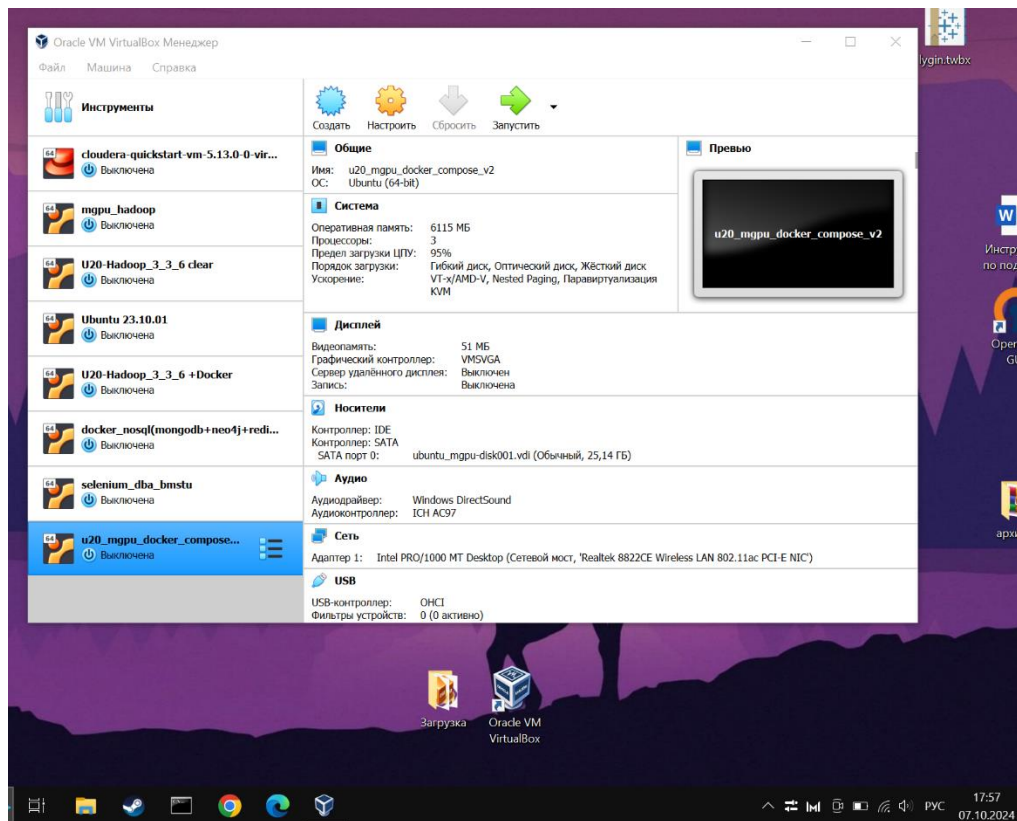
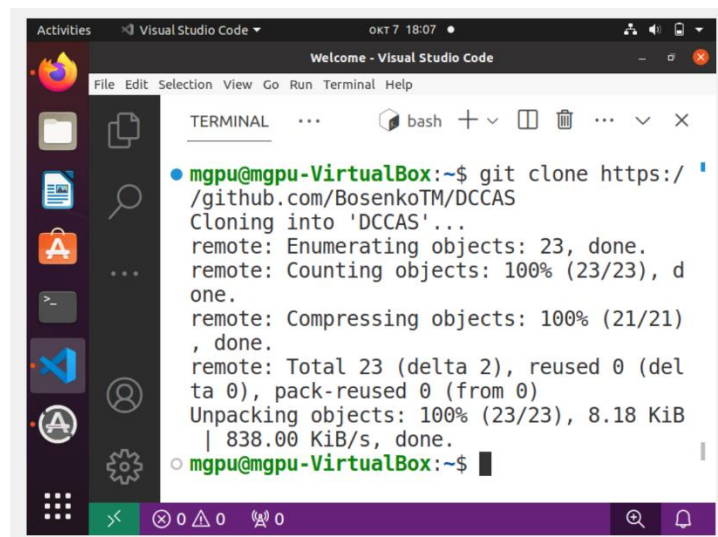


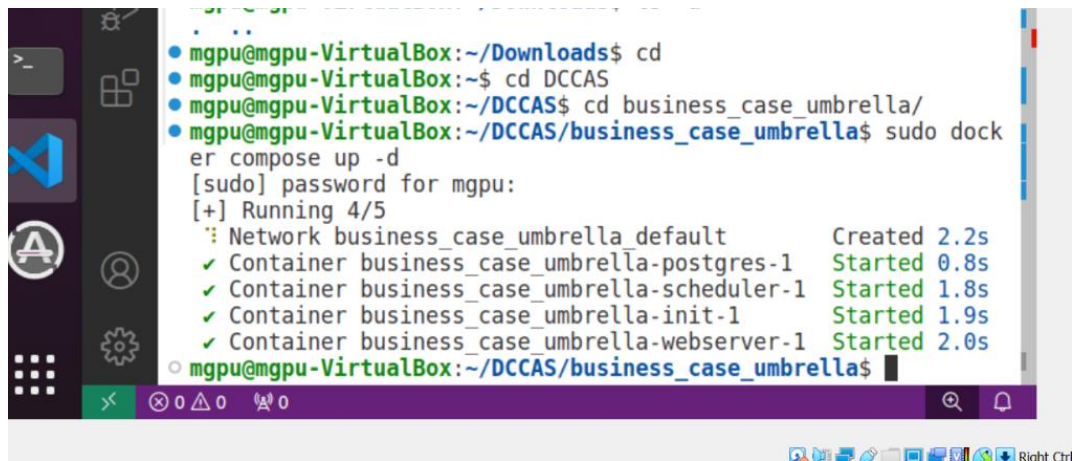
Задание – 1. Развернуть VM ubuntu_mgpu.ova в VirtualBox.



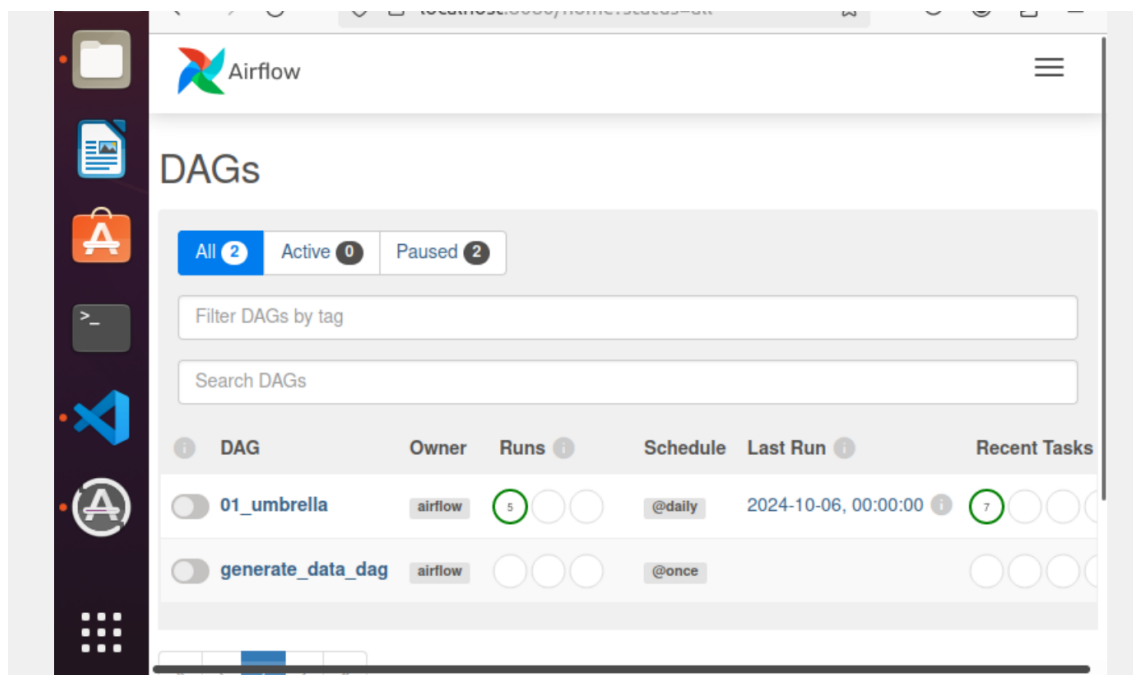
Задание – 2. Клонировать на ПК задание Бизнес кейс Umbrella в домашний каталог VM.



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



```
mgpu@mgpu-VirtualBox:~/Downloads$ cd
mgpu@mgpu-VirtualBox:~$ cd DCCAS
mgpu@mgpu-VirtualBox:~/DCCAS$ cd business_case_umbrella/
mgpu@mgpu-VirtualBox:~/DCCAS/business_case_umbrella$ sudo docker
er compose up -d
[sudo] password for mgpu:
[+] Running 4/5
  ⚙ Network business_case_umbrella default       Created 2.2s
  ✓ Container business_case_umbrella-postgres-1 Started 0.8s
  ✓ Container business_case_umbrella-scheduler-1 Started 1.8s
  ✓ Container business_case_umbrella-init-1      Started 1.9s
  ✓ Container business_case_umbrella-webserver-1 Started 2.0s
mgpu@mgpu-VirtualBox:~/DCCAS/business_case_umbrella$
```



Описание элементов airflow:

1. DAG — это основной элемент в Airflow, представляющий собой граф задач, который показывает порядок их выполнения. Каждая задача в DAG выполняется в соответствии с установленными зависимостями.
2. Owner — это поле указывает на владельца DAG. Обычно это имя пользователя или команды, ответственной за данный DAG.
3. Runs — этот элемент отображает историю выполнения DAG, сколько раз DAG был запущен и каковы результаты этих запусков (успех или ошибка).

4. Schedule — это расписание, по которому DAG будет автоматически запускаться. Оно может быть задано с помощью stop-выражений или других форматов, определяющих частоту выполнения.

5. Last Run — здесь показывается время последнего выполнения DAG.

6. Recent Tasks — этот элемент отображает последние выполненные задачи внутри DAG, включая их статус (успех или ошибка) и время выполнения.

```
6
7 dag = DAG(
8     dag_id="01_umbrella",
9     description="Umbrella example with DummyOperators.",
10    start_date=airflow.utils.dates.days_ago(5),
11    schedule_interval="@daily",
12 )
13 |
```

Описание блока кода – 1. В данном фрагменте кода создаётся объект DAG, который имеет ряд параметров, таких как:

- dag_id - уникальный идентификатор для DAG.
- Description - описание DAG, которое помогает понять его назначение. Здесь указано, что это пример с использованием DummyOperators.
- start_date - дата, с которой начнёт выполняться DAG. Используется функция days_ago(5), DAG начнёт выполняться за 5 дней до текущей даты.
- schedule_interval - интервал, по которому будет запускаться DAG. Здесь указано @daily, что означает, что DAG будет запускаться ежедневно.

Код определяет DAG, который будет запускаться каждый день, начиная с даты, которая была 5 дней назад.

```
12 )
13
14 fetch_weather_forecast = DummyOperator(task_id="fetch_weather_forecast", dag=dag)
15 fetch_sales_data = DummyOperator(task_id="fetch_sales_data", dag=dag)
16 clean_forecast_data = DummyOperator(task_id="clean_forecast_data", dag=dag)
17 clean_sales_data = DummyOperator(task_id="clean_sales_data", dag=dag)
18 join_datasets = DummyOperator(task_id="join_datasets", dag=dag)
19 train_ml_model = DummyOperator(task_id="train_ml_model", dag=dag)
20 deploy_ml_model = DummyOperator(task_id="deploy_ml_model", dag=dag)
```

Описание блока кода - 2: В этом фрагменте кода создаются несколько задач (tasks) для DAG.

1. `fetch_weather_forecast`. Задача, которая отвечает за получение данных о погоде.
2. `fetch_sales_data`. Задача, предназначенная для получения данных о продажах.
3. `clean_forecast_data`. Задача для очистки данных о погоде.
4. `clean_sales_data`. Задача для очистки данных о продажах.
5. `join_datasets`. Задача для объединения очищенных данных о погоде и продажах.
6. `train_ml_model`. Задача, которая отвечает за обучение модели машинного обучения на объединённых данных.
7. `deploy_ml_model`. Задача для развертывания обученной модели машинного обучения.

Все эти задачи организованы в рамках одного DAG и выполняются последовательно.

```
21
22 # Set dependencies between all tasks
23 fetch_weather_forecast >> clean_forecast_data
24 fetch_sales_data >> clean_sales_data
25 [clean_forecast_data, clean_sales_data] >> join_datasets
26 join_datasets >> train_ml_model >> deploy_ml_model
27
```

Описание блока кода – 3. В этом фрагменте кода описывается порядок выполнения задач в DAG.

1. `fetch_weather_forecast >> clean_forecast_data`. Эта строка устанавливает зависимость между задачами. Задача `fetch_weather_forecast` должна быть выполнена перед задачей `clean_forecast_data`.
2. `fetch_sales_data >> clean_sales_data`. Аналогично, пункту 1.

3. `[clean_forecast_data, clean_sales_data] >> join_datasets`. В этой строке указывается, что обе задачи `clean_forecast_data` и `clean_sales_data` должны быть выполнены перед началом задачи `join_datasets`.

4. `join_datasets >>> train_ml_model >> deploy_ml_model`. Здесь устанавливается цепочка зависимостей. Задача `join_datasets` должна завершиться до начала задачи `train_ml_model`, которая, в свою очередь, должна завершиться перед выполнением задачи `deploy_ml_model`.

Задание – 4. Спроектировать верхнеуровневую архитектуру аналитического решения задания в draw.io.

Схема построена на основе варианта – 8 индивидуального задания.

