Министерство образования и науки РФ

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’

Вариант №38491

Выполнил:

Студент группы P33312

Соболев Иван Александрович

Преподаватель:

Осипов Святослав Владимирович

Санкт-Петербург, 2024

**Задание:**

### **Этап 1. Конфигурация**

Настроить репликацию postgres на трех узлах в каскадном режиме A --> B --> C. Для управления использовать pgpool-II. Репликация с A на B синхронная. Репликация с B 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:

1. Симулировать отказ основного узла - выполнить жесткое выключение 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.

### **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:**

Для выполнения сейчас и далее использовался Docker.

## **Настройка рабочего окружения**

Создаем Docker образ, на который будем накатывать узлы:

#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

hostname: ubuntu-a

container\_name: ubuntu-a

networks:

ubuntu-net:

ubuntu-b:

build:

context: .

dockerfile: Dockerfile

hostname: ubuntu-b

container\_name: ubuntu-b

networks:

ubuntu-net:

ubuntu-c:

build:

context: .

dockerfile: Dockerfile

hostname: ubuntu-c

container\_name: ubuntu-c

networks:

ubuntu-net:

ubuntu-pool:

build:

context: .

dockerfile: Dockerfile

hostname: ubuntu-pool

container\_name: ubuntu-pool

networks:

ubuntu-net:

networks:

ubuntu-net:

driver: bridge

**Этап 1. Настройка**

Создаем докер образ:

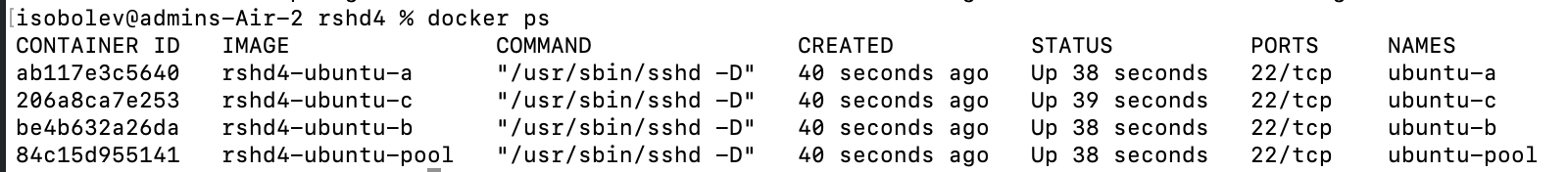
**docker build -t rshd-postgres .**

Создаем новую `bridge` сеть:

**docker network create rshd-bridge**

Запускаем сервисы:

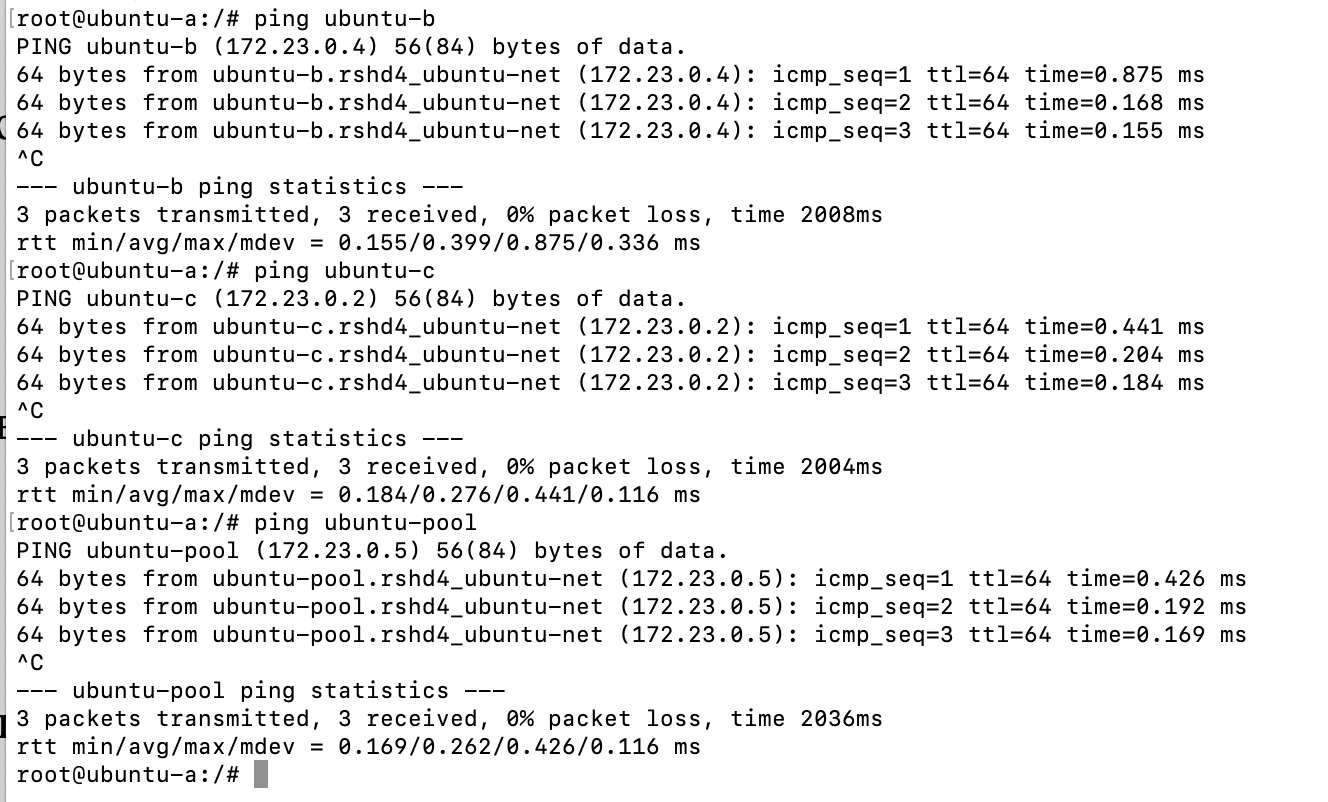
**docker compose up -d**

****

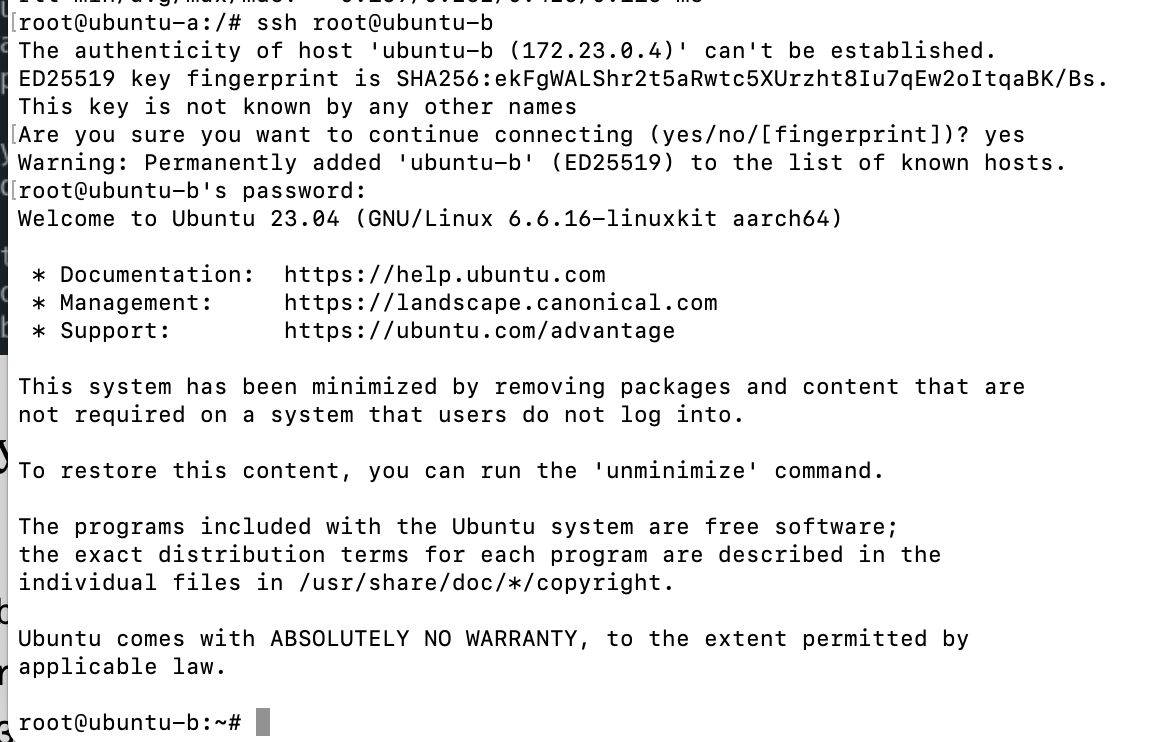
Подключение к запущенному контейнеру:

**docker exec -it ubuntu-a bash**

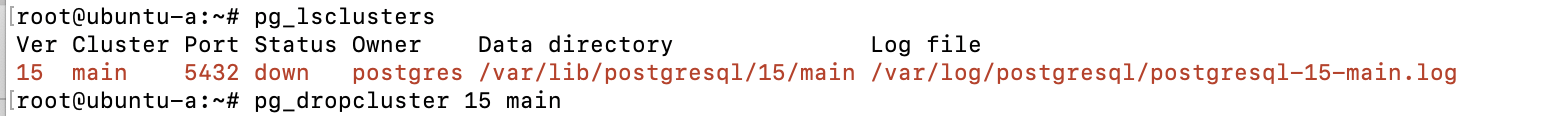
Проверка ping:

****

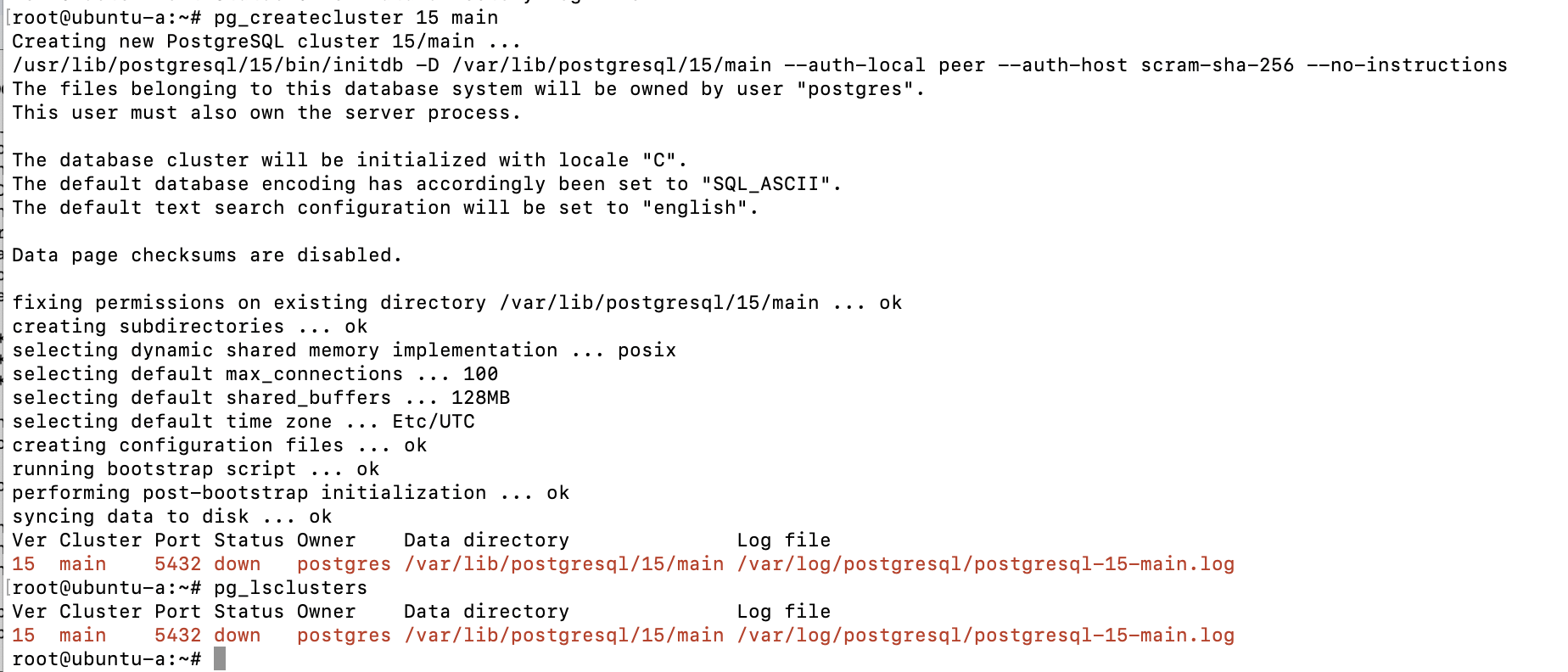
Проверка ssh:

****

Дропаем все существующие кластеры:

****

Создаем новый кластер:

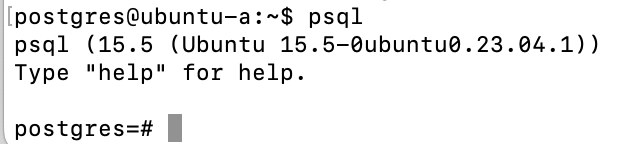
****

Запускаем кластер:

**pg\_ctlcluster 15 main start**

Подключаемся к пользователю `postgres`:

**su - postgres**

****

**Конфигурация `postgresql.conf`:**

Расположение файла конфигурации:

****

**UBUNTU-A**

cluster\_name = 'cluster\_a'

listen\_addresses = '\*'

wal\_level = replica

max\_wal\_senders = 10

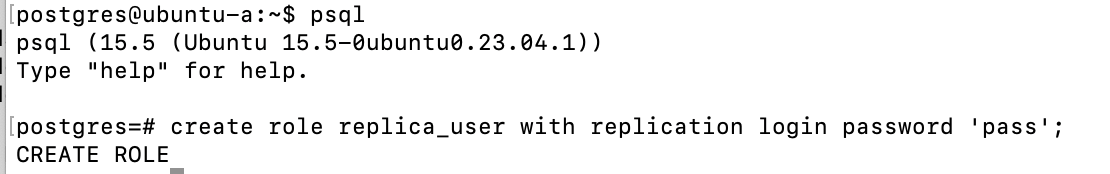
synchronous\_standby\_names = 'cluster\_b'

synchronous\_commit = on

hot\_standby = on

Включаем параметр hot\_standby. Данный параметр будет игнорироваться на master сервере. Но если master сервер станет slave сервером, то данный параметр будет необходим.

Создаем пользователя репликации:

****

Получаем айпишники нод:

**PING ubuntu-a (172.23.0.3) 56(84) bytes of data.**

**PING ubuntu-b (172.23.0.4) 56(84) bytes of data.**

**PING ubuntu-c (172.23.0.2) 56(84) bytes of data.**

Добавляем записи репликаций в `pg\_hba.conf`:

**host replication replica\_user 172.23.0.4/24 md5**

**host replication replica\_user 172.23.0.2/24 md5**

**UBUNTU-B**

cluster\_name = 'cluster\_b'

listen\_addresses = '\*'

wal\_level = replica

max\_wal\_senders = 10

hot\_standby = on

Добавляем записи репликаций в `pg\_hba.conf`:

**host replication replica\_user 172.23.0.3/24 md5**

**host replication replica\_user 172.23.0.2/24 md5**

**UBUNTU-C**

cluster\_name = 'cluster\_c'

listen\_addresses = '\*'

wal\_level = replica

max\_wal\_senders = 10

hot\_standby = on

Добавляем записи репликаций в `pg\_hba.conf`:

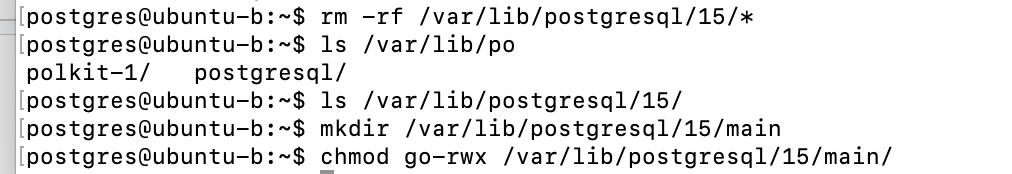
**host replication replica\_user 172.23.0.3/24 md5**

**host replication replica\_user 172.23.0.4/24 md5**

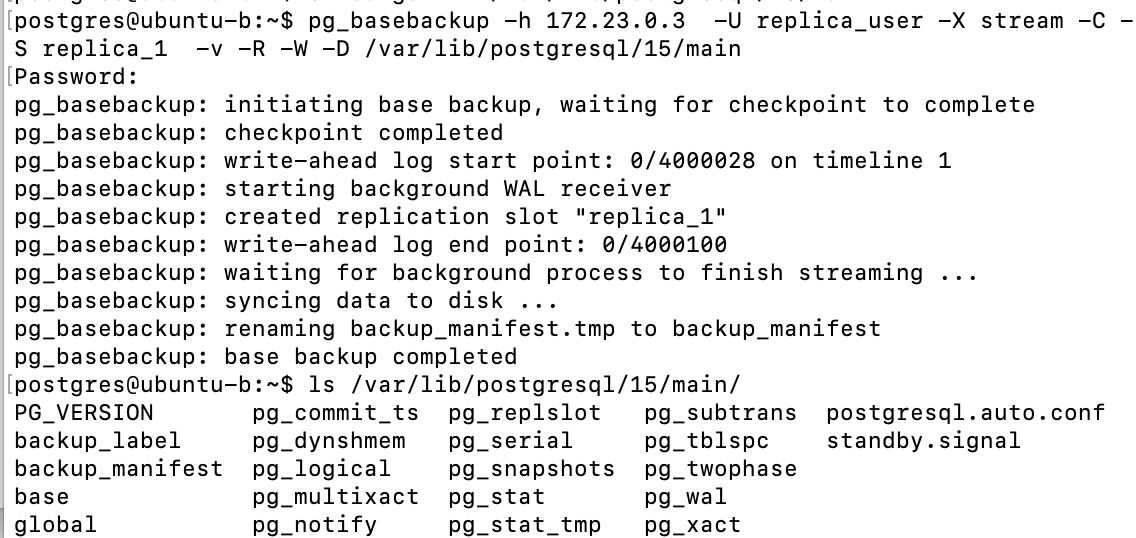
**pg\_basebackup**

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

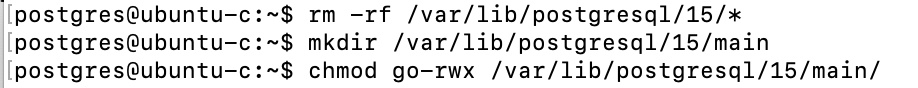
**UBUNTU-B**

****

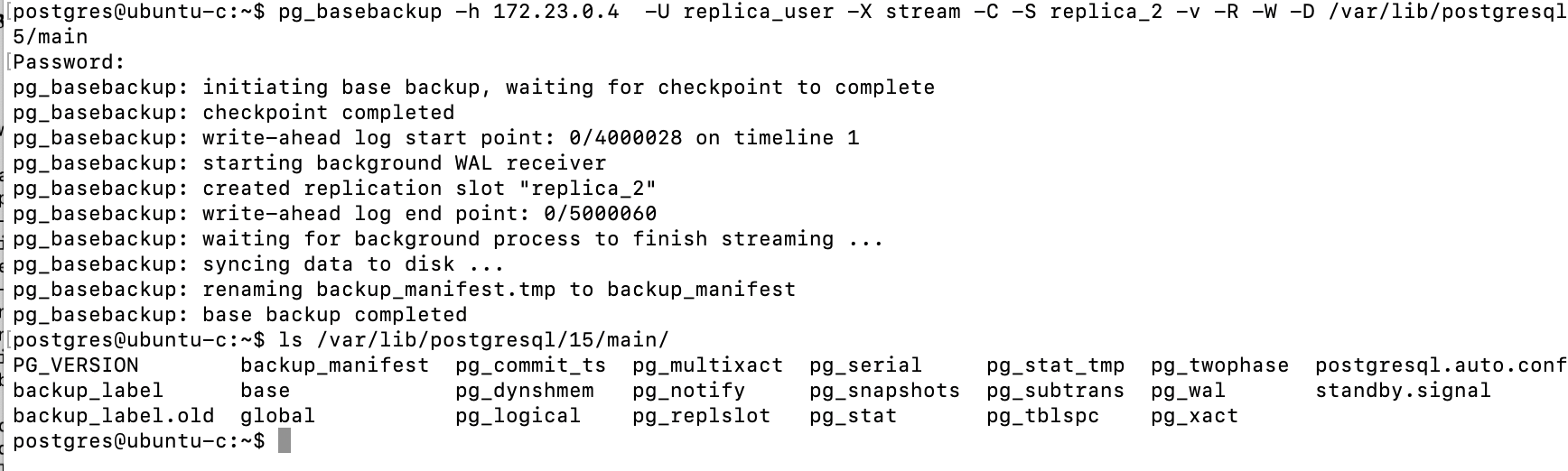
**pg\_basebackup -h 172.23.0.3 -U replica\_user -X stream -C -S replica\_1 -v -R -W -D /var/lib/postgresql/15/main**

****

**UBUNTU-C**

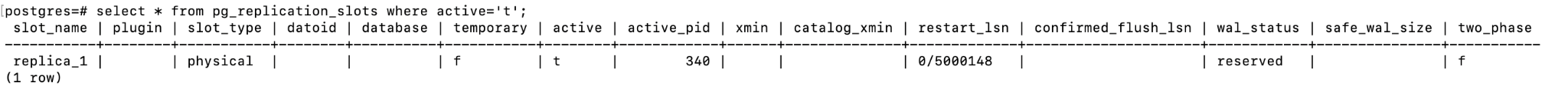
****

**pg\_basebackup -h 172.23.0.4 -U replica\_user -X stream -C -S replica\_2 -v -R -W -D /var/lib/postgresql/15/main**

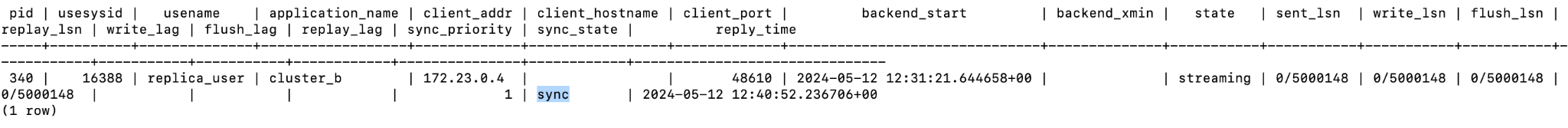
****

**Результат**

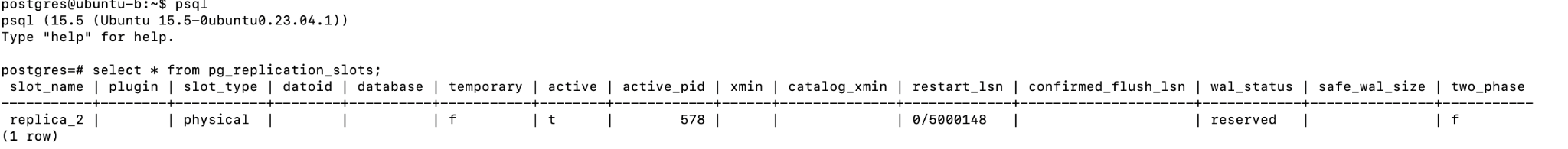
На узле А видим, что репликация идет на узел B:



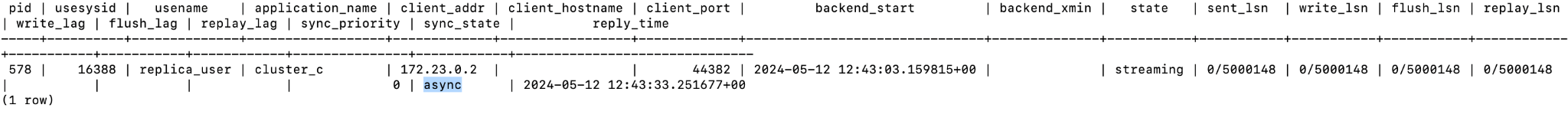
Кластер B подключен синхронно:



На узле B репликация каскадно продолжается на С:



Кластер C подключен асинхронно:



**pgpool**

Добавим информацию о подключении в файл pool\_hba.conf:

host all all 172.23.0.0/24 trust

Теперь внесем изменения в pgpool.conf:

backend\_hostname0 = '172.23.0.3'

# Host name or IP address to connect to for backend 0

backend\_port0 = 5432

# Port number for backend 0

backend\_weight0 = 1

# Weight for backend 0 (only in load balancing mode)

backend\_data\_directory0 = '/var/lib/postgresql/15/main'

# Data directory for backend 0

backend\_flag0 = 'ALLOW\_TO\_FAILOVER'

# Controls various backend behavior

# ALLOW\_TO\_FAILOVER, DISALLOW\_TO\_FAILOVER

# or ALWAYS\_PRIMARY

backend\_application\_name0 = 'node\_a'

# walsender's application\_name, used for "show pool\_nodes" command

backend\_hostname1 = '172.23.0.4'

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 = '172.23.0.2'

backend\_port2 = 5432

backend\_weight2 = 1

backend\_data\_directory2 = '/var/lib/postgresql/15/main'

backend\_flag2 = 'ALLOW\_TO\_FAILOVER'

backend\_application\_name2 = 'node\_c'

failover\_when\_quorum\_exists = off

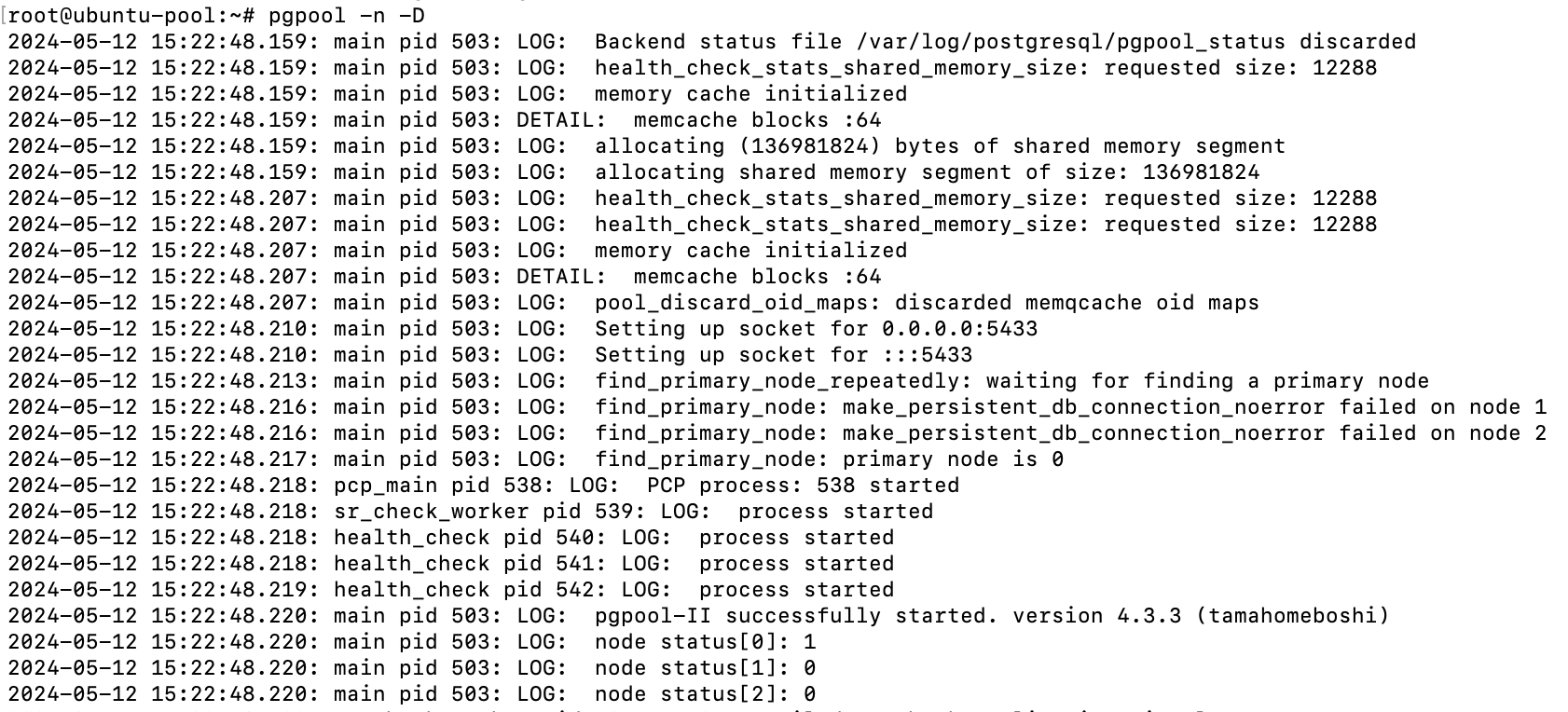
enable\_pool\_hba = on

sr\_check\_user = 'postgres'

sr\_check\_database = 'postgres'

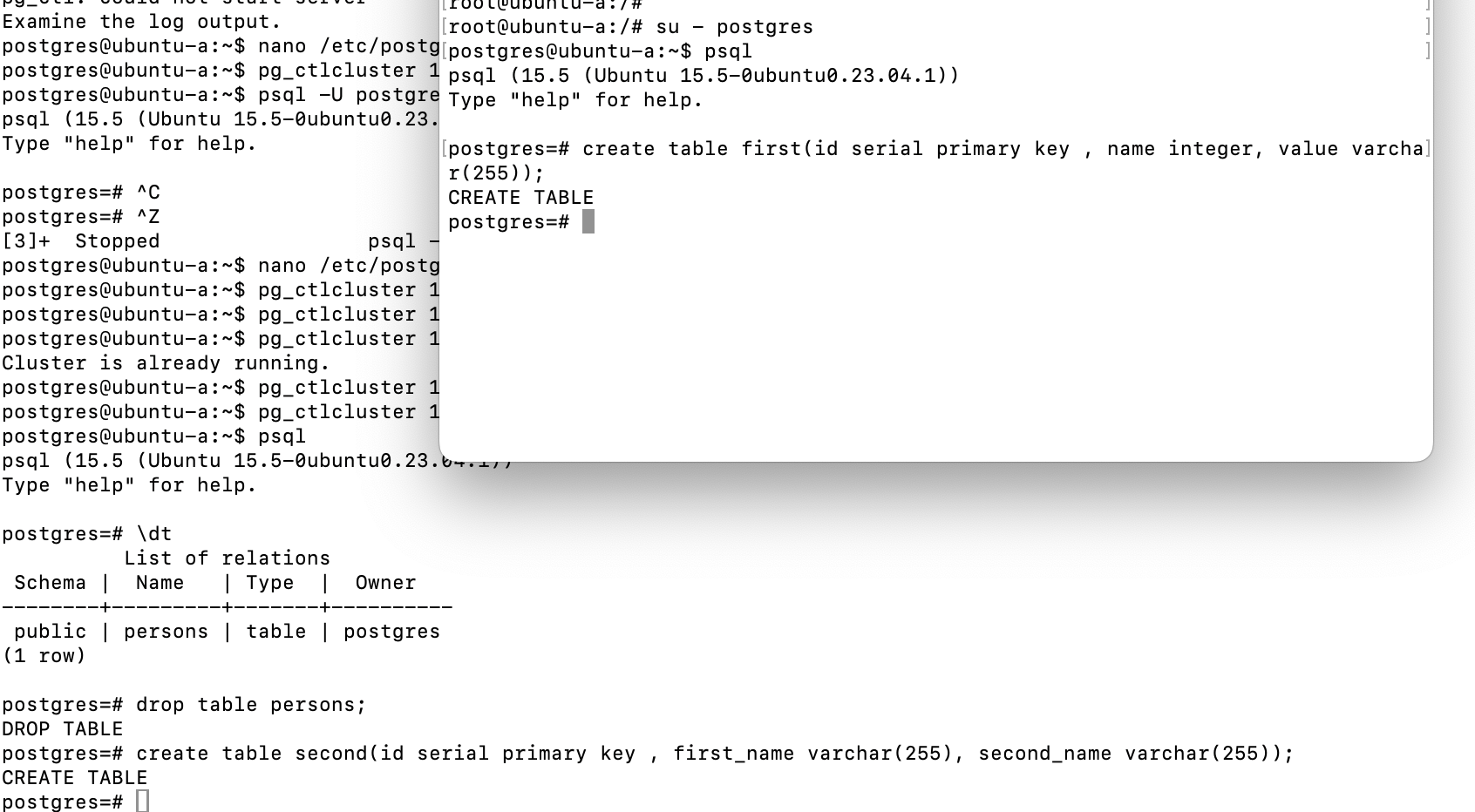
Запускаем pgpool:

pgpool -n -D

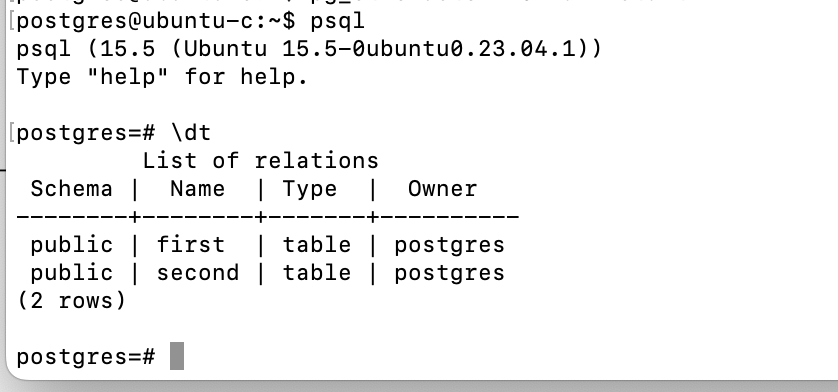


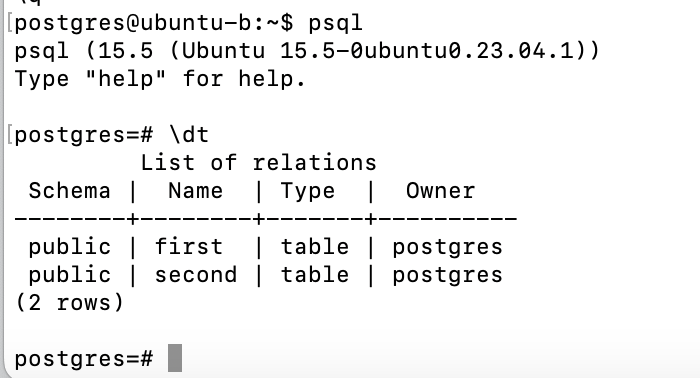
**Этап 2.1. Подготовка**

Создаем таблицы с двух сессий:

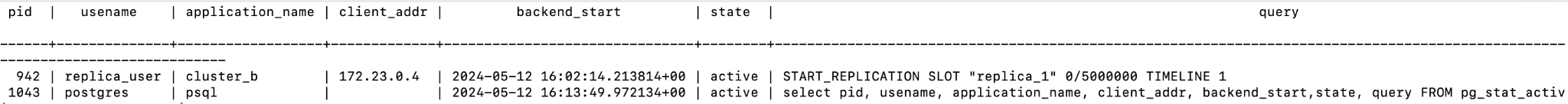


Проверим, что на узлах тоже появились таблицы:

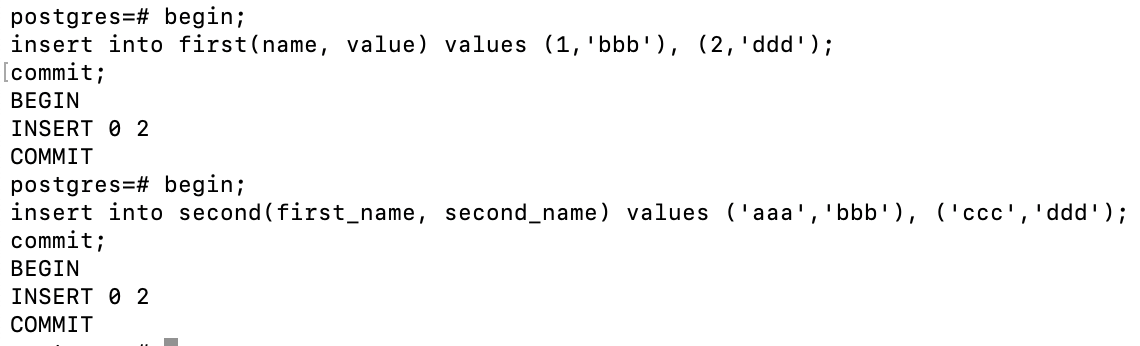




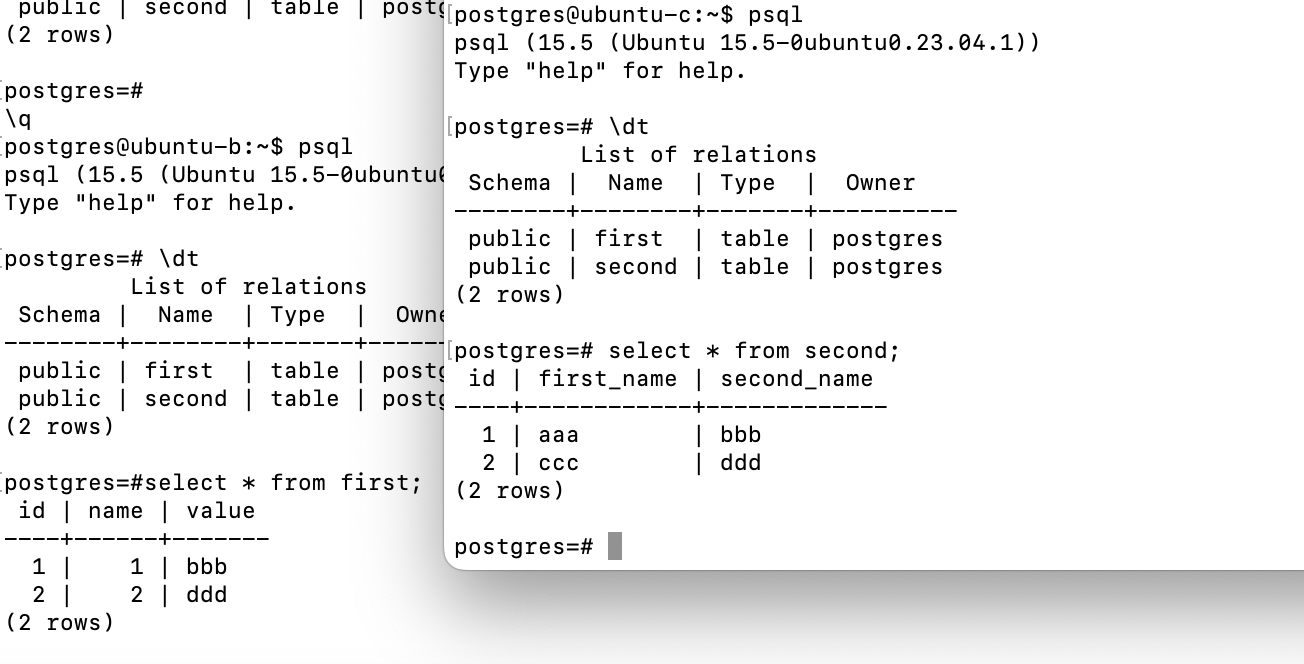
Активные сессии на узле А:



Вставка данных в таблицы:



Проверка вставки:

Как мы можем увидеть, данные реплицируются по узлам.

**Этап 2.2. Сбой**

Создаем снимок контейнера:

docker commit ab117e3c5640 ubuntu-a:latest

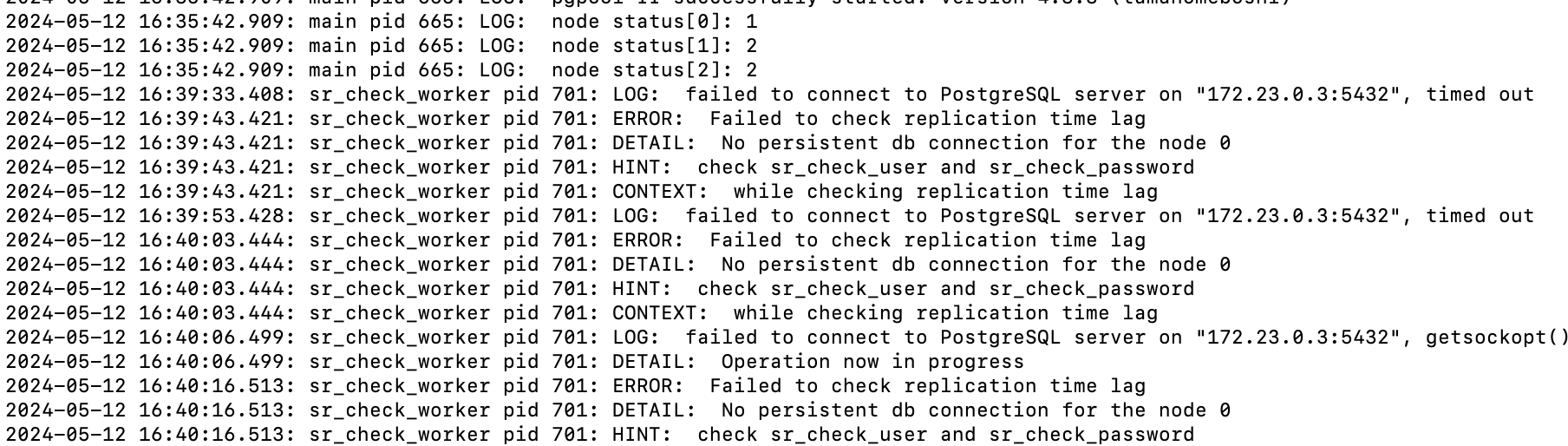


Эмулируем сбой основного узла:

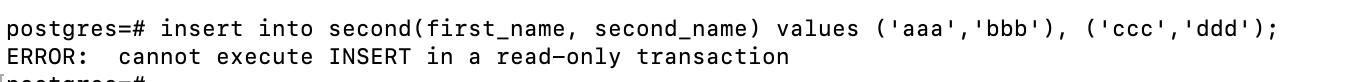
docker stop -s=9 ab117e3c5640 # sigkill

**Этап 2.3. Отработка**

Логи об ошибках:

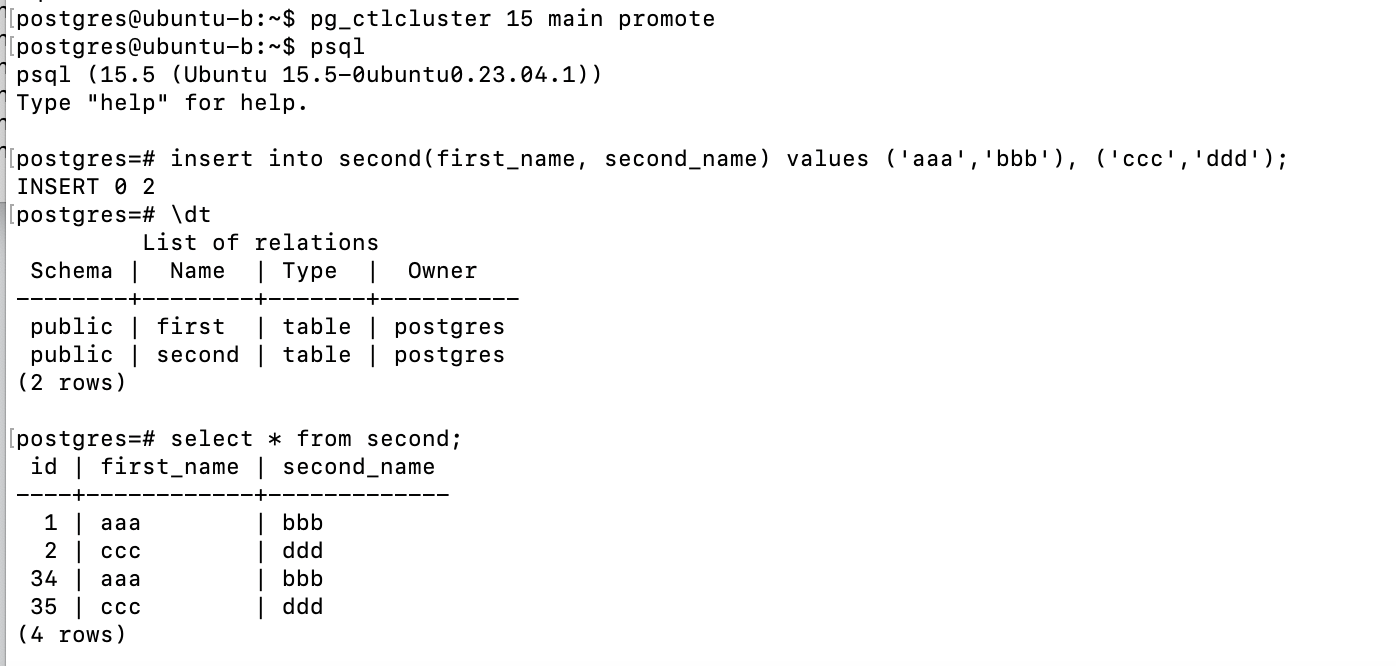


**На резервном узле писать не можем:**

****

**Осуществляем failover. На узле пишем:**

**pg\_ctlcluster 15 main promote**

****

**Этап 3. Восстановление**

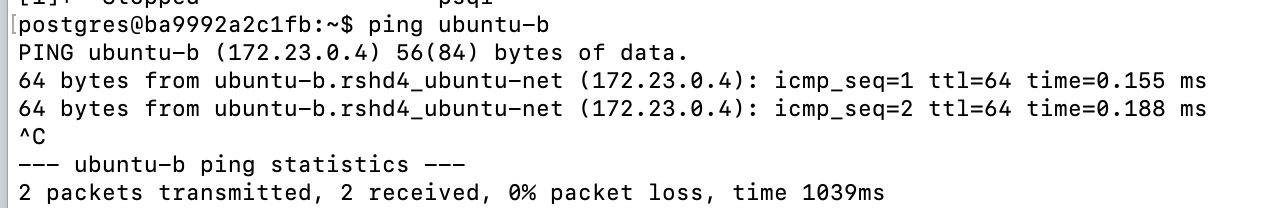
С помощью созданного коммита запускаем убитую основную бд:

**docker run -it -d --network=rshd4\_ubuntu-net ubuntu-a:latest**

Подключаемся к контейнеру:

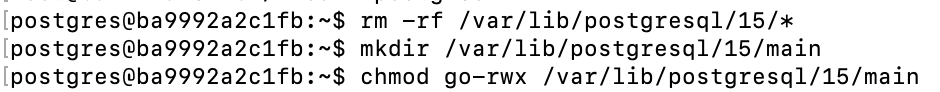
**docker exec -it ba9992a bash**

Проверяем включение в сеть:



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

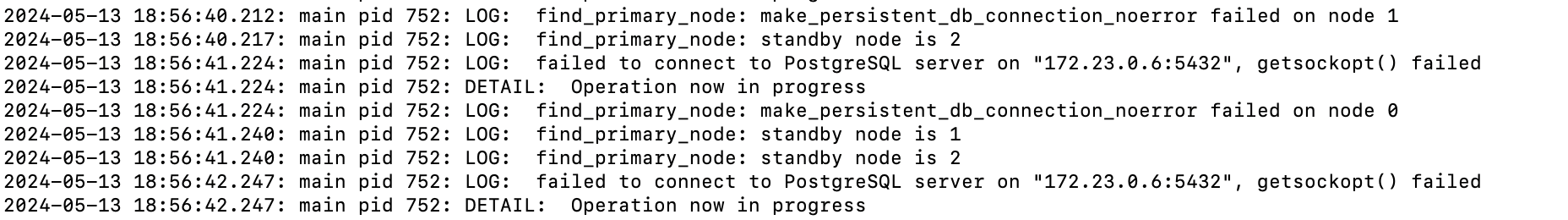
Удаляем PGDATA:

****

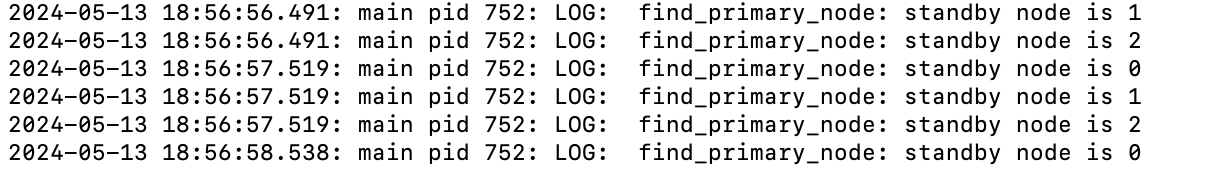
Забираем данные с узла B:

**pg\_basebackup -h 172.23.0.4 -U replica\_user -X stream -C -S back -v -R -W -D /var/lib/postgresql/15/main**

На узле B создаем файл standby.signal, чтобы запустить его в режиме слейва, останавливаем и смотрим в pgpool:

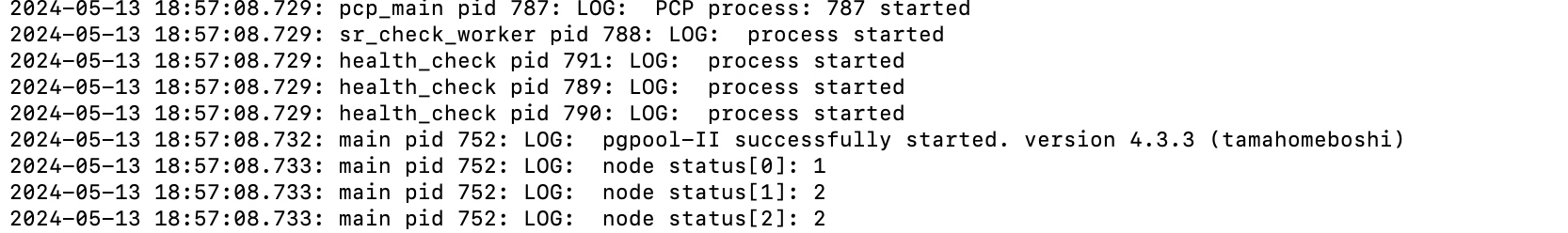
****

Сначала видим, что standby node только один - С, после запуска сервера B их стало две и начала падать ошибка подключения к серверу А. Включаем А:

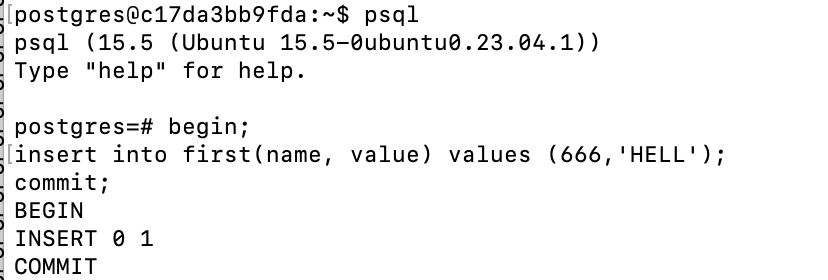


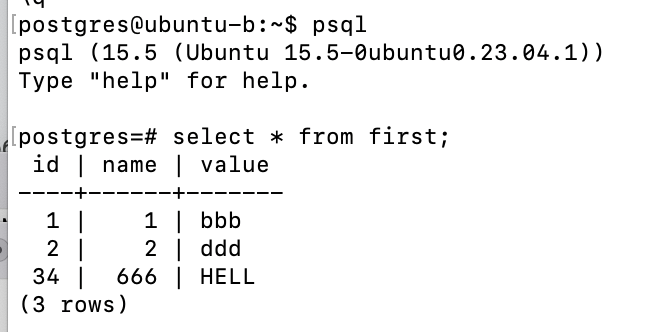
Теперь стало 3 standby ноды, осталось сделать:

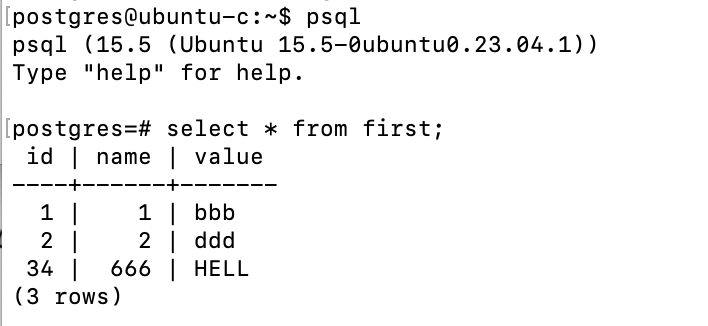
**pg\_ctlcluster 15 main promote**

****

Видим, что мастером стал сервер А. Проверим репликацию:

****

****

****

Все вернулось в исходное состояние!

**Выводы**

В ходе выполнения данной работы я научился строить отказоустойчивые решения на базе СУБД Postgres, получил практические навыки восстановления работы системы после отказа.

Также научился работать с pgpool-II с целью автоматического назначения новой мастер ноды, если текущая не способна работать корректно.

Кроме того, разобрался с различными видами сетей в docker, а также узнал про команду docker commit для создания image с текущим состоянием контейнера.