САНКТ–ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчёт по лабораторной работе 1

по курсу «Администрирование компьютерных сетей»

ЛР3 на Postgres Cluster

Выполнили:
Зотеев М., Иванов В., Скоблилова В.,
Дементьев А., Мельникова Н., Богданов М.
Проверил:
Самохин Н.Ю.

Санкт–Петербург
2025 г.

Содержание

1 Цель работы

Развернуть кластер PostgreSQL с высокой доступностью на базе Patroni и ZooKeeper, проверить репликацию и поведение при отказах, а также настроить HAProxy в качестве entrypoint для клиентских подключений. Дополнительно выполнить условие: после возвращения второй ноды в кластер она должна автоматически догонять данные, записанные в её отсутствие (catch-up по WAL).

2 Исходные условия и окружение

2.1 Окружение

Работа выполнялась на локальной машине (macOS), без обязательного использования виртуальной машины: все компоненты поднимались в Docker контейнерах.

2.2 Инструменты

- Docker + Docker Compose
- DBeaver (для проверки подключений и выполнения SQL)
- PostgreSQL внутри контейнеров (psql вызывается из контейнеров)

2.3 Замечание по `psql` на хосте

На macOS команда `psql` в системе не была установлена, поэтому проверки через entrypoint выполнялись через `psql` внутри контейнеров:

Листинг 1: Отсутствие psql на хосте

```
zsh: command not found: psql
```

3 Структура стенда и общая схема

Стенд состоит из:

- **ZooKeeper** (DCS для Patroni)
- **pg-master** и **pg-slave** (две ноды PostgreSQL под управлением Patroni)
- **HAProxy** (entrypoint) для клиентских подключений на один адрес/порт

4 Развёртывание (Docker Compose)

4.1 Старт кластера

Команда запуска:

Листинг 2: Запуск Docker Compose

```
docker compose up -d --build
```

Первичное предупреждение Compose (не критично, но лучше удалить поле `version` из compose-файла):

Листинг 3: Предупреждение о version

```
WARN[0000] ... docker-compose.yml: the attribute ‘version‘ is obsolete, it will be
    ignored, please remove it to avoid potential confusion
```

4.2 Контейнеры поднялись

Листинг 4: Результат запуска контейнеров

```
 pg-master Built 0.0s
 pg-slave Built 0.0s
 Network lab_1_default Created 0.0s
 Volume "lab_1_pg-master-data" Created 0.0s
 Volume "lab_1_pg-slave-data" Created 0.0s
 Container zoo Started 0.6s
 Container pg-slave Started 0.3s
 Container pg-master Started
```

4.3 Проверка статуса контейнеров и портов

Листинг 5: Проверка docker ps и портов

```
docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"

NAMES STATUS PORTS
pg-slave Up 47 hours 8008/tcp, 0.0.0.0:5434->5432/tcp
pg-master Up 47 hours 8008/tcp, 0.0.0.0:5433->5432/tcp
zoo Up 47 hours 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp, 8080/tcp
```

5 Диагностика проблем и исправления

В процессе выполнения возникли несколько типовых проблем при поднятии кластера на macOS. Ниже приведён полный ход диагностики и исправлений (по шагам, с командами и фактическими выводами).

5.1 Проблема 1: Docker daemon не запущен

Листинг 6: Ошибка подключения к Docker daemon

```
unable to get image ’zookeeper:3.9’: Cannot connect to the Docker daemon at unix:///
    Users/.../docker.sock. Is the docker daemon running?
```

Решение: запустить Docker Desktop, затем повторить `docker compose up`.

5.2 Проблема 2: PostgreSQL на pg-slave не стартует из-за прав на data directory

Попытка подключиться к pg-slave неудачна:

Листинг 7: Подключение к pg-slave через socket

```
docker exec -it pg-slave psql -U postgres -d postgres -c "select pg_is_in_recovery(),
    now();"

psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432"
   failed: No such file or directory
Is the server running locally and accepting connections on that socket?
```

Попытка подключиться по TCP внутри контейнера:

Листинг 8: Подключение к pg-slave по TCP внутри контейнера

```
docker exec -it pg-slave psql -h 127.0.0.1 -p 5432 -U postgres -d postgres -c "select
    pg_is_in_recovery(), now();"

psql: error: connection to server at "127.0.0.1", port 5432 failed: Connection
   refused
Is the server running on that host and accepting TCP/IP connections?
```

Проверка процессов показала, что Patroni запущен, но PostgreSQL не поднимается:

Листинг 9: Проверка процессов внутри pg-slave

```
docker exec -it pg-slave sh -lc "ps aux | egrep 'patroni|postgres' | grep -v egrep"

postgres 1 ... /opt/patroni-venv/bin/python3 ... patroni /postgres1.yml
postgres 6 ... /opt/patroni-venv/bin/python3 ... patroni /postgres1.yml
```

Ключевая причина из логов pg-slave:

Листинг 10: Ошибка прав на /var/lib/postgresql/data

```
docker logs --tail=250 pg-slave
...
FATAL: data directory "/var/lib/postgresql/data" has invalid permissions
DETAIL: Permissions should be u=rwx (0700) or u=rwx,g=rx (0750).
...
```

5.2.1 Решение: принудительная фиксация прав перед запуском Patroni

Для стабильного старта (особенно на macOS с volume) добавлено исправление прав и владельца каталога данных перед запуском Patroni.

Изначально использовали `bash`, но в процессе выяснилось, что:

- `bash` может отсутствовать в образе (поэтому перешли на `sh`)
- `patroni` не всегда лежит в PATH при запуске через shell (поэтому запущен по абсолютному пути)

Симптом: patroni not found

Листинг 11: patroni не найден при запуске через exec

```
docker logs --tail=200 pg-slave
bash: line 1: exec: patroni: not found
```

Итоговое исправление команды запуска Patroni В `docker-compose.yml` для нод PostgreSQL использовано:

Листинг 12: Итоговый запуск Patroni с фиксацией прав и абсолютным путём

```
command: >
  sh -lc "chown -R postgres:postgres /var/lib/postgresql/data
  && chmod 700 /var/lib/postgresql/data
  && exec /opt/patroni-venv/bin/patroni /postgres0.yml"
```

и аналогично для второй ноды (только файл конфигурации другой):

Листинг 13: Итоговый запуск Patroni для второй ноды

```
command: >
  sh -lc "chown -R postgres:postgres /var/lib/postgresql/data
  && chmod 700 /var/lib/postgresql/data
  && exec /opt/patroni-venv/bin/patroni /postgres1.yml"
```

После правки выполнялась пересборка/перезапуск:

Листинг 14: Пересоздание окружения с volumes

```
docker compose down -v
docker compose up -d --build
```

6 Проверка ролей: master/slave

После успешного старта проверяется, что одна нода — лидер (master), а другая — реплика (slave). Основной признак: `pg_is_in_recovery()`.

6.1 Проверка pg-slave

Листинг 15: Проверка что pg-slave в recovery

```
docker exec -it pg-slave psql -h 127.0.0.1 -p 5432 -U postgres -d postgres -c "select
    pg_is_in_recovery(), now();"

Password for user postgres:
pg_is_in_recovery | now
------------------+-------------------------------
t | 2025-12-14 04:41:23.679251+00
(1 row)
```

Значение `t` означает: нода находится в recovery и работает как реплика (read-only).

7 Проверка read-only режима реплики

На реплике запись запрещена. Проверка:

Листинг 16: Попытка INSERT на реплике

```
docker exec -e PGPASSWORD=postgres -it pg-slave psql -h 127.0.0.1 -p 5432 -U postgres
    -d postgres -c "
insert into test_replication(msg) values ('should fail');
"

ERROR: cannot execute INSERT in a read-only transaction
```

Результат соответствует ожидаемому поведению: реплика read-only.

8 Улучшение: автоматический догон данных после возврата ноды

Требование улучшения: после возврата второй ноды (реплики) в кластер она должна автоматически получить данные, записанные во время её отсутствия. Это реализуется механизмом потоковой репликации WAL.

8.1 Сценарий

1. Остановить реплику.
2. Записать данные на лидере.
3. Запустить реплику обратно.
4. Проверить, что новые данные появились на реплике.

8.2 Команды (типовой сценарий)

Листинг 17: Остановка реплики

```
docker stop pg-slave
```

Листинг 18: Запись на мастере во время отсутствия реплики

```
docker exec -e PGPASSWORD=postgres -it pg-master psql -h 127.0.0.1 -p 5432 -U
   postgres -d postgres -c "
insert into test_replication(msg) values ('written while slave down');
"
```

Листинг 19: Возврат реплики

```
docker start pg-slave
```

Листинг 20: Проверка на реплике после возврата

```
docker exec -e PGPASSWORD=postgres -it pg-slave psql -h 127.0.0.1 -p 5432 -U postgres
    -d postgres -c "
select * from test_replication order by id desc limit 5;
"
```

9 HAProxy как entrypoint (часть высокой доступности)

Для максимального результата настроен HAProxy, чтобы клиенты подключались к кластеру через один порт.

9.1 Конфигурация HAProxy (haproxy.cfg)

Файл `haproxy.cfg` в корне проекта:

Листинг 21: haproxy.cfg

```
global
  maxconn 100

defaults
  log global
  mode tcp
  retries 3
  timeout client 30m
  timeout connect 4s
  timeout server 30m
  timeout check 5s

listen stats
  mode http
  bind *:7000
  stats enable
  stats uri /

listen postgres
  bind *:5432
  option httpchk
  http-check expect status 200
  default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
  server postgresql_pg_master_5432 pg-master:5432 maxconn 100 check port 8008
  server postgresql_pg_slave_5432 pg-slave:5432 maxconn 100 check port 8008
```

9.2 Проблема порта 7000: address already in use

При старте HAProxy возник конфликт порта:

Листинг 22: Конфликт порта stats 7000

```
docker compose up -d
...
Error response from daemon: Ports are not available: exposing port TCP 0.0.0.0:7000
    ... bind: address already in use
```

9.3 Решение: переназначение внешнего порта stats

В `docker-compose.yml` внешний порт stats перенесён на `7001`, при этом внутри контейнера stats остаётся на `7000`:

Листинг 23: ports для HAProxy

```
ports:
  - "5432:5432"
  - "7001:7000"
```

9.4 Проверка что HAProxy поднялся

Листинг 24: Проверка портов и контейнеров после добавления HAProxy

```
docker ps --format "table {{.Names}}\t{{.Status}}\t{{.Ports}}"

NAMES STATUS PORTS
postgres_entrypoint Up 38 seconds 0.0.0.0:5432->5432/tcp, 0.0.0.0:7001->7000/tcp
pg-master Up 39 seconds 8008/tcp, 0.0.0.0:5433->5432/tcp
pg-slave Up 39 seconds 8008/tcp, 0.0.0.0:5434->5432/tcp
zoo Up 39 seconds 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp, 8080/tcp
```

9.5 Проверка, что entrypoint ведёт на master

Так как `psql` на хосте отсутствовал, проверка выполнялась через `psql` из контейнера. Подключение осуществляется к `postgres_entrypoint:5432`:

Листинг 25: Проверка роли через entrypoint HAProxy

```
docker exec -e PGPASSWORD=postgres -it pg-master psql -h postgres_entrypoint -p 5432
    -U postgres -d postgres -c "select pg_is_in_recovery();"

 pg_is_in_recovery
-------------------
 f
(1 row)
```

Значение `f` подтверждает, что запрос через HAProxy попал на мастер.

10 Failover через entrypoint (максимальный сценарий)

Цель: показать, что при падении лидера клиенты продолжают работать через HAProxy.

10.1 Определение лидера по логам Patroni

Из логов видно, что лидер — `pg-slave`, а `pg-master` следует за лидером:

Листинг 26: Фрагменты логов: определение лидера

```
... INFO: no action. I am (pg-master), a secondary, and following a leader (pg-slave)
... INFO: no action. I am (pg-slave), the leader with the lock
```

10.2 Остановка лидера

Листинг 27: Остановка текущего лидера pg-slave

```
docker stop pg-slave

pg-slave
```

10.3 Запись через HAProxy после падения лидера

Вставка строки через entrypoint (HAProxy):

Листинг 28: INSERT через HAProxy после падения лидера

```
docker exec -e PGPASSWORD=postgres -it pg-master \
psql -h postgres_entrypoint -p 5432 -U postgres -d postgres \
-c "insert into test_replication(msg) values ('write via haproxy after leader down')
    ;"

INSERT 0 1
```

10.4 Проверка данных через HAProxy

Листинг 29: SELECT через HAProxy: подтверждение результата

```
docker exec -e PGPASSWORD=postgres -it pg-master \
psql -h postgres_entrypoint -p 5432 -U postgres -d postgres \
-c "select id,msg from test_replication order by id desc limit 10;"

 id | msg
----+-------------------------------------
  2 | write via haproxy after leader down
  1 | baseline via haproxy
(2 rows)
```

Данный результат подтверждает:

- Patroni выполнил failover после падения лидера
- HAProxy направил подключение на актуальный мастер
- Клиентская запись успешно выполняется через единый entrypoint

11 Работа в DBeaver (со скриншотами)

В этой части допускается использовать скриншоты. Сюда вставляются изображения процесса подключения и выполнения запросов.

11.1 Подключение к нодам напрямую

- pg-master: `localhost:5433`
- pg-slave: `localhost:5434`

11.2 Подключение через entrypoint (HAProxy)

- HAProxy entrypoint: `localhost:5432`

11.3 Скриншоты

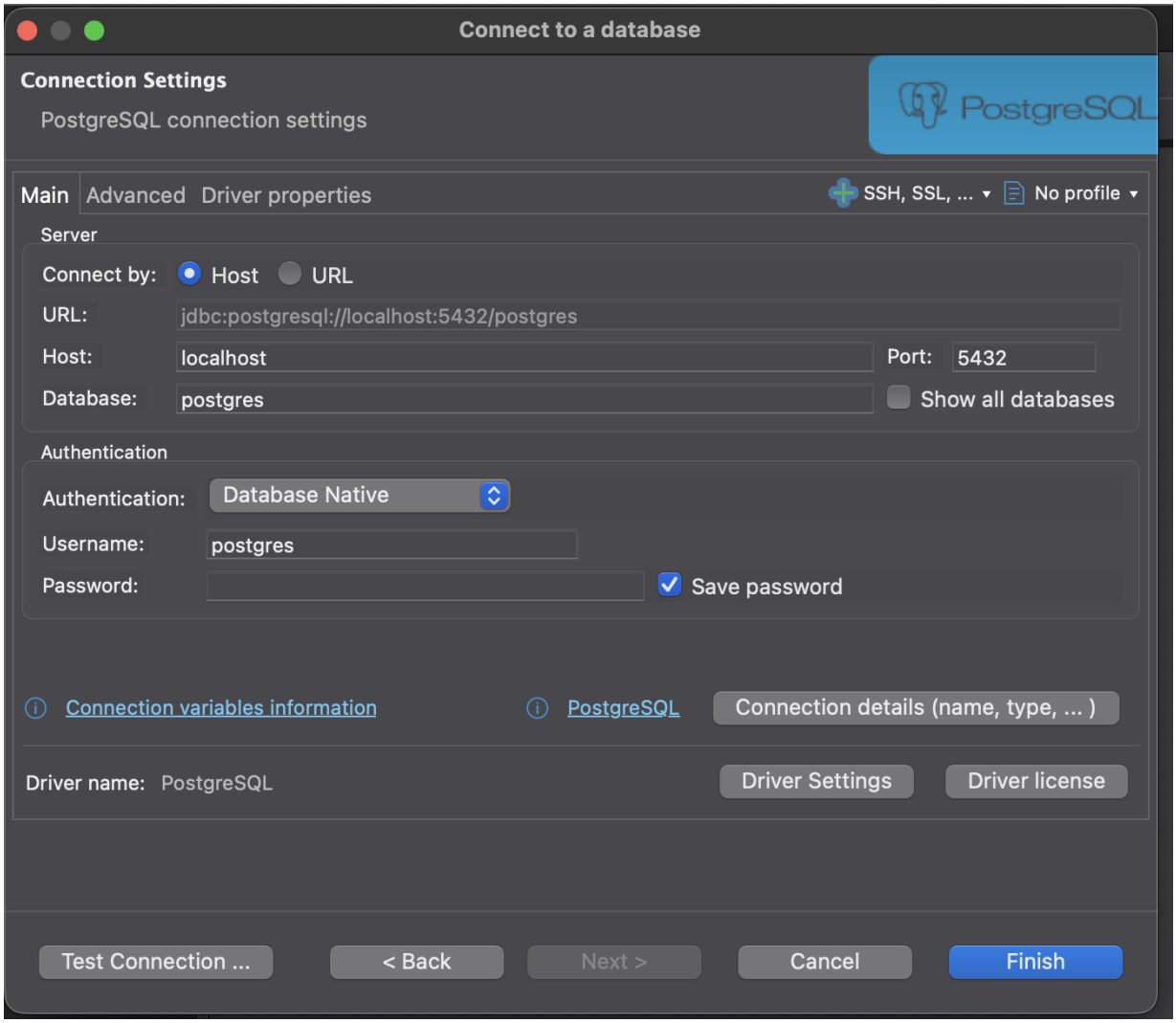

Рис. 1: Настройка подключения к PostgreSQL через HAProxy (localhost:5432) в DBeaver

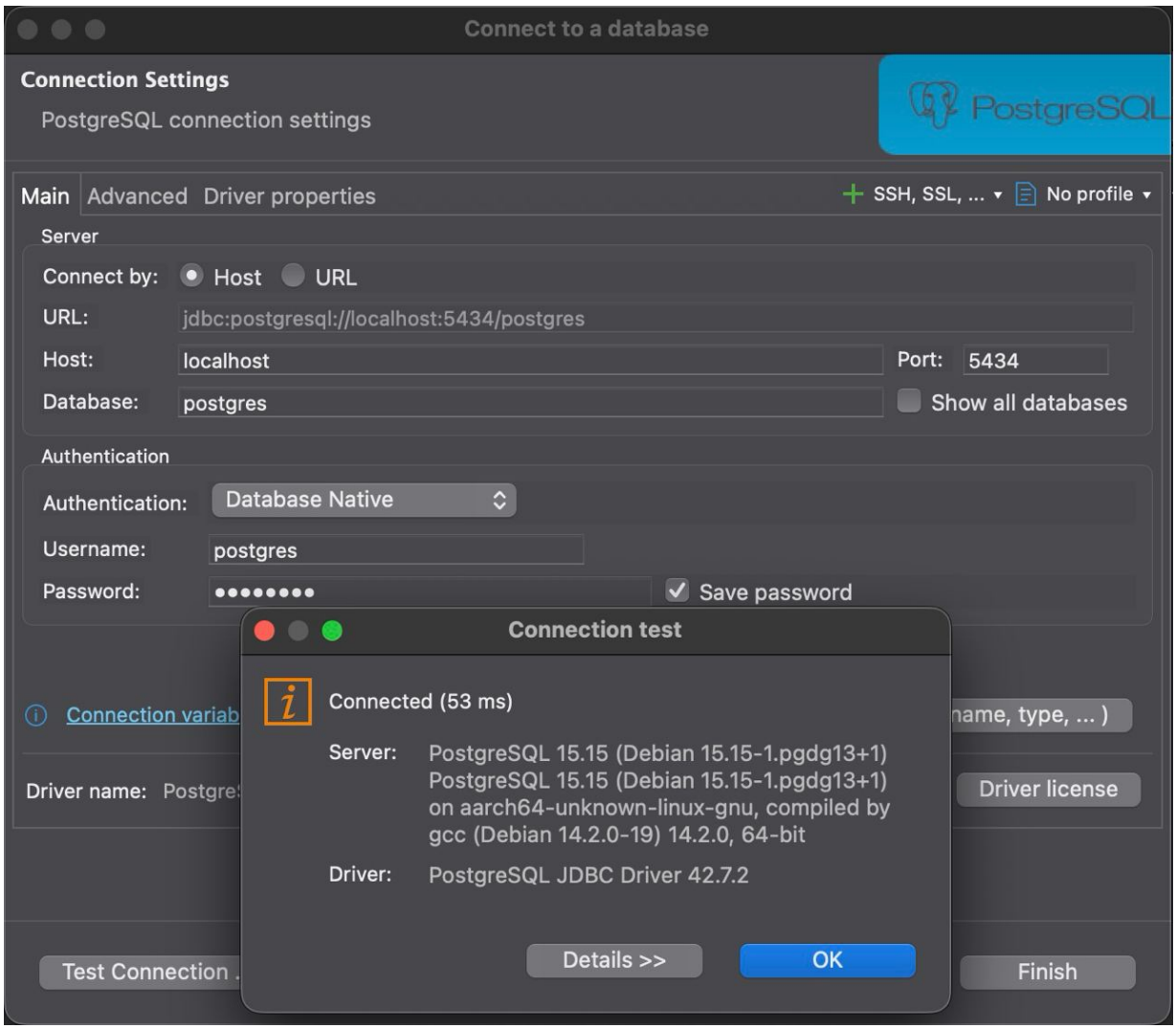

Рис. 2: Успешное подключение к PostgreSQL через HAProxy, подтверждённое тестом соединения в DBeaver

12 Выводы

- Развёрнут кластер PostgreSQL высокой доступности: Patroni + ZooKeeper + две ноды.
- Подтверждены роли нод: мастер (`pg_is_in_recovery() = f`) и реплика (`pg_is_in_recovery() = t`).
- Подтверждено read-only поведение реплики: `INSERT` на реплике приводит к ошибке `cannot execute INSERT in a read-only transaction`.
- Выполнено улучшение: после возврата ноды в кластер данные автоматически догоняются (WAL catch-up).
- Настроен HAProxy как entrypoint. При падении лидера запись через entrypoint продолжает работать, что подтверждено успешным `INSERT` и `SELECT` после остановки лидирующей ноды.

A Приложение А: Блок HAProxy в docker-compose.yml

Листинг 30: haproxy service (пример)

```
haproxy:
  image: haproxy:3.0
  container_name: postgres_entrypoint
  restart: always
  depends_on:
    - pg-master
    - pg-slave
    - zoo
  ports:
    - "5432:5432"
    - "7001:7000"
  volumes:
    - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro
```