

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего
образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Дисциплина: Администрирование систем управления базами данных

Лабораторная работа №4
Вариант 23432

Выполнил:

Серебренникова В. В.
Быковченко С. А.

Группа:

P33202

Проверил:

Николаев В. В.

Санкт-Петербург
2024

Оглавление

Оглавление.....	2
Описание задания	3
Отчет	5
Этап 1.....	5
Этап 2.....	9
Этап 3.....	13
Выводы.....	15

Описание задания

Вариант 23432

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 горячего резерва (Master + Hot Standby). Не использовать дополнительные пакеты. Продемонстрировать доступ в режиме чтение/запись на основном сервере, в режиме чтение на резервном сервере, а также актуальность данных на нём.

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 недоступность основного узла - отключить сетевой интерфейс виртуальной машины, переключить его в изолированную подсеть и т.п.
- 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.

Отчет

Этап 1

Соберем докер-контейнеры с нашими образами Ubuntu 22.04 и запущенными на них PostgreSQL версии 15:

```
PS C:\Users\admin\Desktop\lab4> docker build -t asubd .
[+] Building 1.5s (14/14) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile              0.0s
=> => transferring dockerfile: 388B                               0.0s
=> [internal] load metadata for docker.io/library/ubuntu:22.04  1.4s
=> [auth] library/ubuntu:pull token for registry-1.docker.io    0.0s
=> [internal] load .dockerignore                                 0.0s
=> => transferring context: 2B                                     0.0s
=> [1/9] FROM docker.io/library/ubuntu:22.04@sha256:19478ce7fc2ffbbe89df29fea5725a8d12e57de52eb9ea570890dc5852aac1ac 0.0s
=> CACHED [2/9] RUN apt-get update                               0.0s
=> CACHED [3/9] RUN apt-get install -y apt-file                  0.0s
=> CACHED [4/9] RUN apt-file update                              0.0s
=> CACHED [5/9] RUN apt-get install -y vim                       0.0s
=> CACHED [6/9] RUN apt-get install -y nano                     0.0s
=> CACHED [7/9] RUN apt-get install -y iputils-ping             0.0s
=> CACHED [8/9] RUN apt-get install -y postgresql               0.0s
=> CACHED [9/9] RUN apt-get install -y postgresql-client        0.0s
=> exporting to image                                           0.0s
=> => exporting layers                                           0.0s
=> => writing image sha256:bf4de74b971b0bfbda0130e6ec167ea388c4726ccec6f29fdeefb231f444d396 0.0s
=> => naming to docker.io/library/asubd                          0.0s
```

Рис. 1 - Сборка контейнеров

```
PS C:\Users\admin\Desktop\lab4> docker compose up -d
[+] Running 2/3
- Network lab4_default Created
✓ Container master Started
✓ Container standby Started
PS C:\Users\admin\Desktop\lab4>
```

Рис. 2 - Сборка контейнеров, продолжение

Выполним следующие команды, так как в составе докерфайла докер их пропускает (по неизвестной причине):

```
# Update APT repository and install packages
RUN apt-get update
RUN apt-get install -y apt-file
RUN apt-file update
RUN apt-get install -y vim
```

Рис. 3 - Установка vim

Шаги на хосте MASTER

Создадим пользователя для репликации командой create user rep with replication password 'a':

```
postgres=# create user rep with replication password 'a';
CREATE ROLE
```

Рис. 4 – Создание пользователя для репликации

Изменим файл конфигурации pg_hba.conf командой vi var/lib/postgresql/data/pg_hba.conf. Добавим следующую строку:

```
host replication all 172.21.0.5/32 trust
```

Изменим файл конфигурации postgresql.conf командой vi var/lib/postgresql/data/postgresql.conf. Изменим следующие строки:

```
port = 5432
wal_level=replica
```

Перезагрузим контейнер и снова зайдем в нашу базу данных с мастер-узла. Создадим тестовую таблицу asdf:

```
postgres=# create table asdf (asdf text);
CREATE TABLE
postgres=# \du
```

Role name	Attributes	Member of
postgres	Superuser, Create role, Create DB, Replication, Bypass RLS	{}
rep	Replication	{}

```
postgres=# \d
```

Schema	Name	Type	Owner
public	asdf	table	postgres

```
(1 row)
```

Рис. 5 – Наполнение базы тестовыми данными

Шаги на хосте STANDBY

Очистим папку кластера командой `rm -rf /var/lib/postgresql/data/*`.

Создадим резервную копию с мастер-узла следующей командой:

```
pg_basebackup -D /var/lib/postgresql/data -h master -Xs -R -P -U postgres
```

Описание используемых параметров:

- `-Xs`. `-X` - задание метода копирования WAL-файлов. Параметр `s` (stream, поток) задает передачу WAL-файлов через второе соединение к серверу, через которое будет передаваться журнал предзаписи параллельно с созданием копии.
- `-R` - создает файл `standby.signal` и добавляет параметры конфигурации в файл `postgresql.auto.conf` в целевом каталоге.
- `-P` включает отчёт о прогрессе в процентах выполнения.

```
root@standby:/# pg_basebackup -D /var/lib/postgresql/data -h master -Xs -R -P -U postgres
23025/23025 kB (100%), 1/1 tablespace
root@standby:/# ls var/lib/postgresql/data
PS C:\Users\admin\Desktop\lab4> docker exec -it standby bash
root@standby:/# ls var/lib/postgresql/data
backup_label.old  pg_hba.conf      pg_serial         pg_twophase       postmaster.opts
backup_manifest   pg_ident.conf    pg_snapshots     PG_VERSION        postmaster.pid
base              pg_logical       pg_stat           pg_wal            standby.signal
global            pg_multixact     pg_stat_tmp      pg_xact
pg_commit_ts      pg_notify        pg_subtrans      postgresql.auto.conf
pg_dynshmem       pg_replslot      pg_tblspc        postgresql.conf
```

Рис. 6 – Выполнение бэкапа

Перезапустим контейнер.

Можем проверить, что параметры сработали верно – появился файл `recovery.signal`. Он пустой (так и надо), проверить это мы можем следующей командой:

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

Настроим файл конфигурации `postgresql.conf` на резервном узле командой `vi /var/lib/postgresql/data/postgresql.conf`. Изменим следующие параметры:

```
primary_conninfo = 'host=172.21.0.3 port=5432 user=rep'
hot_standby = on
```

Снова перезапустим standby. Логи на резервном узле при запуске:

```
2024-06-16 23:39:55 2024-06-16 20:39:55.406 UTC [29] LOG: entering standby mode
2024-06-16 23:39:55 2024-06-16 20:39:55.409 UTC [29] LOG: redo starts at 0/2000028
2024-06-16 23:39:55 2024-06-16 20:39:55.409 UTC [29] LOG: consistent recovery state reached at 0/3000000
2024-06-16 23:39:55 2024-06-16 20:39:55.409 UTC [1] LOG: database system is ready to accept read-only connections
2024-06-16 23:39:55 2024-06-16 20:39:55.419 UTC [30] LOG: started streaming WAL from primary at 0/3000000 on timeline 1
```

Рис. 7 - Логи

Проверим, как сработало резервирование:

```
PS C:\Users\admin\Desktop\lab4> docker exec -it standby bash
root@standby:/# su postgres
postgres@standby:/# psql
psql (15.7 (Debian 15.7-1.pgdg120+1))
Type "help" for help.

postgres=# \d
               List of relations
 Schema | Name  | Type  | Owner
-----+-----+-----+-----
 public | asdf  | table | postgres
(1 row)
```

Рис. 8 – Результат резервирования

Продemonстрируем работу узлов в режиме чтение-запись. Запишем строку в таблицу asdf на мастер-узле и проверим, что происходит на резервном:

```
postgres=# insert into asdf values ('asdf');
INSERT 0 1
```

Рис. 9 - Вставка данных на мастер-узле

```
postgres=# select * from asdf;
 asdf
-----
 asdf
(1 row)

postgres=#
```

Рис. 10 - Проверка данных на резервном узле

Всё работает!

Этап 2

Создаем новых клиентов `user_one`, `user_two` в базе данных на мастер-узле следующими командами:

- `create user user_one with password 'pswd';`
- `create user user_two with password 'pswd';`

Дадим пользователям все права на взаимодействия с таблицами и последовательностями в схеме “public”:

- `grant all privileges on all tables in schema public to user_one, user_two;`
- `grant all privileges on all sequences in schema public to user_one, user_two;`

Установим несколько подключений к основной базе данных на мастер-узле с машин наших пользователей (машина `user_one` у первого пользователя и машина `user_two` у второго):

```
root@user_one:/# psql -h master -p 5432 -U user_one postgres
Password for user user_one:
psql (15.7 (Debian 15.7-1.pgdg120+1))
Type "help" for help.

postgres=>
```

Рис. 1 – Подключение первого пользователя

```
root@user_two:/# psql -h master -p 5432 -U user_two postgres
Password for user user_two:
psql (15.7 (Debian 15.7-1.pgdg120+1))
Type "help" for help.

postgres=> █
```

Рис. 12 – Подключение второго пользователя

Создадим таблицу `cats` со столбцами `id`, `name`, `age`. Выполним тестовые записи в одну таблицу с разных пользователей. Проверим, будет ли возникать конфликт и отобразятся ли изменения на standby-узле.

user_one:

```
postgres=> insert into cats (name, age) values ('mark', 3);
INSERT 0 1
postgres=> █
```

Рис. 13 – Создание записи в таблице «Cats» первым пользователем

user_two:

```
postgres=> insert into cats (name, age) values ('meow', 4);
INSERT 0 1
postgres=>
```

Рис. 14 – Создание записи в таблице «Cats» вторым пользователем

Изменения в сущности cats видны с любого хоста.

```
postgres=# select * from cats;
 id | name  | age
----+-----+----
  1 | vasya |   1
  2 | misha |   2
  3 | petya |   3
  4 | mark  |   3
  5 | meow  |   4
(5 rows)

postgres=#
```

Рис. 15 – Проверка корректного отображения сущности с любого хоста

Симулируем недоступность основного узла. Отключим докер-контейнер с машиной, на которой запущен мастер-узел:

<input type="checkbox"/>	lab4	Running (3/4)	0.26%	28 minutes ago			
<input type="checkbox"/>	standby ae0c8672fe2d postgres:15	Running	0%	8080:8080 28 minutes ago			
<input type="checkbox"/>	master 1fde0caf1104 postgres:15	Exited	0.25%	5432:5432 34 minutes ago			
<input type="checkbox"/>	user_one de8473d7ea3c postgres:15	Running	0.01%	8081:8080 40 minutes ago			
<input type="checkbox"/>	user_two c4f2e7566dad postgres:15	Running	0%	8082:8080 40 minutes ago			

Рис. 16 – Сбой хоста Master

Поиск в логах сообщений об ошибках:

```

2024-06-17 15:30:03 2024-06-17 12:30:03.340 UTC [30] LOG:  replication terminated by primary server
2024-06-17 15:30:03 2024-06-17 12:30:03.340 UTC [30] DETAIL:  End of WAL reached on timeline 1 at 0/303E4A8.
2024-06-17 15:30:03 2024-06-17 12:30:03.340 UTC [30] FATAL:  could not send end-of-streaming message to primary: server closed the connection unexpectedly
2024-06-17 15:30:03          This probably means the server terminated abnormally
2024-06-17 15:30:03          before or while processing the request.
2024-06-17 15:30:03          no COPY in progress
2024-06-17 15:30:03 2024-06-17 12:30:03.340 UTC [29] LOG:  invalid record length at 0/303E4A8: wanted 24, got 0
2024-06-17 15:30:03 2024-06-17 12:30:03.346 UTC [47] FATAL:  could not connect to the primary server: connection to server at "master" (172.21.0.3), port 5432 failed: Connection refused
2024-06-17 15:30:03          Is the server running on that host and accepting TCP/IP connections?
2024-06-17 15:30:03 2024-06-17 12:30:03.346 UTC [29] LOG:  waiting for WAL to become available at 0/303E4C0
2024-06-17 15:30:11 2024-06-17 12:30:11.736 UTC [48] FATAL:  could not connect to the primary server: could not translate host name "master" to address: Name or service not known

```

Рис. 17 – Логи

Выполним переключение (failover) на резервный сервер. Эта аварийная смена роли Standby на Primary. Аварийное переключение можно выполнить командой `pg_promote()` на резервном узле:

```

postgres=# SELECT pg_promote();
 pg_promote
-----
 t
(1 row)

postgres=#

```

Рис. 18 – Завершение режима резерва и переключение в обычный режим

В логах на резервном узле видно, что узел перестал ждать новых WAL-файлов с мастер-узла и перешел в основной режим работы:

```

2024-06-17 15:37:04 2024-06-17 12:37:04.454 UTC [29] LOG:  waiting for WAL to become available at 0/303E4C0
2024-06-17 15:37:04 2024-06-17 12:37:04.454 UTC [29] LOG:  redo done at 0/303E430 system usage: CPU: user: 0.02 s, system: 0.03 s, elapsed: 2078.38 s
2024-06-17 15:37:04 2024-06-17 12:37:04.454 UTC [29] LOG:  last completed transaction was at log time 2024-06-17 12:25:55.089252+00
2024-06-17 15:37:04 2024-06-17 12:37:04.459 UTC [29] LOG:  selected new timeline ID: 2
2024-06-17 15:37:04 2024-06-17 12:37:04.508 UTC [29] LOG:  archive recovery complete
2024-06-17 15:37:04 2024-06-17 12:37:04.515 UTC [27] LOG:  checkpoint starting: force
2024-06-17 15:37:04 2024-06-17 12:37:04.520 UTC [1] LOG:  database system is ready to accept connections
2024-06-17 15:37:04 2024-06-17 12:37:04.531 UTC [27] LOG:  checkpoint complete: wrote 2 buffers (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=0.003 s, sync=0.002 s, total=0.017 s; sync files=2, longest=0.001 s, average=0.001 s; distance=0 kB, estimate=10840 kB

```

Рис. 19 – Логи на резервном узле при смене роли

Проверим состояние данных и работу клиентов в режиме чтение/запись.

user_one. К мастеру не подключиться, подключаемся к резервному:

```

root@user_one:/# psql -h master -p 5432 -U user_one postgres
psql: error: could not translate host name "master" to address: Name or service not known
root@user_one:/# psql -h standby -p 5432 -U user_one postgres
Password for user user_one:
psql (15.7 (Debian 15.7-1.pgdg120+1))
Type "help" for help.

postgres=>

```

Рис. 20 – Подключение к резервному хосту

user_two. К мастеру не подключиться, подключаемся к резервному:

```
root@user_two:/# psql -h master -p 5432 -U user_two postgres
psql: error: could not translate host name "master" to address: Name or service not known
root@user_two:/# psql -h standby -p 5432 -U user_two postgres
Password for user user_two:
psql (15.7 (Debian 15.7-1.pgdg120+1))
Type "help" for help.

postgres=>
```

Рис. 21 – Подключение к резервному хосту

Создадим таблицу dogs (id, name, age) и добавим в нее новые записи с разных компьютеров.

user_one:

```
postgres=> insert into dogs (name, age) values ('barney', 7);
INSERT 0 1
```

Рис. 22 – Добавление новой записи

user_two:

```
postgres=> insert into dogs (name, age) values ('victor', 12);
INSERT 0 1
```

Рис. 23 – Добавление новой записи

Проверим

содержимое

таблицы

dogs:

```
postgres=> select * from dogs;
 id |  name  | age
----+-----+----
  1 | kostya |   3
  2 | kolya  |   2
  3 | murka  |   1
 34 | victor |  12
 35 | barney |   7
(5 rows)
```

Рис. 24 – Вывод содержимого таблицы

Этап 3

Восстановим работу основного узла. Заново включим контейнер с мастер-узлом в докере:

<input type="checkbox"/>	lab4			Running (4/4)	0%	0 seconds ago			
<input type="checkbox"/>	standby	ae0c8672fe2d	postgres:15	Running	0%	8080:8080			
<input type="checkbox"/>	master	1fde0caf1104	postgres:15	Running	0%	5432:5432			
<input type="checkbox"/>	user_one	de8473d7ea3c	postgres:15	Running	0%	8081:8080			
<input type="checkbox"/>	user_two	c4f2e7566dad	postgres:15	Running	0%	8082:8080			

Рис. 25 – Запуск master-узла

Актуализируем состояние базы на основном узле командой `pg_basebackup`:

```
Terminal master x standby x user_one x user_two x + v
PS C:\Users\admin\Desktop\lab4> docker exec -it master bash
root@master:/# rm -rf /var/lib/postgresql/data/*
root@master:/# pg_basebackup -D /var/lib/postgresql/data -h standby Xs -P -U postgres
pg_basebackup: error: too many command-line arguments (first is "Xs")
pg_basebackup: hint: Try "pg_basebackup --help" for more information.
root@master:/# pg_basebackup -D /var/lib/postgresql/data -h standby -Xs -P -U postgres
23149/23149 kB (100%), 1/1 tablespace
root@master:/#
```

Рис. 26 – Восстановление на master-узле

Видим, что функция `pg_promote()` удалила `standby.signal`, и нам снова необходимо выполнить команду резервного копирования с `-R` флагом для перевода узла в режим резервирования:

```
Terminal master x standby x user_one x user_two x + v
root@standby:/# ls var/lib/postgresql/data/
backup_label.old base pg_commit_ts pg_hba.conf pg_logical pg_notify pg_serial pg_stat pg_subtrans pg_twophase pg_wal postgresql.auto.conf postmaster.opts
backup_manifest global pg_dynshmem pg_ident.conf pg_multixact pg_replslot pg_snapshots pg_stat_tmp pg_tblspc PG_VERSION pg_xact postgresql.conf postmaster.pid
root@standby:/# rm -rf /var/lib/postgresql/data/*
root@standby:/# pg_basebackup -D /var/lib/postgresql/data -h master -Xs -R -P -U postgres
23149/23149 kB (100%), 1/1 tablespace
```

Рис. 27 – Проверка содержимого директории кластера

Восстанавливаем исправную работу узлов в исходной конфигурации:

```
2024-06-17 15:58:37 2024-06-17 12:58:37.084 UTC [1] LOG: database system is ready to accept read-only connections
2024-06-17 15:58:37 2024-06-17 12:58:37.090 UTC [31] LOG: started streaming WAL from primary at 0/7000000 on timeline 2
```

Рис. 28 – Логи

Проверяем работу клиентов в режиме чтение/запись.

user_one:

```
postgres=# insert into cats (name, age) values ('miron', 20);
INSERT 0 1
postgres=#
```

Рис. 29 – Добавление новой записи

user_two:

```
postgres=> insert into cats (name, age) values ('gosha', 9);
INSERT 0 1
postgres=>
```

Рис. 30 – Добавление новой записи

Прочитаем записи первым пользователем:

```
Terminal  master ×  standby ×  user_one ×  user_two ×  +  ▾
postgres=> select * from cats;
 id | name  | age
----+-----+----
  1 | vasya |   1
  2 | misha |   2
  3 | petya |   3
  4 | mark  |   3
  5 | meow  |   4
 37 | murka |   5
 38 | miron |  20
 39 | gosha |   9
(8 rows)
```

Рис. 31 – Вывод содержимого таблицы

Остановим контейнеры, чтобы не загружать процессор, до консультации у преподавателя.

<input type="checkbox"/>	lab4	Exited	N/A	2 minutes ago	▶	⋮	🗑
<input type="checkbox"/>	standby ae0c8672fe2d	Exited	N/A 8080:8080	2 minutes ago	▶	⋮	🗑
<input type="checkbox"/>	master 1fde0caf1104	Exited	N/A 5432:5432	6 minutes ago	▶	⋮	🗑
<input type="checkbox"/>	user_one de8473d7ea3c	Exited	N/A 8081:8080	1 hour ago	▶	⋮	🗑
<input type="checkbox"/>	user_two c4f2e7566dad	Exited	N/A 8082:8080	1 hour ago	▶	⋮	🗑

Рис. 32 – Завершение работы контейнеров

Выводы

В ходе лабораторной работы мы ~~пострадали~~ научились настраивать узлы баз данных для репликации данных. Мы развернули postgres на двух узлах в режиме горячего резерва и справились с переключением между основным и резервным узлом (failover) в случае аварии на основном узле, и смогли восстановить актуальное состояние базы данных.