федеральное государственное автономное образовательное учреждение
высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЕТ

по лабораторной работе №4

по дисциплине «**Распределенные системы хранения данных**»

Вариант 15

Автор: Кулаков Н. В.

Буторин В.А.

Факультет: ПИиКТ

Группа: P33312

Преподаватель: Шешуков Д. М.

Санкт-Петербург 2023

Постановка задачи и исходные данные

Проверить сетевую связность между всеми узлами (ping, ssh). Для подключения к СУБД (например, через psql), использовать отдельную виртуальную или физическую машину. Перед тем как “сломать” узел на этапе 2, рекомендуется выполнить снапшот виртуальной машины. Для демонстрации наполнения базы, а также доступа на запись (см. задание ниже) использовать не меньше двух таблиц, трёх столбцов, пяти строк, двух транзакций, двух клиентских сессий. Данные не обязаны быть осмысленными, но должны быть легко отличимы - повторяющиеся строки запрещены.

Этап 1 Настройка:

1. 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 2.1 Подготовка:

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

Этап 2.2 Сбой:

1. Симулировать отказ основного узла - выполнить жесткое выключение виртуальной машины.

Этап 2.3 Отработка:

1. Найти продемонстрировать в логах релевантные сообщения об ошибках.
2. Выполнить фейловер на резервный сервер.

3. Продемонстрировать состояние данных и работу клиентов в режиме чтение/запись.

Этап 3 Восстановление:

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

Выполнение

Для выполнения сейчас и далее использовался Docker.

Настройка рабочего окружения

Создаем Docker образ, на который будем накатывать узлы:

```
postgres-node.Dockefile
```
```
#Base Image
FROM ubuntu:23.04

#Update APT repository & Install OpenSSH
RUN apt-get update \
    && apt-get install -y openssh-server \
    && apt-get install -y mysql-client \
    && apt-get -y install curl wget sudo \
    && apt-get -y install ca-certificates gnupg

#Postgres 15
ARG DEBIAN_FRONTEND=noninteractive
```

```
RUN sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release
-cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
RUN wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo
apt-key add -
RUN apt-get -y install postgresql-15
RUN apt-get -y install pgpool2 libpgpool2 postgresql-15-pgpool2

RUN apt-get -y install ssh iputils-ping vim nano

RUN cp -s /usr/lib/postgresql/15/bin/* /usr/bin 2> dev/null; exit 0
#Postgres 15

##Establish the operating directory of OpenSSH
#RUN mkdir /var/run/sshd

#Set Root password
RUN echo 'root:root' | chpasswd

#Allow Root login
RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/' \
    /etc/ssh/sshd_config

#SSH login fix
RUN sed 's@session\s*required\s*pam_loginuid.so@session optional \
    pam_loginuid.so@g' -i /etc/pam.d/sshd

#expose port 22
EXPOSE 22

#Commands to be executed by default
CMD ["/usr/sbin/sshd","-D"]
```

Описание узлов:

```
version: "3.6"

services:
  ubuntu-a:
    build:
```

```
      context: .
      dockerfile: Dockerfile
    ports:
      - "30000:22"
    hostname: ubuntu-a
    container_name: ubuntu-a
    networks:
      ubuntu-net:
  ubuntu-b:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "30001:22"
    hostname: ubuntu-b
    container_name: ubuntu-b
    networks:
      ubuntu-net:
  ubuntu-c:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "30002:22"
    hostname: ubuntu-c
    container_name: ubuntu-c
    networks:
      ubuntu-net:

networks:
  ubuntu-net:
    driver: bridge
```

Этап 1. Настройка

Создаем докер образ:

```
docker build -f=postgres-node.Dockefile . -t rshd-postgres
```

Создаем новую `bridge` сеть:

```
docker network create rshd-bridge
```

Запускаем сервисы:

```
docker compose up -d
```

Подключение к запущенному контейнеру:

```
docker exec -it rshd-node-A bash
```

Проверка ping:

```
root@ubuntu-c:/# ping ubuntu-b
PING ubuntu-b (172.19.0.4) 56(84) bytes of data.
64 bytes from ubuntu-b.lab-4_ubuntu-net (172.19.0.4): icmp_seq=1 ttl=64 time=0.173 ms
64 bytes from ubuntu-b.lab-4_ubuntu-net (172.19.0.4): icmp_seq=2 ttl=64 time=0.077 ms
64 bytes from ubuntu-b.lab-4_ubuntu-net (172.19.0.4): icmp_seq=3 ttl=64 time=0.103 ms
```

Проверка ssh:

```
root@ubuntu-c:/# ssh root@ubuntu-b
root@ubuntu-b's password:
Welcome to Ubuntu 23.04 (GNU/Linux 6.1.19-gentoo-x86_64 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sun Jun  4 06:08:34 2023 from 172.19.0.2
root@ubuntu-b:~#
```

Дропаем все существующие кластеры:

```
root@a0a6dda22bb9:/app# pg_lsclusters
Ver Cluster Port Status Owner    Data directory              Log file
15  main    5432 down   postgres /var/lib/postgresql/15/main
/var/log/postgresql/postgresql-15-main.log
pg_dropcluster 15 main
```

Создаем новый кластер:

```
Ver Cluster Port Status Owner    Data directory              Log file
15  main    5432 down   postgres /var/lib/postgresql/15/main
/var/log/postgresql/postgresql-15-main.log
```

Запускаем кластер:

```
root@a0a6dda22bb9:/app# pg_ctlcluster 15 main start
root@a0a6dda22bb9:/app# pg_lsclusters
Ver Cluster Port Status Owner    Data directory              Log file
15  main    5432 online postgres /var/lib/postgresql/15/main
/var/log/postgresql/postgresql-15-main.log
```

Подключаемся к пользователю `postgres`:

```
su - postgres
psql
```

Конфигурация `postgresql.conf`:

Расположение файла конфигурации:

```
postgres=# show config_file
 /etc/postgresql/14/main/postgresql.conf
```

rshd-node-A

```
cluster_name = 'cluster_a'                      # added to process titles if
nonempty
                                        # (change requires restart)
listen_addresses = '*'          # what IP address(es) to listen on;
wal_level = replica                     # minimal, replica, or logical
max_wal_senders = 10            # max number of walsender processes
synchronous_standby_names = 'cluster_b' # standby servers that provide sync rep
                                # method to choose sync standbys, number of sync
standbys,
```

```
                                # and comma-separated list of application_name
                                # from standby(s); '*' = all
synchronous_commit = on         # synchronization level;
                                        # off, local, remote_write,
remote_apply, or on
```

Создаем пользователя репликации:

```
# create role replica_user with replication login password 'pass';
```

Получаем айпишники нод:

```
postgres@0f4e8f3f3fee:~$ ping rshd-node-A
PING rshd-node-A (172.18.0.2) 56(84) bytes of data.
64 bytes from rshd-node-A.rshd-bridge (172.18.0.2): icmp_seq=2 ttl=64 time=0.119
m

postgres@0f4e8f3f3fee:~$ ping rshd-node-B
PING rshd-node-B (172.18.0.3) 56(84) bytes of data.
64 bytes from 0f4e8f3f3fee (172.18.0.3): icmp_seq=1 ttl=64 time=0.022 ms

postgres@0f4e8f3f3fee:~$ ping rshd-node-C
PING rshd-node-C (172.18.0.4) 56(84) bytes of data.
64 bytes from rshd-node-C.rshd-bridge (172.18.0.4): icmp_seq=1 ttl=64 time=0.102
ms
```

Добавляем записи репликаций в `pg_hba.conf`:

```
host    replication     replica_user    172.18.0.3/32           md5
host    replication     replica_user    172.18.0.4/32           md5
```

Или с другой максой:

```
host replication replica_user 172.19.0.0/24 md5
```

rshd-node-B

```
#listen_addresses = '*'         # what IP address(es) to listen on;
```

```
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost'; use '*' for
all
                                        # (change requires restart)
hot_standby = on                        # "off" disallows queries during
recovery
                                        # (change requires restart)
cluster_name = 'cluster_b'                      # added to process titles if
nonempty
                                        # (change requires restart)
```

rshd-node-C

```
#listen_addresses = '*'         # what IP address(es) to listen on;
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost'; use '*' for
all
                                        # (change requires restart)
hot_standby = on                        # "off" disallows queries during
recovery
                                        # (change requires restart)
cluster_name = 'cluster_c'                      # added to process titles if
nonempty
                                        # (change requires restart)
```

pg_basebackup

Осуществляем backup основного узла для того, что получить зеркало на текущий момент. Для каждого из secondary узлов удаляем `PGDATA` содержимое.

rshd-node-B

```
root@ubuntu-b:/# pg_basebackup -h 172.19.0.4 -U replica_user -X stream -C -S replica_1 -v -R -W -D /var/lib/postgresql/15/main
Password:
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/2000028 on timeline 1
pg_basebackup: starting background WAL receiver
pg_basebackup: created replication slot "replica_1"
pg_basebackup: write-ahead log end point: 0/2000138
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: syncing data to disk ...
pg_basebackup: renaming backup_manifest.tmp to backup_manifest
pg_basebackup: base backup completed
root@ubuntu-b:/#
```

rshd-node-C

```
pg_basebackup -h 172.19.0.4 -U replica_user -X stream -C -S replica_2 -v -R -W -D /var/lib/postgresql/15/main
```

Результат

```
postgres=# select * from pg_replication_slots ;
 slot_name | plugin | slot_type | datoid | database | temporary | active | active_pid | xmin | catalog_xmin | restart_lsn | confirmed_flush_lsn | wal_status | safe_wal_size | two_phase
-----------+--------+-----------+--------+----------+-----------+--------+------------+------+--------------+-------------+---------------------+------------+---------------+-----------
 replica_1 |        | physical  |        |          | f         | f      |            |      |              | 0/C000000   |                     | reserved   |               | f
 replica_2 |        | physical  |        |          | f         | f      |            |      |              | 0/6000000   |                     | reserved   |               | f
(2 rows)
```

```
postgres=# SELECT client_addr, state
FROM pg_stat_replication;
 client_addr |   state
-------------+-----------
 172.18.0.3  | streaming
 172.18.0.4  | streaming
(2 rows)
```

При асинхронной репликации запрос применится сначала на основном узле, а затем данные отправятся на другие узлы. Поэтому, если разорвать соединение с синхронной базой, то основная будет пытаться отправить данные, а поскольку не будет получаться, то она будет ожидать.

pgpool

Настраиваем pgpool:

```
failover_when_quorum_exists = off

enable_pool_hba = on

backend_hostname0 = '172.19.0.4'
backend_port0 = 5432
backend_weight0 = 1
backend_data_directory0 = '/var/lib/postgresql/15/main'
backend_flag0 = 'ALLOW_TO_FAILOVER'
backend_application_name0 = 'node_a'

backend_hostname1 = '172.19.0.2'
backend_port1 = 5432
```

```
backend_weight1 = 1
backend_data_directory1 = '/var/lib/postgresql/15/main'
backend_flag1 = 'ALLOW_TO_FAILOVER'
backend_application_name1 = 'node_b'

backend_hostname2 = '172.19.0.3'
backend_port2 = 5432
backend_weight2 = 1
backend_data_directory2 = '/var/lib/postgresql/15/main'
backend_flag2 = 'ALLOW_TO_FAILOVER'

sr_check_user = 'postgres'
sr_check_database = 'postgres'
```

Добавляем записи в бд и перезапускаем:

```
host all all 172.19.0.0/24 trust
```

Запускаем pgpool:

```
pgpool -n 2> pgpool.log
```

```
mahomeboshi)
2023-06-05 13:00:09.785: main pid 20871: LOG:  node status[0]: 1
2023-06-05 13:00:09.785: main pid 20871: LOG:  node status[1]: 0
2023-06-05 13:00:09.785: main pid 20871: LOG:  node status[2]: 2
root@ubuntu-pool:~# ls -a /var/
```

Подключаемся через:

```
pgpool -p 9999
```

```
 node_id |  hostname  | port | status | pg_status | lb_weight |  role   | pg_role | select_cnt | load_balance_node | replication_delay | replication_state | replication_sync_state | last_statu
s_change
---------+------------+------+--------+-----------+-----------+---------+---------+------------+-------------------+-------------------+-------------------+------------------------+-----------
----------
 0       | 172.19.0.4 | 5432 | up     | unknown   | 0.333333  | primary | unknown | 0          | false             | 0                 |                   |                        | 2023-06-05
 13:00:09
 1       | 172.19.0.2 | 5432 | down   | unknown   | 0.333333  | standby | unknown | 0          | false             | 0                 |                   |                        | 2023-06-05
 13:00:09
 2       | 172.19.0.3 | 5432 | up     | unknown   | 0.333333  | standby | unknown | 1          | true              | 0                 |                   |                        | 2023-06-05
 13:00:09
```

Этап 2.1. Подготовка

Создаем таблицы с двух сессий:

```
postgres=# create table second(id serial primary key, create table first(
postgres(# first_name varchar(255),                   id serial primary key,
postgres(# second_name varchar(255)                   name integer,
postgres(# );                                         value varchar(255)
CREATE TABLE                                          );
```

```
postgres=# \dt
         List of relations
 Schema |  Name  | Type  |  Owner
--------+--------+-------+----------
 public | first  | table | postgres
 public | second | table | postgres
(2 rows)
```

Активные сессии:

```
postgres=# SELECT pid, usename, application_name, client_addr, backend_start, state, query
FROM pg_stat_activity
WHERE state = 'active';
  pid  |   usename    | application_name | client_addr |         backend_start         | state  |                                        query
-------+--------------+------------------+-------------+-------------------------------+--------+-----------------------------------------------------------------------------------------
 21782 | postgres     | psql             | 172.19.0.6  | 2023-06-05 13:14:05.494131+00 | active | SELECT pid, usename, application_name, client_addr, backend_start, state, query+
       |              |                  |             |                               |        | FROM pg_stat_activity                                                                  +
       |              |                  |             |                               |        | WHERE state = 'active';
 29625 | replica_user | cluster_a        | 172.19.0.3  | 2023-06-05 12:45:59.621482+00 | active | START_REPLICATION SLOT "replica_2" 0/5000000 TIMELINE 1
 29633 | replica_user | cluster_b        | 172.19.0.2  | 2023-06-05 12:46:47.850816+00 | active | START_REPLICATION SLOT "replica_1" 0/5000000 TIMELINE 1
(3 rows)
```

Вставка данных в таблицы:

```
postgres=# begin;
BEGIN
postgres=*# insert into first (name, value) values
postgres-*# (10, 'one'), (20, 'two'), (30, 'three'), (40, 'four'), (50, 'five');
INSERT 0 5
postgres=*# commit;
COMMIT
postgres=#
```

```
postgres=# select * from first;
 id | name | value
----+------+-------
  1 |   10 | one
  2 |   20 | two
  3 |   30 | three
  4 |   40 | four
  5 |   50 | five
(5 rows)
```

```
postgres=# select * from second;
 id | first_name | second_name
----+------------+-------------
  1 | abcde      | edcba
  2 | aaaaa      | bbbbb
  3 | ccccc      | ddddd
  4 | asd        | dsa
  5 | five       | four
(5 rows)

postgres=#
```

```
(5 rows)

postgres=# begin
postgres-# ;
BEGIN
postgres=*# insert into second(first_name, second_name) values
postgres-*# ('abcde', 'edcba'), ('aaaaa', 'bbbbb'), ('ccccc', 'ddddd'), ('asd', 'dsa'), ('five',
 'four');
INSERT 0 5
postgres=*# commit;
COMMIT
```

Как мы можем увидеть, данные реплицируются по узлам:

```
postgres=# select * from first;
 id | name | value
----+------+-------
  1 |   10 | one
  2 |   20 | two
  3 |   30 | three
  4 |   40 | four
  5 |   50 | five
(5 rows)

postgres=#
```

```
2 packets transmitted, 2 received, 0% packet loss, time 1037ms
rtt min/avg/max/mdev = 0.049/0.057/0.066/0.008 ms
postgres@ubuntu-c:~$ select * from first;
-bash: syntax error near unexpected token `from'
postgres@ubuntu-c:~$ psql
psql (15.3 (Ubuntu 15.3-0ubuntu0.23.04.1))
Type "help" for help.

postgres=# select * from first;
 id | name | value
----+------+-------
  1 |   10 | one
  2 |   20 | two
  3 |   30 | three
  4 |   40 | four
  5 |   50 | five
(5 rows)
```

Этап 2.2. Сбой

Создаем снимок контейнера:

```
```

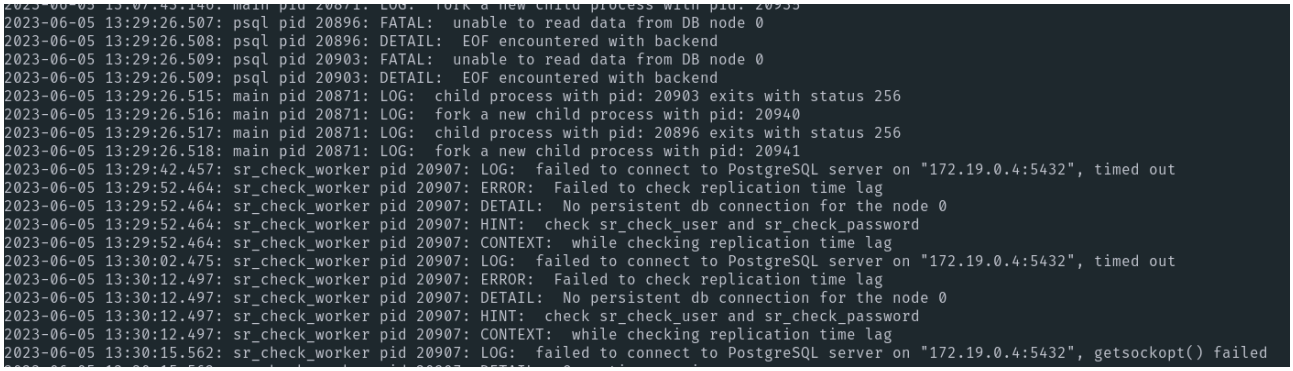
```
docker commit c7b98c5d62e1 rshd-node-a-snap:latest
sha256:365c46730943a93864fbd7cf3f40691f05c3715931f82ec65a50b566cc34a680
```

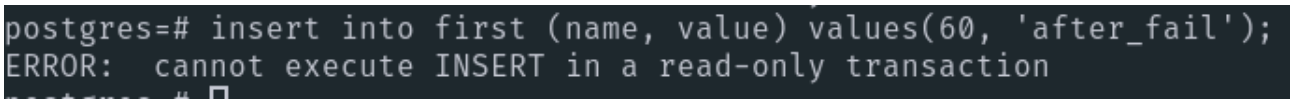
Эмулируем сбой основного узла:

```
docker stop -s=9 c7b98c5d62e1 # sigkill
```

Этап 2.3. Отработка

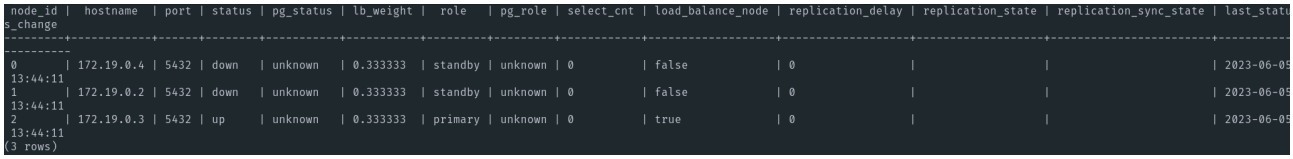
Логи об ошибках:



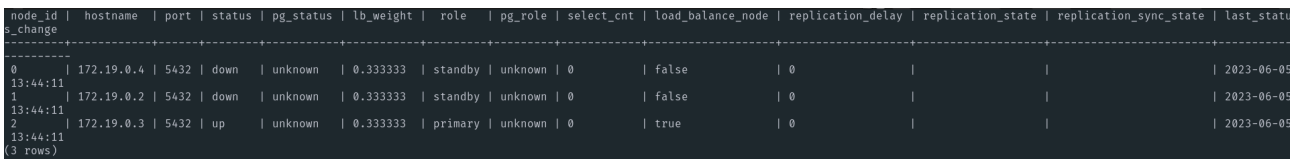


Осуществляем failover. На узле пишем:

```
postgres@ubuntu-c:~$ pg_ctlcluster 15 main promote
```



Теперь мы можем писать:



Этап 3. Восстановление

С помощью созданного коммита запускаем убитую основную бд:

```
nikit@host lab-4 % docker run -it -d --network=lab-4_ubuntu-net rshd-node-a-snap:latest
d966db7bc46b485fb48f69e6d022fda21ff1a336777f7dabca78eb2e621eccd4
nikit@host lab-4 % docker exec -it d966db7bc46b485fb48f69e6d022fda21ff1a336777f7dabca78eb2e621eccd4 bash
```

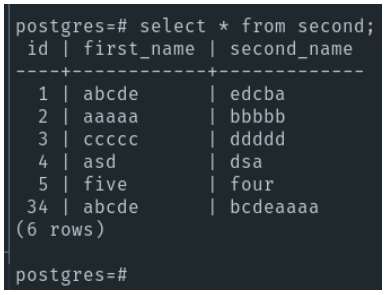
Восстанавливаем ее состояние со бд, в которой уже были произведены изменения:

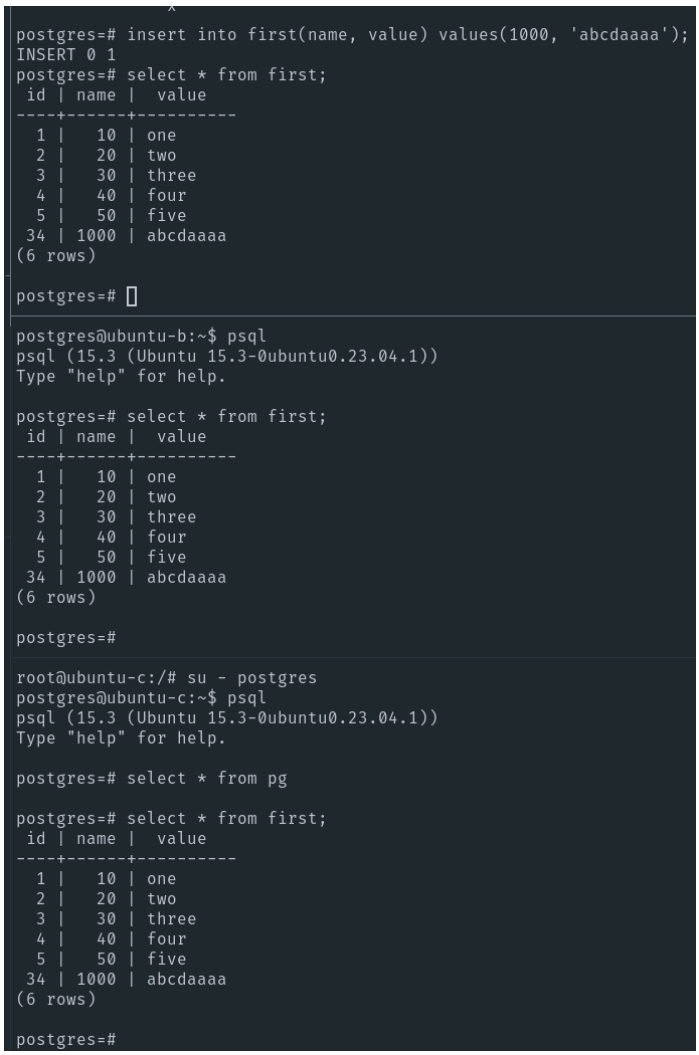
```
pg_basebackup -h 172.19.0.3 -U replica_user -X stream -C -S back -v -R -W -D /var/lib/postgresql/15/main
```

Промоутим:

```
pg_ctlcluster 15 main promote
```

Проверяем состояние:





Вывод

В ходе выполнения данной лабораторной работы мы узнали, как можно настраивать потоковую репликацию в postgresql, а также поработали с pgpool, который является так называемым middleware, умеет распределять нагрузку и изменять бд, в которую осуществляется запись, в случае неисправностей основной. Кроме того, разобрались с различными видами сетей в docker, а также узнали про команду docker commit для создания image с текущим состоянием контейнера