Postgres PROFESSIONAL

Физическое резервирование

Темы

Холодное резервирование средствами ОС

Горячее резервирование (непрерывное архивирование)

Архивация журналов упреждающей записи

Базовая резервная копия

Процедура восстановления

Горячо или холодно?

	Холодное резервирование	↔	Горячее резервирование
Файловая система копируется при...	выключенном сервере	неаккуратно выключенном сервере	работающем сервере
Журналы упреждающей записи...	не нужны	нужны с последней контрольной точки	нужны за время копирования ФС

Холодное резервирование

Аккуратно выключенный сервер

- нужна только копия файловой системы
- время останова можно уменьшить: rsync, останов, снова rsync

Неаккуратно выключенный сервер

- потребуется восстановление согласованного состояния
  (в копию ФС будут входить и необходимые журналы)

Снимок файловой системы

- аналогично неаккуратно выключенному серверу
- все данные должны быть в пределах одного снимка

Восстановление

- разворачиваем копию
- запускаем сервер

Горячее резервирование

В горячую копию попадут несогласованные данные

Сколько журналов нужно для восстановления?

- от последней контрольной точки за все время копирования ФС
- имеет смысл сделать контрольную точку перед копированием
- либо параметр wal_keep_segments должен быть достаточно большим, либо в процессе копирования надо откладывать журналы «в сторону»

В журналах должна быть вся необходимая информация

- по умолчанию в журнал не попадают команды, изменяющие много данных (create table as select, create index, copy from и т. п.) — для них долговечность поддерживается с помощью fsync
- параметр wal_level должен быть archive или выше (minimal < archive < hot_standby < logical, требует рестарта)

pg_basebackup

`pg_basebackup --pgdata=`*`каталог`* `--xlog-method=`*`метод`*

Начало копирования: pg_start_backup()

выполняет контрольную точку

`--checkpoint=fast|spread` — выполнять быстро или медленно

на время работы включает параметр full_page_writes

Копирует файловую систему в указанный *каталог*

`--format=plain|tar` — файлы и каталоги или tar

Копирует в *каталог*/pg_xlog все нужные сегменты WAL

`--xlog-method=fetch` — в самом конце (wal_keep_segments)

`--xlog-method=stream` — в процессе (только для формата plain)

Конец копирования: pg_stop_backup()

Переключается на следующий сегмент WAL

pg_basebackup

Права для запуска

- суперпользователь
- роль с атрибутом REPLICATION

Дополнительная настройка

- в pg_hba.conf должно быть разрешение на подключение для базы данных replication — это специальный протокол для работы с кластером в целом, включая журналы предварительной записи
(разрешения для all не достаточно)
- параметр max_wal_senders должен быть равен как минимум 1
(а если используется метод stream — то как минимум 2)

Восстановление

- разворачиваем копию
- запускаем сервер

pg_basebackup

Табличные пространства

находятся в разных каталогах,
но тоже должны попасть в резервную копию

символьные ссылки в каталоге pg_tblspc должны быть корректными

формат tar

отдельные файлы *OID*.tar для пользовательских табличных пространств, base.tar содержит все остальное

формат plain

пользовательские табличные пространства копируются в те же абсолютные пути (предполагается вызов утилиты с другого сервера)

можно сопоставить табличным пространствам другие пути, и они будут использоваться при восстановлении из копии:

`--tablespace-mapping=`*`старый_путь=новый_путь`*

Непрерывная архивация

Процесс archiver

Параметры

- archive_mode = on
- archive_command — команда shell для копирования сегмента WAL в отдельное хранилище
- archive_timeout — максимальное время для переключения на новый сегмент WAL: pg_switch_xlog()

Алгоритм

- при заполнении сегмента WAL вызывается команда archive_command
- если команда завершается со статусом 0, сегмент удаляется (и создается пустой файл pg_xlog/archive_status/*N*.done)
- если команда возвращает не 0 (в частности, если команда не задана), сегмент остается до тех пор, пока попытка не будет успешной

Непрерывная архивация

Команда копирования

может копировать сегмент в любое удобное место

должна завершаться со статусом 0 только при успехе
(проверить: `command; echo $?`)

не стоит перезаписывать уже существующие файлы

включенный logging_collector запишет сообщения в журнал сервера

%p	полный путь к сегменту WAL
%f	имя файла для сегмента WAL (<> basename %p)
%%	символ %

примеры:

```
test ! -f /архив/%f && cp %p /архив/%f
gzip < %p > /архив/%f
my_backup_script.sh "%p" "%f"
```

Непрерывная архивация

Резервирование

настраиваем непрерывную архивацию

получаем базовую резервную копию с помощью pg_basebackup

```
pg_basebackup --pgdata=каталог
```

ключ `--xlog-method` не указываем
(так как сегменты WAL теперь архивируются отдельно)

`--label="`*`метка`*`"` — запишет название резервной копии в файл метки backup_label

Непрерывная архивация

Восстановление

- останавливаем сервер
- удаляем из $PGDATA все, кроме pg_xlog/
(там могли остаться еще необработанные сегменты)
- восстанавливаем файлы из базовой резервной копии
(кроме pg_xlog/ — старые сегменты не нужны)
- создаем управляющий файл $PGDATA/recovery.conf
- стартуем сервер
- ждем, пока recovery.conf переименуется в recovery.done
- если этого не произошло, ищем причину в журнале сервера

Непрерывная архивация

Файл метки backup_label

- всегда создается при создании резервной копии
- содержит название, время создания копии, начальный сегмент WAL

Управляющий файл recovery.conf

- пример файла в share/recovery.conf.sample (pg_config --sharedir)
- restore_command — команда, «обратная» команде копирования
- должна завершаться со статусом 0 только при успехе
- примеры:

```
cp /архив/%f %p
gunzip < /архив/%f > %p
```

- по умолчанию накатываются все сегменты WAL

Непрерывная архивация

Восстановление в определенной точке (PITR)

recovery_target = 'immediate'	только восстановить согласованность
recovery_target_name = '*имя*'	до именованной точки, созданной pg_create_restore_point('*имя*')
recovery_target_time = '*время*'	до указанного времени
recovery_target_xid = '*xid*'	до указанной транзакции
recovery_target_inclusive = on|off	включать ли указанную точку

Непрерывная архивация

Ветви времени

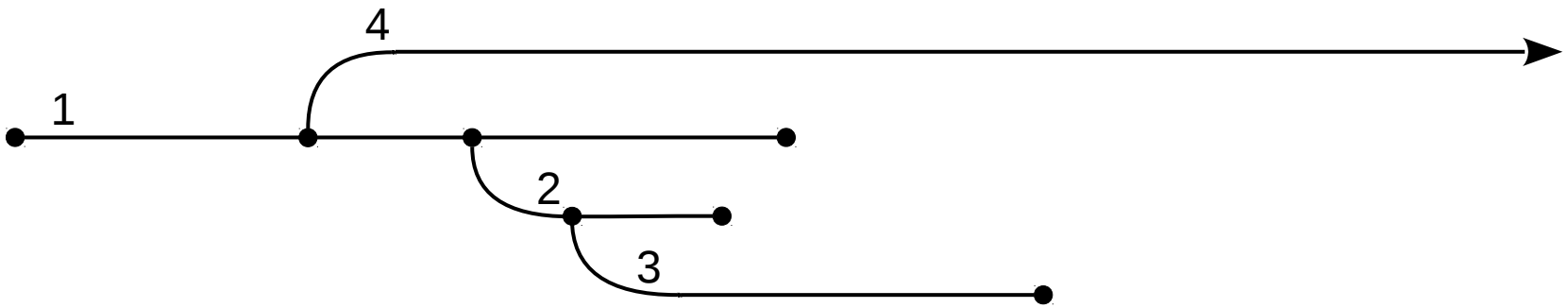
если восстановление происходит на какой-то момент в прошлом, надо отличать сегменты WAL «в прошлой жизни» и «в этой жизни»

при восстановлении создается новая ветвь времени
(файл pg_xlog/*N*.history — архивируется вместе с сегментами WAL)

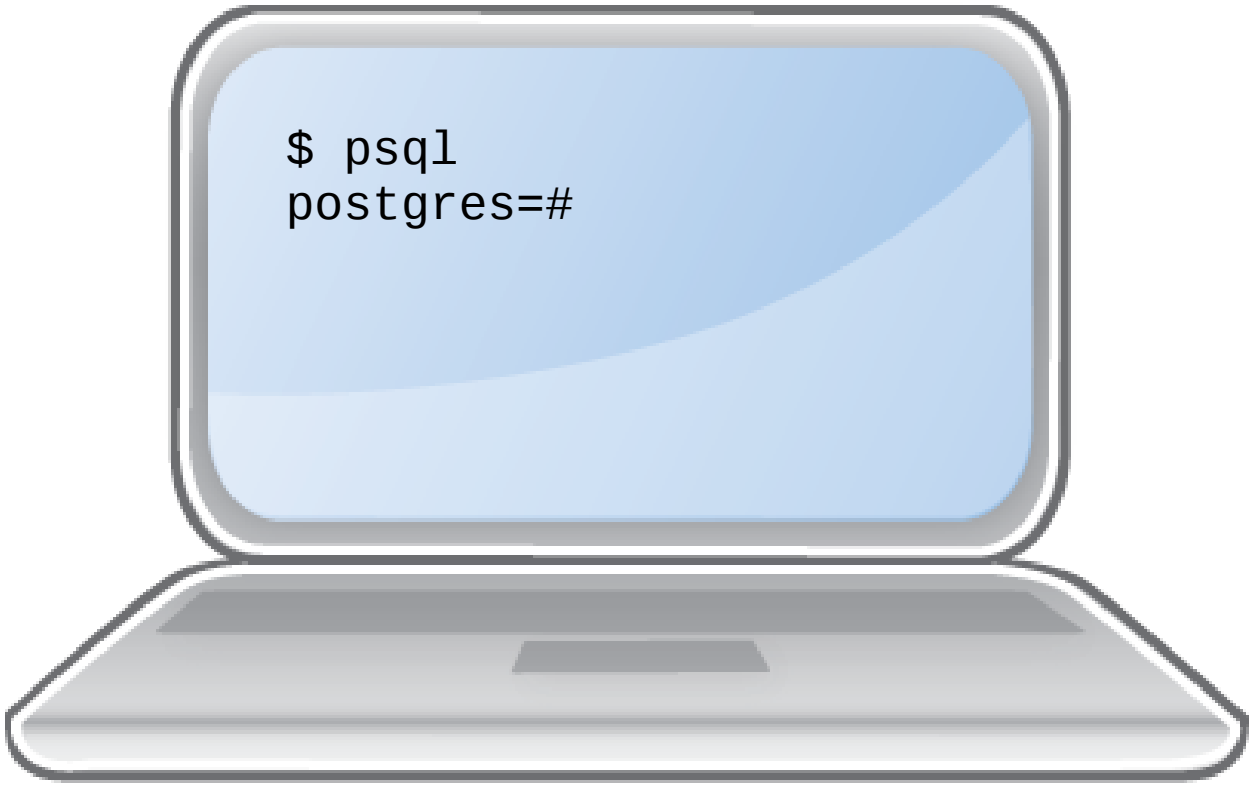
номер сегментов WAL включает номер ветви времени

восстановление:

recovery_target_timeline = 'latest'	по текущей (последней) ветви
recovery_target_timeline = '*номер*'	по указанной ветви



Демонстрация



Итоги

Резервная копия файловой системы может быть согласованной (холодной) и несогласованной (горячей).

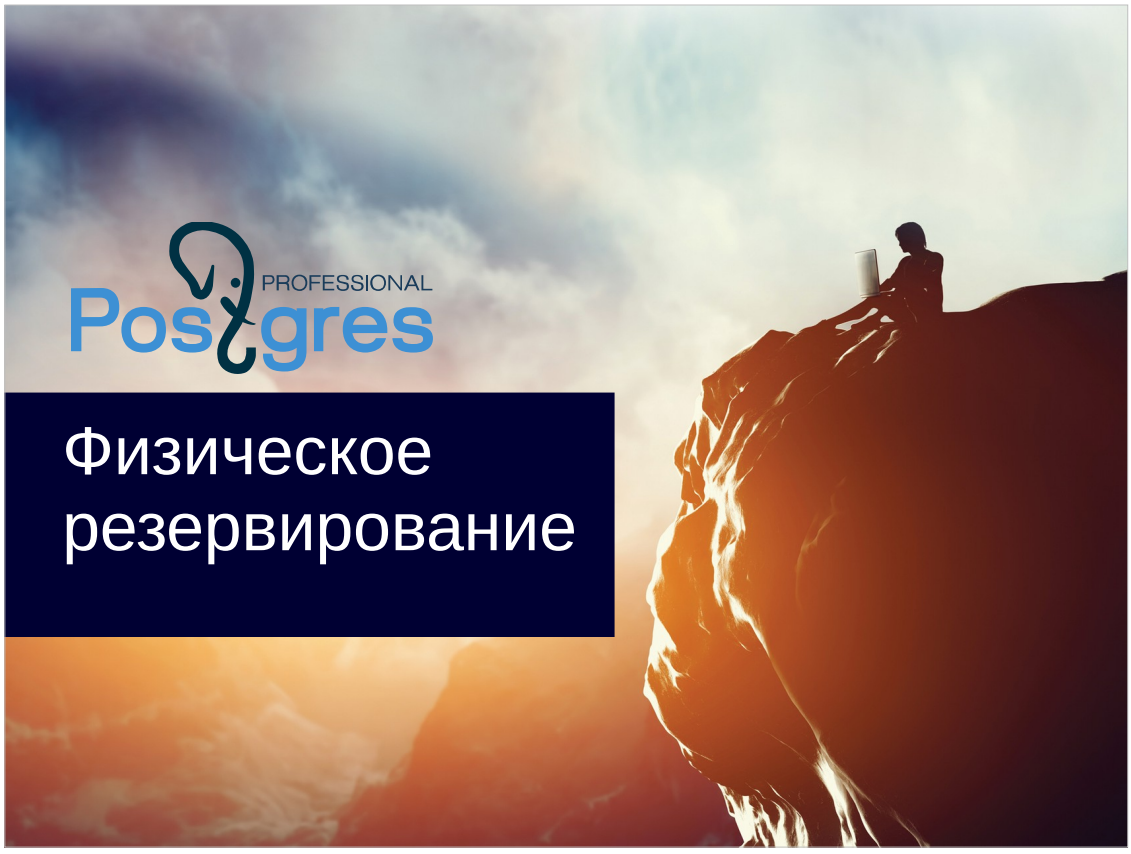
Горячая копия требует журналов упреждающей записи для восстановления согласованности.

Простой вариант: копия содержит необходимые журналы.

Если непрерывно архивировать журналы, можно восстановить систему на любой момент от создания базовой резервной копии.

Практика

1. Убедитесь, что в кластере нет пользовательских табличных пространств.
2. Настройте непрерывное архивирование журналов упреждающей записи в /home/postgres/archivedir/
3. Создайте базовую резервную копию.
4. В базе данных db17 создайте таблицу и вставьте в нее нескольких строк.
5. Создайте именованную точку восстановления.
6. Вставьте еще несколько строк в таблицу.
7. Остановите сервер и восстановите систему, указав созданную ранее точку восстановления.
8. Убедитесь, что таблица восстановлена правильно.



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

Темы

Холодное резервирование средствами ОС

Горячее резервирование (непрерывное архивирование)

Архивация журналов упреждающей записи

Базовая резервная копия

Процедура восстановления

Горячо или холодно?

	Холодное резервирование	↔	Горячее резервирование
Файловая система копируется при...	выключенном сервере	неаккуратно выключенном сервере	работающем сервере
Журналы упреждающей записи...	не нужны	нужны с последней контрольной точки	нужны за время копирования ФС

Физическое резервирование так или иначе предполагает создание копии файловой системы.

Если копия создается при выключенном сервере, она называется «холодной». Такая копия предполагает простое восстановление, но требует останова сервера.

Если копия создается при работающем сервере (что требует определенных действий — просто так копировать файлы нельзя), она называется «горячей». В этом случае процедура сложнее, но позволяет обойтись без останова.

При горячем резервировании копия ФС будет содержать несогласованные данные. Однако механизм восстановления после сбоев можно успешно применить и к восстановлению из резервной копии. Для этого потребуется определенное количество журналов упреждающей записи.

Холодное резервирование

Аккуратно выключенный сервер

- нужна только копия файловой системы
- время останова можно уменьшить: rsync, останов, снова rsync

Неаккуратно выключенный сервер

- потребуется восстановление согласованного состояния (в копию ФС будут входить и необходимые журналы)

Снимок файловой системы

- аналогично неаккуратно выключенному серверу
- все данные должны быть в пределах одного снимка

Восстановление

- разворачиваем копию
- запускаем сервер

Смысл холодного резервирования состоит в том, чтобы сделать копию файловой системы в тот момент, когда она содержит согласованные данные. Восстановление из такой резервной копии происходит просто: файлы разворачиваются и запускается сервер — и он сразу же готов к работе.

К сожалению, единственный вариант сделать такую копию — аккуратно (с выполнением контрольной точки) остановить сервер. Минус понятен: необходим простой сервера. К тому же время простоя может оказаться существенным при большом объеме данных.

Простой можно сократить за счет предварительного выполнения rsync (или аналогичного инструмента) при работающем севере. Тогда после останова сервера rsync докопирует только изменения (которых, предположительно, будет не много). Но простоя все равно не избежать.

Другой вариант — сделать копию несогласованных данных. Такая ситуация может возникнуть при неаккуратном отключении сервера или при создании снимка файловой системы (если ФС имеет такую возможность, и если все необходимые файлы попадают в один снимок).

В этом случае восстановление происходит аналогично, но при старте серверу потребуется выполнить восстановление согласованности. Это не проблема, так как необходимые журналы будут на месте, но может потребовать некоторого времени.

В любом случае, копию ФС нельзя сделать «просто так» при работающем сервере. Восстановиться из такой копии невозможно.

http://www.postgresql.org/docs/current/static/backup-file.html

Горячее резервирование

В горячую копию попадут несогласованные данные

Сколько журналов нужно для восстановления?

- от последней контрольной точки за все время копирования ФС
- имеет смысл сделать контрольную точку перед копированием
- либо параметр wal_keep_segments должен быть достаточно большим, либо в процессе копирования надо откладывать журналы «в сторону»

В журналах должна быть вся необходимая информация

- по умолчанию в журнал не попадают команды, изменяющие много данных (create table as select, create index, copy from и т. п.) — для них долговечность поддерживается с помощью fsync
- параметр wal_level должен быть archive или выше (minimal < archive < hot_standby < logical, требует рестарта)

Чем горячее резервирование принципиально отличается от холодного?

В горячую копию гарантированно попадут несогласованные данные. Но это возможно и при холодном копировании.

Однако при холодном копировании необходимые для восстановления журналы (с последней контрольной точки) гарантированно находятся в файловой системе. Для горячей копии это не так: журнальные файлы (сегменты) постепенно перезаписываются, и за время копирования ФС нужный журнал может будет потерян.

Поэтому надо либо увеличивать параметр wal_keep_segments, либо в процессе копирования «перехватывать» поток журналирования и сохранять эту информацию.

Вторая проблема состоит в том, что по умолчанию параметр wal_level = minimal, что подразумевает оптимизацию, связанную с журналом. Ряд команд, порождающий большой объем изменений (create table as select, create index, alter table set tablespace, copy from и др.), не журналируются. Вместо это данные сразу же записываются на диск (fsync), что само по себе гарантирует долговечность.

Поэтому параметр wal_level необходимо выставить как минимум в archive. Изменение этого параметра требует перезагрузки сервера.

http://www.postgresql.org/docs/current/static/runtime-config-wal.html

pg_basebackup

`pg_basebackup --pgdata=`*`каталог`* `--xlog-method=`*`метод`*

Начало копирования: pg_start_backup()

выполняет контрольную точку

`--checkpoint=fast|spread` выполнять быстро или медленно

на время работы включает параметр full_page_writes

Копирует файловую систему в указанный *каталог*

`--format=plain|tar` файлы и каталоги или tar

Копирует в *каталог*/pg_xlog все нужные сегменты WAL

`--xlog-method=fetch` в самом конце (wal_keep_segments)

`--xlog-method=stream` в процессе (только для формата plain)

Конец копирования: pg_stop_backup()

Переключается на следующий сегмент WAL

Для создания горячей резервной копии существует утилита pg_basebackup.

На самом деле ее возможности шире, но сейчас нас интересует запуск с ключом --xlog_method (и --pgdata, который в любом случае является обязательным).

В начале утилита выполняет контрольную точку (можно задать режим с помощью ключа --checkpoint) и включает параметр full_page_writes (запись в WAL полной страницы при первом ее изменении после контрольной точки).

Эти же начальные действия можно выполнить с помощью функции pg_start_backup(), если хочется большей гибкости, чем дает утилита.

Затем делается копия файловой системы (можно указать формат в ключе --format), а также копия всех необходимых сегментов WAL в каталог pg_xlog. WAL можно получать двумя способами (ключ --xlog-method): либо в самом конце (тогда надо установить параметр wal_keep_segments достаточно большим), либо непосредственно в процессе работы.

Создание копии завершается принудительным переключением на новый сегмент WAL. Это завершающее действие можно выполнить с помощью функции pg_stop_backup().

http://www.postgresql.org/docs/current/static/app-pgbasebackup.html

pg_basebackup

Права для запуска

- суперпользователь
- роль с атрибутом REPLICATION

Дополнительная настройка

- в pg_hba.conf должно быть разрешение на подключение для базы данных replication — это специальный протокол для работы с кластером в целом, включая журналы предварительной записи (разрешения для all не достаточно)
- параметр max_wal_senders должен быть равен как минимум 1 (а если используется метод stream — то как минимум 2)

Восстановление

- разворачиваем копию
- запускаем сервер

Утилита pg_basebackup запускается либо под суперпользователем, либо под ролью с атрибутом replication.

Поскольку утилита работает не с отдельными базами данных, а с кластером в целом (в том числе с журналами WAL, которые относятся ко всему кластеру), она подключается к серверу по специальному протоколу. Чтобы это было возможно, необходимо выполнить две настройки.

Во-первых, разрешить в pg_hba.conf подключение к базе данных replication. Это не название БД, а ключевое слово, обозначающее специальный протокол. Причем разрешения для all недостаточно, replication должен быть разрешен отдельно.

Во-вторых, надо увеличить параметр max_wal_senders (по умолчанию он равен нулю) минимум до единицы, а если для получения журналов используется метод stream — то минимум до 2.

pg_basebackup

Табличные пространства

находятся в разных каталогах,
но тоже должны попасть в резервную копию

символьные ссылки в каталоге pg_tblspc должны быть корректными

формат tar

отдельные файлы *OID*.tar для пользовательских табличных пространств, base.tar содержит все остальное

формат plain

пользовательские табличные пространства копируются в те же абсолютные пути (предполагается вызов утилиты с другого сервера)

можно сопоставить табличным пространствам другие пути, и они будут использоваться при восстановлении из копии:
`--tablespace-mapping=`*`старый_путь=новый_путь`*

Тонкий момент представляют пользовательские табличные пространства, так как они располагаются в отдельных каталогах.

При формате tar каждое такое ТП будет помещено в отдельный tar-файл с именем, равным OID табличного пространства.

Однако при формате plain пользовательские ТП будут копироваться в те же самые абсолютные пути. Если вызывать pg_basebackup локально, это приведет к ошибке directory "…" exists but is not empty. Предполагается, что утилита будет вызываться с другого сервера.

Можно сопоставить табличным пространствам другие пути с помощью ключа --tablespace-mapping. В таком случае именно новые пути будут использоваться при восстановлении из этой копии.

Непрерывная архивация

Процесс archiver

Параметры

- archive_mode = on
- archive_command — команда shell для копирования сегмента WAL в отдельное хранилище
- archive_timeout — максимальное время для переключения на новый сегмент WAL: pg_switch_xlog()

Алгоритм

- при заполнении сегмента WAL вызывается команда archive_command
- если команда завершается со статусом 0, сегмент удаляется (и создается пустой файл pg_xlog/archive_status/*N*.done)
- если команда возвращает не 0 (в частности, если команда не задана), сегмент остается до тех пор, пока попытка не будет успешной

Дальнейшее развитие идеи горячей резервной копии состоит в том, что раз у нас есть копия файловой системы и журналы упреждающей записи, то, добавляя журналы, мы можем восстановить систему не только на момент копирования ФС, но и вообще на произвольный момент. Главное, чтобы в резервную копию входили все необходимые журналы.

Для этого требуется механизм, позволяющий не переписывать сегменты WAL, а откладывать их куда-то в сторону. Такой механизм реализуется фоновым процессом archiver.

PostgreSQL позволяет определить для копирования произвольную команду shell в параметре archive_command. Сам механизм включается параметром achive_mode = on. Параметр archive_timeout позволяет указать максимальное время переключения на новый сегмент WAL — это позволяет при невысокой активности сервера сохранять тем не менее журналы не реже, чем хотелось бы (иными словами, потерять данные максимум за указанное время). Переключение на новый сегмент можно выполнить и вручную с помощью функции pg_switch_log().

Общий алгоритм таков. При заполнении очередного сегмента WAL вызывается функция копирования. Если она завершается с нулевым статусом, то сегмент может быть удален. Если же нет, то сегмент (и следующие за ним) не будет удален, а сервер будет периодически пытаться выполнить команду, пока не получит 0.

http://www.postgresql.org/docs/current/static/continuous-archiving.html

Непрерывная архивация

Команда копирования

может копировать сегмент в любое удобное место

должна завершаться со статусом 0 только при успехе
(проверить: `command; echo $?`)

не стоит перезаписывать уже существующие файлы

включенный logging_collector запишет сообщения в журнал сервера

%p	полный путь к сегменту WAL
%f	имя файла для сегмента WAL (<> basename %p)
%%	символ %

примеры:

```
test ! -f /архив/%f && cp %p /архив/%f
gzip < %p > /архив/%f
my_backup_script.sh "%p" "%f"
```


Команда архивирования может быть произвольной, лишь бы она завершалась с нулевым статусом только в случае успеха. (Статус команды можно проверить с помощью вывода echo $? после нее, но обычно в unix команды устроены именно таким образом.)

Команда не должна перезаписывать уже существующие файлы, так как это скорее всего означает какую-то ошибку и перезапись файла может погубить архив.

Удобно включить сбор сообщений с помощью logging_collector (см. тему «Мониторинг работы», так как в этом случае сообщения об ошибках будут попадать в журнал сервера.

В команде копирования используются символы подстановки %f (имя файла сегмента WAL) и %p (полный путь к этому файлу). Чтобы получить %, надо использовать удвоенный символ: %%.

На слайде приведены несколько примеров команд. Одна просто копирует файл в другой каталог (с проверкой существования). Другая выполняет сжатие файла.

Если подразумевается сложная логика, то ее удобно записать в скрипт и использовать имя скрипта в качестве команды копирования, что иллюстрирует третий пример.

Непрерывная архивация

Резервирование

настраиваем непрерывную архивацию

получаем базовую резервную копию с помощью pg_basebackup

```
pg_basebackup --pgdata=каталог
```

ключ `--xlog-method` не указываем
(так как сегменты WAL теперь архивируются отдельно)

`--label="`*`метка`*`"` — запишет название резервной копии в файл метки backup_label

Таким образом, резервирование состоит в настройке непрерывного архивирования и — после этого — получения базовой резервной копии с помощью pg_basebackup.

При создании резервной копии ключ --xlog-method не указывается, так как нет необходимости включать в копию сегменты WAL, ведь они архивируются отдельно. Все остальные ключи актуальны.

Кроме того, есть еще ключ --label, позволяющий присвоить копии произвольное имя, которое будет записано в специальный файл метки backup_label. Этот файл будет создан в любом случае.

Непрерывная архивация

Восстановление

- останавливаем сервер
- удаляем из $PGDATA все, кроме pg_xlog/
(там могли остаться еще необработанные сегменты)
- восстанавливаем файлы из базовой резервной копии
(кроме pg_xlog/ — старые сегменты не нужны)
- создаем управляющий файл $PGDATA/recovery.conf
- стартуем сервер
- ждем, пока recovery.conf переименуется в recovery.done
- если этого не произошло, ищем причину в журнале сервера

Восстановление происходит следующим образом.

Останавливаем сервер, если он работает.

Удаляем все из $PGDATA (а также очищаем каталоги табличных пространств), кроме каталога pg_xlog с сегментами WAL, так как там могли остаться еще не архивированные сегменты. Каталог pg_xlog имеет смысл скопировать отдельно на всякий случай.

Восстанавливаем файлы из базовой резервной копии, кроме каталога pg_xlog — если он и есть в резервной копии, то в нем находятся старые и уже не нужные сегменты WAL.

Создаем управляющий файл $PGDATA/recovery.conf (об этом подробнее дальше).

Теперь можно стартовать сервер. Если все было сделано правильно, то через некоторое время файл recovery.conf переименуется в recovery.done и сервер будет готов к работе.

Если что-то пойдет не так, сервер остановится (это будет видно, например, по отсутствию файла $PGDATA/postmaster.pid или по процессам ОС. В этом случае надо искать причину ошибки в журнале сервера, исправлять ее и снова запускать сервер.

Непрерывная архивация

Файл метки backup_label

- всегда создается при создании резервной копии
- содержит название, время создания копии, начальный сегмент WAL

Управляющий файл recovery.conf

- пример файла в share/recovery.conf.sample (pg_config --sharedir)
- restore_command — команда, «обратная» команде копирования
- должна завершаться со статусом 0 только при успехе
- примеры:

```
cp /архив/%f %p
gunzip < /архив/%f > %p
```

- по умолчанию накатываются все сегменты WAL

Процессом восстановления управляют два файла.

Во-первых, файл метки backup_label. Он всегда создается при создании базовой резервной копии и содержит название, время создания копии и — самое важное — название сегмента WAL, с которого надо начинать восстановление.

Во-вторых, файл recovery.conf. За образец можно взять файл share/recovery.conf.sample (точное название каталога подскажет утилита pg_config —sharedir). Этот файл совпадает по формату с любым другим конфигурационным файлом, например, postgresql.conf.

В минимальном варианте, в файле достаточно одного параметра restore_command, который определяет команду, «обратную» команде копирование archive_command. Она должна скопировать файл в обратном направлении, и тоже должна завершаться со статусом 0 только при успехе. Это очень важно, так как в процессе восстановления сервер может выполнять команду для файлов, которых не окажется в архиве — это не ошибка, но команда должна завершиться с ненулевым статусом.

В таком минимальном варианте базы данных будут восстановлены максимально близко к моменту сбоя.

Непрерывная архивация

Восстановление в определенной точке (PITR)

recovery_target = 'immediate'	только восстановить согласованность
recovery_target_name = '*имя*'	до именованной точки, созданной pg_create_restore_point('*имя*')
recovery_target_time = '*время*'	до указанного времени
recovery_target_xid = '*xid*'	до указанной транзакции
recovery_target_inclusive = on|off	включать ли указанную точку

Однако процесс восстановления можно остановить и в любой другой момент.

Параметр recovery_target = 'immediate' остановит восстановление, как только будет достигнута согласованность. Фактически, это сводит на нет преимущества непрерывного архивирования, но может пригодиться.

Параметр recovery_target_name позволяет указать именованную точку восстановления, созданную ранее с помощью функции pg_create_restore_point(). Это полезно, если заранее известно, что может потребоваться восстановление.

Параметр recovery_target_time позволяет указать произвольное время (timestamp), а параметр recovery_target_xid — произвольную транзакцию (восстановятся транзакции, завершенные к началу указанной). При этом с помощью параметра recovery_target_inclusive можно включить или исключить саму указанную точку.

Непрерывная архивация

Ветви времени

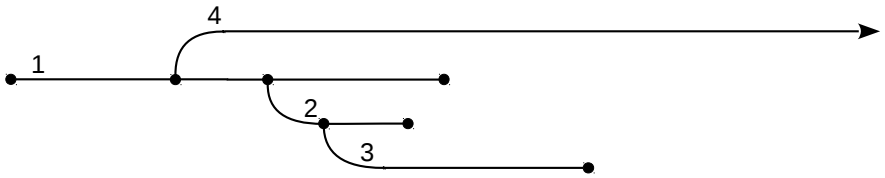
если восстановление происходит на какой-то момент в прошлом, надо отличать сегменты WAL «в прошлой жизни» и «в этой жизни»

при восстановлении создается новая ветвь времени (файл pg_xlog/*N*.history — архивируется вместе с сегментами WAL)

номер сегментов WAL включает номер ветви времени

восстановление:

recovery_target_timeline = 'latest'	по текущей (последней) ветви
recovery_target_timeline = '*номер*'	по указанной ветви

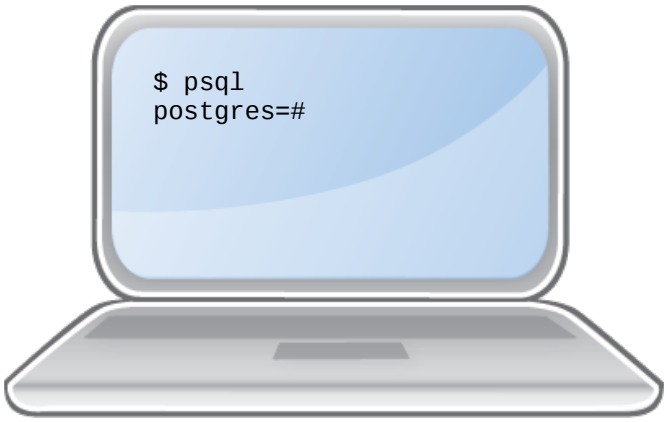


После восстановления в прошлом сервер будет генерировать новые сегменты WAL, которые будут пересекаться с сегментами «из прошлой жизни». Чтобы не потерять сегменты и, вместе с ними, возможность восстановления на другой момент, PostgreSQL вводит понятие ветви времени. После каждого восстановления номер ветви времени увеличивается, и этот номер является частью номера сегментов WAL.

По умолчанию восстановление происходит в текущей (последней) ветви времени. Выбрать другую ветвь можно с помощью параметра recovery_target_timeline.

На рисунке приведен пример ветвления. Первая ветвь имеет номер 1. В некоторый момент произошло восстановление в прошлом и началась ветвь 2. Вскоре и в ней произошло восстановление, что привело к появлению ветви 3. А после этого произошло восстановление назад к ветви 1, и текущая ветвь имеет номер 4. Чтобы такое было возможным, необходимо сохранить все сегменты WAL всех ветвей.

Демонстрация



Итоги

Резервная копия файловой системы может быть согласованной (холодной) и несогласованной (горячей).

Горячая копия требует журналов упреждающей записи для восстановления согласованности.

Простой вариант: копия содержит необходимые журналы.

Если непрерывно архивировать журналы, можно восстановить систему на любой момент от создания базовой резервной копии.

Практика

1. Убедитесь, что в кластере нет пользовательских табличных пространств.
2. Настройте непрерывное архивирование журналов упреждающей записи в /home/postgres/archivedir/
3. Создайте базовую резервную копию.
4. В базе данных db17 создайте таблицу и вставьте в нее нескольких строк.
5. Создайте именованную точку восстановления.
6. Вставьте еще несколько строк в таблицу.
7. Остановите сервер и восстановите систему, указав созданную ранее точку восстановления.
8. Убедитесь, что таблица восстановлена правильно.