Федеральное государственное автономное образовательное
учреждение высшего образования
Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина *«Распределенные системы хранения данных»*

Отчет по лабораторной работе №2

Вариант: 368114

Студент:

Барсуков Максим Андреевич,
группа P3315

Преподаватель:

Харитонова Анастасия Евгеньевна

г. Санкт-Петербург, 2025 г.

Оглавление

Задание

Цель работы – на выделенном узле создать и сконфигурировать новый кластер БД Postgres, саму БД, табличные пространства и новую роль, а также произвести наполнение базы в соответствии с заданием. Отчёт по работе должен содержать все команды по настройке, скрипты, а также измененные строки конфигурационных файлов.

Этап 1. Инициализация кластера БД

- Директория кластера: $HOME/ubi26
- Кодировка: UTF8
- Локаль: русская
- Параметры инициализации задать через переменные окружения

Этап 2. Конфигурация и запуск сервера БД

- Способы подключения: 1) Unix-domain сокет в режиме peer; 2) сокет TCP/IP, только localhost
- Номер порта: 9114
- Способ аутентификации TCP/IP клиентов: по паролю в открытом виде
- Остальные способы подключений запретить.
- Настроить следующие параметры сервера БД:
    - max_connections
    - shared_buffers
    - temp_buffers
    - work_mem
    - checkpoint_timeout
    - effective_cache_size
    - fsync
    - commit_delay
- Параметры должны быть подобраны в соответствии со сценарием OLTP:
  1000 транзакций в секунду размером 32КБ; обеспечить высокую доступность (High Availability) данных.
- Директория WAL файлов: $PGDATA/pg_wal
- Формат лог-файлов: .csv
- Уровень сообщений лога: NOTICE
- Дополнительно логировать: завершение сессий и продолжительность выполнения команд

Этап 3. Дополнительные табличные пространства и наполнение базы

- Создать новые табличные пространства для партицированной таблицы: $HOME/idd21, $HOME/gzp28
- На основе template0 создать новую базу: fakebrownroad
- Создать новую роль, предоставить необходимые права, разрешить подключение к базе.
- От имени новой роли (не администратора) произвести наполнение ВСЕХ созданных баз тестовыми наборами данных. ВСЕ табличные пространства должны использоваться по назначению.
- Вывести список всех табличных пространств кластера и содержащиеся в них объекты.

Выполнение

Этап 1. Инициализация кластера БД

Устанавливаем переменные окружения:

```
export PGDATA=$HOME/ubi26
export PGWAL=$PGDATA/pg_wal
export PGLOCALE=ru_RU.UTF-8
export PGENCODE=UTF8
export PGUSERNAME=postgres1
export PGHOST=pg186
export LANG=ru_RU.UTF-8
export LC_ALL=ru_RU.UTF-8
```

Создаём директорию кластера `ubi26`, инициализируем его и запускаем сервер:

```
# mkdir -p $PGDATA
# mkdir -p $PGWAL

initdb -D "$PGDATA" --encoding=$PGENCODE --locale=$PGLOCALE
--lc-messages=$PGLOCALE --lc-monetary=$PGLOCALE --lc-numeric=$PGLOCALE
--lc-time=$PGLOCALE --no-locale --username=$PGUSERNAME

pg_ctl -D $PGDATA -l $PGDATA/server.log start
```

Теперь можем перейти к настройке файлов конфигурации.

Этап 2. Конфигурация и запуск сервера БД

Конфигурируем порт и прослушиваемые адресы:

```
echo "listen_addresses = '*'" >> $PGDATA/postgresql.conf
echo "port = 9114" >> $PGDATA/postgresql.conf
```

Редактируем `pg_hba.conf`:

```
cat > $PGDATA/pg_hba.conf << EOF
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             127.0.0.1/32            password
host    all             all             ::1/128                 password
local   all             all                                     peer
EOF

pg_ctl -D $PGDATA restart
```

Проверяем возможные подключения:

```
# Подключение через Unix-сокет
psql -h /var/db/postgres1/ubi26 -p 9114 postgres

# Подключение через TCP/IP (IPv4)
psql -h 127.0.0.1 -p 9114 postgres
psql -h 127.0.0.1 -U $PGUSERNAME -p 9114 postgres

# Удалённое подключение с гелиоса
psql -h pg186 -p 9114 -U postgres1 postgres
```

Воспользуемся инструкцией `ALTER SYSTEM`, чтобы записать параметры из консоли `psql`.

При использовании этого варианта параметры запишутся в файл `postgresql.auto.conf`, который имеет приоритет и используется для дополнительной конфигурации после конфигурации через обычный `postgresql.conf`:

```
psql -h 127.0.0.1 -U $PGUSERNAME -p 9114 postgres

-- Принимать подключения с любого IP-адреса
ALTER SYSTEM SET listen_addresses = '*';
-- Номер порта по варианту
ALTER SYSTEM SET port = 9114;

ALTER SYSTEM SET max_connections = 20;
ALTER SYSTEM SET superuser_reserved_connections = 5;
ALTER SYSTEM SET shared_buffers = '40MB';
ALTER SYSTEM SET work_mem = '1MB';
```

```
ALTER SYSTEM SET temp_buffers = '4MB';
ALTER SYSTEM SET effective_cache_size = '100MB';
ALTER SYSTEM SET maintenance_work_mem = '20MB';

ALTER SYSTEM SET wal_level = 'replica';
ALTER SYSTEM SET wal_buffers = '1MB';
ALTER SYSTEM SET wal_compression = 'on';
ALTER SYSTEM SET max_wal_size = '100MB';
-- min_wal_size должен быть минимум вдвое больше wal_segment_size (16MB)
ALTER SYSTEM SET min_wal_size = '40MB';
ALTER SYSTEM SET checkpoint_timeout = '5min';
ALTER SYSTEM SET checkpoint_completion_target = 0.9;
ALTER SYSTEM SET fsync = on;
ALTER SYSTEM SET synchronous_commit = 'remote_write';
ALTER SYSTEM SET wal_log_hints = on;
ALTER SYSTEM SET commit_delay = 1000;
ALTER SYSTEM SET commit_siblings = 2;
ALTER SYSTEM SET default_statistics_target = 50;
ALTER SYSTEM SET random_page_cost = 1.1;
ALTER SYSTEM SET effective_io_concurrency = 2;
ALTER SYSTEM SET max_worker_processes = 2;
ALTER SYSTEM SET max_parallel_workers_per_gather = 0;
ALTER SYSTEM SET bgwriter_delay = '100ms';
ALTER SYSTEM SET bgwriter_lru_maxpages = 100;
ALTER SYSTEM SET bgwriter_lru_multiplier = 2.0;

ALTER SYSTEM SET log_destination = 'csvlog';
ALTER SYSTEM SET logging_collector = on;
ALTER SYSTEM SET log_filename = 'postgresql-%Y-%m-%d_%H%M%S.csv';
ALTER SYSTEM SET log_min_messages = 'notice';
ALTER SYSTEM SET log_connections = on;      -- Начало сессии
ALTER SYSTEM SET log_disconnections = on;   -- Завершение сессии
ALTER SYSTEM SET log_duration = on;         -- Время выполнения команд
ALTER SYSTEM SET log_min_duration_statement = 0;-- Логировать все запросы
ALTER SYSTEM SET log_checkpoints = on;
ALTER SYSTEM SET log_lock_waits = on;
ALTER SYSTEM SET deadlock_timeout = '1s';
ALTER SYSTEM SET idle_in_transaction_session_timeout = '1min';
ALTER SYSTEM SET statement_timeout = '30s';
ALTER SYSTEM SET lock_timeout = '10s';
ALTER USER postgres1 WITH PASSWORD '1234';
\q
```

```
pg_ctl -D "$PGDATA" restart
```

OLTP

Параметры должны быть подобраны в соответствии со сценарием OLTP:
1000 транзакций в секунду размером **32КБ**; обеспечить высокую доступность (High Availability) данных.

Конфигурация под заданные требования:

```
ALTER SYSTEM SET max_connections = 300;
ALTER SYSTEM SET superuser_reserved_connections = 5;

ALTER SYSTEM SET shared_buffers = '8GB';
ALTER SYSTEM SET work_mem = '8MB';
ALTER SYSTEM SET temp_buffers = '64MB';
ALTER SYSTEM SET effective_cache_size = '24GB';
ALTER SYSTEM SET maintenance_work_mem = '1GB';

ALTER SYSTEM SET wal_level = 'replica';
ALTER SYSTEM SET wal_buffers = '16MB';
ALTER SYSTEM SET wal_compression = 'on';
ALTER SYSTEM SET max_wal_size = '8GB';
ALTER SYSTEM SET min_wal_size = '2GB';
ALTER SYSTEM SET checkpoint_timeout = '10min';
ALTER SYSTEM SET checkpoint_completion_target = 0.9;
ALTER SYSTEM SET fsync = on;
ALTER SYSTEM SET synchronous_commit = 'remote_write';
ALTER SYSTEM SET wal_log_hints = on;

ALTER SYSTEM SET commit_delay = 10000;
ALTER SYSTEM SET commit_siblings = 5;
ALTER SYSTEM SET default_statistics_target = 100;
ALTER SYSTEM SET random_page_cost = 1.1;
ALTER SYSTEM SET effective_io_concurrency = 200;
ALTER SYSTEM SET max_worker_processes = 8;
ALTER SYSTEM SET max_parallel_workers_per_gather = 4;

ALTER SYSTEM SET bgwriter_delay = '10ms';
ALTER SYSTEM SET bgwriter_lru_maxpages = 1000;
ALTER SYSTEM SET bgwriter_lru_multiplier = 5.0;

ALTER SYSTEM SET log_checkpoints = on;
ALTER SYSTEM SET log_lock_waits = on;
ALTER SYSTEM SET deadlock_timeout = '1s';
ALTER SYSTEM SET idle_in_transaction_session_timeout = '1min';
ALTER SYSTEM SET statement_timeout = '30s';
ALTER SYSTEM SET lock_timeout = '10s';
```

Попробую обосновать выбранные мною значения:

Память	
`shared_buffers = 8GB`	Это 25% от 32 ГБ RAM сервера — стандартная рекомендация для OLTP. Размер позволяет кэшировать ~262 тыс. транзакций по 32 КБ, снижая частые чтения с диска. Баланс между кэшем СУБД и памятью ОС для буферизации WAL и временных файлов.
`work_mem = 8MB`	Рассчитано как (RAM − shared_buffers) / max_connections × 0.1 = (32GB − 8GB)/300 × 0.1 ≈ 8MB. Ограничивает память на сортировки/хэширование для 300 сессий, предотвращая OOM-ошибки при параллельных операциях.
`temp_buffers = 64MB`	Эмпирическое значение для временных таблиц и сортировок в OLTP. Позволяет хранить ~2000 временных объектов по 32 КБ на сессию без записи на диск (64MB × 300 = 19.2GB в пределах RAM), минимизируя I/O.
`maintenance_work_mem = 1GB`	Выделяет память для операций обслуживания (VACUUM, CREATE INDEX). Для OLTP с частыми обновлениями увеличение до 1 ГБ ускоряет очистку «мертвых» строк, предотвращая разрастание таблиц. Оптимально для сервера с 32 ГБ RAM.
`effective_cache_size = 24GB`	Указывает планировщику, что 75% RAM (32GB − 8GB) доступно для кэша ОС. Помогает выбирать оптимальные планы запросов (например, индексные сканы вместо seqscan), так как реальные данные часто кэшируются ОС.

WAL	
`wal_compression = on`	Для снижения нагрузки на диск и сеть при 1000 транзакций/сек размером 32 КБ включено сжатие WAL. Алгоритм LZ4 уменьшает объем записываемых данных на 30-40%, что критично для HA-кластера с репликацией, сокращая задержки и потребление пропускной способности. Это дает баланс между CPU-нагрузкой и производительностью.
`wal_buffers = 16MB`	Размер буфера для WAL перед записью на диск. Для 1000 TPS × 32 КБ = 32 МБ/сек: 16 МБ хватает на ~0.5 сек группировки записей. Автонастройка (1/32 shared_buffers = 256 МБ) избыточна для OLTP.
`checkpoint_timeout = 10min`	Частые контрольные точки (каждые 10 мин) снижают объем данных для восстановления при сбое (HA-требование). В сочетании с max_wal_size = 8GB обеспечивает плавную запись WAL (8GB / 600 сек = ~13.6MB/сек), что подходит для SATA SSD.
`checkpoint_completion_target = 0.9`	Растягивает запись checkpoint на 90% интервала между контрольными точками. Равномерно распределяет нагрузку на диск, избегая «шторков» I/O.
`fsync = on`	Критично для High Availability: гарантирует, что данные физически записаны на диск перед подтверждением транзакции. Несмотря на снижение TPS на 10-15%, исключает потерю данных при аварийном отключении.
`max_wal_size = 8GB` `min_wal_size = 2GB`	Ограничивают размер WAL-сегментов. При `checkpoint_timeout = 10min` 8 ГБ позволяют генерировать до 13.6 МБ/сек WAL (8 ГБ / 600 сек), что покрывает пиковую нагрузку 32 МБ/сек (1000 × 32 КБ) с запасом.
`synchronous_commit = 'remote_write'`	Гарантирует запись WAL в буфер реплики перед подтверждением транзакции. Баланс

	между надежностью HA-кластера и производительностью: снижает задержку по сравнению с on, но безопаснее off.
Транзакции, параллелизм, управление ресурсами	
`max_connections = 300`	Для обработки 1000 транзакций/сек я выбрал 300 подключений, так как OLTP-системы часто используют пулы соединений. При средней длительности транзакции ~3 мс, 300 одновременных сессий обеспечат пропускную способность 1000 TPS (1000 / (1/0.003) ≈ 300) с запасом для пиковых нагрузок, исключая ошибки «too many connections». (с учетом пулов соединений (pgBouncer), реальная нагрузка: ~3-5 транзакций на соединение)
`commit_delay = 10000`	Группирует коммиты транзакций, сокращая количество операций записи в WAL. При 1000 TPS за 10ms накапливается ~10 транзакций (1000/100), уменьшая нагрузку на диск в 10 раз. Баланс между задержкой и производительностью.
`commit_siblings = 5`	Активирует группировку коммитов (commit_delay) только при наличии ≥5 активных транзакций. Для OLTP с 1000 TPS это позволяет объединять до 10-20 операций в одну запись WAL, снижая нагрузку на диск без увеличения задержки для изолированных запросов.
`statement_timeout = '30s'`	Прерывает запросы, выполняющиеся дольше 30 секунд. Защищает от "зависших" операций, которые могут блокировать ресурсы и исчерпать пул подключений (300 сессий × 30 сек = 9000 транзакций). Для OLTP с быстрыми транзакциями (обычно <1 сек) это предотвращает деградацию производительности.
`idle_in_transaction_session_timeout = '1min'`	Автоматически завершает сессии, оставшиеся в открытой транзакции без активности более 1 минуты. Устраняет

	блокировки строк/таблиц из-за "забытых" транзакций, критично для HA-кластера, где каждая реплика должна обрабатывать актуальные данные.
`max_worker_processes = 8`	Максимальное число фоновых процессов (для параллельных запросов, репликации). На сервере с 8+ ядрами CPU это позволяет обрабатывать до 8 параллельных запросов.
`max_parallel_workers_per_gather = 4`	Ограничивает число потоков на один запрос. Для OLTP с короткими транзакциями 4 потока предотвращают «распыление» ресурсов CPU, сохраняя ядра для обработки новых запросов.
Высокая доступность	
`hot_standby = on`	Разрешает выполнение запросов на чтение на репликах. Для OLTP-нагрузки это позволяет распределить SELECT-запросы между узлами, снижая нагрузку на мастер. Реплики работают с небольшим отставанием (1-2 сек), сохраняя доступность при сбоях.
`archive_command = 'gzip < %p > /wal_archive/%f.gz'`	Архивирует WAL-логи в сжатом виде для восстановления на конкретный момент времени (PITR). Для HA критично хранить полную историю изменений: при потере всех узлов кластер можно восстановить из архива. Интеграция с облачным хранилищем (например, aws s3 cp) повышает отказоустойчивость.
`wal_level = 'replica'`	minimal + point-in-time recovery Включает логирование, необходимое для репликации. Минимальный уровень для HA, позволяет создавать физические реплики. Для логической репликации потребовалось бы logical.
Оптимизация запросов	
`default_statistics_target = 100`	Увеличивает точность статистики для планировщика запросов. Для OLTP с частыми точечными запросами (по индексам) улучшает выбор плана, снижая

	риск full scan.
`random_page_cost = 1.1`	Отражает стоимость случайного чтения для SSD/NVMe. Значение 1.1 (близко к seq_page_cost = 1) стимулирует использование индексных сканов вместо фильтрации больших таблиц.
`effective_io_concurrency = 200`	Задает количество параллельных операций ввода-вывода. Для NVMe дисков (очереди до 256) 200 позволяет полностью использовать их пропускную способность при массовых операциях.
Настройки фоновой записи	
`bgwriter_delay = '10ms'`	Уменьшает интервал между циклами фоновой записи с 200 мс до 10 мс. Для OLTP с высокой частотой обновлений это позволяет быстрее освобождать shared_buffers, снижая риск конфликтов при записи. Однако требует больше CPU.
`bgwriter_lru_maxpages = 1000`	Разрешает записывать до 1000 страниц за цикл. Для 32 КБ транзакций (1000 TPS) это покрывает ~32 МБ данных за цикл, предотвращая накопление грязных страниц и снижая нагрузку на checkpoint.
`bgwriter_lru_multiplier = 5.0`	Агрессивно освобождает буферы: записывает в 5 раз больше страниц, чем оценка свободных буферов. Для HA-кластера с репликацией это минимизирует задержки при синхронной записи.

Память: Баланс между кэшированием (shared_buffers) и оперативными вычислениями (work_mem) предотвращает перегрузку RAM и свопинг.
WAL: Частые контрольные точки + группировка WAL-записей снижают latency транзакций при гарантии сохранности данных.
Транзакции: Настройки под SSD/NVMe и параллелизм максимизируют скорость обработки 1000 TPS.

Проверка производительности:

```
createdb -h localhost -p 9114 pgbench_db
pgbench -h localhost -p 9114 -U $PGUSERNAME -i -s 100 pgbench_db
```

`-s 100`: Создает тестовые таблицы с масштабом 100 (≈1.5 GB данных).
Для OLTP с 1000 TPS масштаб должен быть ≥100, чтобы данные не помещались полностью в RAM.

```
pgbench -h localhost -p 9114 -U $PGUSERNAME \
  -c 200 -j 8 -T 600 -M prepared -r -P 1 \
  -S -s 100 -W 32 pgbench_db
```

Параметры:
`-c 200`: 200 параллельных клиентов (2/3 от max_connections = 300).
`-j 8`: 8 потоков (по числу ядер CPU).
`-T 600`: Длительность теста — 10 минут.
`-M prepared`: Использование prepared statements.
`-r`: Показывать задержки (latency) для каждой операции.
`-P 1`: Выводить прогресс каждую секунду.
`-S`: Только SELECT-запросы (имитация OLTP-нагрузки).
`-W 32`: Размер транзакции — 32 КБ (как в задании).

На целевом сервере проверка через pgbench должна показывать такие ожидаемые результаты:

Параметр	**Целевое значение**	**Обоснование**
TPS (без задержек)	≥ 900	1000 TPS с запасом для HA-настроек
Avg Latency	≤ 15 мс	Для 32 КБ транзакций на NVMe
95th Percentile Latency	≤ 30 мс	Гарантия стабильности при пиках
Buffers Hit Rate	≥ 99%	Эффективность shared_buffers и кэша ОС

Для HA-кластера критично сохранять tps ≥ 900 даже при активации синхронной репликации.

Этап 3. Дополнительные табличные пространства и наполнение базы

Создание табличных пространств:

```
pg_ctl -D $PGDATA start
export PGPASSWORD=1234

mkdir -p $HOME/idd21
mkdir -p $HOME/gzp28

psql -U postgres -h localhost -p 9114 -U postgres1 postgres -c "CREATE
TABLESPACE ts1 LOCATION '$HOME/idd21';"
psql -U postgres -h localhost -p 9114 -U postgres1 postgres -c "CREATE
TABLESPACE ts2 LOCATION '$HOME/gzp28';"
```

Создание базы данных:

```
psql -U postgres1 -h localhost -p 9114 -d postgres

CREATE DATABASE fakebrownroad TEMPLATE template0 ENCODING 'UTF8'
LC_COLLATE='ru_RU.UTF-8' LC_CTYPE='ru_RU.UTF-8';
```

Создание роли:

```
CREATE ROLE data_user LOGIN PASSWORD '1234';
GRANT CONNECT ON DATABASE fakebrownroad TO data_user;
GRANT CONNECT ON DATABASE template0 TO data_user;

\c fakebrownroad

GRANT ALL ON SCHEMA public TO data_user;

GRANT CREATE ON TABLESPACE ts1 TO data_user;
GRANT CREATE ON TABLESPACE ts2 TO data_user;
```

Подключение от лица новой роли:

```
psql -U data_user -d fakebrownroad -h localhost -p 9114
```

Создание тестовых таблиц:

```
CREATE TABLE table1 (id serial PRIMARY KEY, name text) TABLESPACE ts1;
CREATE TABLE table2 (id serial PRIMARY KEY, value integer) TABLESPACE ts2;
CREATE TABLE table3 (id serial PRIMARY KEY, info text) TABLESPACE ts2;

INSERT INTO table1 (name) SELECT 'Имя ' || g FROM generate_series(1, 100) g;
INSERT INTO table2 (value) SELECT g * 10 FROM generate_series(1, 100) g;
INSERT INTO table3 (info) SELECT 'Инфо ' || g FROM generate_series(1, 100) g;
```

Проверка списков табличных пространств:

```
SELECT spcname, pg_tablespace_location(oid) FROM pg_tablespace;
```

Проверка объектов в табличных пространствах:

```
SELECT t.spcname, c.relname
FROM pg_class c
JOIN pg_tablespace t ON c.reltablespace = t.oid
WHERE c.relkind = 'r';
```

Скрипт очистки:

```
rm -rf $HOME/ubi26
rm -rf $HOME/idd21
rm -rf $HOME/gzp28
rm -rf $HOME/logs
```

Демонстрация

Инициализируем кластер:

```
[postgres1@pg186 ~]$ initdb -D "$PGDATA" --encoding=$PGENCODE --locale=$PGLOCALE --lc-messages=$PGLOCALE --lc-
monetary=$PGLOCALE --lc-numeric=$PGLOCALE --lc-time=$PGLOCALE --no-locale --username=$PGUSERNAME

pg_ctl -D $PGDATA -l $PGDATA/server.log start
Файлы, относящиеся к этой СУБД, будут принадлежать пользователю "postgres1".
От его имени также будет запускаться процесс сервера.

Кластер баз данных будет инициализирован со следующими параметрами локали:
  провайдер:   libc
  LC_COLLATE:  C
  LC_CTYPE:    C
  LC_MESSAGES: ru_RU.UTF-8
  LC_MONETARY: ru_RU.UTF-8
  LC_NUMERIC:  ru_RU.UTF-8
  LC_TIME:     ru_RU.UTF-8
Выбрана конфигурация текстового поиска по умолчанию "english".

Контроль целостности страниц данных отключён.

создание каталога /var/db/postgres1/ubi26 ... ок
создание подкаталогов ... ок
выбирается реализация динамической разделяемой памяти ... posix
выбирается значение max_connections по умолчанию ... 100
выбирается значение shared_buffers по умолчанию ... 128MB
выбирается часовой пояс по умолчанию ... Europe/Moscow
создание конфигурационных файлов ... ок
выполняется подготовительный скрипт ... ок
выполняется заключительная инициализация ... ок
сохранение данных на диске ... ок

initdb: предупреждение: включение метода аутентификации "trust" для локальных подключений
initdb: подсказка: Другой метод можно выбрать, отредактировав pg_hba.conf или ещё раз запустив initdb с ключом
 -A, --auth-local или --auth-host.

Готово. Теперь вы можете запустить сервер баз данных:

    pg_ctl -D /var/db/postgres1/ubi26 -l файл_журнала start

ожидание запуска сервера.... готово
сервер запущен
[postgres1@pg186 ~]$
```

Подключаемся:

```
[postgres1@pg186 ~]$ psql -h 127.0.0.1 -U $PGUSERNAME -p 9114 postgres
psql (16.4)
Введите "help", чтобы получить справку.

postgres=#
```

Изменение методов аутентификации:

```
[postgres1@pg186 ~]$
cat > $PGDATA/pg_hba.conf << EOF
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             127.0.0.1/32            password
host    all             all             ::1/128                 password
local   all             all                                     peer
EOF

pg_ctl -D $PGDATA restart

ожидание завершения работы сервера.... готово
сервер остановлен
ожидание запуска сервера....2025-04-07 13:16:14.057 MSK [96045] СООБЩЕНИЕ:  передача вывода в протокол процесс
у сбора протоколов
2025-04-07 13:16:14.057 MSK [96045] ПОДСКАЗКА:  В дальнейшем протоколы будут выводиться в каталог "log".
 готово
сервер запущен
```

```
[postgres1@pg186 ~]$ psql -h 127.0.0.1 -U $PGUSERNAME -p 9114 postgres
Пароль пользователя postgres1:
psql (16.4)
Введите "help", чтобы получить справку.

postgres=#
```

Создание новой БД и роли:

```
postgres=#
postgres=# CREATE DATABASE fakebrownroad TEMPLATE template0 ENCODING 'UTF8'
 LC_COLLATE='ru_RU.UTF-8' LC_CTYPE='ru_RU.UTF-8';
CREATE DATABASE
postgres=#
postgres=# GRANT CONNECT ON DATABASE fakebrownroad TO data_user;
GRANT CONNECT ON DATABASE template0 TO data_user;"
GRANT
GRANT
postgres"# \c fakebrownroad

GRANT ALL ON SCHEMA public TO data_user;

GRANT CREATE ON TABLESPACE ts1 TO data_user;
GRANT CREATE ON TABLESPACE ts2 TO data_user;
postgres"# ^C
postgres=# ^C
postgres=# \q
[postgres1@pg186 ~]$ psql -U postgres1 -h localhost -p 9114 -d postgres
psql (16.4)
Введите "help", чтобы получить справку.

postgres=#
postgres=#
postgres=# \c fakebrownroad
Вы подключены к базе данных "fakebrownroad" как пользователь "postgres1".
fakebrownroad=# GRANT ALL ON SCHEMA public TO data_user;

GRANT CREATE ON TABLESPACE ts1 TO data_user;
GRANT CREATE ON TABLESPACE ts2 TO data_user;
GRANT
GRANT
GRANT
fakebrownroad=# \q
[postgres1@pg186 ~]$
psql -U data_user -d fakebrownroad -h localhost -p 9114

psql (16.4)
Введите "help", чтобы получить справку.

fakebrownroad⇒
```

Проверка объектов в табличных пространствах:

```
fakebrownroad=>
SELECT spcname, pg_tablespace_location(oid) FROM pg_tablespace;

  spcname   | pg_tablespace_location
------------+------------------------
 pg_default |
 pg_global  |
 ts1        | /var/db/postgres1/idd21
 ts2        | /var/db/postgres1/gzp28
(4 строки)

fakebrownroad=>
SELECT t.spcname, c.relname
FROM pg_class c
JOIN pg_tablespace t ON c.reltablespace = t.oid
WHERE c.relkind = 'r';

  spcname  |        relname
-----------+-----------------------
 ts1       | table1
 ts2       | table2
 ts2       | table3
 pg_global | pg_authid
 pg_global | pg_subscription
 pg_global | pg_database
 pg_global | pg_db_role_setting
 pg_global | pg_tablespace
 pg_global | pg_auth_members
 pg_global | pg_shdepend
 pg_global | pg_shdescription
 pg_global | pg_replication_origin
 pg_global | pg_shseclabel
 pg_global | pg_parameter_acl
(14 строк)

fakebrownroad=>
```

Исходный код

https://github.com/maxbarsukov/itmo/tree/master/6%20pcxд/лабораторные/lab2

Вывод

В ходе выполнения лабораторной работы был успешно создан и настроен кластер PostgreSQL с использованием выделенного узла.