

**Инструкция администратора по
подготовке PostgreSQL 17.6 на Debian
GNU/Linux, созданию роли artdba и
расширений, а также восстановлению
базы данных из дампа, входящего в
комплект ПО**

Содержание

Область применения	3
Требования к среде	3
Комплект для развёртывания	3
Обозначения	3
1. Установка PostgreSQL из комплекта	4
1.1. Проверка комплекта и архитектуры	4
1.2. Локальная установка	4
1.3. Запуск и проверка кластера	4
2. Базовая настройка и доступ	5
2.1. Пути и настройки	5
2.2. Локальная аутентификация	5
2.3. Если приложению нужен сетевой доступ	5
3. Настройка памяти по объёму RAM	6
3.1. Определить доступную серверу память	6
3.2. Параметры для выделенного сервера	6
3.3. Память запросов и фоновых операций	6
3.4. Фоновые задачи <code>pg_background</code>	7
4. Применение и проверка параметров памяти	8
4.1. Записать параметры в конфигурацию	8
4.2. Проверить и применить конфигурацию	8
4.3. Проверить запас памяти под рабочей нагрузкой	8
5. Подготовка файлов роли и базы	10
5.1. Создание <code>artdba</code> на целевом сервере	10
5.2. Создание новой базы	10
6. Подготовка к восстановлению структуры	11
6.1. Проверить доступность расширений	11
6.2. Проверить состав дампа	11
6.3. Порядок создания объектов	11
7. Восстановление дампа	12
7.1. Восстановить структуру данные и индексы	12
7.2. Проверить созданные расширения	12
7.3. Повторный запуск после ошибки	12
8. Проверка и запуск ПО	13
8.1. Обновить статистику и проверить объекты	13

8.2. Проверка доступа и функций	13
8.3. Критерии готовности	13
8.4. Действия при неуспешной проверке	13
9. Сопровождение и источники	15
9.1. Резервное копирование	15
9.2. Мониторинг и обновления	15
9.3. Типовые ошибки	15
10. Официальная документация	16
11. Стороннее ПО	1

Область применения

Инструкция предназначена для первичного развёртывания в новую пустую базу. Обновление действующей базы выполняется по регламенту обновления ПО. Формирование дампа, установка серверных компонентов расширений и перенос данных из действующей системы в документ не входят.

Требования к среде

Операционная система — Debian GNU/Linux, архитектура x86_64 (amd64). Выпуск ОС должен соответствовать системным требованиям ПО и локальному комплекту DEB-пакетов.

СУБД — PostgreSQL 17.6. Необходимые расширения и их серверные зависимости должны быть подготовлены до выполнения этой инструкции.

Администратору необходимы доступ sudo и локальные файлы комплекта. Объём памяти и свободного диска определяется системными требованиями ПО и размером развёрнутой базы с учётом индексов, WAL и временных файлов.

Комплект для развёртывания

Комплект содержит: packages — DEB-пакеты PostgreSQL 17.6 и зависимостей; 00_database.sql — создание базы appdb_restore с владельцем artdba и параметрами кодировки и locale; appdb.dump — дамп формата custom; SHA256SUMS — контрольные суммы всех файлов комплекта, включая packages. Состав и версии расширений, кодировка, locale provider и locale базы, а также контрольные показатели данных зафиксированы в сопроводительной информации к комплекту.

Используйте файлы одного выпуска ПО. Схемы и команды создания расширений входят в дампы и восстанавливаются вместе со структурой базы.

Обозначения

appdb_restore — имя целевой базы, /srv/pg-transfer — каталог файлов поставки, порт — 5432, кластер — 17/main. Команды Bash выполняются в терминале сервера, SQL и команды с обратной косой чертой — в psql.

Роль artdba создаётся с административными правами, предусмотренными конфигурацией ПО. Для подключений приложения используйте отдельную роль с согласованными правами. SUPERUSER обходит RLS даже при NOBYPASSRLS. [3]

1. Установка PostgreSQL из комплекта

Установка выполняется на подготовленный сервер Debian GNU/Linux amd64 из локальных DEB-пакетов PostgreSQL 17.6 и зависимостей. Пакеты находятся в каталоге packages распакованного комплекта. Подключение внешних репозитория и загрузка пакетов из сети не требуются.

1.1. Проверка комплекта и архитектуры

Откройте терминал в корневом каталоге распакованного комплекта. Проверьте контрольные суммы и архитектуру системы. Все файлы должны пройти проверку; команда dpkg должна вывести amd64.

```
sha256sum -c SHA256SUMS
dpkg --print-architecture
dpkg-deb -f packages/postgresql-17_*.deb \
```

Package Version Architecture

Версия серверного пакета должна соответствовать PostgreSQL 17.6; архитектура — amd64. В packages размещается по одному пакету каждой устанавливаемой версии и полный набор необходимых зависимостей для используемого выпуска Debian. При несовпадении контрольных сумм, версии или архитектуры установку не выполняйте.

1.2. Локальная установка

```
cd packages
sudo apt --no-download install ./*.deb
cd ..
```

Параметр --no-download запрещает загрузку пакетов. АРТ устанавливает локальные пакеты и разрешает зависимости в пределах подготовленного комплекта и уже установленных компонентов ОС. Ошибка зависимостей означает, что комплект не соответствует подготовленной среде; не запускайте загрузку недостающих пакетов из внешних репозитория.

1.3. Запуск и проверка кластера

```
pg_lsclusters
sudo systemctl enable postgresql
sudo systemctl start postgresql@17-main
sudo systemctl status postgresql@17-main --no-pager
sudo -u postgres /usr/lib/postgresql/17/bin/psql \
  -X -p 5432 -d postgres -c 'SELECT version();'
/usr/lib/postgresql/17/bin/pg_restore --version
```

Результат проверки: кластер 17/main работает на порту 5432, сервер и pg_restore имеют версию 17.6. Отсутствующий кластер создайте командой sudo pg_createcluster 17 main --start, затем повторите проверку. Существующий кластер повторно не создавайте. Эта инструкция применяется к первичному развёртыванию, а не к замене работающей СУБД.

2. Базовая настройка и доступ

2.1. Пути и настройки

```
sudo -u postgres psql -X -p 5432 -d postgres
SHOW config_file;
SHOW hba_file;
SHOW data_directory;
SHOW password_encryption;
SHOW shared_preload_libraries;
\q
```

Для стандартного 17/main конфигурация находится в /etc/postgresql/17/main. Сохраните копию файлов перед изменениями. В postgresql.conf задайте или проверьте параметры:

```
listen_addresses = 'localhost'
password_encryption = 'scram-sha-256'
fsync = on
full_page_writes = on
synchronous_commit = on
autovacuum = on
```

Для локального восстановления внешний доступ не нужен. Не отключайте надёжность записи ради ускорения импорта. Параметры памяти настройте по разделам 3 и 4 до восстановления базы.

2.2. Локальная аутентификация

Сохраните административный доступ через peer. В pg_hba.conf проверьте правила ниже и их порядок: применяется первое подходящее правило. Не заменяйте весь файл этим фрагментом. [7]

local	all	postgres		peer
host	appdb_restore	artdba	127.0.0.1/32	scram-sha-256
host	appdb_restore	artdba	:::1/128	scram-sha-256

2.3. Если приложению нужен сетевой доступ

Добавьте реальный приватный адрес сервера в listen_addresses. Настройте серверный TLS-сертификат и ключ, включите ssl = on и разрешите только адрес приложения в firewall и pg_hba.conf, например hostssl appdb_restore app_runtime APP_IP/32 scram-sha-256. Роль app_runtime нужно предварительно создать и выдать ей необходимые права.

Клиенту задайте sslmode=verify-full, доверенный корневой сертификат и DNS-имя, соответствующее сертификату. SCRAM обеспечивает аутентификацию, но не заменяет TLS для шифрования трафика. [7, 8]

```
sudo -u postgres psql -X -d postgres -c \
    "SELECT line_number,error FROM pg_hba_file_rules WHERE error IS NOT NULL;"
sudo systemctl restart postgresql@17-main
sudo -u postgres pg_isready -p 5432
sudo journalctl -u postgresql@17-main -n 50 --no-pager
```

Перезапуск нужен для listen_addresses и ряда серверных параметров; для изменения только НБА достаточно reload. Выполняйте restart в согласованное окно обслуживания.

3. Настройка памяти по объёму RAM

Профиль ниже предназначен для одного экземпляра PostgreSQL на выделенном сервере БД с RAM от 4 до 128 GiB. Приложение размещается отдельно. Значения задают исходную конфигурацию развёртывания; допустимая параллельная нагрузка проверяется перед вводом в эксплуатацию.

3.1. Определить доступную серверу память

```
free -h
awk '/MemTotal/ {printf "RAM: %.2f GiB\n", $2/1024/1024}' \
  /proc/meminfo
systemctl show postgresql@17-main -p MemoryMax
```

Для виртуальной машины используйте выделенную ей RAM, а не память гипервизора. При ограничении памяти службы расчёт ведётся по меньшему из значений RAM и MemoryMax; infinity означает отсутствие лимита службы. Swap в объём RAM не включается. Этот профиль не применяется к нескольким кластерам или к совместному размещению приложения и БД без отдельного бюджета памяти.

3.2. Параметры для выделенного сервера

shared_buffers задайте равным 25% RAM. effective_cache_size задайте равным 75% RAM. Для промежуточного объёма вычислите значения в MiB и округлите вниз. maintenance_work_mem вычислите как RAM / 32 с верхним пределом 2048MB. В таблице приведены результаты этих правил.

RAM GiB	shared_buffers	effective_cache_size	maintenance_work_mem
4	1GB	3GB	128MB
8	2GB	6GB	256MB
16	4GB	12GB	512MB
32	8GB	24GB	1024MB
64	16GB	48GB	2048MB
128	32GB	96GB	2048MB

Единицы GB и MB в postgresql.conf кратны 1024 и соответствуют GiB и MiB в расчёте. shared_buffers — реальная общая память кластера; 25% — рекомендуемая отправная точка для выделенного сервера. [6]

effective_cache_size — оценка доступного кэша для планировщика, включающая shared_buffers и файловый кэш ОС. Этот параметр не выделяет память и не прибавляется к shared_buffers в бюджете RAM. Значение 75% является правилом данного профиля. [12]

3.3. Память запросов и фоновых операций

Для всех строк таблицы установите work_mem = 4MB, hash_mem_multiplier = 2.0, temp_buffers = 8MB, autovacuum_work_mem = 128MB, autovacuum_max_workers = 3 и max_connections = 500. Лимит 500 включает резервные административные подключения; max_connections не означает, что 500 тяжёлых запросов могут выполняться одновременно без проверки памяти.

work_mem действует на операцию сортировки или хеширования и на участвующие процессы, а не на всю базу. autovacuum_work_mem выделяется каждому рабочему процессу autovacuum. maintenance_work_mem ограничивает отдельную операцию обслуживания. [6]

3.4. Фоновые задачи pg_background

Конфигурация предусматривает до 500 клиентских подключений и дополнительно до 30 одновременно выполняющихся задач pg_background в кластере. Задачи pg_background используют слоты background worker из max_worker_processes, а не обычные клиентские слоты max_connections. Их память и нагрузка на CPU учитываются дополнительно. [6, 13]

3.4.1. Количество фоновых процессов

```
max_connections = 500
max_worker_processes = 64
max_parallel_workers = 8
max_parallel_workers_per_gather = 2
```

Из общего лимита 64 планируется до 30 процессов pg_background и до 8 параллельных worker. Остаётся 26 слотов для других расширений и процессов, использующих этот общий пул, в том числе логической репликации. Это расчёт бюджета, а не техническое резервирование слотов: другие потребители не должны занимать ёмкость, необходимую для 30 задач.

autovacuum обслуживается отдельным лимитом autovacuum_max_workers и не вычитается из max_worker_processes. Его потребление памяти включается в общий бюджет RAM. max_parallel_workers_per_gather = 2 ограничивает параллельных worker одного узла Gather; общий предел параллельных worker кластера равен 8. [6]

3.4.2. Ограничение числа задач

Диспетчер фоновых заданий должен ограничивать суммарное число одновременно запущенных задач pg_background значением 30 для всех сеансов. Остальные задания остаются в очереди. max_worker_processes = 64 сам по себе не ограничивает pg_background числом 30 и не гарантирует наличие свободных слотов.

В версиях pg_background, предоставляющих параметр pg_background.max_workers, задайте его равным 30 в каждом запускающем сеансе до запуска задач. Этот параметр ограничивает один сеанс и не заменяет общий лимит диспетчера. Он отсутствует в старых версиях расширения; доступность определяется версией компонента в комплекте. [13]

3.4.3. Память фоновых запросов

Каждая задача выполняет SQL в отдельном процессе и может расходовать work_mem, память хеш-операций, временных таблиц и обслуживания. Используйте общесерверные настройки памяти из раздела 4. Не рассчитывайте на наследование параметров, изменённых командой SET в запускающем сеансе: необходимые сеансовые значения задаются внутри SQL фоновой задачи. [13]

3.4.4. Контроль при вводе в эксплуатацию

```
SELECT backend_type, state, count(*)
FROM pg_stat_activity
GROUP BY backend_type, state
ORDER BY backend_type, state;
```

На нагрузочной проверке запустите предусмотренные 30 задач одновременно с клиентской нагрузкой. Проверьте очередь заданий, число процессов, память, отсутствие ошибок регистрации background worker и завершение всех задач. Отслеживайте также потребителей общего пула других расширений.

4. Применение и проверка параметров памяти

4.1. Записать параметры в конфигурацию

Сохраните копию postgresql.conf, затем измените существующие параметры в файле. Не создавайте повторяющиеся записи одного параметра. Ниже приведён полный блок для сервера с 16 GiB RAM; для другого объёма замените три значения по таблице раздела 7.

```
sudo cp -a /etc/postgresql/17/main/postgresql.conf \
    /etc/postgresql/17/main/postgresql.conf.before-memory
sudoedit /etc/postgresql/17/main/postgresql.conf
shared_buffers = '4GB'
effective_cache_size = '12GB'
maintenance_work_mem = '512MB'
work_mem = '4MB'
hash_mem_multiplier = 2.0
temp_buffers = '8MB'
autovacuum_work_mem = '128MB'
autovacuum_max_workers = 3
max_connections = 500
max_worker_processes = 64
max_parallel_workers = 8
max_parallel_workers_per_gather = 2
```

4.2. Проверить и применить конфигурацию

```
sudo -u postgres psql -X -d postgres -c \
    "SELECT sourcefile,sourceline,error FROM pg_file_settings
    WHERE error IS NOT NULL;"
sudo systemctl restart postgresql@17-main
sudo -u postgres pg_isready -p 5432
sudo -u postgres psql -X -d postgres
SELECT name, setting, unit, sourcefile, pending_restart
FROM pg_settings
WHERE name IN ('shared_buffers','effective_cache_size','work_mem',
    'maintenance_work_mem','autovacuum_work_mem','temp_buffers',
    'hash_mem_multiplier','autovacuum_max_workers','max_connections',
    'max_worker_processes','max_parallel_workers',
    'max_parallel_workers_per_gather')
ORDER BY name;
\q
```

До restart запрос к pg_file_settings должен вернуть 0 строк. Изменение shared_buffers, max_connections и max_worker_processes требует перезапуска. После restart проверьте значения с учётом unit: для shared_buffers единицей может быть 8kB. pending_restart должен быть false. sourcefile показывает файл, из которого взято действующее значение: настройки в postgresql.auto.conf могут переопределять postgresql.conf.

4.3. Проверить запас памяти под рабочей нагрузкой

Бюджет должен покрывать shared_buffers, память активных операций запросов, обслуживание, autovacuum, процессы подключений, расширения и ОС. effective_cache_size в эту сумму не включается. Для оценки хеш-операций используйте: активные запросы × одновременные операции × процессы на

запрос $\times$ work_mem $\times$ hash_mem_multiplier. Это оценка компонента нагрузки, а не жёсткий предел всего PostgreSQL.

Расчётный пример пика: 500 активных клиентских запросов и 30 задач pg_background, по 3 одновременно потребляющие память хеш-операции на процесс, work_mem = 4MB и hash_mem_multiplier = 2.0. Получается $(500 + 30) \times 3 \times 4\text{MB} \times 2 = 12720\text{MB}$, то есть около 12,4 GiB только на эти операции. До 8 параллельных worker при тех же допущениях добавляют ещё 192MB. Это пример оценки, а не гарантированный верхний предел.

К полученной величине добавьте shared_buffers, память самих backend-процессов и расширений, до 384MB для трёх autovacuum-worker, параллельные операции обслуживания и резерв ОС. Таблица по RAM определяет размеры параметров, но не подтверждает достаточность 4–16 GiB для 500 одновременно активных запросов и 30 фоновых задач. Для заданной нагрузки достаточность RAM подтверждается нагрузочным испытанием.

На нагрузочной проверке контролируйте MemAvailable через free -h, обмен со swap через vmstat 1 и сообщения OOM в журнале ядра. Рост временных файлов анализируйте вместе с планами запросов. Не увеличивайте work_mem пропорционально RAM автоматически: сначала оцените число одновременно активных операций и запас памяти.

5. Подготовка файлов роли и базы

На целевом сервере подготовьте закрытый каталог и разместите в нём файлы поставки через согласованный защищённый канал. Откройте терминал в корневом каталоге комплекта, проверенного в разделе 1.

```
sudo install -d -o postgres -g postgres -m 700 /srv/pg-transfer
sudo install -o postgres -g postgres -m 600 \
  appdb.dump 00_database.sql \
  /srv/pg-transfer/
```

Файлы в рабочем каталоге доступны пользователю postgres. Восстановление выполняет SQL из дампа с административными правами; используйте только проверенный комплект.

5.1. Создание artdba на целевом сервере

Роль с LOGIN одновременно является пользователем PostgreSQL: отдельная команда CREATE USER не нужна. Блок рассчитан на отсутствие artdba. Если роль уже есть, проверьте её атрибуты и используйте согласованный ALTER ROLE; не удаляйте роль с зависимыми объектами. [3]

```
sudo -u postgres psql -X -v ON_ERROR_STOP=1 -d postgres
-- DROP ROLE artdba;
CREATE ROLE artdba WITH
  SUPERUSER CREATEDB CREATEROLE INHERIT LOGIN REPLICATION
  NOBYPASSRLS CONNECTION LIMIT -1;
ALTER ROLE artdba SET search_path TO public, oracle, fee_gen;
SET password_encryption = 'scram-sha-256';
\password artdba
\du artdba
\q
```

Пароль задайте через интерактивную команду \password и сохраните в хранилище секретов. Строку DROP ROLE оставьте комментарием. Эта роль предназначена для администрирования базы; для работы приложения используйте отдельную роль с ограниченными правами.

5.2. Создание новой базы

Создайте базу файлом 00_database.sql из комплекта. В файле заданы имя appdb_restore, владелец artdba, шаблон template0, кодировка и locale базы. Выполнение SQL-файла исключает ручной выбор параметров, не соответствующих дампу.

```
sudo -u postgres psql -X -v ON_ERROR_STOP=1 \
  -d postgres -f /srv/pg-transfer/00_database.sql
sudo -u postgres psql -X -v ON_ERROR_STOP=1 -d postgres \
  -c 'REVOKE CONNECT ON DATABASE appdb_restore FROM PUBLIC;'
```

Базу создавайте вне транзакции. Не используйте существующую рабочую базу в качестве цели для этих команд.

6. Подготовка к восстановлению структуры

6.1. Проверить доступность расширений

```
sudo -u postgres psql -X -d appdb_restore
SELECT name, default_version, installed_version
FROM pg_available_extensions
WHERE name IN ('plpgsql','dblink','hstore','orafce','pg_aq',
               'pg_background','pg_variables','postgres_fdw','tablefunc',
               'uuid-osspl','pg_trgm','pg_similarity','fuzzystrmatch')
ORDER BY name;
\q
```

Сверьте наличие расширений и их версии по умолчанию с перечнем компонентов комплекта. Дамп обычно содержит CREATE EXTENSION без указания VERSION, поэтому при восстановлении используется версия по умолчанию из установленного компонента. Для расширений с явно заданной VERSION соответствующая версия должна быть доступна на сервере. При несовпадении состава или версий восстановление не выполняйте.

6.2. Проверить состав дампа

```
sudo -u postgres bash
cd /srv/pg-transfer
/usr/lib/postgresql/17/bin/pg_restore --list appdb.dump > toc.list
/usr/lib/postgresql/17/bin/pg_restore \
  --section=pre-data --no-owner --no-acl --no-tablespaces \
  --file=pre-data-review.sql appdb.dump
exit
```

Проверьте toc.list и pre-data-review.sql: дамп должен содержать пользовательские схемы, таблицы, функции и команды создания требуемых расширений. Файл pre-data-review.sql служит для проверки и отдельно не выполняется. Не исключайте SCHEMA и EXTENSION из списка восстановления.

6.3. Порядок создания объектов

Раздел pre-data восстанавливает структуру до загрузки данных: схемы, расширения и необходимые определения объектов. pg_restore использует зависимости из архива, чтобы создать объекты в требуемом порядке. Схемы, входящие в состав самих расширений, создаются их установочными SQL-командами. [5, 9]

Пользовательские схемы не создаются отдельным SQL-файлом. Расширения также создаются командами из дампа. Установка серверных компонентов расширений в данной инструкции не рассматривается; их доступность проверяется до восстановления.

plpgsql уже присутствует в базе, созданной из template0. Наличие расширения не заменяет проверку его версии и схемы после восстановления. Не выполняйте ALTER EXTENSION UPDATE во время развёртывания.

7. Восстановление дампа

7.1. Восстановить структуру данные и индексы

Выполните блок полностью. Три этапа используют один дамп: сначала pre-data создаёт структуру и расширения, затем data загружает данные, затем post-data создаёт индексы, триггеры и ограничения, относящиеся к этому разделу. При ошибке следующий этап не запускается.

```
sudo -u postgres bash <<'RESTORE'
set -e
cd /srv/pg-transfer
for section in pre-data data post-data; do
    echo "Restoring: $section"
    if /usr/lib/postgresql/17/bin/pg_restore \
        -p 5432 --dbname=appdb_restore --role=artdba \
        --no-owner --no-acl --no-tablespaces \
        --section="$section" --exit-on-error --verbose \
        appdb.dump > "restore-$section.log" 2>&1; then
        tail -n 20 "restore-$section.log"
    else
        rc=$?
        tail -n 60 "restore-$section.log"
        echo "Restore failed: $section, exit code: $rc" >&2
        exit "$rc"
    fi
done
echo 'Restore completed successfully'
RESTORE
```

Успех — завершение всех трёх этапов с кодом 0 и сообщение Restore completed successfully. Сохраните restore-pre-data.log, restore-data.log и restore-post-data.log. Данные конфигурационных таблиц расширений, включённые в дампы, загружаются вместе с разделом data.

--role=artdba задаёт роль для создания объектов. --no-owner и --no-acl отключают перенос прежних владельцев и прав; --no-tablespaces использует целевое размещение. Владелец восстанавливаемых пользовательских объектов становится artdba. Доступ приложения настройте после восстановления. [5]

7.2. Проверить созданные расширения

```
sudo -u postgres psql -X -d appdb_restore
SELECT e.extname, e.extversion, n.nspname AS schema
FROM pg_extension e
JOIN pg_namespace n ON n.oid=e.extnamespace
ORDER BY e.extname;
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
\q
```

Имена, версии и схемы расширений должны совпадать с перечнем компонентов комплекта.

7.3. Повторный запуск после ошибки

--exit-on-error останавливает текущий этап, но не откатывает уже выполненные команды. Устраните причину ошибки и повторите все этапы в заново созданной пустой базе. Не запускайте восстановление поверх частично заполненной базы и не используйте --clean для рабочей базы.

8. Проверка и запуск ПО

8.1. Обновить статистику и проверить объекты

```
sudo -u postgres /usr/lib/postgresql/17/bin/vacuumdb \  
-p 5432 -d appdb_restore --analyze-in-stages  
sudo -u postgres psql -X -d appdb_restore  
SELECT pg_size_pretty(pg_database_size(current_database()));  
SELECT schemaname, count(*) AS tables  
FROM pg_tables  
WHERE schemaname NOT IN ('pg_catalog','information_schema')  
GROUP BY schemaname ORDER BY schemaname;  
SELECT indexrelid::regclass FROM pg_index WHERE NOT indisvalid;  
SELECT conrelid::regclass, conname  
FROM pg_constraint WHERE NOT convalidated;  
SELECT nspname, pg_get_userbyid(nspowner) AS owner  
FROM pg_namespace  
WHERE nspname IN ('public','oracle','fee_gen');  
\q  
psql -X -h 127.0.0.1 -p 5432 -U artdba -d appdb_restore  
SHOW search_path;  
SELECT current_schemas(false);  
\q
```

Проверьте `search_path` новым подключением `artdba`: `ALTER ROLE` действует на новые сеансы. `current_schemas(false)` показывает реально доступные схемы; отсутствие `oracle` или `fee_gen` нельзя скрывать только наличием имени в `search_path`.

Сопоставьте версии и схемы расширений, число таблиц и контрольные показатели данных с контрольными показателями комплекта. Проверьте точные `COUNT(*)` и бизнес-агрегаты для таблиц, перечисленных в контрольных показателях. Невалидные индексы и неожиданные невалидированные ограничения требуют выяснения причины до запуска ПО.

8.2. Проверка доступа и функций

Создайте роль приложения и выдайте ей `CONNECT`, `USAGE`, необходимые права на таблицы, последовательности и функции согласно регламенту настройки доступа ПО. Проверьте `default privileges` для будущих объектов. Роль `artdba` предназначена для административных операций.

Выполните проверки чтения, записи, генерации идентификаторов и функций расширений. Для `postgres_fdw` и `dblink` проверьте адреса внешних систем, `user mappings`, секреты и сетевой доступ. Не запускайте фоновые задания до проверки их настроек.

8.3. Критерии готовности

База готова к запуску ПО, если `pg_restore` завершился с кодом 0, расширения соответствуют поставке, контрольные показатели данных совпали, права доступа проверены и функциональные проверки завершились успешно. Сохраните журнал развёртывания и фактические версии компонентов.

Укажите целевую базу в настройках ПО, запустите приложение и проконтролируйте журнал ошибок. До допуска пользователей создайте резервную копию развёрнутой базы.

8.4. Действия при неуспешной проверке

Не допускайте пользователей к базе. Сохраните журналы и зарегистрируйте обращение в техническую поддержку с описанием ошибки и версиями компонентов. Повторное развёртывание выполняйте в

новую пустую базу после устранения причины. Удаление ошибочно созданной базы допустимо только после проверки, что она не содержит рабочих данных.

9. Сопровождение и источники

9.1. Резервное копирование

Настройте регулярное резервное копирование, срок хранения и пробное восстановление. Копии храните отдельно от сервера СУБД с ограничением доступа. Для восстановления на момент времени необходимы базовая физическая копия и архив WAL; порядок настройки определяется эксплуатационным регламентом. [11]

9.2. Мониторинг и обновления

Контролируйте свободное место, рост WAL, ошибки PostgreSQL, подключения, длительные транзакции, блокировки и autovacuum. Устанавливайте обновления PostgreSQL и расширений из комплектов обновления ПО по регламенту сопровождения. Перед обновлением создавайте резервную копию базы. [1]

9.3. Типовые ошибки

role already exists — проверить существующую роль, не удалять её автоматически. extension is not available или version not available — среда не соответствует поставке. already exists — проверить, что восстановление запущено в новую пустую базу и ни один этап не выполняется повторно. unsupported version in file header — проверить версию pg_restore и формат дампа. Ошибки владельцев, ACL или зависимостей — сохранить журнал ошибки и проверить соответствие файлов выпуска ПО.

10. Официальная документация

1. Политика версий PostgreSQL — <https://www.postgresql.org/support/versioning/>
3. CREATE ROLE — <https://www.postgresql.org/docs/17/sql-createrole.html>
5. pg_restore — <https://www.postgresql.org/docs/17/app-pgrestore.html>
6. Настройки памяти — <https://www.postgresql.org/docs/17/runtime-config-resource.html>
7. pg_hba.conf — <https://www.postgresql.org/docs/17/auth-pg-hba-conf.html>
8. TLS — <https://www.postgresql.org/docs/17/ssl-tcp.html>
9. CREATE EXTENSION — <https://www.postgresql.org/docs/17/sql-createextension.html>
10. Схемы и search_path — <https://www.postgresql.org/docs/17/ddl-schemas.html>
11. Архивирование WAL и PITR — <https://www.postgresql.org/docs/17/continuous-archiving.html>
12. Оценка кэша планировщиком — <https://www.postgresql.org/docs/17/runtime-config-query.html>
13. pg_background — https://github.com/vibhorkum/pg_background/blob/master/README.md

11. Стороннее ПО

Сторонний компонент/сервис	Название лицензии	Ссылка на лицензию	Ссылка на репозиторий
PostgreSQL 17.6	PostgreSQL License	https://www.postgresql.org/about/licence/ https://github.com/postgres/postgres/blob/master/COPYRIGHT	https://github.com/postgres/postgres