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 3](#_4wv7arx2x4xz)

[Выполнение 4](#_z5ouzacohqk7)

[Подготовка окружения 4](#_p7xzq2k699zz)

[Этап 1. Конфигурация 6](#_2gkk9zdm7y4j)

[primary](#_nfrs2am4jmyf) 6

[standby](#_jigp5ffzcwcm) 7

[Запуск 9](#_qglp3vq57vly)

[Наполнение базы 10](#_ofq7oqwewzwl)

[Этап 2. Симуляция и обработка сбоя 11](#_wcl09aujm2vt)

[2.1 Подготовка 11](#_9lsfh4yqatyo)

[2.2 Сбой 13](#_hqdgvtcijmlh)

[2.3 Обработка 13](#_mrkvc2rmsrdi)

[Этап 3. Восстановление 15](#_ydelrnterpup)

[Вывод 18](#_9nxi2mdwsuhw)

# **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 |
| --- |

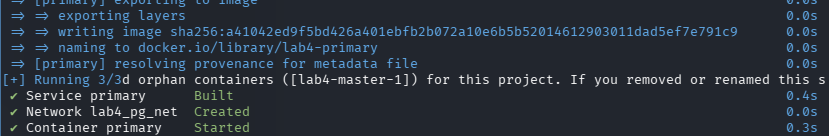
pg\_hba.conf

| # 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**

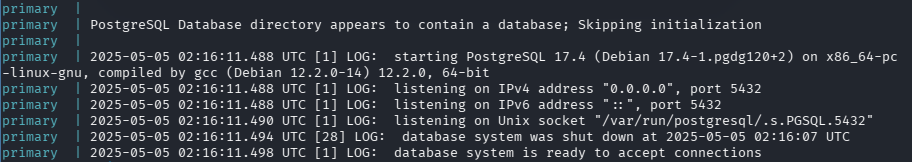
Запуск primary:

| docker-compose up -d --build primary |
| --- |



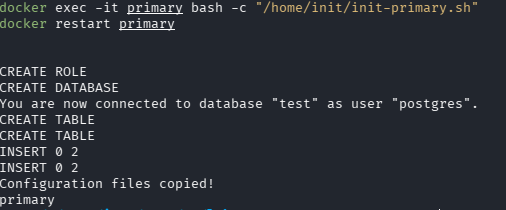
Ожидаем лога "database system is ready to accept connections":

| docker-compose logs primary |
| --- |

****

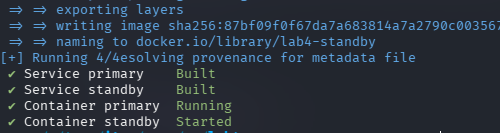
Запуск init-primary.sh:

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



Запуск standby:

| docker-compose up -d --build standby |
| --- |



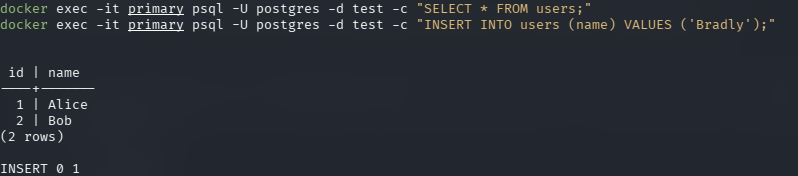
### **Наполнение базы**

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'); |
| --- |

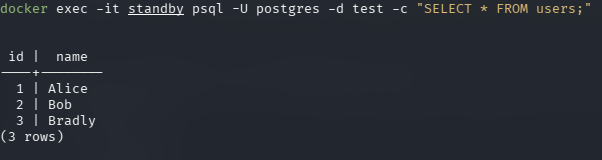
Проверим данные на primary:

| 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');" |
| --- |

****

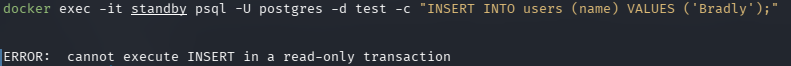
Проверим режим чтения на standby:

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

****

Попробуем записать данные на standby:

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

****

## **Этап 2. Симуляция и обработка сбоя**

### **2.1 Подготовка**

Cкрипты 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 |
| --- |

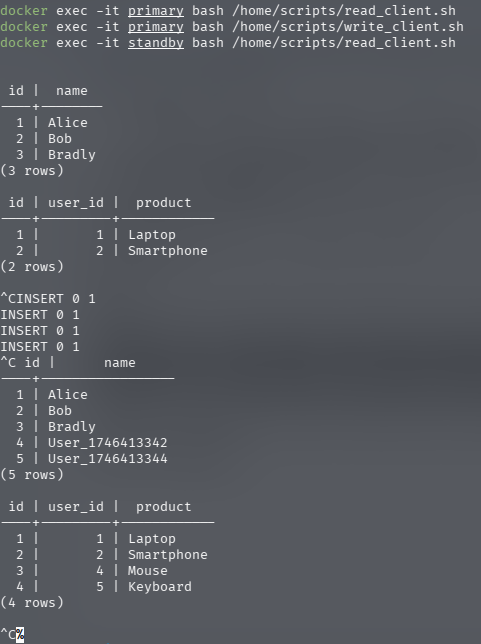
write\_client.sh

| #!/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 |
| --- |

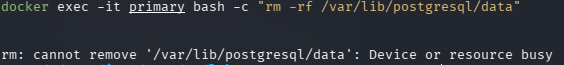
Ожидаем что в standby чтение будет автоматически показывать новые данные из primary:

****

### **2.2 Сбой**

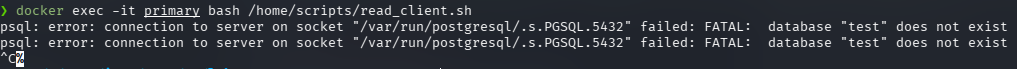
Симулируем ошибку диска на основном узле – удаляем директорию PGDATA со всем содержимым.

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

****

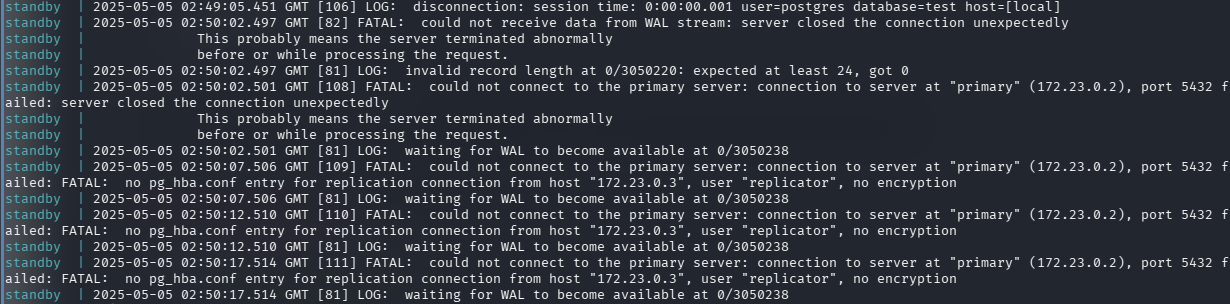
При этом чтение в primary будет невозможно:

| docker exec -it primary bash /home/scripts/read\_client.sh |
| --- |

****

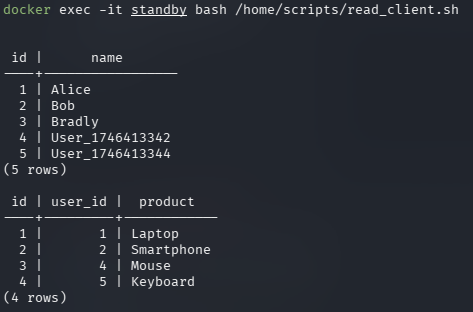
### **2.3 Обработка**

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:

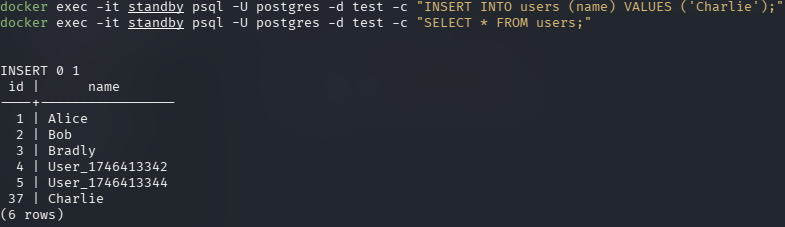
| docker exec -d standby bash -c "/home/scripts/auto\_promote.sh" |
| --- |

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;" |
| --- |

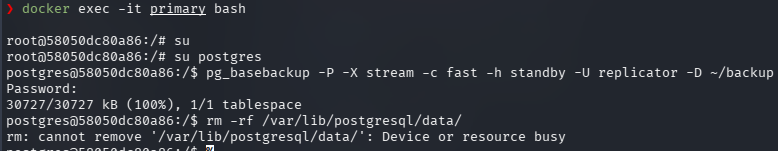
****

Как и ожидалось, запись и чтение работают.

## **Этап 3. Восстановление**

Восстанавливаем работу primary. В консоли primary:

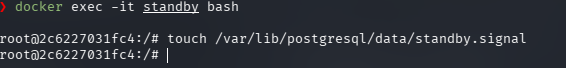
| docker exec -it primary bash  su  su postgres  pg\_basebackup -P -X stream -c fast -h standby -U replicator -D ~/backup  # password: replicator\_password  rm -rf /var/lib/postgresql/data/  mv ~/backup/\* /var/lib/postgresql/data/ |
| --- |

****

Выходим из контейнера primary.

В консоли standby:

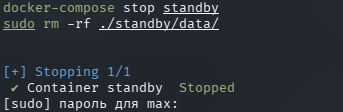
| docker exec -it standby bash  touch /var/lib/postgresql/data/standby.signal |
| --- |

****

Выходим из контейнера standby.

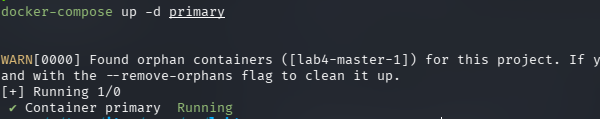
Останавливаем standby:

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

****

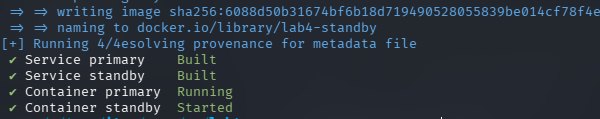
Заново запускаем мастера:

| docker-compose up -d primary |
| --- |

****

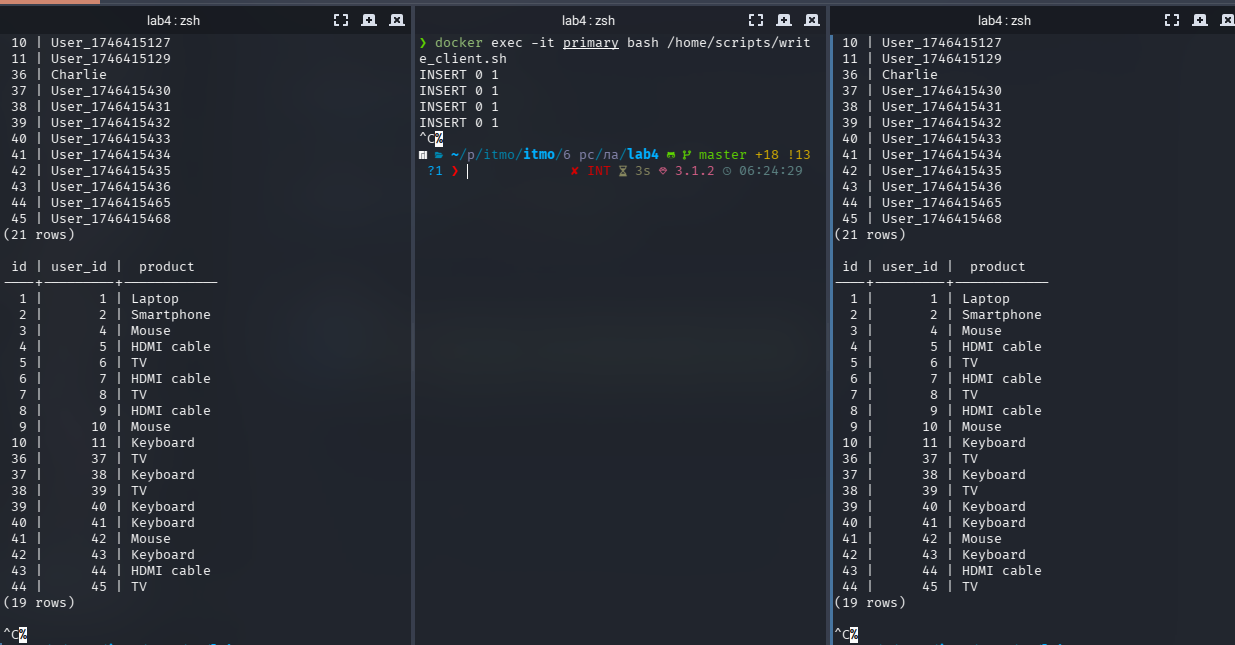
Пересоздаем standby чтобы заново стал standby:

| docker-compose up -d --build standby |
| --- |

****

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%20рсхд/лабораторные/lab4](https://github.com/maxbarsukov/itmo/tree/master/6%20%D1%80%D1%81%D1%85%D0%B4/%D0%BB%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D1%8B%D0%B5/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.