Базы данных

Лекция 11 – Администрирование сервера

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2024

2024.12.16

Оглавление

Установка сервера

- установка из двоичных пакетов;
- установка из исходного кода.

Установка из двоичных пакетов

PostgreSQL доступен в виде двоичных пакетов для самых распространённых операционных систем. Устанавливать PostgreSQL рекомендуется именно из них. Сборка из исходного кода рекомендуется только для разработчиков PostgreSQL или расширений для него.

Дистрибутивы ОС Linux и других юникс-подобных ОС содержат все необходимые пакеты для установки. Вместе с дистрибутивом PostgreSQL как правило идет инструкция по установке и первичной настройке.

На декабрь 2024 года текущая версия 17.2. В дистрибутивах FC 16.3 (16.6).

```
$ head /usr/share/pgsql/sql_features.txt #  что умеет и что не умеет
B011    Embedded Ada            NO
B012    Embedded C              YES
B013    Embedded COBOL          NO
B014    Embedded Fortran        NO
B015    Embedded MUMPS          NO
B016    Embedded Pascal         NO
B017    Embedded PL/I           NO
B021    Direct SQL              YES
B030    Enhanced dynamic SQL    NO
B031    Basic dynamic SQL       NO
```

Установка в приложении к конкретному дистрибутиву обычно описана в сопровождающей документации. Для дистрибутивов на основе rpm это может быть файл

`/usr/share/doc/postgresql/README.rpm-dist`

Быстрый старт

При установке создается системный пользователь `postgres` и каталог `/usr/lib/pgdql`, где сервер будет держать свои файлы, в том числе и файлы баз данных.

```
$ ls -ld pgsql
drwx------. 4 postgres postgres 4096 дек 01 00:00 pgsql
```

Первый шаг – инициализация кластера от root

```
$ sudo postgresql-setup --initdb
 * Initializing database in '/var/lib/pgsql/data'
 * Initialized, logs are in /var/lib/pgsql/initdb_postgresql.log
```

В каталоге среди прочих будет создан файл initdb_postgresql.log, в котором описан следующий шаг, пригодный для любых установок, однако для rpm-based дистрибутивов есть более другой способ с использованием `systemd`:

```
$ sudo systemctl start postgresql.service
$ sudo systemctl enable postgresql.service
Created symlink /etc/systemd/system/multi-user.target.wants/postgresql.service \
→ /usr/lib/systemd/system/postgresql.service.
```

Сервер запущен и будет автоматически запускаться при презагрузке системы.

В процессе инициализации также создается файл /var/lib/pgsql/.bash_profile

```
$ sudo cat /var/lib/pgsql/.bash_profile
[ -f /etc/profile ] && source /etc/profile

PGDATA=/var/lib/pgsql/data
export PGDATA
```

Установка из исходного кода

Пререквизиты для сборки кратко описаны в документации, детальное описание доступно в исходных кодах.

Где берем:

`https://www.postgresql.org/ftp/`

`rsync://ftp.postgresql.org/pgsql-ftp/`

```
$ rsync rsync://ftp.postgresql.org/pgsql-ftp/
drwxr-xr-x          4.096 2024/11/21 17:04:26 .
-rw-r--r--          1.730 2024/01/01 03:00:01 README
lrwxrwxrwx             12 2024/11/21 17:04:26 latest
-rw-r--r--             24 2024/12/15 12:21:07 sync_timestamp
drwxr-xr-x          4.096 2017/12/28 18:56:45 dev
drwxrwsr-x          4.096 2024/05/07 16:41:50 odbc
drwxr-xr-x          4.096 2021/07/12 18:01:08 old
drwxr-xr-x          4.096 2017/04/13 13:58:54 pgadmin
drwxrwsr-x          4.096 2024/09/29 05:55:37 postgis
drwxr-xr-x          4.096 2019/07/11 19:41:42 projects
drwxr-xr-x          4.096 2017/10/25 11:28:37 repos
drwxr-xr-x          4.096 2024/07/05 15:30:34 snapshot
drwxr-xr-x         16.384 2024/11/21 17:04:09 source
```

`postgresql-17.2.tar.gz` – документация в исходниках (sgml)

`postgresql-17.2-docs.tar.gz` – документация отдельно (html)

`https://www.postgresql.org/docs/17` – документация онлайн (en)

`https://postgrespro.ru/docs/postgresql/17` – документация онлайн (en, ru)

Процесс сборки стандартный

```
$ tar xf postgresql-17.2.tar.gz
$ cd postgresql-17.2
$ mkdir build
$ cd build
$ ../configure --help > configure.help
```

Изучаем содержимое `configure.help` и соответсвующую главу документации, после чего запускаем конфигурационный скрипт с требуемыми нам параметрами, если стандартная конфигурация нас не удовлетворяет. Если удовлетворяет, просто говорим

```
$ ../configure
```

В стандартной конфигурации собираются сервер и утилиты, а также клиентские приложения и интерфейсы, которым требуется только компилятор C. Все файлы по умолчанию устанавливаются в /usr/local/pgsql.

Далее

```
$ make       # будет собрано без документации и man'ов
$ make world # будет собрано все, что можно собрать
$ make check # регрессионные тесты
```

И установка

```
$ sudo make install       # для main
$ sudo make install-world # для main world
```

Можно установить только клиентскую часть без сервера (см. Документацию).

Параметры скрипта configure

Стандартные параметры (--prefix, каталоги установки компонет) как обычно.

Функциональные

`--enable-nls[=LANGS]` – поддержка национальных языков (NLS, Native Language Support). Значение LANGS представляет необязательный список кодов языков через пробел, поддержка которых требуется, например: `--enable-nls='ru'`. Если список не задаётся, устанавливаются все доступные переводы.

`--with-{perl|python|tcl}` – подключает поддержку языков PL/Perl, PL/Python, PL/Tcl на стороне сервера.

`--with-systemd` – подключает поддержку служебных уведомлений для systemd. Это улучшает интеграцию с системой, если сервер запускается под управлением systemd, и не оказывает никакого влияния в противном случае.

`--with-selinux` – сборка с поддержкой Selinux.

`--with-ssl=openssl` – подключает поддержку соединений SSL (зашифрованных).

`--with-pam` – подключает поддержку PAM (Pluggable Authentication Modules, подключаемых модулей аутентификации).

`--with-pgport=НОМЕР` – задаёт НОМЕР порта по умолчанию для сервера и клиентов. Стандартное значение — 5432. Этот порт всегда можно изменить позже, но если указать здесь другой номер, и сервер, и клиенты будут скомпилированы с одним значением. Менять это значение имеет смысл, только если необходимо запускать в одной системе несколько серверов PostgreSQL.

{--with-blocksize|--with-wal-blocksize}=РАЗМЕР_БЛОКА – задаёт размер блока в килобайтах. Эта величина будет единицей хранения и ввода/вывода данных таблиц и журнала WAL. Значение по умолчанию, 8 килобайт, достаточно универсально, но в особых случаях может быть оправдан другой размер блока. Это значение должно быть степенью 2 от 1 до 32 (в килобайтах).

ВАЖНО!!! С изменением этого значения нарушается совместимость формата БД на диске, и для перехода к сборке с другим размером не получится использовать pg_upgrade .

Действия после установки

В некоторых системах необходимо указать системе, как найти недавно установленные совместно используемые библиотеки. Это не требуется в FreeBSD, Linux, NetBSD, OpenBSD и Solaris. Тем не менее, в linux зарегистирировать их можно так:

```
$ sudo /sbin/ldconfig -m /usr/local/pgsql/lib
```

Путь поиска разделяемых библиотек на разных платформах может устанавливаться по-разному, но наиболее распространённый способ — установить переменную окружения LD_LIBRARY_PATH:

```
LD_LIBRARY_PATH=/usr/local/pgsql/lib
export LD_LIBRARY_PATH
```

или в csh, tcsh:

```
setenv LD_LIBRARY_PATH /usr/local/pgsql/lib
```

Переменные окружения

Если целевым каталогом был выбран `/usr/local/pgsql` или какой-то другой, по умолчанию отсутствующий в пути поиска, следует добавить `/usr/local/pgsql/bin` (или другой путь, переданный в указании `--bindir`) в переменную PATH. Это не обязательно, но использовать postgres в этом случае будет удобнее.

Для этого следует добавить в скрипт запуска оболочки учетной записи, например `~/.bash_profile` или в `/etc/profile` если это нужно всем пользователям:

```
PATH=/usr/local/pgsql/bin:$PATH
export PATH
```

Для оболочек csh или tcsh команда должна быть такой:

```
set path = ( /usr/local/pgsql/bin $path )
```

Чтобы система могла найти документацию man, нужно добавить в скрипт запуска оболочки примерно следующие строки, если только она не установлена в размещение, просматриваемое по умолчанию:

```
MANPATH=/usr/local/pgsql/share/man:$MANPATH
export MANPATH
```

Адрес компьютера и порт сервера базы данных для клиентских приложений, переопределяя стандартные значения, задают переменные окружения PGHOST и PGPORT.

Если планируется запускать клиентские приложения удалённо, пользователям, которые будут использовать определённый сервер, будет удобно, если они установят PGHOST.

Создание кластера баз данных

Кластер баз данных[1] – область хранения баз данных на диске.

Кластер баз данных представляет собой набор баз, управляемых одним экземпляром работающего сервера.

После инициализации кластер будет содержать базу данных с именем `postgres`, предназначенную для использования по умолчанию утилитами, пользователями и сторонними приложениями. Сам сервер баз данных не требует наличия базы `postgres`, но многие внешние вспомогательные программы рассчитывают на её существование.

При инициализации в каждом кластере создаются ещё две базы: `template1` и `template0`.

Как можно понять из их имён, они применяются впоследствии в качестве шаблонов создаваемых баз данных.

С точки зрения файловой системы кластер баз данных представляет собой один каталог, в котором будут храниться все данные. Какого-либо стандартного пути не существует, но часто данные размещаются в `/usr/local/pgsql/data` или в `/var/lib/pgsql/data`.

Прежде чем с каталогом данных можно будет работать, его нужно инициализировать, используя программу `initdb`, которая устанавливается в составе сервера.

```
$ sudo su - postgres
$ initdb -D /usr/local/pgsql/data
```

Можно перед запуском initdb установить переменную окружения `PGDATA` и не указывать `-D`.

---

1) В SQL применяется термин «кластер каталога»

Если для запуска и остановки сервера используется pg_ctl (18.3), годится вариант:

```
$ pg_ctl -D /usr/local/pgsql/data initdb
```

Создание каталога, с котором будет работать учетка postgres

```
$ sudo mkdir /usr/local/pgsql
$ sudo chown postgres: /usr/local/pgsql
$ sudo chmod 700 /usr/local/pgsql
$ sudo su - postgres
postgres$ initdb -D /usr/local/pgsql/data
Файлы, относящиеся к этой СУБД, будут принадлежать пользователю "postgres".
От его имени также будет запускаться процесс сервера.

Кластер баз данных будет инициализирован с локалью "ru_RU.UTF-8".
Кодировка БД по умолчанию, выбранная в соответствии с настройками: "UTF8".
...
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 /usr/local/pgsql/data -l файл_журнала start
```

initdb не будет работать, если указанный каталог данных уже существует и содержит файлы — эта мера предохранения от случайной перезаписи существующей инсталляции.

Локали

Программа initdb устанавливает для кластера баз данных локаль по умолчанию.

Если ничего ей не указывать, она берет параметры локали из текущего окружения и применяет их к инициализируемой базе данных. Однако можно выбрать и другую локаль для базы данных. Также можно указать локаль для отдельных категорий.

Для этого при сборке необходимо указывать `configure --enable-nls`.

LC_COLLATE	Порядок сортировки строк
LC_CTYPE	Классификация символов
LC_MESSAGES	Язык сообщений
LC_MONETARY	Форматирование валютных сумм
LC_NUMERIC	Форматирование чисел
LC_TIME	Форматирование даты и времени

Например,

```
initdb –locale=ru_RU.UTF-8 --lc_messages=C --lc_collate=C --lc_cype=C
```

Для локалей, отличных от C и POSIX, порядок сортировки символов зависит от системной библиотеки локализации, а он, в свою очередь, влияет на порядок ключей в индексах.

Поэтому кластер нельзя перевести на несовместимую версию библиотеки ни путём восстановления снимка, ни через двоичную репликацию, ни перейдя на другую операционную систему или обновив её версию. Использование локалей, отличающихся от C или POSIX, влияет на производительность. В частности, замедляет обработку символов и мешает LIKE использовать обычные индексы. По этой причине следует использовать локали только в том случае, если они действительно необходимы.

Файловые системы

PostgreSQL может использовать любую файловую систему с семантикой POSIX.

NFS

Каталог данных PostgreSQL может размещаться и в файловой системе NFS. PostgreSQL не подстраивается специально под NFS, что означает, что с NFS он работает точно так же, как и с локально подключёнными устройствами. При этом функциональность файловых систем, которая имеет свои особенности в NFS, например блокировки файлов, не используется.

При использовании NFS с PostgreSQL следует монтировать файловую систему в режиме hard. В режиме soft в случаях перебоев в сети системные вызовы будут прерываться, а поскольку PostgreSQL не повторяет вызовы, прерванные таким образом, это будет проявляться как ошибки ввода/вывода. В режиме hard в случае сетевых проблем процессы могут на неопределённое время «зависать».

iSCSI

Можно использовать протокол iSCSI[2]. Это транспортный протокол, обеспечивающий передачу протокола хранения данных SCSI по TCP/IP через сетевое соединение, которым обычно является Ethernet. iSCSI работает как метод организации распределенных хранилищ.

---

2) https://selectel.ru/blog/iscsi-protocol/, https://ru.wikipedia.org/wiki/ISCSI

Запуск сервера баз данных

Программа сервера называется postgres.

Самый прямолинейный вариант запуска сервера вручную — просто выполнить непосредственно postgres, указав расположение каталога данных в ключе -D, например:

```
$ postgres -D /usr/local/pgsql/data
```

В результате сервер продолжит работу на переднем плане.

Запускать сервер следует под именем учётной записи postgres.

Без параметра -D сервер попытается использовать каталог данных, указанный в переменной окружения PGDATA. Если и эта переменная не определена, сервер не запустится.

Запуск postgres в фоновом режиме

```
$ postgres -D /usr/local/pgsql/data > logfile 2>&1 & # классический вариант UNIX
$ postgres -D /usr/local/pgsql/data &> logfile &     # более короткий (linux)
```

Использование утилиты pg_ctl

```
pg_ctl start -l logfile
```

запустит сервер в фоновом режиме и направит выводимые сообщения сервера в указанный файл журнала. Параметр -D для неё имеет то же значение, что и для программы postgres. С помощью pg_ctl также можно остановить сервер, посмотрть его состояние.

```
pg_ctl {initdb|start|stop|restart|reload|status|promote|logrotate|kill}
```

Проблемы с подключениями клиентов

Ряд проблем может быть связан непосредственно с тем, как был запущен сервер.

```
psql: ошибка: не удалось подключиться к серверу "server.joe.com"
(123.123.123.123) по порту 5432: Connection refused
      Он действительно работает по адресу "server.joe.com" (123.123.123.123) и
принимает TCP-соединения (порт 5432)?
```

Это общая проблема «я не могу найти сервер и начать взаимодействие с ним».

Указанное выше сообщение говорит о попытке установить подключение по TCP/IP.

Очень часто объясняется это тем, что сервер просто забыли настроить для работы по протоколу TCP/IP.

При попытке установить подключение к локальному серверу через Unix-сокет можно получить такое сообщение:

```
psql: ошибка: не удалось подключиться к серверу через сокет "/tmp/.s.PGSQL.5432":
No such file or directory
      Он действительно работает локально и принимает соединения через этот сокет?
```

Если сервер действительно работает, следует проверить, что указываемый клиентом путь сокета (здесь /tmp) соответствует значению `unix_socket_directories`.

Сообщение об ошибке подключения всегда содержит адрес сервера или путь к сокету, и это помогает понять, куда именно подключается клиент.

Если сервер на самом деле не принимает подключения по этому адресу, обычно выдаётся сообщение ядра Connection refused (В соединении отказано) или No such file or directory (Нет такого файла или каталога), приведённое выше.

Управление ресурсами ядра (18.4)

Postgres иногда может исчерпывать некоторые ресурсы операционной системы до предела, особенно при запуске нескольких копий сервера в одной системе или при работе с очень большими базами.

Совместно используемая память и семафоры

Postgres требует, чтобы операционная система предоставляла средства межпроцессного взаимодействия (IPC), в частности, совместно используемую память и семафоры. Системы семейства Unix обычно предоставляют функции IPC в стиле «System V» или функции IPC в стиле «POSIX» или и те, и другие. В Windows эти механизмы реализованы по-другому.

По умолчанию PostgreSQL запрашивает очень небольшой объём совместно используемой памяти System V и намного больший объём анонимной совместно используемой памяти mmap.

Возможен также вариант использования одной большой области памяти System V (см. shared_memory_type). Помимо этого при запуске сервера создаётся значительное количество семафоров (в стиле System V или POSIX). В настоящее время семафоры POSIX используются в системах Linux и FreeBSD, а на других платформах используются семафоры System V.

Функции IPC в стиле System V обычно сталкиваются с лимитами на уровне системы. Когда PostgreSQL превышает один из этих лимитов, сервер отказывается запускаться, но должен выдать полезное сообщение, говорящее об ошибке и о том, что с ней делать.

Linux использует преимущественно стиль mmap и совем не использует семафоров System V.

Выключение сервера

Сервер баз данных можно отключить несколькими способами. На практике они все сводятся к отправке сигнала управляющему процессу postgres.

Если сервер был установлен в виде готового продукта и был запущен предусмотренными в этом продукте средствами, то останавливать сервер необходимо этими же средствами.

В системе, где сервер запускается с пом. systemctl, следует использовать именно его.

```
sudo systemctl stop postgresql.service
```

Тем не менее, можно выбрать тот или иной вариант отключения, посылая разные сигналы главному процессу postgres:

SIGTERM

Запускает так называемое *умное выключение*. Получив SIGTERM, сервер перестаёт принимать новые подключения, но позволяет всем существующим сеансам закончить работу в штатном режиме. Сервер будет отключён только после завершения всех сеансов.

Если, получив этот сигнал, сервер находится в процессе восстановления, восстановление и потоковая репликация будут прерваны только после завершения всех обычных сеансов.

SIGINT

Запускает *быстрое выключение*. Сервер запрещает новые подключения и посылает всем работающим серверным процессам сигнал SIGTERM, в результате чего их транзакции прерываются и сами процессы завершаются. Управляющий процесс ждёт, пока будут завершены все эти процессы, и затем завершается сам.

SIGQUIT

Запускает *немедленное выключение*. Сервер отправляет всем дочерним процессам сигнал SIGQUIT и ждёт их завершения. Если какие-либо из них не завершаются в течение 5 секунд, им посылается SIGKILL.

Главный процесс сервера завершается, как только будут завершены все дочерние процессы, без выполнения обычной процедуры остановки БД. В результате при последующем запуске будет запущен процесс восстановления (воспроизведения изменений из журнала). Такой вариант выключения рекомендуется только в экстренных ситуациях.

Отправить эти сигналы можно, используя программу pg_ctl.

Кроме того, в системах, отличных от Windows, соответствующий сигнал можно отправить с помощью команды kill. PID основного процесса postgres можно узнать, воспользовавшись программой ps, либо прочитав файл postmaster.pid в каталоге данных. Например, можно выполнить быстрое выключение так:

Для завершения отдельного сеанса, не прерывая работу других сеансов, следует использовать функцию `pg_terminate_backend()` или отправить сигнал SIGTERM дочернему процессу, обслуживающему этот сеанс.

Обновление кластера PostgreSQL

- обновление данных с применением pg_dumpall;
- обновление данных с применением pg_upgrade;
- обновление данных с применением репликации.

Текущие номера версий PostgreSQL состоят из номеров основной и корректирующей (дополнительной) версии (15.10, 16.3, 17.2).

В корректирующих выпусках внутренний формат хранения никогда не меняется, и они всегда совместимы с предыдущими и последующими выпусками той же основной версии. Например, версия 16.2 совместима с версией 16.3 и версией 16.6.

Для обновления версии на совместимую достаточно заменить исполняемые файлы при выключенном сервере и затем запустить сервер. Каталог данных при этом не затрагивается.

При обновлении основных версий PostgreSQL внутренний формат данных может меняться.

Традиционный способ переноса данных в новую основную версию — выгрузить данные из старой версии, а затем загрузить их в новую (долго).

Утилита pg_upgrade позволяет выполнить обновление быстрее.

Для обновления также можно использовать репликацию.

Если используется Postgres в виде готового продукта, в нём могут содержаться вспомогательные скрипты для обновления основной версии.

Изменения основной версии обычно приносят какие-либо видимые пользователю несовместимости, которые могут требовать доработки приложений. Все подобные изменения описываются в замечаниях к выпуску (Приложение E).

Обновление данных с применением pg_dumpall

Один из вариантов обновления – выгрузка данных из одной основной версии в файл и загрузка их из этого файла в другую.

Для этого необходимо использовать программу логического копирования.

Для создания копии рекомендуется применять программы `pg_dump` и `pg_dumpall` от новой версии сервера. Текущие версии этих программ способны читать данные любой версии сервера, начиная с 9.2.

Далее предполагается, что сервер установлен в каталоге `/usr/local/pgsql`, а данные находятся в `/usr/local/pgsql/data`. Если это не так, нужно подставить свои пути.

1) При запуске резервного копирования следует убедиться в том, что в базе данных не производятся изменения. Изменения не повлияют на целостность полученной копии, но изменённые данные в неё не попадут.

Возможно, следует изменить разрешения в файле `/usr/local/pgsql/data/pg_hba.conf`, чтобы исключить другие подключения к серверу.

Получение копии всех данных:

```
pg_dumpall > outputfile
```

Для создания резервной копии можно воспользоваться программой pg_dumpall от текущей версии сервера. Однако для лучшего результата имеет смысл использовать pg_dumpall из версии 17.2. Это наиболее полезно, если предполагается установка новой версии рядом со старой. В этом случае можно выполнить установку как обычно, а перенести данные позже. Это также сократит время обновления.

2) Остановка старого сервера:

```
pg_ctl stop
```

В системах, где PostgreSQL запускается при загрузке, должен быть скрипт запуска, с помощью которого можно сделать то же самое. Например, в системах на базе init

```
/etc/rc.d/init.d/postgresql stop
```

а на базе systemd

```
systemctl stop postgresql.service
```

3) При восстановлении из резервной копии следует удалить или переименовать старый каталог, где был установлен сервер, если его имя не привязано к версии. Разумнее будет переименовать каталог, а не удалять его, чтобы его можно было восстановить в случае проблем. Однако учтите, что этот каталог может занимать много места на диске. Переименовать каталог можно, например так:

```
mv /usr/local/pgsql /usr/local/pgsql.old
```

4) Установка новой версии сервера
5) Создание нового кластера баз данных от имени рользователя сервера (postgres)

```
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
```

6) Перенос изменений, внесённых в предыдущие версии pg_hba.conf и postgresql.conf.
7) Запуск сервера от имени рользователя сервера (postgres)

```
/usr/local/pgsql/bin/postgres -D /usr/local/pgsql/data
```

8) Восстановление данных из резервной копии, используя новый psql:

```
/usr/local/pgsql/bin/psql -d postgres -f выходной_файл
```

Минимизировать время отключения сервера можно, установив новый сервер в другой каталог и запустив параллельно оба сервера, старый и новый на разных портах. После этого можно будет перенести данные, используя пайпы:

```
pg_dumpall -p 5432 | psql -d postgres -p 5433
```

pg_upgrade — обновить экземпляр сервера PostgreSQL

```
pg_upgrade -b oldbindir [-B newbindir] \
           -d oldconfigdir \
           -D newconfigdir [option...]
```

pg_upgrade позволяет обновить данные в каталоге базы данных до последней основной версии без операции выгрузки/восстановления данных при обновлениях основной версии.

С выходом новых основных версий в PostgreSQL регулярно добавляются новые возможности, которые часто меняют структуру системных таблиц, но внутренний формат хранения меняется редко. Учитывая этот факт, pg_upgrade позволяет выполнить быстрое обновление, создавая системные таблицы заново, но сохраняя старые файлы данных.

Если при обновлении основной версии формат хранения данных изменится так, что данные в старом формате окажутся нечитаемыми, pg_upgrade не сможет произвести такое обновление, однако в этом случае в замечаниях к выпуску (Приложение E) этот факт будет отмечен, а такде предложен способ выполнить обновление.

Параметры

`-b bindir, --old-bindir=bindir` – каталог с исполняемыми файлами старой версии. Также может использоваться переменная окружения PGBINOLD.

`-B bindir, --new-bindir=bindir` – каталог с исполняемыми файлами новой версии. По умолчанию это каталог, в котором располагается pg_upgrade. Также может использоваться переменная окружения PGBINNEW.

`-c, --check` – только проверить кластеры, не изменять никакие данные.

`-d configdir, --old-datadir=configdir` – каталог конфигурации старого кластера (переменная окружения PGDATAOLD).
`-D configdir, --new-datadir=configdir` – каталог конфигурации нового кластера (переменная окружения PGDATANEW).

Прежде чем переходить на новую версию полностью следует протестировать свои клиентские приложения с новой версией. Для этого нужно установить рядом старую и новую версии.

Процесс обновления с использованием pg_upgrade:

1) Переместить старый кластер (необязательно).

Если каталог инсталляции привязан к версии, например `/opt/PostgreSQL/17`, перемещать его не требуется. Все графические инсталляторы выбирают при установке каталоги, привязанные к версии.

Если каталог инсталляции не привязан к версии, например `/usr/local/pgsql`, необходимо переместить каталог текущей инсталляции, чтобы он не конфликтовал с новой. Когда текущий сервер отключён, каталог этой инсталляции можно безопасно переместить; если старый каталог `/usr/local/pgsql`, его можно переименовать, выполнив:

```
mv /usr/local/pgsql /usr/local/pgsql.old
```

2) Собрать новую версию при установке из исходного кода

Новую версию PostgreSQL следует собирать с флагами `configure`, совместимыми с флагами старого кластера. Тем не менее, прежде чем начинать обновление pg_upgrade проверит результаты pg_controldata, чтобы убедиться, что все параметры совместимы.

3) Установить новые исполняемые файлы PostgreSQL

Программа pg_upgrade включена в новую инсталляцию по умолчанию.

Если при установке из исходного кода требуется разместить сервер в новом размещении, следует воспользоваться переменной prefix:

```
make prefix=/usr/local/pgsql.new install
```

4) Инициализировать новый кластер PostgreSQL

Для этого используется initdb. Однако, для initdb следует указать флаги, совместимые с флагами в старом кластере. Многие готовые инсталляторы выполняют это действие автоматически. Запускать новый кластер не требуется.

5) Установка .so файлов расширений.

Многие расширения и пользовательские модули, как из contrib, так и из других источников, используют .so файлы (или библиотеки DLL), например, pgcrypto.so. Если они использовались в старом кластере, файлы, соответствующие новому исполняемому файлу сервера, должны быть установлены в новом кластере (обычно средствами операционной системы). Если обновления расширений доступны, `pg_upgrade` сообщит об этом и создаст скрипт, который можно будет запустить позже, чтобы обновить эти расширения.

6) Копирование пользовательских файлов полнотекстового поиска

Копирование словарей, тезаурусов, списков синонимов и стоп-слов из старого кластера в новый.

7) Настройка аутентификации

Программа `pg_upgrade` будет подключаться к новому и старому серверу несколько раз, так что имеет смысл установить режим аутентификации peer в `pg_hba.conf` или использовать файл ~/.pgpass (32.16).

8) Подготовка к обновлению публикующего сервера (см. документацию)

9) Подготовка к обновлению подписчика (см. документацию)

10 Остановка обоих серверов

```
pg_ctl -D /opt/PostgreSQL/16 stop
pg_ctl -D /opt/PostgreSQL/17 stop
```

11) Подготовиться к обновлению ведомых серверов (см. документацию)

12) Запуск pg_upgrade

Программу pg_upgrade всегда следует запускать от нового сервера, а не от старого.

pg_upgrade требует указания каталогов данных старого и нового кластера, а также каталогов исполняемых файлов (bin). Также можно определить имя пользователя и номера портов, и нужно ли копировать файлы данных (по умолчанию), клонировать их или создавать на них ссылки (hardlink).

Если выбрать вариант со ссылкой на данные, обновление выполнится гораздо быстрее (так как файлы не копируются) и потребует меньше места на диске, но возможность обращаться к старому кластеру после обновленияисчезнет.

Этот вариант также требует, чтобы каталоги данных старого и нового кластера располагались в одной файловой системе.

Вариант с клонированием работает так же быстро и экономит место на диске, но позволяет сохранить рабочее состояние старого кластера при запуске нового. Для этого варианта тоже требуется, чтобы старый и новый каталоги данных находились в одной файловой системе.

При запуске `pg_upgrade` проверит два кластера на совместимость и, если всё в порядке, выполнит обновление.

Также возможно запустить `pg_upgrade --check`, чтобы ограничиться только проверками (при этом старый сервер может продолжать работать).

`pg_upgrade --check` также сообщит, какие коррективы нужно будет внести вручную после обновления. Программе pg_upgrade требуются права на запись в текущий каталог.

Если при восстановлении схемы базы данных происходит ошибка, pg_upgrade завершает свою работу и необходимо вернуться к старому кластеру (п. 19).

Чтобы попробовать pg_upgrade ещё раз, необходимо внести коррективы в старом кластере, чтобы pg_upgrade могла успешно восстановить схему.

Если проблема возникла в модуле contrib, может потребоваться удалить этот модуль в старом кластере, а затем установить его в новом после обновления (предполагается, что этот модуль не хранит пользовательские данные).

13) Обновление ведомых серверов с потоковой репликацией и трансляцией журнала

14) Восстановление pg_hba.conf

Если pg_hba.conf изменялся (peer), необходимо восстановить его исходное состояние.

Также может потребоваться скорректировать другие файлы конфигурации в новом кластере, чтобы они соответствовали старому, например, postgresql.conf, файлы, включаемые в него и postgresql.auto.conf.

15) Запуск нового сервера

16) Действия после обновления

Если после обновления требуются какие-то дополнительные действия, программа pg_upgrade выдаст предупреждения об этом по завершении работы.

Она также сгенерирует файлы скриптов, которые должны запускаться администратором. Эти скрипты будут подключаться к каждой базе данных, требующей дополнительных операций. Каждый такой скрипт следует выполнять командой:

```
psql --username=postgres --file=script.sql postgres
```

Эти скрипты могут выполняться в любом порядке, и после выполнения их можно удалить.

17) Статистика

Так как статистика оптимизатора не передаётся в процессе работы pg_upgrade, будет выдано указание запустить соответствующую команду для воссоздания этой информации после обновления. Возможно, для этого понадобится установить параметры подключения к новому кластеру.

Такую статистику эффективно генерирует команда `vacuumdb --all –analyze-only`.

18) Удаление старого кластера

Если все завершилось нормально и старые приложения работают с новыс сервером, можно удалить каталоги данных старого кластера, запустив скрипт, упомянутый в выводе `pg_upgrade` после обновления. Также можете удалить каталоги старой инсталляции.

19) Возврат к старому кластеру

Если выполнив команду pg_upgrade, возникла необходимость вернуться к старому кластеру, возможны следующие варианты:

1) Если использовался ключ `--check`, в старом кластере ничего не меняется; его можно просто перезапустить.

2) Если не использовался ключ **--**link, в старом кластере ничего не изменилось и его можно просто перезапустить.

3) Если использовался ключ **--**link, у старого и нового кластера могут оказаться общие файлы данных:

- если работа pg_upgrade была прервана до начала создания ссылок, в старом кластере ничего не изменилось и его можно просто перезапустить.

- если новый кластер не запускался, старый кластер не претерпел никаких изменений, за исключением того, что при создании ссылки на данные к имени `$PGDATA/global/pg_control` был добавлен суффикс `.old`. Чтобы продолжить использование старого кластера, достаточно убрать суффикс `.old` из имени файла `$PGDATA/global/pg_control`; после этого старый кластер можно будет перезапустить.

- если вы запускали новый кластер, он внёс изменения в общие файлы, и использовать старый кластер небезопасно. В этом случае старый кластер нужно будет восстановить из резервной копии.

Обновление бинарной установки (на примере FC)

Для обновления второстепенной версии следует просто установить новые RPM; обычно это не более того. Обновление основного релиза требует больше усилий.

Если обновление выполняется более чем на одну основную версию, чтобы перенести данные в новую версию, необходимо следовать «традиционному» процессу дампа и перезагрузки:

1) перед обновлением следует запустить pg_dumpall, чтобы извлечь все данные в файл;
2) завершить работу старого сервера;
3) обновить RPM до новой версии;
4) выполнить initdb;
5) запустить файл дампа через psql, чтобы восстановить данные.

Для некоторых основных релизов RPM поддерживается более быстрое обновление с конкретного подмножества предыдущих релизов. Можно запустить

```
$ postgresql-setup --upgrade-ids
```

чтобы посмотреть, с каких предыдущих версий можно обновиться. Это намного быстрее, чем дамп и перезагрузка базы из него. Чтобы сделать более быстрое обновление:

1) завершние работы старого сервера, работающего со старыми данными;
2) при желании создание резервной копию каталога данных (рекомендуется);
3) установка RPM новой версии (все, что у вас были раньше, плюс postgresql-upgrade);
4) запуск от root'а «`postgresql-setup --upgrade [--upgrade-from ID]`»

5) обновление файлов конфигурации `/var/lib/pgsql/data/*.conf` в отношении настроек, которые были ранее (старые файлы конфигурации находятся в старом каталоге данных или в `/var/lib/pgsql/data-old/`, если обновление выполнялось на месте);

6) запуск от root'а «`systemctl start postgresql.service`»

7) старый каталог данных можно удалить после завершения обновления, как и пакет postgresql-upgrade.

ПРИМЕЧАНИЕ: Процесс обновления на месте новый и относительно плохо протестирован, поэтому если данные критически важны, то очень хорошей идеей будет сделать резервную копию tarball старого каталога данных перед запуском обновления.

Это позволит вернуться туда, где кластер был в случае катастрофы.

Защита сервера

- с применением ssl;
- с применением GSSAPI;
- с применением туннелей SSH.

Защита соединений TCP/IP с применением туннелей SSH

При правильном подходе это позволяет обеспечить должный уровень защиты сетевого трафика, даже для клиентов, не поддерживающих SSL.

На компьютере с сервером PostgreSQL должен работать сервер SSH и к нему можно подключаться через ssh каким-нибудь пользователем.

Клиент организует защищённый туннел с этим удалённым сервером. Туннель прослушивает локальный порт и перенаправляет весь трафик в порт удалённого компьютера.

Трафик, передаваемый в удалённый порт, может поступать как на адрес localhost, так и на другой привязанный адрес, если это необходимо. При этом не будет видно, что трафик поступает с вашего компьютера.

Для установки туннеля между клиентским компьютером и удалённой машиной foo.com используется команда

```
ssh -L 63333:localhost:5432 joe@foo.com
```

-L – режим тунеллирования;

63333 — это локальный номер порта туннеля (любой незанятый порт из разрешенных IANA для частного использования – с 49152 по 65535).

Имя или IP-адрес после него обозначает привязанный адрес, к которому подключается клиент, в данном случае это localhost (и это же значение по умолчанию).

5432 — порт, на котором принимает запросы сервер.

Чтобы подключиться к этому серверу через созданный тоннель, нужно подключиться к порту 63333 на локальном компьютере:

```
psql -h localhost -p 63333 postgres
```

Для сервера баз данных это будет выглядеть так, как будто к нему подключается пользователь joe компьютера foo.com и он будет применять ту процедуру проверки подлинности, которая установлена для подключений данного пользователя к этому привязанному адресу.

Трафик между сервером SSH и сервером PostgreSQL не защищён, однако это не должно нести какие-то дополнительные риски, поскольку эти серверы работают на одном компьютере.

Аутентификация клиентского приложения

При подключении к серверу базы данных, клиентское приложение указывает имя пользователя PostgreSQL, аналогично тому, как это имеет место при обычном входе пользователя на компьютер с ОС Unix.

При работе в среде SQL по имени пользователя определяется, какие у него есть права доступа к объектам базы данных.

PostgreSQL управляет правами и привилегиями, используя так называемые «роли».

В отношении подключения к серверу подразумевается «роль с привилегией LOGIN».

Аутентификация это процесс идентификации клиента сервером базы данных, а также определение того, может ли клиентское приложение (или пользователь запустивший приложение) подключиться с указанным именем пользователя.

PostgreSQL предлагает несколько различных методов аутентификации клиентов. Метод аутентификации конкретного клиентского соединения может основываться на адресе компьютера клиента, имени базы данных, имени пользователя.

Имена пользователей базы данных PostgreSQL не имеют прямой связи с пользователями операционной системы на которой запущен сервер.

Если у всех пользователей базы данных заведена учётная запись в операционной системе сервера, то имеет смысл назначить им точно такие же имена для входа в PostgreSQL. Однако сервер, принимающий удалённые подключения, может иметь большое количество пользователей базы данных, у которых нет учётной записи в ОС. В таких случаях соответствия между именами пользователей базы данных и именами пользователей операционной системы не требуется.

Файл pg_hba.conf

Аутентификация клиентов управляется конфигурационным файлом, который традиционно называется `pg_hba.conf`[3] и расположен в каталоге с данными кластера базы данных.

`pg_hba.conf`, со стандартным содержимым, создаётся командой `initdb` при инициализации каталога с данными. Однако его можно разместить в любом другом месте (конфигурационный параметр hba_file).

`pg_hba.conf` прочитывается при запуске системы, а также в тот момент, когда основной сервер получает сигнал SIGHUP. Если во время работы системы файл `pg_hba.conf` редактировался, необходимо послать процессу postmaster этот сигнал (pg_ctl reload, вызвав SQL-функцию pg_reload_conf() или выполнив kill -HUP), чтобы он прочел обновлённый файл.

```
$ sudo systemctl status postgresql
...
   Main PID: 88741 (postmaster)
...
```

Общий формат файла pg_hba.conf — набор записей, по одной на строку.

Пустые строки игнорируются, как и любой текст комментария после знака #.

Запись может быть продолжена на следующей строке, для этого нужно завершить строку обратной косой чертой `'\'`.

Запись состоит из нескольких полей, разделённых пробелами и/или табуляциями.

Поля могут содержать пробелы, если содержимое этих полей заключено в кавычки.

---

3) host-based authentication — аутентификации по имени узла

Если в кавычки берётся какое-либо ключевое слово в поле базы данных, пользователя или адресации (например, all или replication), то слово теряет своё особое значение и просто обозначает базу данных, пользователя или сервер с данным именем.

Обратная косая черта обозначает перенос строки даже в тексте в кавычках или комментариях.

Каждая запись аутентификации обозначает:

- тип соединения
- диапазон IP-адресов клиента (если он соотносится с типом соединения)
- имя базы данных
- имя пользователя
- способ аутентификации, который будет использован для соединения в соответствии с этими параметрами.

Для аутентификации применяется первая запись с соответствующим типом соединения, адресом клиента, указанной базой данных и именем пользователя.

Процедур «fall-through» или «backup» не предусмотрено – если выбрана запись и аутентификация не прошла, последующие записи не рассматриваются.

Если же ни одна из записей не подошла, в доступе будет отказано.

Каждая запись может быть директивой включения или записью аутентификации.

Директивы включения указывают, что нужно обработать дополнительные файлы с записями и записи из них будут вставлены вместо директив включения. Директивы включения содержат только два поля: директиву `include`, `include_if_exists` или `include_dir` и включаемый файл или каталог. Путь к файлу или каталогу может задаваться и как абсолютный, и как относительный и заключаться в двойные кавычки. Для формы `include_dir` будут включены все файлы, не начинающиеся с точки `'.'` и заканчивающиеся на `.conf`.

Набор файлов во включаемом каталоге обрабатывается по порядку имён локали C.
Записи могут иметь следующие форматы:

```
local           database  user              auth-method [auth-options]
host            database  user  address     auth-method [auth-options]
hostssl         database  user  address     auth-method [auth-options]
hostnossl       database  user  address     auth-method [auth-options]
hostgssenc      database  user  address     auth-method [auth-options]
hostnogssenc    database  user  address     auth-method [auth-options]
host            database  user  IP-address  IP-mask      auth-method   [auth-options]
hostssl         database  user  IP-address  IP-mask      auth-method   [auth-options]
hostnossl       database  user  IP-address  IP-mask      auth-method   [auth-options]
hostgssenc      database  user  IP-address  IP-mask      auth-method   [auth-options]
hostnogssenc    database  user  IP-address  IP-mask      auth-method   [auth-options]
include         file
include_if_exists file
include_dir     directory
```

`local` – управляет подключениями через Unix-сокеты. Без подобной записи подключения через Unix-сокеты невозможны.

`host` – управляет подключениями, устанавливаемыми по TCP/IP. Записи host соответствуют подключениям с SSL и без SSL, а также подключениям, защищённым механизмами GSSAPI и не защищённым GSSAPI.

Удалённое соединение по TCP/IP невозможно, если сервер запущен без определения соответствующих значений для параметра конфигурации `listen_addresses`.

database – определяет, каким именам баз данных соответствует эта запись.
Значения:

all – подходят все базы данных;

sameuser – соответствует, только если имя запрашиваемой базы данных совпадает с именем запрашиваемого пользователя;

samerole – запрашиваемый пользователь должен быть членом роли с таким же именем, как и у запрашиваемой базы данных;
Суперпользователи, несмотря на то, что они имеют полные права, считаются включёнными в samerole, только когда они явно входят в такую роль, непосредственно или косвенно;

replication – запись соответствует, когда запрашивается подключение для физической репликации, но не когда запрашивается подключение для логической.

Для подключений физической репликации не указывается какая-то конкретная база данных, в то время как для подключений логической репликации должна указываться конкретная база.

Любое другое значение воспринимается как имя определённой базы PostgreSQL или регулярное выражение. Несколько регулярных выражений и/или имён баз данных можно указать, разделяя их запятыми.

Если имя базы данных начинается с косой черты (/), оставшаяся часть имени обрабатывается как регулярное выражение.

Также можно задать отдельный файл с именами баз данных и/или регулярными выражениями, написав имя файла после знака @.

user – указывает, какому имени (или именам) пользователя базы данных соответствует эта запись.

all указывает, что запись соответствует всем пользователям. Любое другое значение задаёт либо имя конкретного пользователя базы данных, либо имя группы[4] (если это значение начинается с +). Если имя пользователя начинается с косой черты (/), оставшаяся часть имени обрабатывается как регулярное выражение.

Также можно задать отдельный файл с именами пользователей и/или регулярными выражениями, написав имя файла после знака @.

address – указывает адрес (или адреса) клиентской машины, которым соответствует данная запись. Это поле может содержать или имя компьютера, или диапазон IP-адресов, или одно ключевых слов (см. документацию).

Диапазон IP-адресов указывается в виде начального адреса диапазона, дополненного косой чертой (/) и длиной маски CIDR. Длина маски задаёт количество старших битов клиентского IP-адреса, которые должны совпадать с битами IP-адреса диапазона. Биты, находящиеся правее, в указанном IP-адресе должны быть нулевыми. Между IP-адресом, знаком / и длиной маски CIDR не должно быть пробельных символов.

Типичные примеры диапазонов адресов IPv4, указанных таким образом:

172.20.143.89/32 для одного компьютера;

172.20.143.0/24 для небольшой сети;

10.6.0.0/16 для крупной сети.

Чтобы указать один компьютер, для IPv4 следует использовать длину маски 32.

all – любой IP-адрес, samehost – любые IP-адреса данного сервера, samenet – любой адрес любой подсети, к которой сервер подключён напрямую.

---

4) В PostgreSQL нет никакой разницы между пользователем и группой

Если определено имя компьютера (всё, что не является диапазоном IP-адресов или специальным ключевым словом, воспринимается как имя компьютера), то оно сравнивается с результатом обратного преобразования IP-адреса клиента (например, обратного DNS-запроса, если используется DNS).

При сравнении имён компьютеров регистр не учитывается.

Если имена совпали, выполняется прямое преобразование имени (например, прямой DNS-запрос) для проверки, относится ли клиентский IP-адрес к адресам, соответствующим имени.

В качестве имени узла в файле pg_hba.conf должно указываться то, что возвращается при преобразовании IP-адреса клиента в имя, иначе строка не будет соответствовать узлу.

Некоторые базы данных имён позволяют связать с одним IP-адресом несколько имён узлов, но операционная система при попытке разрешить IP-адрес возвращает только одно имя.

Указание имени, начинающееся с точки (.), соответствует суффиксу актуального имени узла. Т.е. `.example.com` будет соответствовать `foo.example.com` (а не только `example.com`).

Когда в `pg_hba.conf` указываются имена узлов, следует добиться, чтобы разрешение имён выполнялось достаточно быстро. Для этого может быть полезен локальный кеш разрешения имён, например, `nscd`.

Эти поля не применимы к записям local.

IP-address, **IP-mask** – эти два поля могут быть использованы как альтернатива записи IP-адрес/длина-маски. Вместо того, чтобы указывать длину маски, в отдельном столбце указывается сама маска. Например, 255.0.0.0 представляет собой маску CIDR для IPv4 длиной 8 бит, а 255.255.255.255 представляет маску CIDR длиной 32 бита.

Эти поля не применимы к записям local.

auth-method – метод аутентификации. Подробности в (20.3).

Все значения воспринимаются с учётом регистра и должны быть записаны в нижнем регистре, даже аббревиатуры типа `ldap`.

trust – разрешает безусловное подключение. Этот метод позволяет тому, кто может подключиться к серверу с базой данных PostgreSQL, войти под любым желаемым пользователем PostgreSQL без введения пароля и без какой-либо другой аутентификации.

reject – отклоняет подключение безусловно. Эта возможность полезна для «фильтрации» некоторых серверов группы, например, строка `reject` может отклонить попытку подключения одного компьютера, при этом следующая строка позволяет подключиться остальным компьютерам в той же сети.

scram-sha-256 – проверяет пароль пользователя, производя аутентификацию SCRAM-SHA-256 (20.5).

md5 – проверяет пароль пользователя, производя аутентификацию SCRAM-SHA-256 или MD5 (20.5).

password – требует для аутентификации введения клиентом незашифрованного пароля. Поскольку пароль посылается простым текстом через сеть, такой способ не стоит использовать, если сеть не вызывает доверия (20.5).

gss – для аутентификации использует GSSAPI. Способ доступен только для подключений по TCP/IP (20.6). Может применяться в сочетании с шифрованием GSSAPI.

sspi – для аутентификации использует SSPI. Доступен только для Windows (20.7).

ident – получает имя пользователя операционной системы клиента, связываясь с сервером Ident, и проверяет, соответствует ли оно имени пользователя базы данных. Аутентификация ident может использоваться только для подключений по TCP/IP. Для локальных подключений применяется аутентификация peer (20.8).

peer – получает имя пользователя операционной системы клиента из операционной системы и проверяет, соответствует ли оно имени пользователя запрашиваемой базы данных.

Доступно только для локальных подключений (20.9).

ldap – проводит аутентификацию, используя сервер LDAP (20.10).

radius – проводит аутентификацию, используя сервер RADIUS (20.11).

cert – проводит аутентификацию, используя клиентский сертификат SSL (20.12).

pam – роводит аутентификацию, используя службу подключаемых модулей аутентификации (PAM), предоставляемую операционной системой (20.13).

bsd – проводит аутентификацию, используя службу аутентификации BSD, предоставляемую операционной системой (20.14).

Записи файла pg_hba.conf рассматриваются последовательно для каждого подключения, поэтому порядок записей имеет большое значение.

Обычно более ранние записи определяют чёткие критерии для соответствия параметров подключения, но для методов аутентификации допускают послабления.

Записи более поздние смягчают требования к соответствию параметров подключения, но усиливают их в отношении методов аутентификации.

Например, некто желает использовать `trust` аутентификацию для локального подключения по TCP/IP, но при этом запрашивать пароль для удалённых подключений по TCP/IP. В этом случае запись, устанавливающая аутентификацию `trust` для подключения адреса 127.0.0.1, должна предшествовать записи, определяющей аутентификацию по паролю для более широкого диапазона клиентских IP-адресов.

[auth-options] – См. в документации.

Пример 20.1. Примеры записей pg_hba.conf

```
# Позволяет любому пользователю локальной системы подключаться к любой
# базе данных, используя любое имя пользователя баз данных, через
# Unix-сокеты (по умолчанию для локальных подключений).
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             all                                     trust

# То же, но для локальных "замкнутых" подключений по TCP/IP.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             127.0.0.1/32            trust

# То же самое, но с использованием имени узла
# (обычно покрывает и IPv4, и IPv6).
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             localhost               trust

# То же самое с использованием регулярного выражения для DATABASE, позволяющего
# подключитьсяк любым базам данных с именем, начинающимся с "db" и
# заканчивающимся на число, содержащее от 2 до 4 цифр ("db1234" или "db12").
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    "/^db\d{2,4}$"  all             localhost               trust
```

```
# Позволяет любому пользователю любого компьютера с IP-адресом 192.168.93.x
# подключаться к базе данных "postgres" с именем, которое сообщает для данного
# подключения ident (как правило, это имя пользователя операционной системы).
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    postgres        all             192.168.93.0/24         ident

# Позволяет любому пользователю компьютера 192.168.12.10 подключаться
# к базе данных "postgres", если он передаёт правильный пароль.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    postgres        all             192.168.12.10/32        scram-sha-256

# Позволяет любым пользователям с компьютеров в домене example.com
# подключаться к любой базе данных, если передаётся правильный пароль.
#
# Для всех пользователей требуется аутентификация SCRAM, за исключением
# пользователя 'mike', который использует старый клиент, не поддерживающий
# аутентификацию SCRAM.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             mike            .example.com            md5
host    all             all             .example.com            scram-sha-256

# В случае отсутствия предшествующих строчек с "host", следующие две строки
# откажут в подключении с 192.168.54.1 (поскольку данная запись будет
# выбрана первой), но разрешат подключения GSSAPI с любых других
# адресов. С нулевой маской ни один бит из IP-адреса компьютера
```

```
# не учитывается, так что этой строке соответствует любой компьютер.
# Незашифрованные средствами GSSAPI подключения (они "опускаются"
# до третьей строки, так как записи "hostgssenc"
# соответствуют только
# зашифрованные подключения) принимаются, но только с адреса 192.168.12.10.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             192.168.54.1/32         reject
hostgssenc all          all             0.0.0.0/0               gss
host    all             all             192.168.12.10/32        gss

# Позволяет пользователям с любого компьютера 192.168.x.x подключаться
# к любой базе данных, если они проходят проверку ident. Если же ident
# говорит, например, что это пользователь "bryanh" и он запрашивает
# подключение как пользователь PostgreSQL "guest1", подключение
# будет разрешено, если в файле pg_ident.conf есть сопоставление
# "omicron", позволяющее пользователю "bryanh" подключаться как "guest1".
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             192.168.0.0/16          ident map=omicron

# Если для локальных подключений предусмотрены только эти четыре строки,
# они позволят локальным пользователям подключаться только к своим
# базам данных (базам данных с именами, совпадающими с
# именами пользователей баз данных), кроме пользователей, чьё имя
# заканчивается на "helpdesk", администраторов
# или членов роли "support", которые могут подключиться к любой БД.
# Список имён администраторов содержится в файле $PGDATA/admins.
```

```
# Пароли требуются в любом случае.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   sameuser        all                                     md5
local   all             /^.*helpdesk$                           md5
local   all             @admins                                 md5
local   all             +support                                md5

# Последние две строчки выше могут быть объединены в одну:
local   all             @admins,+support                        md5

# В столбце DATABASE также могут указываться списки и имена файлов:
local   db1,db2,@demodbs  all                                   md5
```

Методы аутентификации

PostgreSQL предлагает к использованию широкий набор методов аутентификации пользователей:

trust – сервер доверяет пользователям, никак не проверяя их.

password – требуется ввод пароля пользователем.

GSSAPI – используется библиотека безопасности, совместимая с GSSAPI. Обычно этот метод применяется при использовании специальной службы аутентификации, Kerberos или Microsoft Active Directory.

SSPI – используется протокол, подобный GSSAPI, но предназначенный для Windows.

ident – используется служба, реализующая «Identification Protocol» (RFC 1413) на клиентском компьютере. Для подключений через локальный сокет Unix этот метод работает как peer.

peer – полагается на средства операционной системы, позволяющие узнать пользователя процесса на другой стороне локального подключения. Для удалённых подключений она не поддерживается.

LDAP – работает с сервером аутентификации LDAP.

RADIUS – работает с сервером аутентификации RADIUS.

по сертификату – требуется использования клиентами SSL-подключения. Построена на проверке передаваемых ими сертификатов SSL.

PAM – реализуется с использованием библиотеки PAM.

BSD – основвывается на использовании механизма аутентификации BSD (в настоящее время поддерживается только в системе OpenBSD).

Для локальных подключений обычно рекомендуется использовать метод peer, хотя в некоторых обстоятельствах может быть достаточно и режима trust.

Для удалённых подключений самой простой будет аутентификация по паролю.

Все остальные варианты требуют использования некоторой внешней инфраструктуры безопасности (обычно это служба аутентификации или центр сертификации, выдающий сертификаты SSL) либо поддерживаются не на всех платформах.

Аутентификация password

Существует несколько методов аутентификации по паролю. Они работают примерно одинаково, но различаются тем, как пароли пользователей хранятся на сервере и как пароль передаётся от клиента по каналу связи.

scram-sha-256 – выполняется аутентификация SCRAM-SHA-256, как описано в RFC 7677. Она производится по схеме вызов-ответ, которая предотвращает перехват паролей через недоверенные соединения и поддерживает хранение паролей на сервере в виде криптографического хеша, что считается безопасным. Это наиболее безопасный из существующих на данный момент методов, но он не поддерживается старыми клиентами.

md5 – для метода md5 реализован менее безопасный механизм вызов-ответ. Он предотвращает перехват паролей и предусматривает хранение паролей на сервере в зашифрованном виде, но не защищает в случае похищения хешей паролей с сервера. Кроме того, алгоритм хеширования MD5 в наши дни уже может и не защитить от целенаправленных атак.

Если в качестве метода аутентификации в **pg_hba.conf** указан md5, а пароль пользователя на сервере зашифрован для SCRAM, автоматически будет производиться аутентификация на базе SCRAM.

password – пароль передаётся в открытом виде и поэтому является уязвимым для атак с перехватом трафика. Его следует избегать всегда, если это возможно. Однако если подключение защищено SSL, метод password может быть безопасен.

Пароли баз данных PostgreSQL отделены от паролей пользователей операционной системы.

Пароли всех пользователей базы данных хранятся в системном каталоге `pg_authid`.

Управлять паролями можно либо используя SQL-команды CREATE ROLE и ALTER ROLE, например,

```
CREATE ROLE foo WITH LOGIN PASSWORD '1234';
```

либо с помощью команды `psql \password`.

Если пароль для пользователя не задан, вместо него хранится NULL, и пройти аутентификацию по паролю этот пользователь не сможет.