**Отчет по выполнению домашнего задания №7 "Настройка DAG для выгрузки данных из API в PostgreSQL (Yandex Cloud)"**

**1. Цель работы**

Настроить автоматический data pipeline в Apache Airflow, который:

- Получает данные о положении МКС из публичного API («http://api.open-notify.org/iss-now.json»).

- Сохраняет их в PostgreSQL (Yandex Managed Database).

- Работает по расписанию (каждые 30 минут).

**2. Выполненные шаги**

**2.1. Развертывание инфраструктуры в Yandex Cloud**

1. Создана виртуальная машина с Apache Airflow 2.0 (Ubuntu 20.04, 2 vCPU, 4 GB RAM).
2. Создано Хранилище DAG-файлов. Имя бакета: bucket-af
3. Настроена группа безопасности (Security Groups)

Группы безопасности в Yandex Cloud работают как firewall. Нужно разрешить исходящий трафик на api.open-notify.org:80 и api.wheretheiss.at:443.

1. Открыть Yandex Cloud Console → Virtual Private Cloud (VPC) → Security Groups.
2. Найти группу безопасности, привязанную к кластеру Kubernetes (обычно k8s-security-group).
3. Добавлено правило

на исходящий и исходящий трафик:

* + Протокол: TCP
  + Диапазон портов: 80, 443 (HTTP/HTTPS )
  + Назначение: CIDR 0.0.0.0/0 (разрешить все внешние адреса)

на входящий трафик для Postgres

* + Протокол: TCP
  + Диапазон портов: 6432
  + Назначение: CIDR 0.0.0.0/0 (разрешить все внешние адреса)

1. Создан NAT-шлюз:
   * VPC → Gateway → Create Gateway → NAT Gateway.
   * Выбрана зону доступности (например, ru-central1-a).
   * Привязана к нужной сети.
2. Добавлен маршрут в таблицу маршрутизации:
   1. VPC → Routing tables → Edit.
   2. Добавить маршрут:
      1. Destination: 0.0.0.0/0
      2. Next hop: NAT Gateway (выбрать созданный шлюз).

6. Развернут Managed PostgreSQL в Yandex Cloud:

- Кластер: 1 нода, 2 vCPU, 4 GB RAM.

- Создана БД: «analytics».

- Пользователь: «user1» с правами на запись.

7. Настроено подключение Airflow → PostgreSQL:

- В UI Airflow («Admin → Connections») добавлен Connection ID: «analytics\_postgres».

- Указаны параметры (хост, порт, логин, пароль, SSL).

**2.2. Создание таблицы в PostgreSQL**

Перед запуском DAG создана таблица «iss\_positions»:

CREATE TABLE iss\_positions (

id SERIAL PRIMARY KEY,

timestamp BIGINT,

latitude FLOAT,

longitude FLOAT,

created\_at TIMESTAMP,

status VARCHAR(20),

error\_message TEXT

);

**3. Реализация DAG**

**3.1. Описание DAG («DAG\_MKS\_final.py»)**

DAG состоит из 3 задач:

1. «test\_postgres\_connection» – проверяет доступность PostgreSQL.

2. «fetch\_iss\_data» – получает данные с API МКС.

3. «save\_to\_postgres» – сохраняет данные в БД.

Ключевые особенности DAG:

Расписание: «schedule\_interval=timedelta(minutes=30)»

Обработка ошибок:

- Если API недоступен, сохраняется запись с «status='failed'» и описанием ошибки.

- При проблемах с PostgreSQL задача повторяется («retries=3»).

Автоматическое создание таблицы: Если таблицы нет, DAG создает её при первом запуске.

**4. Проверка работоспособности**

**4.1. Запуск DAG через Airflow UI**

- DAG активирован и запускается каждые 30 минут.

- В логах задач видно:

plaintext

INFO - Testing connection to user1@<postgres\_host>:6432/analytics

INFO - Success: PostgreSQL 15.3

INFO - Successfully saved ISS data to PostgreSQL

**4.2. Проверка данных в PostgreSQL**

Выполнен запрос:

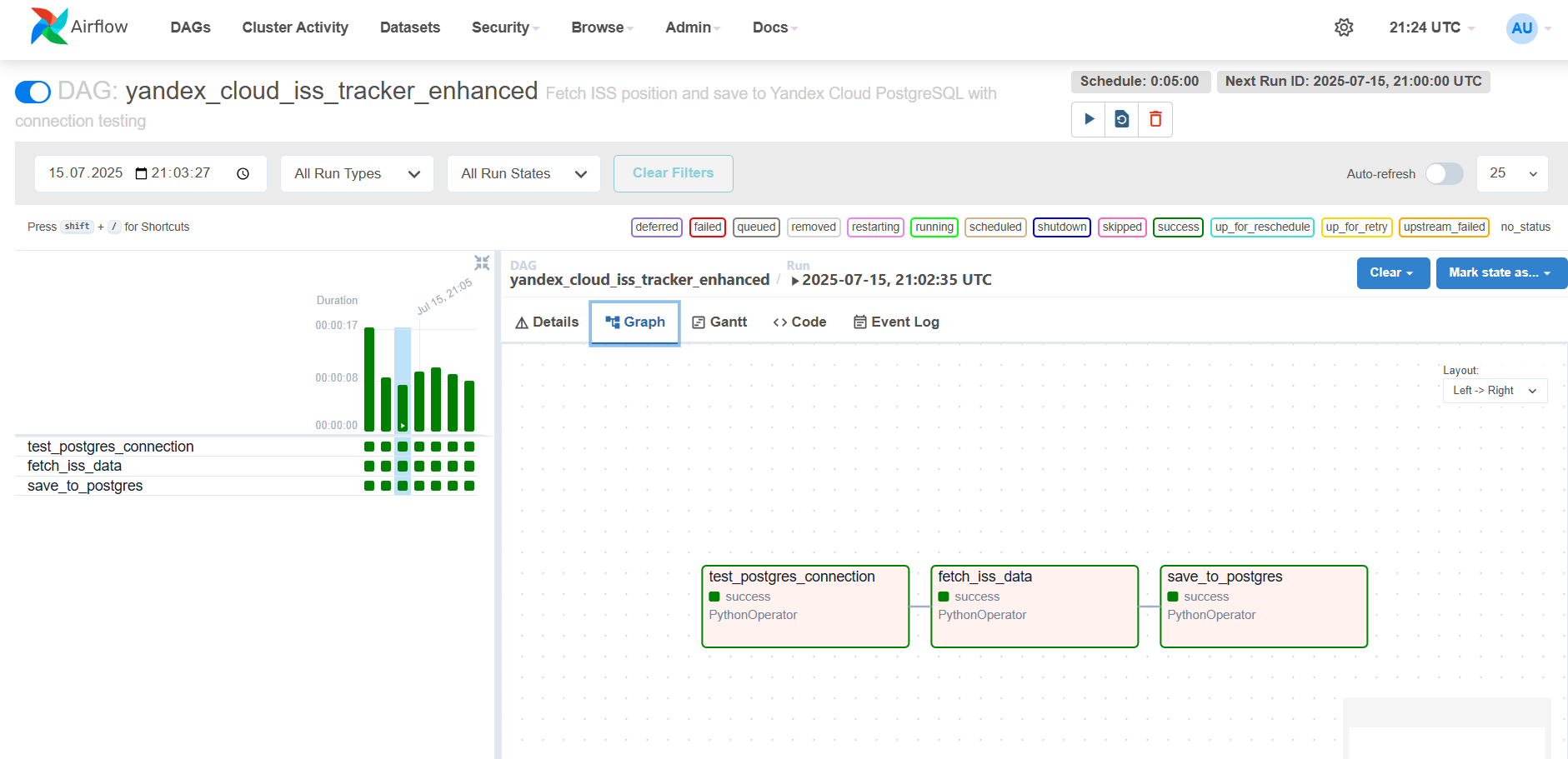
SELECT \* FROM iss\_positions ORDER BY created\_at DESC;

**5. Итог**

* DAG работает стабильно (запускается каждые 30 минут).
* Данные успешно сохраняются в PostgreSQL («analytics.iss\_positions»).
* Ошибки обрабатываются (если API недоступен, записывается статус «failed»).
* Проверка подключения к БД перед записью данных.

**Скриншоты для проверки**

* Airflow UI: Успешные запуски DAG (зеленые кружки).



* PostgreSQL: Данные за последние 7 периодов.

