Федеральное государственное автономное образовательное
учреждение высшего образования
Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина *«Распределенные системы хранения данных»*

Отчет по лабораторной работе №4

Вариант: 368127

Студент:

Барсуков Максим Андреевич,
группа P3315

Преподаватель:

Харитонова Анастасия Евгеньевна

г. Санкт-Петербург, 2025 г.

Оглавление

Задание

Цель работы - ознакомиться с методами и средствами построения отказоустойчивых решений на базе СУБД Postgres; получить практические навыки восстановления работы системы после отказа.

Работа рассчитана на двух человек и выполняется в три этапа: настройка, симуляция и обработка сбоя, восстановление.

Требования к выполнению работы

- В качестве хостов использовать одинаковые виртуальные машины.
- В первую очередь необходимо обеспечить сетевую связность между ВМ.
- Для подключения к СУБД (например, через psql), использовать отдельную виртуальную или физическую машину.
- Демонстрировать наполнение базы и доступ на запись на примере **не менее, чем двух** таблиц, столбцов, строк, транзакций и клиентских сессий.

Этап 1. Конфигурация

Развернуть postgres на двух узлах в режиме трансляции логов. Не использовать дополнительные пакеты. Продемонстрировать доступ в режиме чтение/запись на основном сервере. Продемонстрировать, что новые данные синхронизируются на резервный сервер.

Этап 2. Симуляция и обработка сбоя

2.1 Подготовка:

- Установить несколько клиентских подключений к СУБД.
- Продемонстрировать состояние данных и работу клиентов в режиме чтение/запись.

2.2 Сбой:

Симулировать ошибку диска на основном узле - удалить директорию PGDATA со всем содержимым.

2.3 Обработка:

- Найти и продемонстрировать в логах релевантные сообщения об ошибках.
- Выполнить переключение (failover) на резервный сервер.
- Продемонстрировать состояние данных и работу клиентов в режиме чтение/запись.

Этап 3. Восстановление

- Восстановить работу основного узла - откатить действие, выполненное с виртуальной машиной на этапе 2.2.
- Актуализировать состояние базы на основном узле - накатить все изменения данных, выполненные на этапе 2.3.
- Восстановить исправную работу узлов в исходной конфигурации (в соответствии с этапом 1).
- Продемонстрировать состояние данных и работу клиентов в режиме чтение/запись.

Выполнение

Подготовка окружения

Структура:

```
├── docker-compose.yml
├── primary
│   ├── conf
│   │   ├── pg_hba.conf
│   │   └── postgresql.conf
│   ├── Dockerfile
│   ├── init
│   │   └── init-primary.sh
│   └── scripts
│       ├── init-db.sql
│       ├── read_client.sh
│       └── write_client.sh
└── standby
    ├── conf
    │   ├── pg_hba.conf
    │   └── postgresql.conf
    ├── Dockerfile
    ├── init
    │   └── init-standby.sh
    └── scripts
        ├── auto_promote.sh
        └── read_client.sh
```

Хосты настроены через docker-compose.yml:

```
services:
  primary:
    container_name: primary
    build:
      context: ./primary
    restart: unless-stopped
    ports:
      - "5432:5432"
    environment:
      - PGDATA=/var/lib/postgresql/data
      - PGENCODING=UTF8
      - PGLOCALE=en_US.UTF8
      - PGUSERNAME=postgres
      - POSTGRES_PASSWORD=postgres
    volumes:
      - ./primary/data:/var/lib/postgresql/data
    networks:
      - pg_net

  standby:
    container_name: standby
    build:
      context: ./standby
    restart: unless-stopped
    ports:
      - "5433:5432"
    depends_on:
      - primary
    environment:
      - PGDATA=/var/lib/postgresql/data
      - PGENCODING=UTF8
      - PGLOCALE=en_US.UTF8
      - PGUSERNAME=postgres
      - POSTGRES_PASSWORD=postgres
    volumes:
      - ./standby/data:/var/lib/postgresql/data
    networks:
      - pg_net

networks:
  pg_net:
    driver: bridge
```

Соединение между хостами настроено через сеть `pg_net` и порты 5432 и 5433 соответственно.

Этап 1. Конфигурация

primary

Dockerfile

```
FROM postgres:latest
COPY conf/* /etc/postgresql/
COPY scripts/* /home/scripts/
COPY init/init-primary.sh /home/init/init-primary.sh
RUN chmod +x /home/scripts/read_client.sh
RUN chmod +x /home/scripts/write_client.sh
RUN chmod +x /home/init/init-primary.sh
```

init-primary.sh

```
#!/bin/bash
set -e

psql -v ON_ERROR_STOP=1 --username "postgres" -c "CREATE ROLE replicator
WITH REPLICATION PASSWORD 'replicator_password' LOGIN;"
psql -v ON_ERROR_STOP=1 --username "postgres" -f
"/home/scripts/init-db.sql"
cp /etc/postgresql/postgresql.conf "$PGDATA/postgresql.conf"
cp /etc/postgresql/pg_hba.conf "$PGDATA/pg_hba.conf"
echo "Configuration files copied!"
```

postgresql.conf

```
listen_addresses = '*'
wal_level = replica
wal_keep_size = 64MB
max_wal_senders = 10
max_replication_slots = 10
archive_mode = on
archive_command = 'echo "dummy command, archive_command called"'
log_connections = on
log_disconnections = on
log_duration = on
wal_log_hints = on
```

pg_hba.conf

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             all                                     trust
host    all             all             0.0.0.0/0               md5
host    replication     replicator      0.0.0.0/0               md5
```

standby

Dockerfile

```
FROM postgres:latest
COPY conf/* /etc/postgresql/
COPY scripts/* /home/scripts/
COPY init/init-standby.sh /docker-entrypoint-initdb.d/init-standby.sh
RUN chmod +x /home/scripts/read_client.sh
RUN chmod +x /docker-entrypoint-initdb.d/init-standby.sh
RUN chmod +x /home/scripts/auto_promote.sh
```

init-standby.sh

```
#!/bin/bash
set -e

until pg_isready -h primary -p 5432 -U postgres; do
  echo "Waiting for primary to be ready..."
  sleep 2
done

pg_ctl stop -D "$PGDATA"

rm -rf "$PGDATA"/*
echo "Data directory cleaned up"

PGPASSWORD='replicator_password' pg_basebackup -h primary -D
/var/lib/postgresql/data -U replicator -v -P --wal-method=stream
echo "Base backup completed"

touch "$PGDATA/standby.signal"

chown -R postgres:postgres "$PGDATA"

cp /etc/postgresql/postgresql.conf "$PGDATA/postgresql.conf"
cp /etc/postgresql/pg_hba.conf "$PGDATA/pg_hba.conf"
echo "Conf files copied"

pg_ctl -D "$PGDATA" start
```

postgresql.conf

```
listen_addresses = '*'
hot_standby = on
primary_conninfo = 'host=primary port=5432 user=replicator
password=replicator_password'
```

```
wal_log_hints = on
log_connections = on
log_disconnections = on
log_duration = on
wal_log_hints = on
```

pg_hba.conf

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             all                                     trust
host    all             all             0.0.0.0/0               md5
host    replication     replicator      0.0.0.0/0               md5
```

Запуск

Запуск primary:

```
docker-compose up -d --build primary
```

```
 ⇒ ⇒ exporting layers                                                                    0.0s
 ⇒ ⇒ writing image sha256:a41042ed9f5bd426a401ebfb2b072a10e6b5b52014612903011dad5ef7e791c9  0.0s
 ⇒ ⇒ naming to docker.io/library/lab4-primary                                            0.0s
 ⇒ [primary] resolving provenance for metadata file                                      0.0s
[+] Running 3/3d orphan containers ([lab4-master-1]) for this project. If you removed or renamed this s
 ✔ Service primary      Built                                                            0.4s
 ✔ Network lab4_pg_net  Created                                                          0.0s
 ✔ Container primary    Started                                                          0.3s
```

Ожидаем лога "database system is ready to accept connections":

```
docker-compose logs primary
```

```
primary  |
primary  | PostgreSQL Database directory appears to contain a database; Skipping initialization
primary  |
primary  | 2025-05-05 02:16:11.488 UTC [1] LOG:  starting PostgreSQL 17.4 (Debian 17.4-1.pgdg120+2) on x86_64-pc
-linux-gnu, compiled by gcc (Debian 12.2.0-14) 12.2.0, 64-bit
primary  | 2025-05-05 02:16:11.488 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
primary  | 2025-05-05 02:16:11.488 UTC [1] LOG:  listening on IPv6 address "::", port 5432
primary  | 2025-05-05 02:16:11.490 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
primary  | 2025-05-05 02:16:11.494 UTC [28] LOG:  database system was shut down at 2025-05-05 02:16:07 UTC
primary  | 2025-05-05 02:16:11.498 UTC [1] LOG:  database system is ready to accept connections
```

Запуск init-primary.sh:

```
docker exec -it primary bash -c "/home/init/init-primary.sh"
docker restart primary
```

```
docker exec -it primary bash -c "/home/init/init-primary.sh"
docker restart primary


CREATE ROLE
CREATE DATABASE
You are now connected to database "test" as user "postgres".
CREATE TABLE
CREATE TABLE
INSERT 0 2
INSERT 0 2
Configuration files copied!
primary
```

Запуск standby:

```
docker-compose up -d --build standby
```

```
 ⇒ ⇒ exporting layers
 ⇒ ⇒ writing image sha256:87bf09f0f67da7a683814a7a2790c003567
 ⇒ ⇒ naming to docker.io/library/lab4-standby
[+] Running 4/4esolving provenance for metadata file
 ✔ Service primary    Built
 ✔ Service standby    Built
 ✔ Container primary  Running
 ✔ Container standby  Started
```

Наполнение базы

На примере не менее, чем двух таблиц, столбцов, строк, транзакций и клиентских сессий, `init-db.sql`:

```
CREATE DATABASE test;
\c test;
CREATE TABLE users (id SERIAL PRIMARY KEY, name VARCHAR(255));
CREATE TABLE orders ( id SERIAL PRIMARY KEY, user_id INT REFERENCES
users(id), product VARCHAR(255));
INSERT INTO users (name) VALUES ('Alice'), ('Bob');
INSERT INTO orders (user_id, product) VALUES (1, 'Laptop'), (2,
'Smartphone');
```

Проверим данные на primary:

```
docker exec -it primary psql -U postgres -d test -c "SELECT * FROM
users;"
docker exec -it primary psql -U postgres -d test -c "INSERT INTO users
(name) VALUES ('Bradley');"
```

```
docker exec -it primary psql -U postgres -d test -c "SELECT * FROM users;"
docker exec -it primary psql -U postgres -d test -c "INSERT INTO users (name) VALUES ('Bradly');"


 id | name
----+------
  1 | Alice
  2 | Bob
(2 rows)

INSERT 0 1
```

Проверим режим чтения на standby:

```
docker exec -it standby psql -U postgres -d test -c "SELECT * FROM
users;"
```

```
docker exec -it standby psql -U postgres -d test -c "SELECT * FROM users;"


 id |  name
----+--------
  1 | Alice
  2 | Bob
  3 | Bradly
(3 rows)
```

Попробуем записать данные на standby:

```
docker exec -it standby psql -U postgres -d test -c "INSERT INTO users
(name) VALUES ('Bradley');"
```

```
docker exec -it standby psql -U postgres -d test -c "INSERT INTO users (name) VALUES ('Bradly');"


ERROR:  cannot execute INSERT in a read-only transaction
```

Этап 2. Симуляция и обработка сбоя

2.1 Подготовка

Скрипты для симуляции чтения и записи из primary и чтение из standby:

read_client.sh

```
#!/bin/bash
while true; do
    psql -U postgres -d test -c "SELECT * FROM users;"
    psql -U postgres -d test -c "SELECT * FROM orders;"
    sleep 2
done
```

write_client.sh

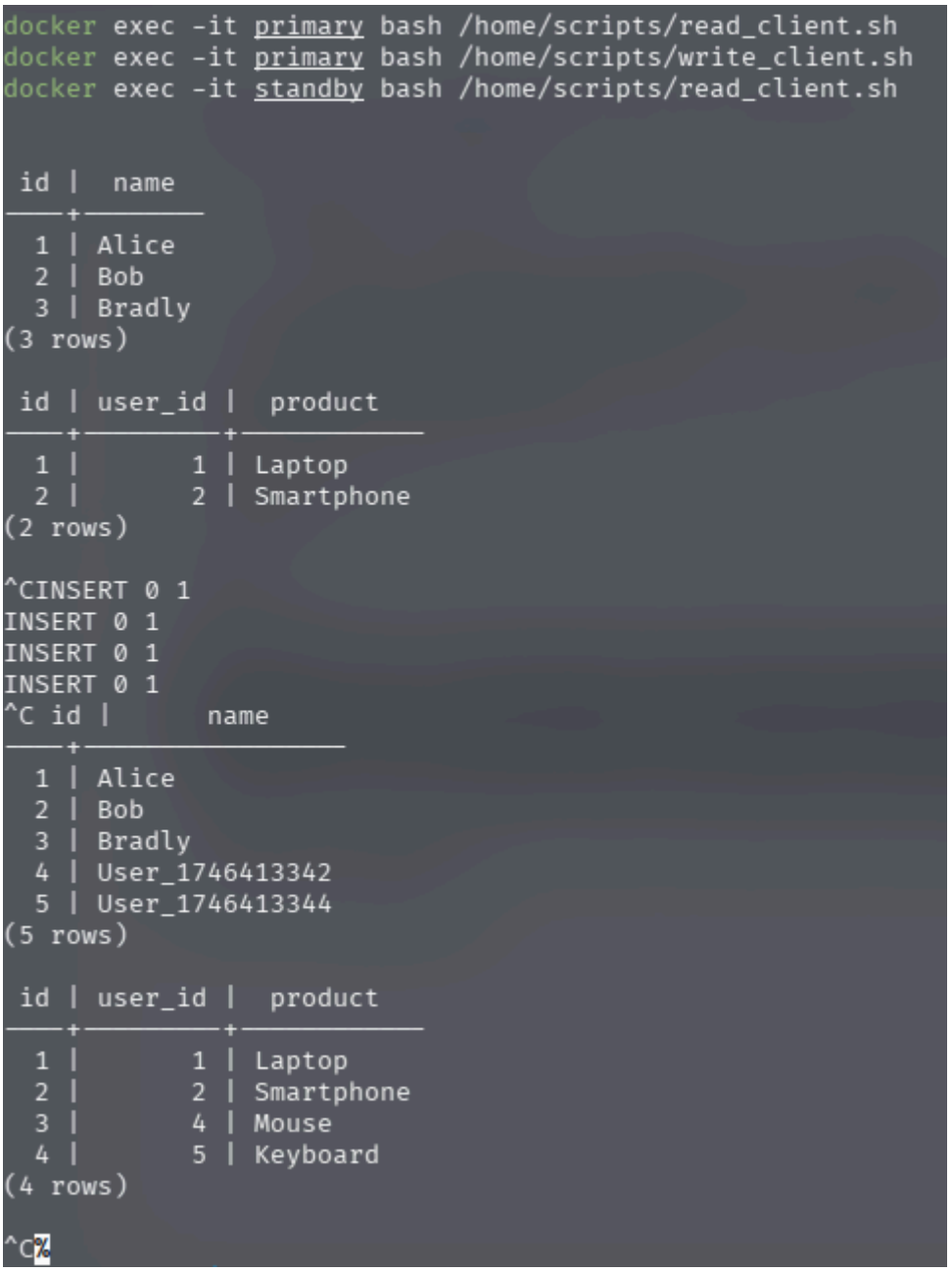
```
#!/bin/bash
while true; do
    psql -U postgres -d test -c "INSERT INTO users (name) VALUES
('User_$(date +%s)');"

    items=("TV" "Mouse" "Keyboard" "HDMI cable")
    random_item=${items[$RANDOM % ${#items[@]}]}
    last_user_id=$(psql -U postgres -d test -t -c "SELECT id FROM users
ORDER BY id DESC LIMIT 1;")
    psql -U postgres -d test -c "INSERT INTO orders (user_id, product)
VALUES ($last_user_id, '$random_item');"
    sleep 2
done
```

Запуск:

```
docker exec -it primary bash /home/scripts/read_client.sh
docker exec -it primary bash /home/scripts/write_client.sh
docker exec -it standby bash /home/scripts/read_client.sh
```

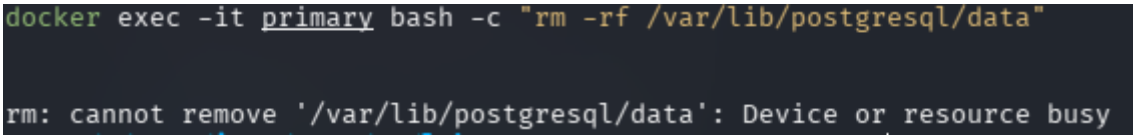
Ожидаем что в standby чтение будет автоматически показывать новые данные из primary:



2.2 Сбой

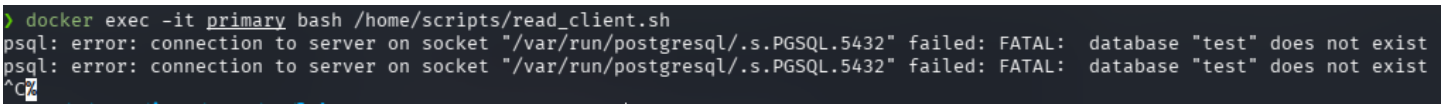
Симулируем ошибку диска на основном узле – удаляем директорию PGDATA со всем содержимым.

```
docker exec -it primary bash -c "rm -rf /var/lib/postgresql/data"
```



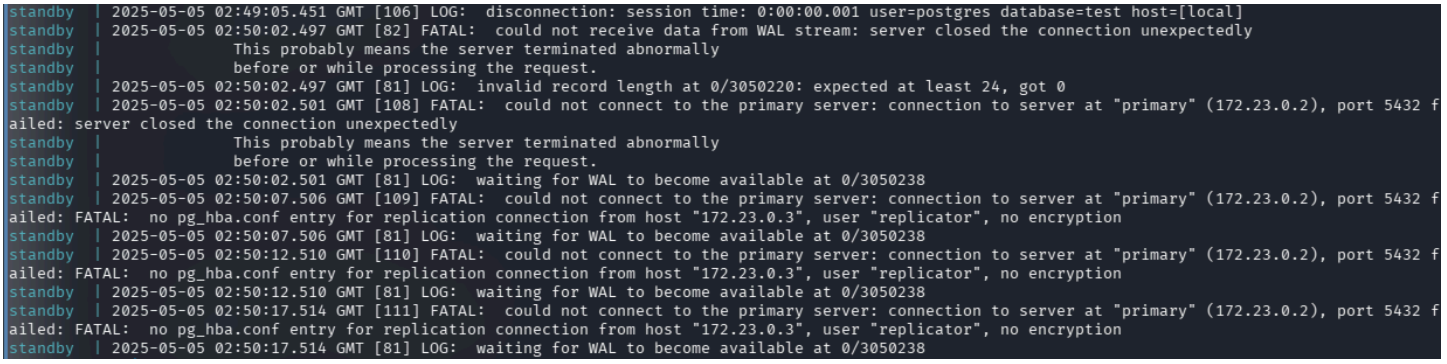
При этом чтение в primary будет невозможно:

```
docker exec -it primary bash /home/scripts/read_client.sh
```



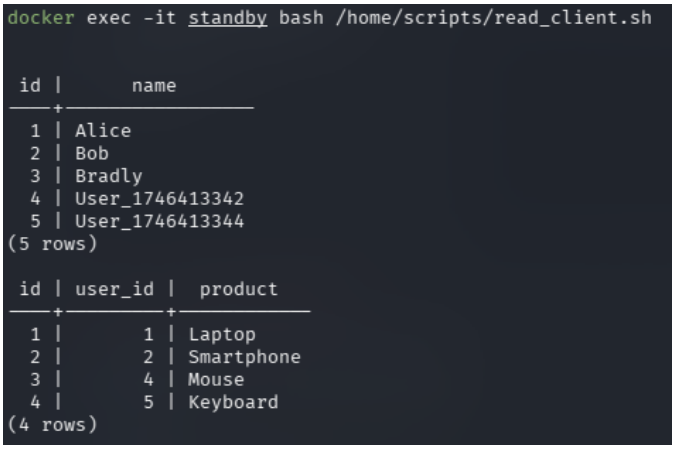
2.3 Обработка

В standby видим логи:



При этом чтение работает:

```
docker exec -it standby bash /home/scripts/read_client.sh
```



При работающем standby запустим скрипт автоматического promote:

```
docker exec -d standby bash -c "/home/scripts/auto_promote.sh"
```

auto_promote.sh

```
#!/bin/bash
set -e
PRIMARY_HOST="primary"
PG_CONNECT_TIMEOUT=3
CHECK_INTERVAL=5
while true; do
    if ! pg_isready -h $PRIMARY_HOST -p 5432 -t $PG_CONNECT_TIMEOUT -q;
then
        echo "Primary is unavailable. Attempting promote..."
        if [ -f $PGDATA/standby.signal ]; then
            if pg_ctl promote -D $PGDATA; then
                echo "Promote successful. This node is now master."
                exit 0
            else
                echo "Promote failed!" >&2
                exit 1
            fi
        else
            echo "Not in standby mode. Cannot promote." >&2
            exit 1
        fi
    fi
    sleep $CHECK_INTERVAL
done
```

Проверим чтение и запись на standby после promote:

```
docker exec -it standby psql -U postgres -d test -c "INSERT INTO users
(name) VALUES ('Charlie');"
docker exec -it standby psql -U postgres -d test -c "SELECT * FROM
users;"
```

```
docker exec -it standby psql -U postgres -d test -c "INSERT INTO users (name) VALUES ('Charlie');"
docker exec -it standby psql -U postgres -d test -c "SELECT * FROM users;"


INSERT 0 1
 id |      name
----+-----------------
  1 | Alice
  2 | Bob
  3 | Bradly
  4 | User_1746413342
  5 | User_1746413344
 37 | Charlie
(6 rows)
```

Как и ожидалось, запись и чтение работают.

Этап 3. Восстановление

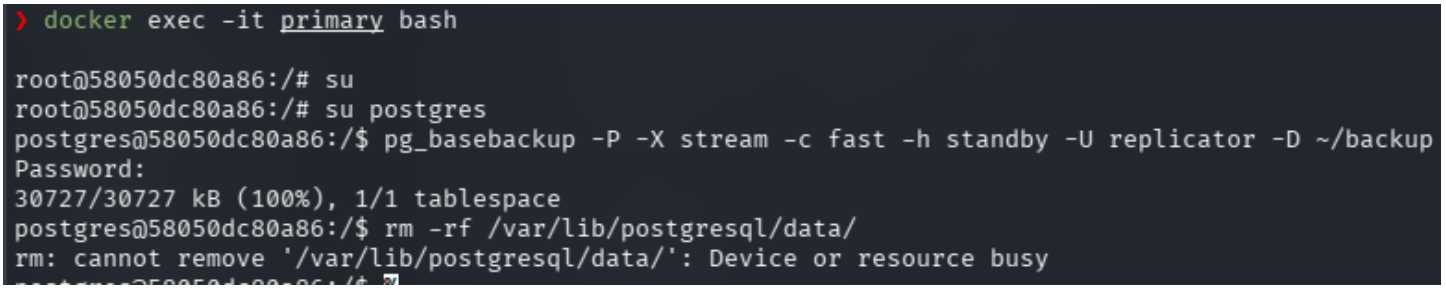
Восстанавливаем работу primary. В консоли primary:

```
docker exec -it primary bash

su
su postgres

pg_basebackup -P -X stream -c fast -h standby -U replicator -D ~/backup
# password: replicator_password

rm -rf /var/lib/postgresql/data/
mv ~/backup/* /var/lib/postgresql/data/
```



Выходим из контейнера primary.

В консоли standby:

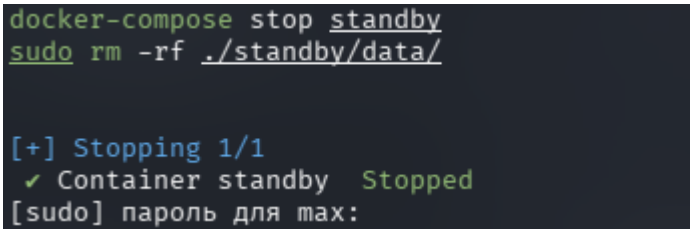
```
docker exec -it standby bash

touch /var/lib/postgresql/data/standby.signal
```

Выходим из контейнера standby.

Останавливаем standby:

```
docker-compose stop standby
sudo rm -rf ./standby/data/
```



Заново запускаем мастера:

```
docker-compose up -d primary
```

```
docker-compose up -d primary


WARN[0000] Found orphan containers ([lab4-master-1]) for this project. If y
and with the --remove-orphans flag to clean it up.
[+] Running 1/0
 ✔ Container primary  Running
```

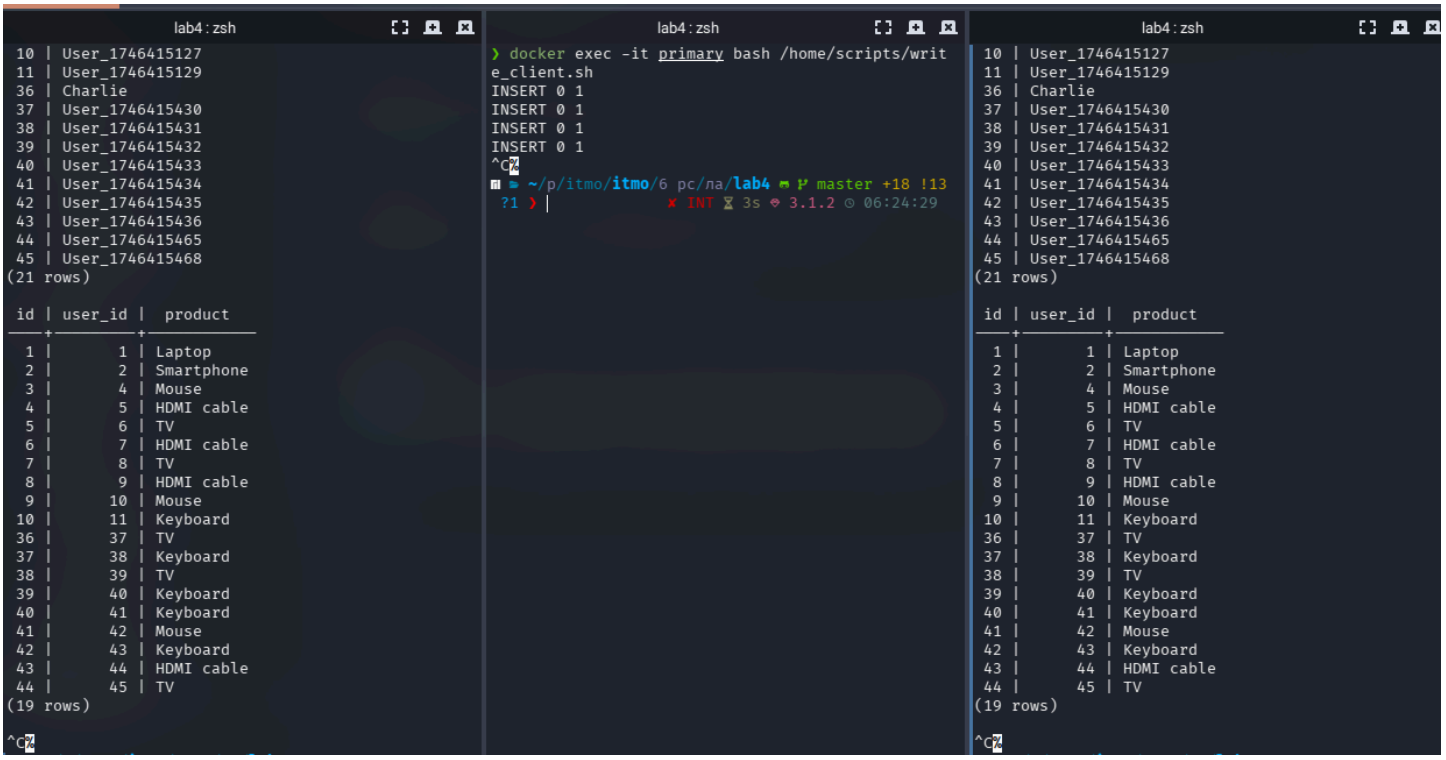
Пересоздаем standby чтобы заново стал standby:

```
docker-compose up -d --build standby
```

```
 ⇒ ⇒ writing image sha256:6088d50b31674bf6b18d719490528055839be014cf78f4e
 ⇒ ⇒ naming to docker.io/library/lab4-standby
[+] Running 4/4esolving provenance for metadata file
 ✔ Service primary    Built
 ✔ Service standby    Built
 ✔ Container primary  Running
 ✔ Container standby  Started
```

Проверяем:

```
docker exec -it primary bash /home/scripts/read_client.sh
docker exec -it primary bash /home/scripts/write_client.sh
docker exec -it standby bash /home/scripts/read_client.sh
```



Исходный код

https://github.com/maxbarsukov/itmo/tree/master/6%20pcxд/лабораторные/lab4

Вывод

В ходе лабораторной работы в сети виртуальных машин была успешно настроена отказоустойчивая конфигурация PostgreSQL с использованием трансляции логов между основным и резервным серверами для синхронизации двух кластеров. Проведена симуляция сбоя, отработано аварийное переключение на резервный узел с сохранением работоспособности системы. Восстановление основного сервера выполнено через повторную синхронизацию данных с резервного. Работа подтвердила важность репликации логов для минимизации потерь данных при отказах и продемонстрировала практические аспекты восстановления PostgreSQL-кластера. Полученные навыки позволяют эффективно обеспечивать отказоустойчивость СУБД в реальных условиях.