|  |
| --- |
| 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.

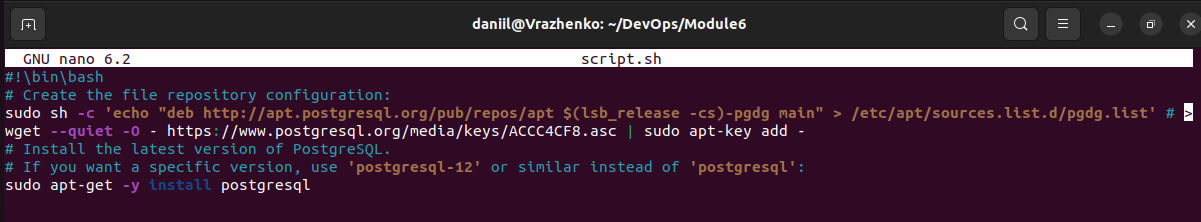
В рамках практики основное внимание было уделено автоматизации процессов администрирования: создание Ansible-ролей для развертывания кластеров, настройка потоковой репликации, интеграция с системами мониторинга, а также разработка скриптов для резервного копирования и восстановления. Эти задачи направлены 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

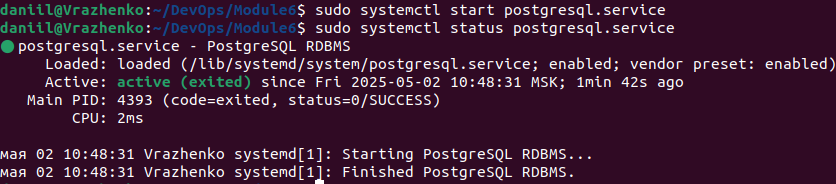
**1. Установка PostgreSQL 12 на Ubuntu 22.04**

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.

Рисунок 1 - Установка PostgreSQL

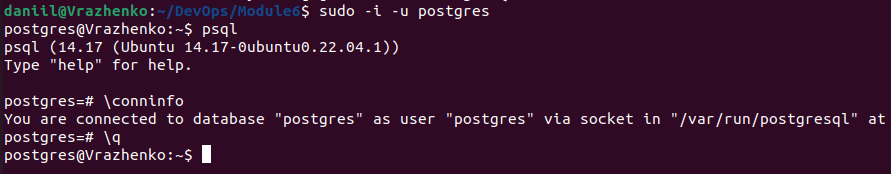
**2. Проверка статуса службы PostgreSQL**

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

Рисунок 2 - Проверка работоспособности

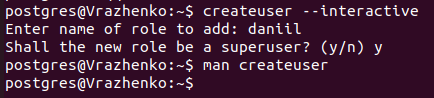
**3. Подключение к серверу через psql**

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.

Рисунок 3 - Вход в postgres

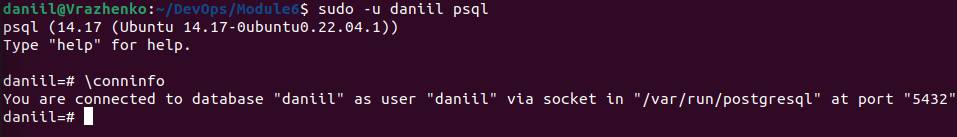
**4. Создание пользователя с правами суперпользователя**

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.

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.

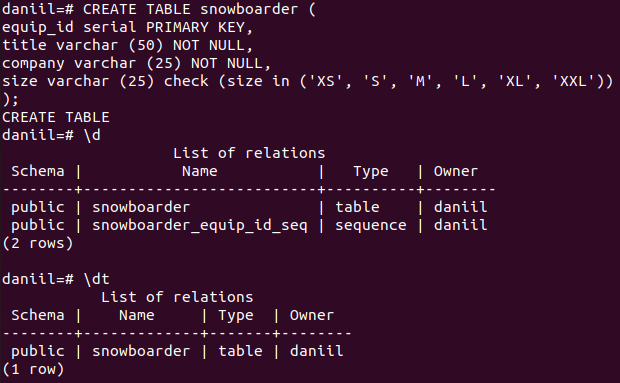
Рисунок 5 - Создание базы данных

**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.

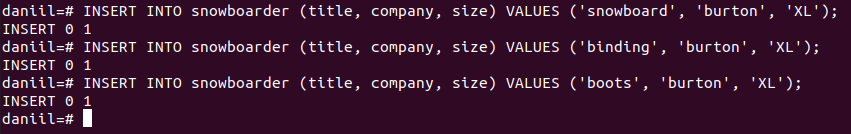
Рисунок 6 - Создание и просмотр базы данных

**7. Начальное заполнение таблицы**

В таблицу добавлены три тестовые записи:

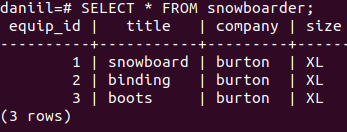
* **Снаряжение**: сноуборд, крепления, ботинки;
* **Производитель**: Burton;
* **Размер**: XL.

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

Рисунок 7 - Заполнение таблицы данными

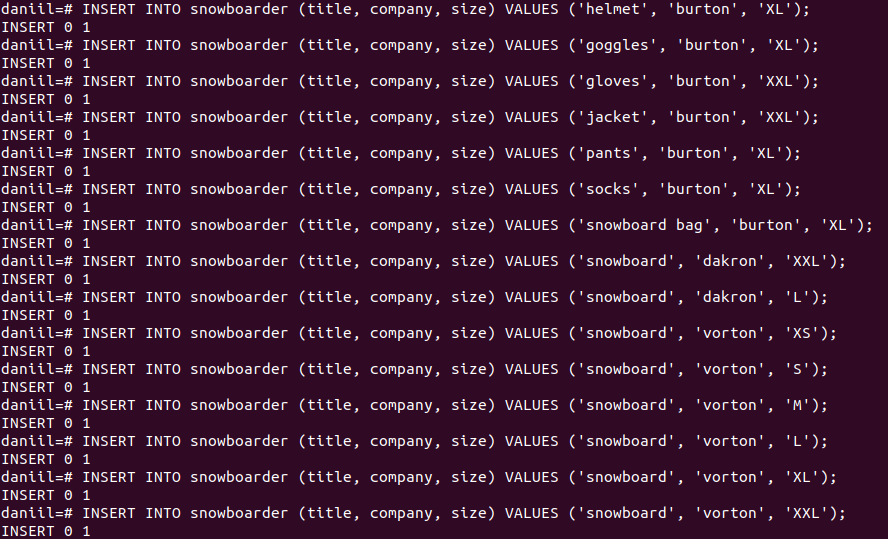
**8. Проверка данных после начального заполнения**

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

Рисунок 8 - Проверка таблицы

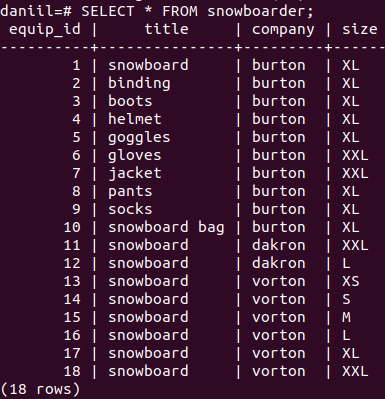
**9. Дополнение таблицы новыми данными**

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

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.

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.