. Выгрузка в Excel

with pd.ExcelWriter('resul_data.xlsx') as writer:
    # Записываем первый DataFrame на вкладку "Продажи по категориям"
    sales_by_category.to_excel(writer, sheet_name='Продажи по категориям', index=True)

    # Записываем второй DataFrame на вкладку "Продажи по годам"
    avg_sales_by_year.to_excel(writer, sheet_name='Продажи по годам', index=True)

# Определение задач

task_1 = DummyOperator(task_id="start", dag=dag)
task_2 = PythonOperator(task_id="ETL", python_callable=extract_and_transform, dag=dag)

```

```
task_3 = DummyOperator(task_id="end", dag=dag)

# Установка зависимостей между задачами
task_1 >> task_2 >> task_3
```

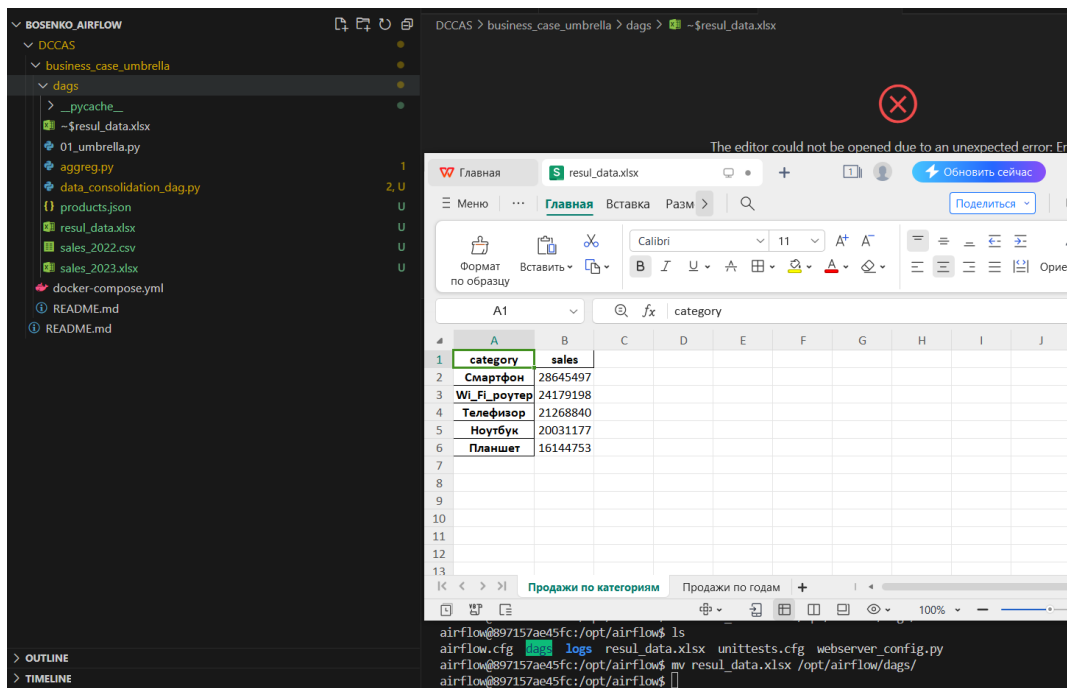
Получившийся ДАГ

DAGs

[illegible]

Результат выполнения ДАГА

```
Cepren@DESKTOP-03G3897 MINGW64 /c/Bosenko_airflow/DCCAS/business_case_umbrella (main)
$ docker exec -it 897157ae45fc bash
airflow@897157ae45fc:/opt/airflow$ ls
airflow.cfg  dags  logs  resul  data.xlsx  unittests.cfg  webserver_config.py
airflow@897157ae45fc:/opt/airflow$
```

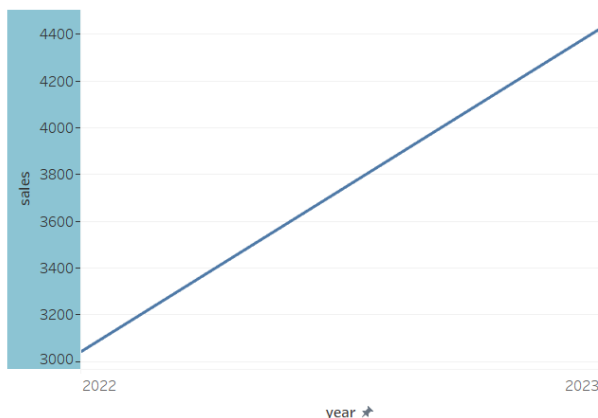


Визуализация

Динамика продаж. Лабораторная 3. Вариант 18.
Кузьменко Сергей



Динамика продаж



Архитектура решения:

