Авторы: Егор Рогов, Павел Лузанов

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

Нумерация версий и общие замечания

Обновление на дополнительный выпуск

Обновление основной версии

Версии сервера

Основные версии (9.6, 10, 11, ...)

- поддержка сообщества в течение 5 лет
- добавляется и изменяется функционал
- новая версия не совместима на двоичном уровне с предыдущей
- для обновления всегда требуются специальные действия

Дополнительные выпуски (10.1, 10.2, ...)

- только исправление ошибок и проблем безопасности
- гарантируется двоичная совместимость
- обычно достаточно установить новые исполняемые файлы

Номер версии PostgreSQL состоит из двух частей: номер основной версии (major release) и номер дополнительного выпуска (minor release). До версии 10 основной номер состоял из двух чисел (9.5, 9.6), затем перешли на одно (10, 11). Номер дополнительного выпуска — последнее число через точку (например, 5 в 10.5).

Дополнительные выпуски служат только и исключительно для исправления ошибок, найденных в основной версии. Для них гарантируется сохранение двоичной совместимости (на одной платформе). Поэтому обновление на следующий дополнительный выпуск делается максимально просто и рекомендуется, как только дополнительный выпуск появляется.

Новая основная версия привносит изменение функционала: какие-то возможности добавляются, изменяются, реже — удаляются. В этом случае двоичная совместимость отсутствует. Если попробовать подключить новую версию исполняемых файлов к старой версии кластера БД, PostgreSQL откажется с ним работать.

Для обновления основной версии требуется предпринимать специальные шаги. Не исключено, что помимо обновления сервера БД, потребуется внесение изменений и в приложение. Мотивацией для перехода на новую версию служит появление новых возможностей и окончание поддержки старой версии (которая распространяется на 5 лет с момента выпуска). Из-за сложности процесса часто пропускают несколько версий, например, переходят с 9.5 сразу на 10 и т. п.

https://postgrespro.ru/docs/postgresql/10/upgrading

Общие соображения

«Замечания к выпуску»

- необходимо изучить раздел «Миграция» всех промежуточных версий

Проверка обновления на тестовом окружении

- заранее обнаружить возможные проблемы
- автоматизировать процесс для сокращения времени простоя

Запасной вариант

- на случай проблем при обновлении производственного сервера

Независимо от того, какое обновление выполняется, стоит обратить внимание на некоторые моменты.

Всегда следует внимательно ознакамливаться с приложением «Замечания к выпуску» (Release Notes) документации. В нем в разделе «Миграция» описаны все несовместимости и нестандартные действия, необходимые при обновлении.

Например, если планируется переход с 9.6.8 на 10.5 (последний доп. выпуск выбранной основной версии), надо прочитать все замечания к версиям 9.6.9, 9.6.10 (последний доп. выпуск для 9.6), 10.0, …, 10.5.

Обновление следует проверять на тестовой среде, эквивалентной производственной, с последующим тестированием приложения, что позволит заранее выявить возможные проблемы и «обкатать» и автоматизировать процедуру обновления.

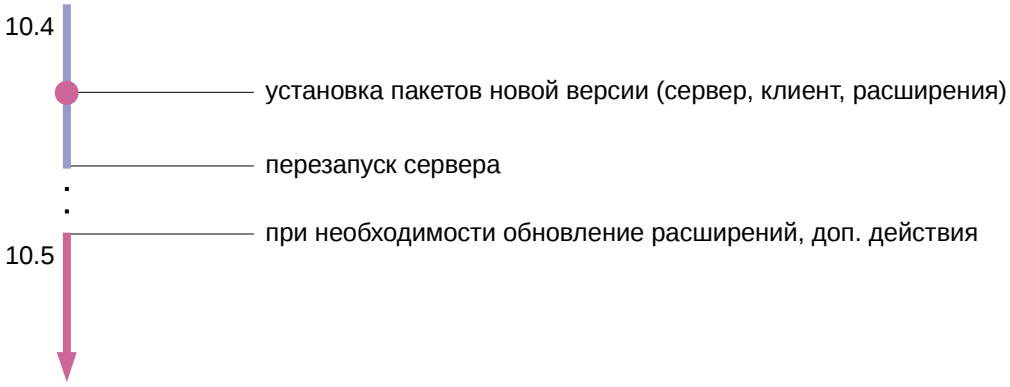
Это существенно снижает риск возникновения проблем при обновлении производственного сервера, но не исключает его полностью. Поэтому всегда необходимо иметь (протестированный) запасной вариант на случай, если обновление пойдет не так. Таким вариантом, например, может быть готовая реплика или резервная копия, в зависимости от требований к возможному времени простоя.

Дополнительный выпуск

Обновление на дополнительный выпуск

Использование физических реплик

Дополнительный выпуск



\+ простота

– вынужденное прерывание обслуживания

Для того, чтобы обновить сервер до очередного дополнительного выпуска, требуется:

1. Установить новые исполняемые файлы.
Если PostgreSQL был установлен из пакета, необходимо установить новые версии пакетов (сервер, клиент, расширения). Некоторые пакетные менеджеры могут по умолчанию автоматически перезапускать сервер; обычно это нежелательно. Если PostgreSQL был собран из исходных кодов, то необходимо выполнить make install.

2. Перезапустить сервер. После перезапуска он продолжит работу уже на новой версии.
Время перезапуска составляет время прерывания обслуживания клиентов.

3. После перезапуска может потребоваться обновление расширений (ALTER EXTENSION … UPDATE) и выполнение дополнительных действий, обозначенных в «Замечаниях к выпуску». Однако в большинстве случаев это не требуется.

Уменьшение простоя

Процесс перезапуска сервера

запрещаются новые соединения

smart ожидает завершения всех сеансов,
fast принудительно отключает все сеансы (по умолчанию)

выполняется контрольная точка

Контрольная точка

вручную до перезапуска сервера

PgBouncer

«пауза» на время перезапуска сервера

открытые транзакции продолжают выполняться,
все новые — приостанавливаются

Время прерывания обслуживания можно сократить, а в некоторых случаях и устранить полностью.

Что происходит при перезапуске сервера?

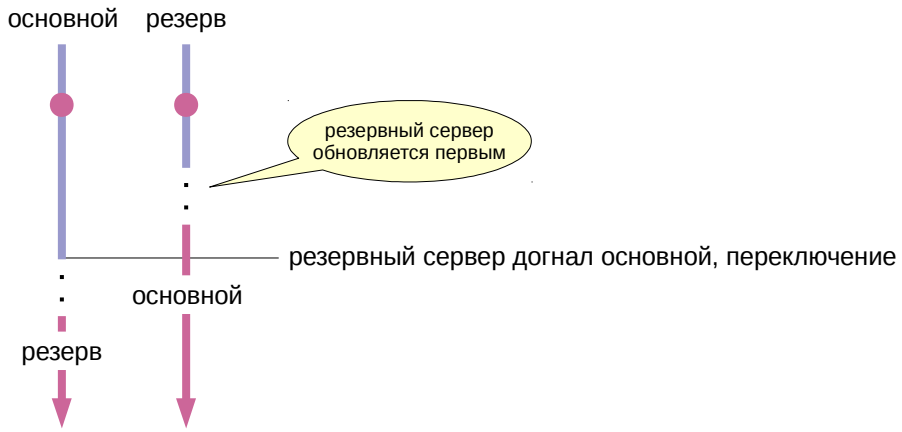
Во-первых, PostgreSQL перестает принимать новые подключения (клиенты будут получать ошибку «shutdown in progress»).

Во-вторых, в зависимости от режима останова, сервер либо дожидается завершения активных сеансов, либо принудительно завершает их. Последнее подразумевается по умолчанию утилитой pg_ctl, но, вероятно, это не лучший выбор для производственной среды, если сервер в основном имеет дело с OLTP-нагрузкой (короткие запросы).

В-третьих, выполняется финальная контрольная точка (shutdown checkpoint), чтобы синхронизировать данные из буферов в оперативной памяти с диском (как рассматривалось в модуле «Журналирование»). При большом размере буферного кэша этот процесс может занимать значительное время. Поэтому может оказаться выгодным сначала выполнить контрольную точку вручную (CHECKPOINT), а затем перезапустить сервер. В этом случае финальная контрольная точка все равно будет выполнена, но пройдет значительно быстрее.

Если используется pgbouncer, имеет смысл поставить его на «паузу» на время перезагрузки. При этом (подразумевая режим «transaction») работающие транзакции продолжат выполнение, но все новые будут отложены. Таким образом (для OLTP-приложения) перезапуск будет выглядеть, как увеличившееся время отклика СУБД.

Репликация



+ возможно обновление без или с минимальным прерыванием обслуживания

Если в системе используется физическая репликация и предусмотрен механизм плавного переключения пользователей между серверами, процедуру обновления можно провести и без прерывания обслуживания.

Для этого сначала выполняют обновление резервного сервера. Во время его перезапуска вся нагрузка ложится на основной сервер. Поскольку версии двоично-совместимы, репликация работает между уже обновленным резервным сервером и еще не обновленным основным. В абсолютном большинстве случаев репликация будет работать и в другом направлении (от обновленного сервера к серверу со старой версией), но от ошибок обратной совместимости никто не застрахован. Поэтому сначала лучше обновлять именно резервный сервер.

Затем, когда резервный сервер «догоняет» после перезагрузки основной сервер, основной сервер останавливается и выполняется переключение на резервный. После запуска бывшего основного сервера он подключается к новому основному в качестве резервного (это может потребовать запуска утилиты pg_rewind; вся процедура рассматривается подробно в курсе DBA3).

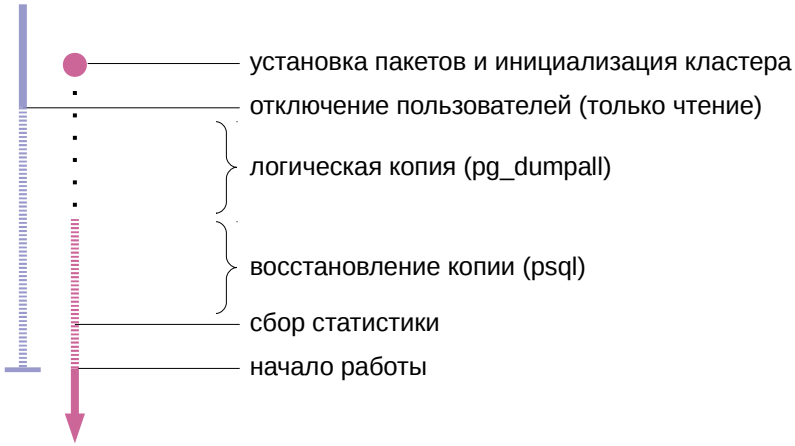
Основная версия

Логическая резервная копия pg_dumpall

Утилита pg_upgrade

Логическая репликация

pg_dumpall



\+ можно сменить платформу, разрядность, кодировку баз данных и т. п.

– большое время обновления, требуется много дискового пространства

10

Самый простой способ обновления на основную версию — сделать *логическую* резервную копию и развернуть ее на другом сервере с новой версией. (Логическая копия — это команды SQL, воссоздающие базу данных; подробнее см. курс DBA3).

Сначала устанавливаем новый сервер и инициализируем кластер (некоторые пакетные менеджеры выполняют инициализацию сразу при установке).

Если новый сервер разворачивается на том же компьютере, надо позаботиться, чтобы каталоги данных не пересекались. Обычно пакеты собираются с учетом этого требования (например, /var/lib/pgsql/9.6/ и /var/lib/pgsql/10/). Потребуется изменить конфигурационные файлы нового кластера, внеся в них изменения с работающего кластера.

Резервную копию надо делать при работающем сервере, но требуется отключить всех пользователей (кроме только читающих), поскольку изменения, сделанные после запуска резервного копирования, не попадут в копию.

После этого можно запускать новый сервер и в нем разворачивать подготовленную резервную копию. И затем собрать статистику — без этого оптимизатор не сможет строить адекватные планы запросов.

После всех необходимых проверок каталоги старого сервера (исполняемые файлы, содержимое PGDATA, пользовательские табличные пространства) можно удалить.

pg_dumpall

Ускорение переноса данных

```
pg_dumpall --globals-only

pg_dump --format=d --jobs=N
pg_restore --jobs=N
```

} для каждой базы данных отдельно

Экономия места

```
pg_dumpall | psql параметры-соединения-с-новым-кластером
```

Сбор статистики

```
$ vacuumdb --analyze-in-stages
```

расширение `dump_stat`

Быстрая проверка

```
pg_dumpall --schema-only
```


Хотя утилита pg_dumpall и выполняет резервную копию всего кластера, она всегда выполняется в один поток. Восстановление также выполняется в один поток утилитой psql.

Если аппаратные ресурсы позволяют, можно сохранить глобальные объекты кластера с помощью pg_dumpall, а затем выполнить копии каждой БД отдельно в параллельном режиме с помощью утилиты pg_dump в формате directory. В таком случае и восстановление можно выполнять в параллельном режиме утилитой pg_restore. Подробности см. в курсе DBA3.

Если стоит задача сэкономить место на диске, можно не сохранять резервную копию, а просто направить выход pg_dumpall на вход psql.

Чтобы как можно раньше собрать хоть какую-то статистику, можно выполнять сбор в несколько (3) этапов. Можно попробовать начать работу с СУБД уже после второго этапа, сократив тем самым время прерывания обслуживания. Еще одна возможность — использовать внешнее расширение dump_stat.

При проверке обновления на тестовой среде сначала можно выполнить обновление с переносом только схемы данных (без самих данных). Таким образом можно достаточно быстро выявить часть возможных проблем.

Заметим, что лучше использовать все утилиты от новой (уже установленной) версии, поскольку они могут содержать улучшения, недоступные в предыдущей версии.

pg_upgrade

Условия применимости

- в новой версии изменяется формат только служебной информации
- в остальном сохраняется двоичная совместимость, включая представление типов и файлы данных
- в базах данных не используются REG-типы
- обновление с версии 8.4 или более поздней
- обновление до версии, соответствующей утилите

Второй, наиболее распространенный, способ обновления — утилита pg_upgrade, которая умеет приводить файлы кластера к виду, совместимому с новой версией.

Возможность применения утилиты обусловлена тем, что при обновлении основной версии (как правило) меняются служебные таблицы, относящиеся к системному каталогу, но форматы файлов данных и индексов остаются без изменений. Поэтому достаточно поправить только небольшую часть данных, оставив основной массив как есть, без изменений.

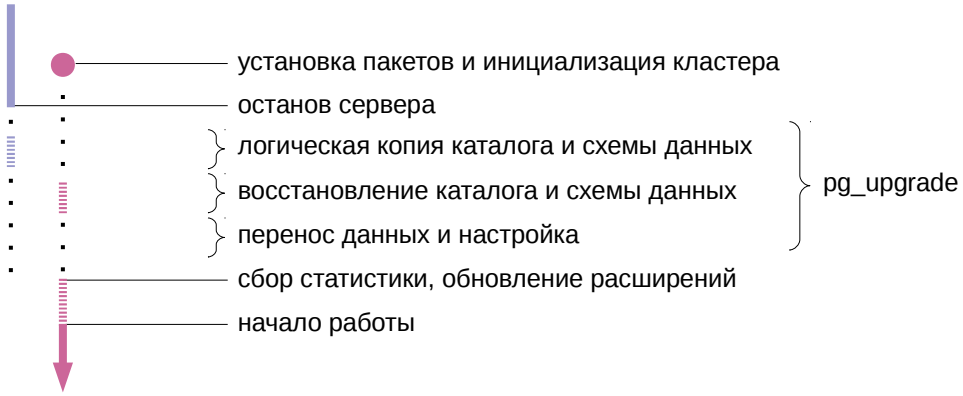
При этом, конечно, новый кластер должен быть совместим со старым по разрядности, кодировке и локалям.

В некоторых версиях изменяется двоичное представление типов данных. Например, в 9.4 тип jsonb изменился по сравнению с бета-версией 9.4. В таких случаях утилита не сможет помочь, но по крайней мере перечислит, где используется проблемный тип. Это поможет избавиться от него вручную перед обновлением.

Есть и дополнительные ограничения. Например, утилита не сможет обновить кластер, если в одной из баз данных используются REG-типы (кроме regtype, regclass и regenum).

Утилитой поддерживается обновление любой версии, начиная с 8.4, на версию, соответствующую утилите. Таким образом, pg_upgrade от версии 10 может обновить старый сервер только до 10, но не до 9.6.

pg_upgrade



\+ скорость обновления, обычно не требуется дополнительное место

– ограниченная применимость

13

Вначале рассмотрим общую (упрощенную) последовательность действий при обновлении с помощью pg_upgrade.

Сначала необходимо установить новый сервер и инициализировать (но не запускать) кластер. Этот шаг выполняется так же, как и при обновлении с помощью pg_dumpall.

Затем старый сервер останавливается и запускается утилита pg_upgrade. Пока обозначим схематически, что она делает:

- Старый сервер запускается на время, чтобы сделать логическую резервную копию общих объектов кластера и схемы данных всех БД. Это как раз та информация, для которой теряется двоичная совместимость.

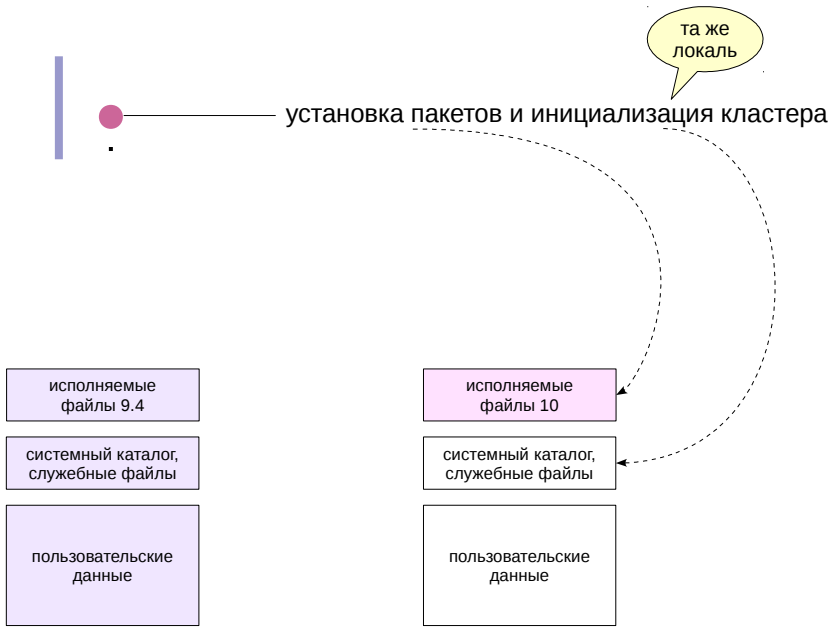
- Временно запускается новый сервер, выполняются необходимые проверки применимости, из резервной копии восстанавливаются глобальные объекты и схема данных.

- Выполняется перенос данных: либо копирование, либо (что гораздо удобнее и быстрее) простановка жестких ссылок (hard link).

В результате работы утилиты новый кластер становится обновленной версией старого. Далее выполняются заключительные действия, такие, как сбор статистики и обновление расширений.

Огромный плюс pg_upgrade состоит в возможности очень быстрого обновления кластера любого размера без дополнительного места на диске (при использовании жестких ссылок). Из минусов отметим то, что утилита применима не всегда.

pg_upgrade



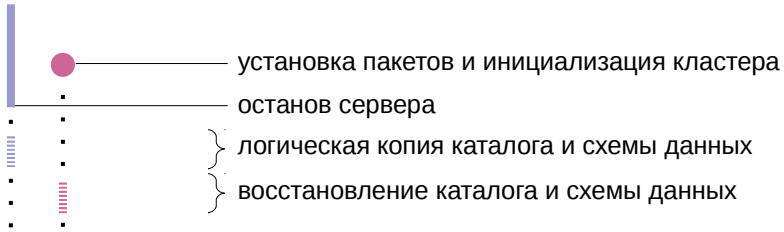
Чуть более подробно рассмотрим, что происходит с файловыми системами старого и нового кластера при обновлении.

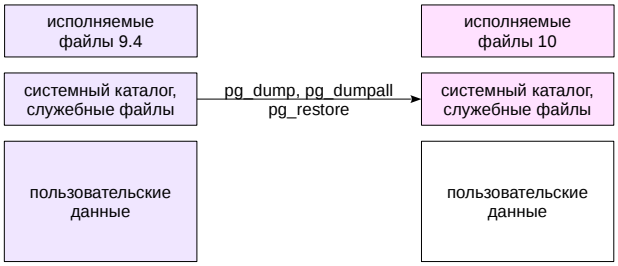
Шаг 1. Установлены пакеты с новой версией и инициализирован кластер. Новый сервер выключен и не содержит никаких пользовательских объектов, только стандартные базы данных и системный каталог.

Старый сервер продолжает работать.

Важный момент: новый кластер должен быть инициализирован с той же локалью, что и старый.

pg_upgrade



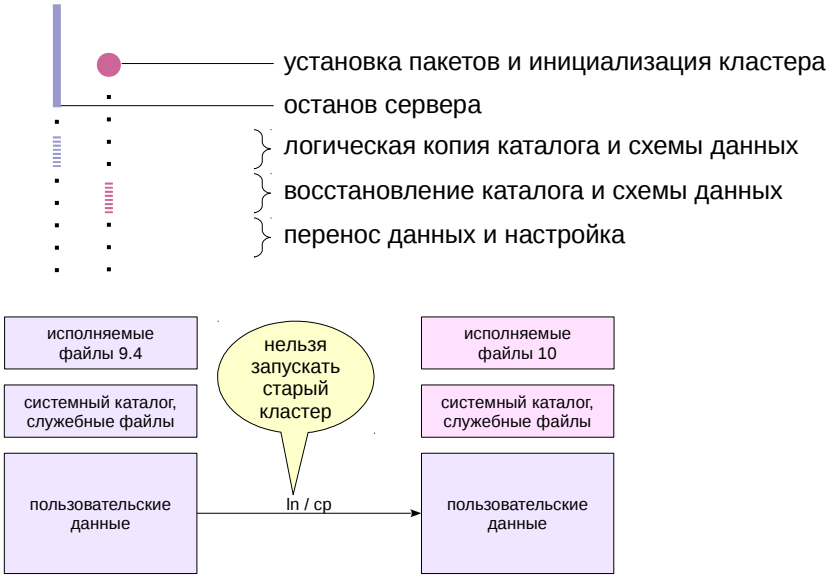


Шаг 2. Старый сервер останавливается, запускается утилита pg_upgrade. Она выполняет резервную копию каталога и схем данных всех баз и восстанавливает ее на новом сервере. Фактически для этого используются обычные утилиты pg_dump, pg_dumpall и pg_restore, которые рассматривались выше в этой теме (и подробно освещены в курсе DBA3).

После этого системный каталог нового кластера соответствует системному каталогу старого кластера с точностью до изменения формата данных (именно поэтому для него и используется логическая резервная копия);

Фактически, в новом кластере созданы все базы данных и все объекты внутри них, но системный каталог ссылается на еще несуществующие файлы данных.

pg_upgrade



Шаг 3. Утилита pg_upgrade выполняет перенос пользовательских данных.

Теперь файлы данных нового кластера либо скопированы со старого, или на них проставлены жесткие ссылки.

Заметим, что если файлы данных копируются, то старый кластер никак не затрагивается обновлением и можно в любой момент вернуться к нему, если при обновлении что-то пошло не так.

Но при использовании жестких ссылок файлы данных фактически становятся общими — точнее, одноименные файлы двух кластеров ссылаются на однаковые индексные дескрипторы файловой системы (inode). Поэтому, как только новый кластер будет запущен первый раз, старый уже невозможно запустить безопасным образом. Такой режим обновления требует наличия другого «плана Б».

pg_upgrade

Режим переноса пользовательских данных

по умолчанию — копирование файлов

обычно используются жесткие ссылки
`pg_upgrade --link`

Проверка обновления

```
pg_upgrade --check [--link]
```

Настройки доступа

на время обновления к обоим кластерам должен быть открыт суперпользовательский доступ

в процессе обновления оба кластера периодически запускаются на порту 50432 (или `$PGPORTOLD`, `$PGPORTNEW`)

Обычно при обновлении используются жесткие ссылки, поскольку это выполняется очень быстро (время обновления, по сути, не зависит от объема данных) и не требует дополнительного места на диске. Чтобы утилита работала в таком режиме, ей передается ключ --link.

Мы уже говорили о том, что обновление требуется проверять.
У pg_upgrade есть ключ --check, который выполняет предварительные проверки, но не собственно обновление — причем это можно сделать, не останавливая старый сервер. На самом деле проверка будет проведена еще раз при обновлении, но лучше заранее убедиться, что все готово.

Заметим, что если при обновлении будут использоваться жесткие ссылки, то вместе с ключом --check надо указывать и --link.

Еще один момент, на который следует обратить внимание — настройка доступа. Утилита pg_upgrade в процессе своей работы запускает и останавливает серверы, и для этого нужно, чтобы к обоим кластерам у нее был локальный суперпользовательский доступ (возможно, для этого потребуется временно изменить файл pg_hba.conf). Серверы поднимаются на порту 50432, чтобы не допустить случайного подключения к ним пользователей; при необходимости номер порта можно указать явно.

pg_upgrade

Подробности внутренней работы

- копирование общих объектов старого кластера:
  `pg_dumpall --globals-only --binary-upgrade`
- копирование схем данных каждой базы старого кластера:
  `pg_dump --schema-only --binary-upgrade`
- заморозка всех транзакций нового кластера
- копирование статуса транзакций (XACT)
- копирование мультитранзакций
- установка значения счетчика транзакций нового кластера
- восстановление общих объектов и схем данных: `pg_restore`
- перенос пользовательских данных
- установка значения OID нового кластера: `pg_resetwal -o`

На самом деле утилита pg_upgrade выполняет много довольно тонких операций. Знать о них необязательно, но полезно в случае возникновения нештатной ситуации.

Для создания резервной копии pg_upgrade запускает утилиту pg_dump в специальном режиме --binary-upgrade, который сохраняет нумерацию файлов (этот режим не нужен для обычной работы и официально не поддерживается).

Далее утилита pg_upgrade:

- замораживает все транзакции нового кластера (чтобы не зависеть от номеров транзакций, которые там выполнялись);

- копирует статусы транзакций (XACT);

- копирует информацию о мультитранзакциях;

- счетчик транзакций устанавливается в значение со старого кластера.

Затем в новом кластере восстанавливаются из резервной копии глобальные объекты и схемы данных каждой базы. Это не занимает много времени, поскольку объекты создаются, но не заполняются данными.

После переноса пользовательских данных остается выставить счетчик OID в значение со старого кластера (это выполняется утилитой pg_resetwal).

Физическая репликация

Реплики нужно создать заново

- нельзя не обновлять (нет двоичной совместимости)
- но использовать pg_upgrade тоже нельзя (недетерминированность)

Вариант 1 (pg_basebackup)

- развернуть новые реплики из физической резервной копии, сделанной после обновления основного сервера
- надежно, но долго

Вариант 2 (rsync)

- воспользоваться тем, что файлы данных не изменяются
- применимо только для Unix и только для жестких ссылок
- быстро, но сложно

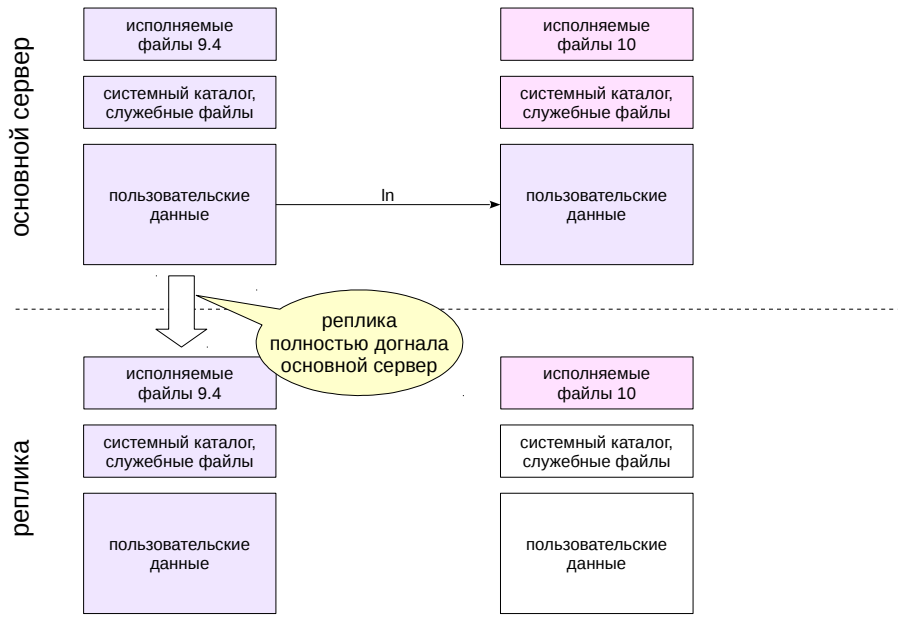
Отдельные трудности возникают при использовании физической репликации. Дело в том, что и мастер, и реплики необходимо обновлять одновременно. Соединение мастера одной версии с репликой другой версии скорее всего приведет к потере данных.

Еще одна тонкость: реплику нельзя обновлять с помощью pg_upgrade. Такое обновление не дает 100% гарантии того, что два идентичных экземпляра после обновления также получатся идентичными.

Обычный способ состоит в том, чтобы забыть про существующие реплики и создать их заново. Для этого надо выполнить базовую резервную копию мастера и развернуть из нее новую реплику (это подробно рассматривается в курсе DBA3). Понятно, что это достаточно длительная процедура.

Если речь идет об операционной системе семейства Unix, и при обновлении мастера использовался режим жестких ссылок, то время развертывания новой реплики можно существенно ускорить, получив примерно такой же выигрыш по времени, который дает pg_upgrade для мастера. Для этого надо правильным образом воспользоваться утилитой rsync.

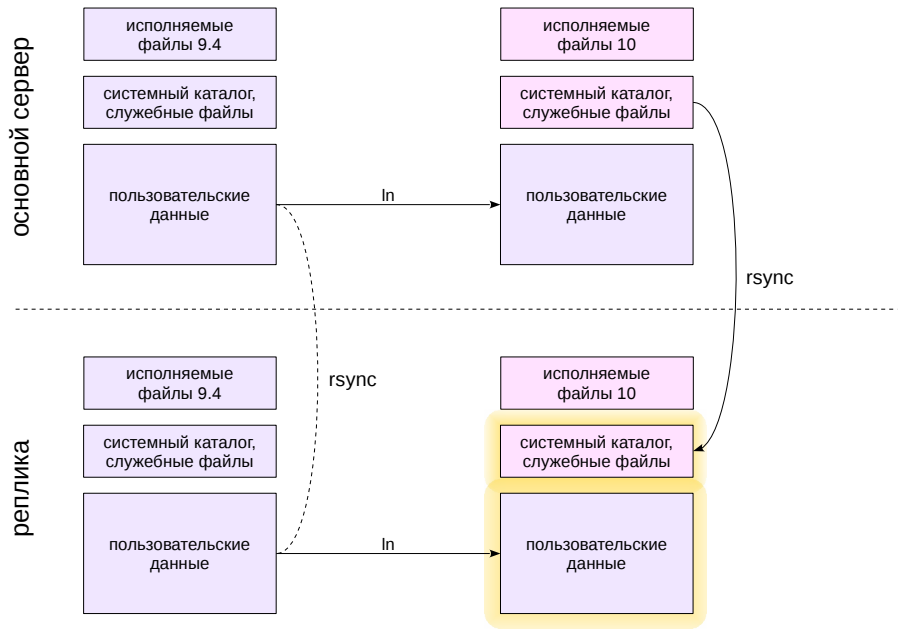
Физическая репликация



Исходная позиция: мастер обновлен, но новый сервер еще не запускался. Реплика выключена, причем перед выключением она полностью догнала мастер (таким образом файлы данных совпадают).

На реплике устанавливается новая версия сервера, но кластер не инициализируется.

Физическая репликация



Дальше запускается rsync так, чтобы в одной команде скопировать с мастера *оба кластера,* старый и новый, на реплику.

В таком режиме rsync понимает, что:

- старый и новый кластеры разделяют общие файлы данных;

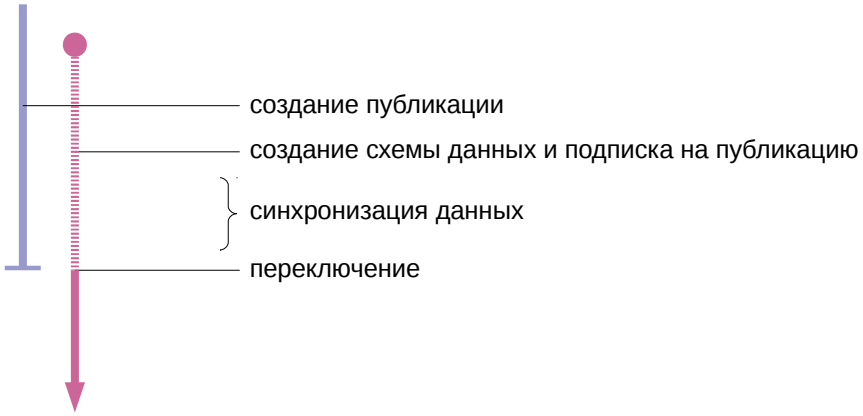
- файлы данных старого кластера на мастере и на реплике совпадают (чтобы не зависеть от объема, сравниваются только размеры файлов, а не их содержимое);

- следовательно, файлы данных не надо физически копировать на реплику, а достаточно проставить жесткие ссылки.

И rsync остается скопировать только файлы, относящиеся к системному каталогу, а они имеют небольшой размер.

Этот процесс описан в документации к pg_upgrade, в том числе и конкретные команды, которые необходимо выполнить. К сожалению, документация не объясняет, почему необходимы именно такие команды. Для того, чтобы разобраться в этом вопросе детально, полезно ознакомиться с перепиской разработчиков.

Логическая репликация



\+ возможно обновление без или с минимальным прерыванием обслуживания

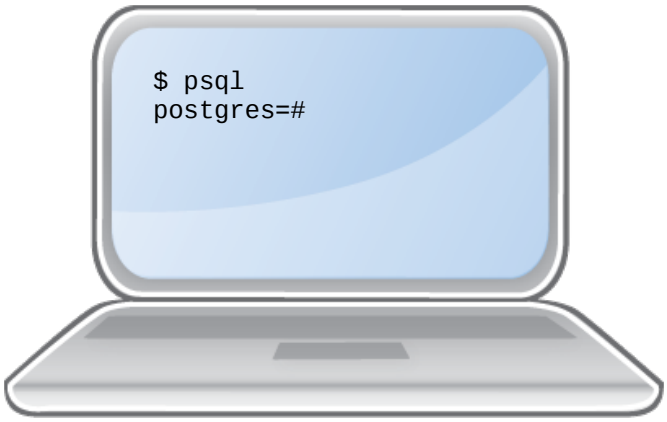
– сложная настройка, не реплицируются схема данных и команды DDL, требуется дополнительное место на диске и т. д.

Можно выполнить обновление основной версии без (или с минимальным) прерывания обслуживания, если использовать логическую репликацию.

Для этого требуется установить и настроить необходимым образом новый экземпляр. На старом сервере создаются публикации всех изменений во всех базах данных. На новом сервере создаются подписки на эти публикации с синхронизацией данных. После того, как данные синхронизированы, остается переключить пользователей на новый сервер, удалить подписку и остановить старый сервер.

Штатная логическая репликация появилась в PostgreSQL начиная с версии 10. К сожалению, это решение пока далеко от идеала: не реплицируются схема данных, команды DDL, команда TRUNCATE, данные последовательностей. Все это делает такой подход более сложным и менее привлекательным, чем использование pg_upgrade.

Демонстрация



Итоги

Обновление на дополнительный выпуск — простая замена исполняемых файлов

- физическая репликация, чтобы не прерывать обслуживание

Обновление основной версии:

- резервная копия (pg_dumpall)
- утилита обновления (pg_upgrade)
- логическая репликация
- наличие физической репликации усложняет процедуру
- сразу после — сделать физическую резервную копию

Практика

1. В кластере PostgreSQL 9.4 создайте пользователя с аутентификацией по паролю.
2. От имени нового пользователя создайте в базе данных таблицу со столбцом типа hstore.
3. Обновите кластер 9.4 на версию 10 с помощью логической резервной копии. При обновлении сделайте так, чтобы все данные оказались размещены в отдельном табличном пространстве.
4. Проверьте корректность обновления: доступность сервера, аутентификацию пользователя, содержимое таблицы.

1. PostgreSQL 9.4 уже установлен в /usr/local/pgsql-9.4/, кластер инициализирован. Необходимое окружение выставляется командой

```
9.4
```

При создании пользователя укажите пароль:

```
CREATE USER имя_пользователя PASSWORD 'пароль';
```

А в начало файла pg_hba.conf добавьте строку:

```
local all имя_пользователя md5
```

Тип данных hstore доступен в одноименном расширении.

2. PostgreSQL 10 уже установлен в /usr/local/pgsql-10/, кластер инициализирован. Необходимое окружение выставляется командой

```
10
```

Чтобы разместить данные в отдельном табличном пространстве, предварительно создайте его и отредактируйте файл с резервной копией (добавьте табличное пространство по умолчанию для БД).

3. Чтобы сохранить аутентификацию, внесите в файл pg_hba.conf те же изменения, что были сделаны в старом кластере.