МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине

«РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ ХРАНЕНИЯ ДАННЫХ»

Выполнили:
Студенты группы P3318
Рамеев Тимур Ильгизович
Горло Евгений Николаевич
Преподаватель:
Заболотная Ольга Михайловна

Санкт-Петербург 2025

Оглавление

Оглавление 2
Задание 3
Требования: 3
Этапы: 3
Выполнение 4
Этап 1. Конфигурация 4
Этап 2. Симуляция и обработка сбоя. 8
Этап 3. Восстановление 10
Вывод 13

Задание

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

Требования:

- 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.

Этапы:

Этап 1. Конфигурация

Настроить репликацию postgres на трёх узлах: A - основной, B и C - резервные. Для управления использовать pgpool-II. Репликация с A на B синхронная. Репликация с A на C асинхронная. Продемонстрировать, что новые данные реплицируются на B в синхронном режиме, а на C с задержкой.

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 неожиданное отключение основного узла - выполнить Power Off виртуальной машины.

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.

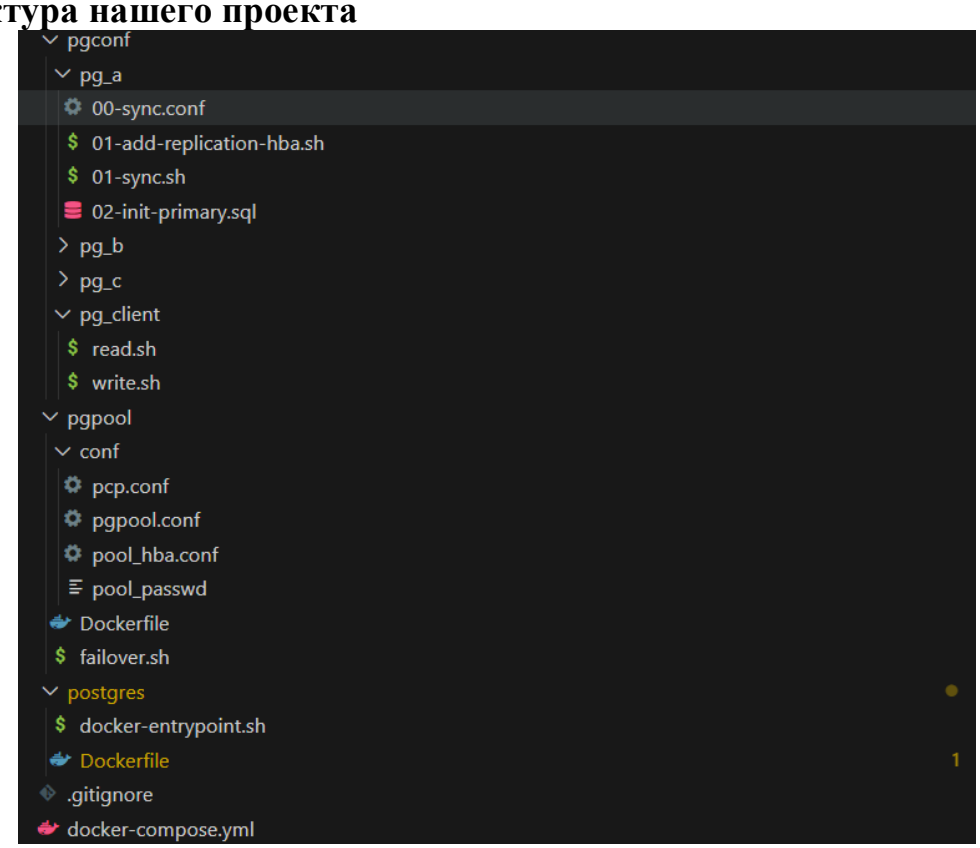
Выполнение

Этап 1. Конфигурация

Репозиторий с кодом: https://github.com/Stt1xX/ReplicaPostgreSQL
Работа была выполнена с использованием платформы **Docker**. Было развернуто 5 виртуальных машин:

1. Pg-a контейнер primary-backend основной узел.
2. Pg-b контейнер standby-backend вспомогательный узел, с синхронной репликацией.
3. Pg-c контейнер standby-backend вспомогательный узел, с асинхронной репликацией
4. Pg-pool контейнер с запущенным pg-pool-ll, через который идет взаимодействие с кластером.
5. Pg-client контейнер который нагружает наш кластер клиентскими сессиями.

Структура нашего проекта



1. docker-compose.yml – основной docker – файл для сборки и запуска всех контейнеров.
2. Pgconf/pg-a | pg-b |pg-c pg-client – директории, файлы которых нужны для корректной работы соответствующих контейнеров. Все эти файлы будут вмонтированы в соответствующие контейнеры, а затем выполнены.
    a. 00-sync.conf – фрагмент параметров для указания узла pg-b как синхронной реплики.
    b. 01-add-replication-hba.sh – скрипт, который указывает несколько правил для pg-hba.conf, благодаря которым наши реплики смогут беспрепятственно подключиться к pg-a и сделать basebackup.
    c. 01-sync.sh – скрипт, который вставит фрагмент 00-sync.conf (п. a) в postgresql.conf
    d. 02-init-primary.sql – sql-скрипт с тестовыми данными + созданием бд и роли replicator
    e. Read.sh – sh скрипт эмулирующий постоянные запросы на чтение тестовых данных из кластера (через pgpool)
    f. Write.sh – sh скрипт эмулирующий постоянные запросы на запись новых тестовых данных в кластер (через pgpool)
    g. Restore-as-replica.sh – sh скрипт, позволяющий восстановить pg-a узел после его подения (восстановить в состояние реплики)
3. Pgpool/conf Все эти файлы будут вмонтированы в контейнер pgpool, а затем выполнены.
    a. Pgpool.conf – файл, содержащий основные настройки нашего кластера (pgpool-a). Данные обо всех узлах кластера, порты, credentials, failover command и т. д.
    b. Pool-hba.conf – файл, содержит данные о аутентификации на шаге клиент – pgpool (разрешено всем в сети docker).
    c. Pool-passwd – файл, содержит логин и пароль для подключения pgpool к узлам. (У узлов эти credentials должны быть)
    d. Dockerfile – файл-конфиг для настройки + установки компонентов в наш контейнер pgpool.
    e. Failover.sh – скрипт, который будет обрабатывать failover.
4. Postgres
    a. docker-entrypoint.sh – скрипт кастомный entrypoint, вызывается автоматически при создании контейнеров pg-a, pg-b, pg-c. Если он понимает, что он запущен в рамках создания реплики (pg-b, pg-c) выполняет basebackup основного узла.
    b. Dockerfile – файл-конфиг для настройки + установки компонентов в наши контейнеры pg-a, pg-b, pg-c

Итак, выше была кратко описана структура проекта (подробнее ее можно изучить в репозитории (ссылка тоже выше)). Далее мы переходим к работе с проектом.

Сборка проекта:

```
PS C:\Users\ramti\Desktop\ITMO\Studies\DistributedDataStorageSystems\Lab4> docker-compose build --no-cache
```

Запуск проекта:

```
PS C:\Users\ramti\Desktop\ITMO\Studies\DistributedDataStorageSystems\Lab4> docker-compose up
[+] Running 1/1
 ✓ client Pulled
[+] Running 9/9
 ✓ Network lab4_pgnet        Created
 ✓ Volume "lab4_pg-a-data"   Created
 ✓ Volume "lab4_pg-b-data"   Created
 ✓ Volume "lab4_pg-c-data"   Created
 ✓ Container pg-a            Created
 ✓ Container pg-c            Created
 ✓ Container pg-b            Created
 ✓ Container pgpool          Created
 ✓ Container pg-client       Created
```

После того, как мы запустили наш проект, можем подключиться к основному узлу через pg-pool и посмотреть информацию о репликах:

```
PS C:\Users\ramti\Desktop\ITMO\Studies\DistributedDataStorageSystems\Lab4> docker exec -it pg-client /bin/bash
root@1f57a1e09f9b:/# psql -h pgpool -p 9999 -U postgres -d demo
psql (15.14 (Debian 15.14-1.pgdg13+1))
Type "help" for help.

demo=#
```

```
demo=# SELECT application_name, client_addr, sync_state
FROM pg_stat_replication;
 application_name | client_addr | sync_state
------------------+-------------+------------
 pg_b             | 172.21.0.3  | sync
 pg_c             | 172.21.0.4  | async
(2 rows)
```

Также можем подключиться к контейнеру с pg-pool и через pcp утилиту посмотреть состояние наших узлов:

```
bfec856d6bf0:/opt/pgpool-II/bin# /opt/pgpool-II/bin/pcp_node_info -U pgpool
Password:
pg-a 5432 1 0.333333 waiting up primary primary 0 none none 2025-09-16 23:44:22
pg-b 5432 1 0.333333 waiting up standby standby 0 none none 2025-09-16 23:44:22
pg-c 5432 1 0.333333 waiting up standby standby 0 none none 2025-09-16 23:44:22
```

Для наглядности можем провести эксперимент, подтверждающий статус синхронной\асинхронной реплики:

При отключении B реплики, A реплика не может подтвердить успешную запись на B, что приводит к зависанию. При включении B, работа продолжается. А при отключении C реплики, работа продолжается в штатном режиме:

```
demo=# insert into accounts values (10, 'bob', 12);
```

```
demo=# insert into accounts values (9, 'bob', 12);
INSERT 0 1
```

Это абсолютно нормальное поведение.

Ну и можем проверить, что наши реплики актуальны – скрипт docker-entrypoint.sh сработал, и в них есть данные, которые мы вносили в pg-a:

```
PS C:\Users\ramti\Desktop\ITMO\Studies\DistributedDataStorageSystems\Lab4> docker exec -it pg-b psql -U postgres -d demo -c "select * from accounts"
 id | name  | balance
----+-------+---------
  1 | alice |     100
  2 | bob   |      50
(2 rows)

PS C:\Users\ramti\Desktop\ITMO\Studies\DistributedDataStorageSystems\Lab4> docker exec -it pg-a psql -U postgres -d demo -c "select * from accounts"
 id | name  | balance
----+-------+---------
  1 | alice |     100
  2 | bob   |      50
(2 rows)
```

Для демонстрации нескольких клиентских соединений написаны два скрипта read.sh и write.sh:

pgconf > pg_client > $ read.sh

```
#!/bin/bash

while true; do
    echo "[$(date)] Reading current data..."
    psql -h pgpool -p 9999 -U postgres -d demo <<EOF
SELECT * FROM accounts ORDER BY id DESC LIMIT 5;
SELECT * FROM logs ORDER BY created_at DESC LIMIT 5;
EOF

    sleep 5
done
```

pgconf > pg_client > $ write.sh

```
#!/bin/bash

while true; do
    echo "[$(date)] Inserting new data..."
    psql -h pgpool -p 9999 -U postgres -d demo <<EOF
INSERT INTO accounts (name, balance) VALUES ('user_$(date +%s)', (random()*100)::int);
INSERT INTO logs (note) VALUES ('inserted via pgpool at $(date)');
EOF

    sleep 5
done
```

Скрипты запускаются с клиента и обращаются к pg-pool:

```
PS C:\Users\ramti\Desktop\ITMO\Studies\DistributedDataStorageSystems\Lab4> docker exec -it pg-client /bin/bash
root@1f57a1e09f9b:/# /scripts/write.sh &
[1] 37
root@1f57a1e09f9b:/# /scripts/read.sh &
[2] 38
root@1f57a1e09f9b:/# [Wed Sep 17 02:30:46 AM UTC 2025] Inserting new data...
[Wed Sep 17 02:30:46 AM UTC 2025] Reading current data...
 id | name  | balance
----+-------+---------
  2 | bob   |      50
  1 | alice |     100
(2 rows)

 id | note  |          created_at
----+-------+-------------------------------
  1 | init1 | 2025-09-17 02:22:23.133058+00
  2 | init2 | 2025-09-17 02:22:23.133058+00
(2 rows)

INSERT 0 1
INSERT 0 1
[Wed Sep 17 02:30:51 AM UTC 2025] Reading current data...
[Wed Sep 17 02:30:51 AM UTC 2025] Inserting new data...
 id |      name       | balance
----+-----------------+---------
  3 | user_1758076246 |       4
  2 | bob             |      50
  1 | alice           |     100
```

Этап 2. Симуляция и обработка сбоя.

Для симуляции сбоя достаточно выключить контейнер с основным узлом:

```
PS C:\Users\ramti\Desktop\ITMO\Studies\DistributedDataStorageSystems\Lab4> docker stop pg-a
pg-a
```

Давайте посмотрим логи pgpool-a:

```
PS C:\Users\ramti\Desktop\ITMO\Studies\DistributedDataStorageSystems\Lab4> docker logs pgpool
2025-09-17 05:55:59.664: health_check0 pid 79: LOG:  health check failed on node 0 (timeou
t:0)
2025-09-17 05:55:59.664: health_check0 pid 79: LOG:  received degenerate backend request f
or node_id: 0 from pid [79]
2025-09-17 05:55:59.664: health_check0 pid 79: LOG:  signal_user1_to_parent_with_reason(0)
2025-09-17 05:55:59.664: main pid 59: LOG:  Pgpool-II parent process received SIGUSR1
2025-09-17 05:55:59.664: main pid 59: LOG:  Pgpool-II parent process has received failover
 request
2025-09-17 05:55:59.665: main pid 59: LOG:  === Starting degeneration. shutdown host pg-a(
5432) ===
2025-09-17 05:55:59.677: main pid 59: LOG:  Restart all children
2025-09-17 05:55:59.677: main pid 59: LOG:  execute command: /scripts/failover.sh  0 pg-a
 5432 /var/lib/postgresql/data  1 pg-b 0 0 5432 /var/lib/postgresql/data pg-b 0
[Failover script] Failover triggered!
[Failover script] New primary: pg-b (5432)
[Failover script] Old primary: pg-a (5432)
ALTER SYSTEM
ALTER SYSTEM
server signaled
2025-09-17 05:56:00.190: sr_check_worker pid 78: WARNING:  failed to connect to PostgreSQL
 server, getaddrinfo() failed with error "Try again"
waiting for server to promote..... done
server promoted
[Failover script] Promoted pg-b
[Failover script] Reconfiguring standby pg-c to follow pg-b...
server signaled
[Failover script] Failover completed.
2025-09-17 05:56:06.514: main pid 59: LOG:  find_primary_node_repeatedly: waiting for find
ing a primary node
2025-09-17 05:56:06.540: main pid 59: LOG:  find_primary_node: primary node is 1
2025-09-17 05:56:06.540: main pid 59: LOG:  find_primary_node: standby node is 2
2025-09-17 05:56:06.540: main pid 59: LOG:  failover: set new primary node: 1
2025-09-17 05:56:06.540: main pid 59: LOG:  failover: set new main node: 1
2025-09-17 05:56:06.543: main pid 59: LOG:  === Failover done. shutdown host pg-a(5432) ==
=
```

В логах четко видно, как pgpool теряет связь с основным узлом, происходит триггер скрипта failover, который по сети вызывает promote standby узла.

Несмотря на то, что мы отключили наш основной узел, скрипты, которые мы вызвали на стороне клиента, продолжают работать, после небольшого перебоя:

```
 16 | inserted via pgpool at Wed Sep 17 05:55:45 AM UTC 2025 | 2025-09-17 05:55:45.971314+
00
 15 | inserted via pgpool at Wed Sep 17 05:55:42 AM UTC 2025 | 2025-09-17 05:55:43.017161+
00
 14 | inserted via pgpool at Wed Sep 17 05:55:40 AM UTC 2025 | 2025-09-17 05:55:40.869484+
00
 13 | inserted via pgpool at Wed Sep 17 05:55:37 AM UTC 2025 | 2025-09-17 05:55:37.91983+0
0
(5 rows)

[Wed Sep 17 05:55:53 AM UTC 2025] Inserting new data...
psql: error: connection to server at "pgpool" (172.19.0.5), port 9999 failed: server close
d the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
[Wed Sep 17 05:55:54 AM UTC 2025] Reading current data...
[Wed Sep 17 05:55:59 AM UTC 2025] Inserting new data...
psql: error: connection to server at "pgpool" (172.19.0.5), port 9999 failed: server close
d the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
psql: error: connection to server at "pgpool" (172.19.0.5), port 9999 failed: server close
d the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
[Wed Sep 17 05:56:04 AM UTC 2025] Inserting new data...
[Wed Sep 17 05:56:04 AM UTC 2025] Reading current data...
INSERT 0 1
INSERT 0 1
 id |      name       | balance
----+-----------------+---------
 37 | user_1758088564 |       5
 36 | user_1758088564 |      88
 18 | user_1758088550 |      27
 17 | user_1758088548 |      25
 16 | user_1758088545 |      51
```

Скрин не вставляем, но данные в нашем кластере не потерялись.

Состояние узлов наблюдаем через pcp:

```
e56e56a7a05e:/opt/pgpool-II/bin# ./pcp_node_info -U pgpool
Password:
pg-a 5432 3 0.333333 down down standby unknown 0 none none 2025-09-17 07:09:15
pg-b 5432 2 0.333333 up up primary primary 0 none none 2025-09-17 07:09:15
pg-c 5432 2 0.333333 up up standby standby 0 none none 2025-09-17 07:08:49
e56e56a7a05e:/opt/pgpool-II/bin#
```

Этап 3. Восстановление

Итак, для начала мы восстановили узел в состояние standby, но уже с актуальными данными. Для этого мы перезапускаем контейнер pg-a с переменной окружения - *RESTORE=REPLICA* .

```
# Если RESTORE=1, запускаем restore и выходим, не стартуем второй postgres
if [ "$RESTORE" = "1" ]; then
  echo "[entrypoint] RESTORE mode активирован, запускаем restore_as_replica.sh"
  /scripts/restore_as_replica.sh
  exec docker-entrypoint.sh postgres
fi
```

Наш кастомный entrypoint при наличии переменной окружения RESTORE запускает особый скрипт restore_as_replica.sh, который очищает нерелевантный узел, делает basebackup с нынешнего primary узла и запускает его в режиме standby (standby.signal).

docker-compose build pg-a
docker-compose up pg-a

Немного логов:

```
pg-a  | [entrypoint] RESTORE mode активирован, запускаем restore_as_replica.sh
pg-a  | [restore] --- Начало восстановления ---
pg-a  | [restore] Остановка PostgreSQL...
pg-a  | pg_ctl: could not send stop signal (PID: 34): No such process
pg-a  | [restore] Ошибка остановки через pg_ctl
pg-a  | [restore] Очистка данных...
pg-a  | [restore] Получение базы с primary (pg-b)...
pg-a  | pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg-a  | pg_basebackup: checkpoint completed
pg-a  | pg_basebackup: write-ahead log start point: 0/F000028 on timeline 2
pg-a  | pg_basebackup: starting background WAL receiver
pg-a  | pg_basebackup: created temporary replication slot "pg_basebackup_2644"
pg-a  |    80/30787 kB (0%), 0/1 tablespace (.../lib/postgresql/data/base/4/2675)
pg-a  | 30798/30798 kB (100%), 0/1 tablespace (...ostgresql/data/global/pg_control)
pg-a  | 30798/30798 kB (100%), 1/1 tablespace
pg-a  | pg_basebackup: write-ahead log end point: 0/F000100
pg-a  | pg_basebackup: waiting for background process to finish streaming ...
pg-a  | pg_basebackup: syncing data to disk ...
pg-a  | pg_basebackup: renaming backup_manifest.tmp to backup_manifest
pg-a  | pg_basebackup: base backup completed
pg-a  | [restore] Создание standby.signal...
pg-a  | [restore] standby.signal создан
pg-a  | [restore] Запись primary_conninfo...
pg-a  | [restore] --- Восстановление завершено ---
```

Теперь можем заметить, что узел pg-a запущен в slave режиме:

```
PS C:\Users\ramti\Desktop\ITMO\Studies\DistributedDataStorageSystems\Lab4> docker exec -it pgpool
 /opt/pgpool-II/bin/pcp_node_info -U pgpool
Password:
pg-a 5432 2 0.333333 up up standby standby 0 none none 2025-09-17 10:59:40
pg-b 5432 2 0.333333 up up primary primary 0 none none 2025-09-17 10:59:40
pg-c 5432 2 0.333333 up up standby standby 0 none none 2025-09-17 10:59:40
PS C:\Users\ramti\Desktop\ITMO\Studies\DistributedDataStorageSystems\Lab4>
```

Теперь нам осталось вернуть прежнюю конфигурацию: узел pg-a должен вновь стать primary, а узел pg-b должен стать standby.

Для этого мы перезагружаем наш узел pg-a с переменной окружения *RESTORE=PRIMARY*. Данная процедура также запускает скрипт promote_and_reconfigure.sh, который осуществляет promote узла a и basebackup реплик. Логов очень много, поэтому прикреплять смысла не видим, можно будет показать на сдаче. Результатом этой операции будет такая конфигурация кластера:

```
PS C:\Users\ramti\Desktop\ITMO\Studies\DistributedDataStorageSystems\Lab4> docker exec -it pgpool /opt/pgpool-II/bin/pcp_node_info -U pgpool
Password:
pg-a 5432 2 0.333333 up up primary primary 0 none none 2025-09-19 08:22:24
pg-b 5432 2 0.333333 up up standby standby 0 none none 2025-09-19 08:22:24
pg-c 5432 2 0.333333 up up standby standby 0 none none 2025-09-19 08:22:24
```

Для тестирования системы мы создали powershell скрипт *reset_cluster.ps1*, который исполняет задачи, поставленные в лр. Также скрипт на каждом этапе создает нагрузку на сервер в виде запросов на чтение и запись.

```
$composeFile = "docker-compose.yml"

(Get-Content $composeFile) -replace '(^\s*#?\s*- RESTORE=.*)', '# $1' |
    Set-Content $composeFile

docker compose down -v
Start-Sleep -Seconds 3
docker compose build
Start-Sleep -Seconds 10
docker compose up -d
Start-Sleep -Seconds 10

docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "INSERT INTO accounts(name, balance) VALUES ('volodya', 1000);"
docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "INSERT INTO accounts(name, balance) VALUES ('nicolya', 1000);"
docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "INSERT INTO accounts(name, balance) VALUES ('bodyan', 1000);"
docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "SELECT * FROM accounts;"

(Get-Content $composeFile) -replace '^\s*#?\s*- RESTORE=.*', '      - RESTORE=REPLICA' |
    Set-Content $composeFile

# 8. Запустить pg-a
docker compose up -d --force-recreate pg-a
Start-Sleep -Seconds 10
```

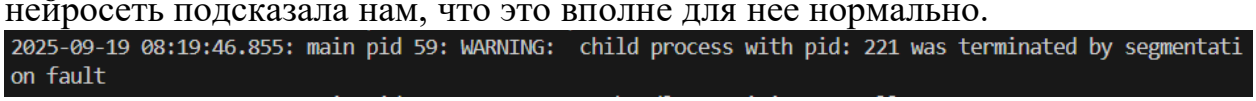
```
docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "INSERT INTO accounts(name, balance) VALUES ('bob', 1000);"
docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "INSERT INTO accounts(name, balance) VALUES ('pit', 1000);"
docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "INSERT INTO accounts(name, balance) VALUES ('taras', 1000);"
docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "INSERT INTO accounts(name, balance) VALUES ('tristan', 1000);"
docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "SELECT * FROM accounts;"

(Get-Content $composeFile) -replace '^\s*- RESTORE=.*', '      - RESTORE=PRIMARY' |
  Set-Content $composeFile

docker compose up -d --force-recreate pg-a
Start-Sleep -Seconds 10

docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "INSERT INTO accounts(name, balance) VALUES ('aaaaaa', 1000);"
docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "INSERT INTO accounts(name, balance) VALUES ('bbbbbb', 1000);"
docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "INSERT INTO accounts(name, balance) VALUES ('cccccc', 1000);"
docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "INSERT INTO accounts(name, balance) VALUES ('dddddd', 1000);"
docker exec pg-client psql -h pgpool -p 9999 -U postgres -d demo -c "SELECT * FROM accounts;"
```

В итоге потом можно посмотреть, что все данные были записаны:

```
PS C:\Users\ramti\Desktop\ITMO\Studies\DistributedDataStorageSystems\Lab4> docker exec -it pg-client psql -U postgres -d demo -p 9999 -h pgpool -c "select * from accounts;"
 id |  name   | balance
----+---------+---------
  1 | alice   |     100
  2 | bob     |      50
  3 | volodya |    1000
  4 | nicolya |    1000
  5 | bodyan  |    1000
 36 | bob     |    1000
 37 | pit     |    1000
 38 | taras   |    1000
 39 | tristan |    1000
 69 | aaaaaa  |    1000
 70 | bbbbbb  |    1000
 71 | dddddd  |    1000
(12 rows)
```

Стоит сразу оговориться, в ходе выполнения лр мы столкнулись с проблемой, связанной с устаревшей версией pgpool. На Docker более свежей

версии не нашлось, чем версия 4.4.2. Да можно было заморочиться и поставить актуальную версию через исходники, но мы этим заниматься не стали. Так вот, эта версия периодически выдает segmentation fault на клиентский запрос на чтение\запись. Мы думали, что это связано с неправильной конфигурацией сервера (буферов, кешей, памяти), однако похоже это не так. Другие пользователи также сталкивались с такой проблемой на этой версии, и нейросеть подсказала нам, что это вполне для нее нормально.



Вывод

Мы организовали полноценный распределенный кластер под управлением pgpool. Настроили корректную обработку failover. И научились полностью восстанавливать конфигурацию сервера после failover (switchover) без потери данных.