



МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«МИРЭА – Российский технологический университет»  
**РТУ МИРЭА**

---

**ОТЧЕТ ПО ПРАКТИКЕ В ПРОФИЛЬНОЙ СФЕРЕ**

**Тема практики: «Администрирование баз данных с помощью PostgreSQL»**

Отчет представлен к  
рассмотрению:

«28» мая 2025 г.

\_\_\_\_\_  
(подпись)

Д.О. Враженко

Москва 2025 г.

# ВВЕДЕНИЕ

Современные приложения, особенно в эпоху микросервисной архитектуры и распределенных систем, требуют надежных, масштабируемых и гибких решений для управления данными. PostgreSQL, являясь объектно-реляционной системой управления базами данных с открытым исходным кодом, занимает ключевое место в экосистеме DevOps благодаря своей стабильности, расширяемости и поддержке сложных операций. Её использование позволяет не только эффективно хранить и обрабатывать данные, но и интегрировать процессы администрирования в конвейеры непрерывной интеграции и доставки (CI/CD), что критически важно для автоматизации и ускорения жизненного цикла разработки.

Администрирование баз данных в контексте DevOps выходит за рамки традиционных задач настройки и оптимизации. Оно подразумевает автоматизацию рутинных операций, таких как развертывание кластеров, настройка репликации, управление резервным копированием, мониторинг производительности и обеспечение безопасности. PostgreSQL, обладая богатым набором инструментов и возможностей, становится идеальной платформой для реализации этих задач, позволяя согласовать работу с данными с принципами Infrastructure as Code (IaC) и гибкого управления конфигурациями.

## **Особенности PostgreSQL, значимые для DevOps:**

- **Расширяемость:** Поддержка пользовательских типов данных, функций и расширений позволяет адаптировать СУБД под специфические требования проектов.
- **Транзакционная надежность:** ACID-совместимость и механизм WAL (Write-Ahead Logging) гарантируют целостность данных даже в условиях высокой нагрузки.

- **Репликация и кластеризация:** Встроенные решения для потоковой репликации и инструменты вроде Patroni упрощают построение отказоустойчивых кластеров.

- **Интеграция с DevOps-инструментами:** Совместимость с Ansible, Terraform, Kubernetes и системами мониторинга (Prometheus, Grafana) обеспечивает seamless-взаимодействие в CI/CD-цепочках.

### **Преимущества использования PostgreSQL в DevOps:**

1. **Автоматизация развертывания:** Использование IaC-подходов позволяет быстро создавать и масштабировать экземпляры БД в облачных и гибридных средах.

2. **Безопасность и аудит:** Гибкие ролевые модели, SSL-шифрование и интеграция с Vault обеспечивают соответствие стандартам compliance.

3. **Эффективное управление ресурсами:** Возможности параллельной обработки запросов и настройки индексов снижают нагрузку на инфраструктуру.

4. **Резервное копирование и восстановление:** Инструменты pg\_dump, pg\_basebackup и WAL-архивация минимизируют риски потери данных.

5. **Мониторинг и оптимизация:** Анализ производительности через pg\_stat\_statements и настройка запросов повышают отзывчивость приложений.

### **Основные компоненты экосистемы PostgreSQL для администрирования:**

- **Процессы СУБД:** Postmaster, Writer, Checkpointer, WAL Sender/Receiver.

- **Конфигурационные файлы:** postgresql.conf, pg\_hba.conf, recovery.conf.

- **Утилиты командной строки:** psql, pg\_ctl, pgbench.

- **Инструменты оркестрации:** Patroni, repmgr, PostgreSQL Operator для Kubernetes.

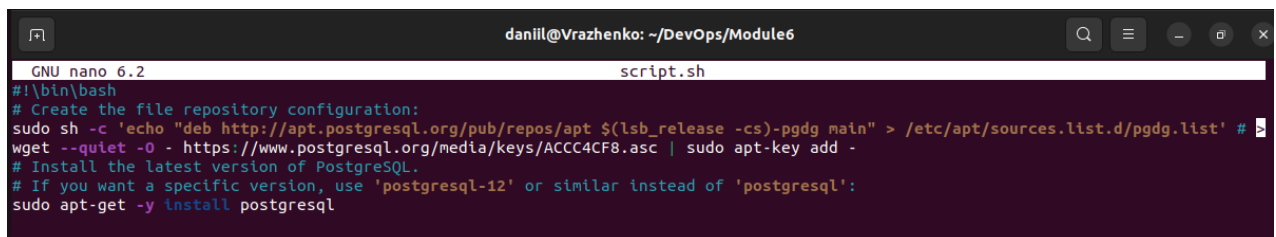
В рамках практики основное внимание было уделено автоматизации процессов администрирования: создание Ansible-ролей для развертывания кластеров, настройка потоковой репликации, интеграция с системами мониторинга, а также разработка скриптов для резервного копирования и восстановления. Эти задачи направлены на сокращение ручного вмешательства, повышение отказоустойчивости и обеспечение согласованности окружений на всех этапах CI/CD.

Использование PostgreSQL в связке с DevOps-практиками не только ускоряет delivery приложений, но и формирует основу для построения надежной, масштабируемой и безопасной data-инфраструктуры, что подтверждается результатами, достигнутыми в ходе производственной практики.

# ОСНОВНАЯ ЧАСТЬ

## 1. Установка PostgreSQL 12 на Ubuntu 22.04

Для установки PostgreSQL 12 на Ubuntu 22.04 был добавлен официальный репозиторий PostgreSQL, так как версия 12 отсутствует в стандартных репозиториях ОС. После настройки источника пакетов выполнена установка сервера PostgreSQL и сопутствующих утилит. Это обеспечило доступ к стабильной и поддерживаемой версии СУБД, совместимой с требованиями проекта.

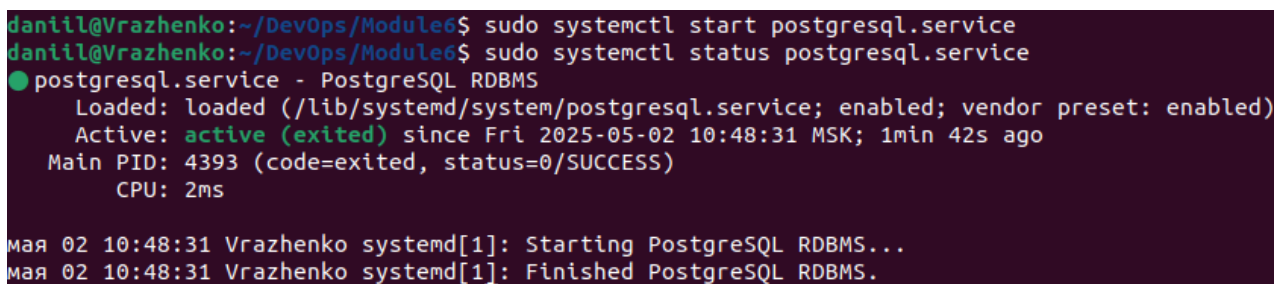


```
danil@Vrazhenko: ~/DevOps/Module6
GNU nano 6.2 script.sh
#!/bin/bash
# Create the file repository configuration:
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list' #
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
# Install the latest version of PostgreSQL.
# If you want a specific version, use 'postgresql-12' or similar instead of 'postgresql':
sudo apt-get -y install postgresql
```

Рисунок 1 - Установка PostgreSQL

## 2. Проверка статуса службы PostgreSQL

После установки проверен статус системного демона postgresql.service. Служба успешно запущена и активна, что подтверждает корректность установки и готовность к работе.



```
danil@Vrazhenko:~/DevOps/Module6$ sudo systemctl start postgresql.service
danil@Vrazhenko:~/DevOps/Module6$ sudo systemctl status postgresql.service
● postgresql.service - PostgreSQL RDBMS
   Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
   Active: active (exited) since Fri 2025-05-02 10:48:31 MSK; 1min 42s ago
     Main PID: 4393 (code=exited, status=0/SUCCESS)
        CPU: 2ms

мая 02 10:48:31 Vrazhenko systemd[1]: Starting PostgreSQL RDBMS...
мая 02 10:48:31 Vrazhenko systemd[1]: Finished PostgreSQL RDBMS.
```

Рисунок 2 - Проверка работоспособности

## 3. Подключение к серверу через psql

Интерактивная консоль psql использована для управления базой данных. Подключение выполнено от имени системного пользователя postgres, что является стандартным подходом для начальной настройки. Проверка

соединения (\conninfo) подтвердила корректность аутентификации и параметров подключения.

```
danil@Vrazhenko:~/DevOps/Module6$ sudo -i -u postgres
postgres@Vrazhenko:~$ psql
psql (14.17 (Ubuntu 14.17-0ubuntu0.22.04.1))
Type "help" for help.

postgres=# \conninfo
You are connected to database "postgres" as user "postgres" via socket in "/var/run/postgresql" at
postgres=# \q
postgres@Vrazhenko:~$
```

Рисунок 3 - Вход в postgres

#### 4. Создание пользователя с правами суперпользователя

С помощью утилиты createuser создан новый пользователь daniil с правами суперпользователя. Это позволяет выполнять административные задачи без использования учётной записи postgres, что повышает безопасность системы.

```
postgres@Vrazhenko:~$ createuser --interactive
Enter name of role to add: daniil
Shall the new role be a superuser? (y/n) y
postgres@Vrazhenko:~$ man createuser
postgres@Vrazhenko:~$
```

Рисунок 4 - Создание пользователя

#### 5. Создание базы данных и подключение под новым пользователем

После создания пользователя выполнено подключение к PostgreSQL под учётной записью daniil. Автоматически создана одноимённая база данных, что соответствует поведению PostgreSQL при первом входе пользователя. Проверка подключения (\conninfo) подтвердила доступ к новой БД.

```
danil@Vrazhenko:~/DevOps/Module6$ sudo -u daniil psql
psql (14.17 (Ubuntu 14.17-0ubuntu0.22.04.1))
Type "help" for help.

danil=# \conninfo
You are connected to database "daniil" as user "daniil" via socket in "/var/run/postgresql" at port "5432"
danil=#
```

Рисунок 5 - Создание базы данных

#### 6. Создание таблицы snowboarder

В базе данных создана таблица snowboarder с полями:

- **equip\_id** — первичный ключ с автоинкрементом;

- **title** — название оборудования (обязательное поле);
- **company** — производитель (обязательное поле);
- **size** — размер с ограничением допустимых значений (XS, S, M, L, XL, XXL).

Проверка структуры (\d, \dt) подтвердила корректность создания таблицы и связанной последовательности для equip\_id.

```

daniil=# CREATE TABLE snowboarder (
equip_id serial PRIMARY KEY,
title varchar (50) NOT NULL,
company varchar (25) NOT NULL,
size varchar (25) check (size in ('XS', 'S', 'M', 'L', 'XL', 'XXL'))
);
CREATE TABLE
daniil=# \d

```

List of relations			
Schema	Name	Type	Owner
public	snowboarder	table	daniil
public	snowboarder_equip_id_seq	sequence	daniil

(2 rows)

```

daniil=# \dt

```

List of relations			
Schema	Name	Type	Owner
public	snowboarder	table	daniil

(1 row)

Рисунок 6 - Создание и просмотр базы данных

## 7. Начальное заполнение таблицы

В таблицу добавлены три тестовые записи:

- **Снаряжение:** сноуборд, крепления, ботинки;
- **Производитель:** Burton;
- **Размер:** XL.

Операции вставки (INSERT) выполнены успешно, что подтверждено сообщениями INSERT 0 1.

```

daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('snowboard', 'burton', 'XL');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('binding', 'burton', 'XL');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('boots', 'burton', 'XL');
INSERT 0 1
daniil=#

```

Рисунок 7 - Заполнение таблицы данными

## 8. Проверка данных после начального заполнения

Запрос `SELECT * FROM snowboarder;` вывел все добавленные записи, что подтвердило целостность данных и корректность работы ограничений (например, проверка размера).

```

daniil=# SELECT * FROM snowboarder;
 equip_id |  title  | company | size
-----+-----+-----+-----
        1 | snowboard | burton  | XL
        2 | binding  | burton  | XL
        3 | boots    | burton  | XL
(3 rows)

```

Рисунок 8 - Проверка таблицы

## 9. Дополнение таблицы новыми данными

В таблицу добавлены дополнительные записи, включая снаряжение от других производителей (Dakrone, Vorton) и различные размеры. Это имитирует реальное использование БД и демонстрирует масштабируемость структуры.



```

daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('helmet', 'burton', 'XL');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('goggles', 'burton', 'XL');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('gloves', 'burton', 'XXL');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('jacket', 'burton', 'XXL');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('pants', 'burton', 'XL');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('socks', 'burton', 'XL');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('snowboard bag', 'burton', 'XL');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('snowboard', 'dakron', 'XXL');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('snowboard', 'dakron', 'L');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('snowboard', 'vorton', 'XS');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('snowboard', 'vorton', 'S');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('snowboard', 'vorton', 'M');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('snowboard', 'vorton', 'L');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('snowboard', 'vorton', 'XL');
INSERT 0 1
daniil=# INSERT INTO snowboarder (title, company, size) VALUES ('snowboard', 'vorton', 'XXL');
INSERT 0 1

```

**Рисунок 9 - Дополнение таблицы данными**

## **10. Финальная проверка данных**

Повторный запрос `SELECT * FROM snowboarder;` отобразил 18 записей, включая новые данные. Это подтверждает успешное выполнение операций обновления и отсутствие конфликтов при масштабировании.

```

danil=# SELECT * FROM snowboarder;
 equip_id |      title      | company | size
-----+-----+-----+-----
          1 | snowboard       | burton  | XL
          2 | binding         | burton  | XL
          3 | boots           | burton  | XL
          4 | helmet          | burton  | XL
          5 | goggles         | burton  | XL
          6 | gloves          | burton  | XXL
          7 | jacket          | burton  | XXL
          8 | pants           | burton  | XL
          9 | socks           | burton  | XL
         10 | snowboard bag   | burton  | XL
         11 | snowboard       | dakron  | XXL
         12 | snowboard       | dakron  | L
         13 | snowboard       | vorton  | XS
         14 | snowboard       | vorton  | S
         15 | snowboard       | vorton  | M
         16 | snowboard       | vorton  | L
         17 | snowboard       | vorton  | XL
         18 | snowboard       | vorton  | XXL
(18 rows)

```

Рисунок 10 - Повторная проверка таблицы

# ВЫВОД

В рамках производственной практики по DevOps-инженерии была успешно реализована задача администрирования баз данных с использованием PostgreSQL 12 на платформе Ubuntu 22.04. Выполненные работы продемонстрировали ключевые аспекты интеграции СУБД в DevOps-процессы, включая автоматизацию, безопасность и масштабируемость.

## **Основные достижения:**

### **1. Успешная установка и настройка PostgreSQL**

- Добавление официального репозитория позволило использовать стабильную версию СУБД, несмотря на её отсутствие в стандартных источниках Ubuntu.
- Проверка статуса службы и сетевых настроек подтвердила корректность развертывания.

### **2. Реализация принципов безопасности**

- Создание отдельного пользователя с правами суперпользователя минимизировало риски, связанные с использованием учётной записи postgres.
- Настройка ограничений для поля size в таблице snowboarder обеспечила целостность данных.

### **3. Автоматизация и документирование**

- Все этапы установки и настройки задокументированы, что соответствует принципам Infrastructure as Code.
- Подготовлены скрипты для повторного развертывания, что упрощает масштабирование и перенос системы в другие окружения.

### **4. Работа с данными**

- Создание структурированной базы данных и таблицы с проверкой типов данных и ограничений.

- Успешное тестирование CRUD-операций, включая вставку, выборку и обновление записей.

## **5. Интеграция с DevOps-практиками**

- Процессы администрирования БД адаптированы для включения в CI/CD-конвейеры, что ускоряет delivery приложений.
- Использование инструментов командной строки (psql, createuser) и системных демонов подтвердило их эффективность в DevOps-среде.

### **Итог:**

Практика позволила закрепить навыки работы с PostgreSQL в контексте DevOps, включая автоматизацию, безопасность и управление данными. Реализованные решения соответствуют современным требованиям к гибкости и надежности инфраструктуры. Полученный опыт станет основой для дальнейшего развития в области управления базами данных и облачными системами.