Работа в сети

PostgreSQL

Роман Гордиенко
Старший инженер-программист в Factory5

Роман Гордиенко

О спикере:

- старший инженер-программист
- занимаюсь разработкой программных продуктов с 2008 года
- опыт работы с БД с 2007

Вспоминаем прошлое занятие

Вопрос: какая основная цель мониторинга состояния БД?

Вспоминаем прошлое занятие

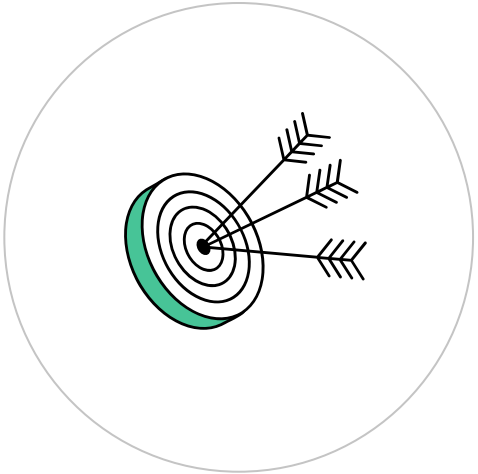
Вопрос: какая основная цель мониторинга состояния БД?

Ответ:

- следить за работой БД, чтобы гарантировать её рабочее состояние
- получать оповещения при возникновении системной ошибки и при избыточном использовании ресурсов

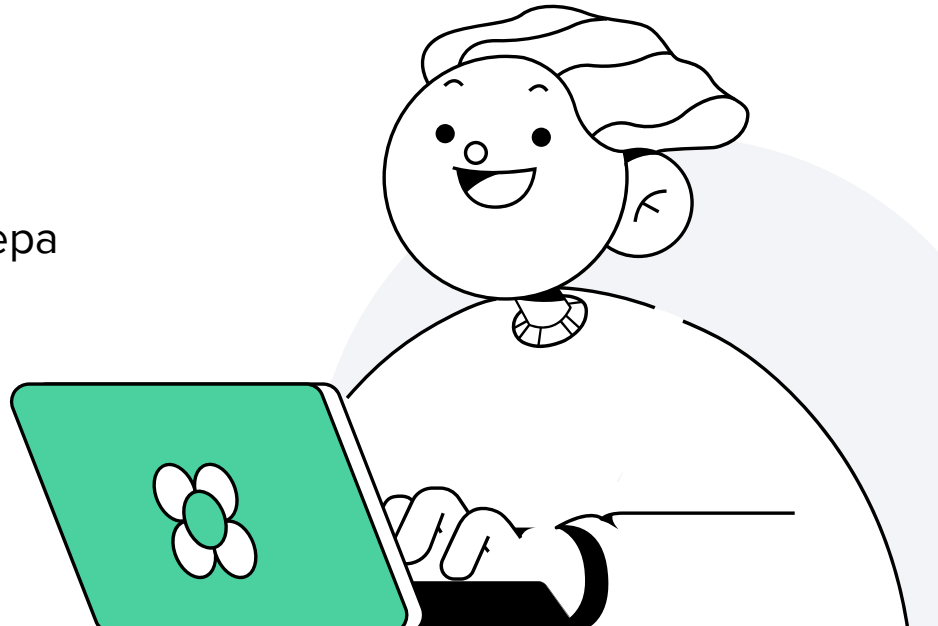
Цели занятия

- Вспомнить архитектуру PostgreSQL
- Разобраться в работе с балансировщиком нагрузки PgBouncer
- Узнать, как настраивать доступ и безопасность PostgreSQL
- Научиться настраивать отказоустойчивый кластер



План занятия

1. Введение
2. Архитектура PostgreSQL
3. Балансировщик нагрузки PgBouncer
4. Настройка доступа при работе в сети
5. Построение отказоустойчивого кластера
6. Итоги
7. Домашнее задание



Введение

1

Введение

Современные СУБД в основном используют подключения типа «клиент-сервер». PostgreSQL не исключение.

Поэтому важно понимать, **как правильно работает и конфигурируется эта СУБД**

Архитектура PostgreSQL

2

Архитектура PostgreSQL

При подключении к серверу клиент соединяется с процессом `postmaster`.

В задачи этого процесса входит:

- порождение других процессов
- присмотр за ними

Таким образом, postmaster порождает серверный процесс, и дальше клиент работает уже с ним

Архитектура PostgreSQL

На каждое соединение создаётся по серверному процессу, поэтому при большом числе соединений следует использовать пул (например, с помощью расширения `PgBouncer`).

Postmaster также порождает ряд служебных процессов. Об основных этих процессах поговорим немного позже в этом разделе.

Дерево процессов можно увидеть с помощью команды `ps fax`

Архитектура PostgreSQL

У экземпляра СУБД есть общая для всех серверных процессов память. Большую её часть занимает буферный кеш (shared buffers), необходимый для ускорения работы с данными на диске

Архитектура PostgreSQL

Обращение к дискам происходит через операционную систему, которая тоже кеширует данные в оперативной памяти.

PostgreSQL полностью полагается на операционную систему и сам не управляет устройствами. Он считает, что вызов `fsync()` гарантирует попадание данных из памяти на диск

Архитектура PostgreSQL

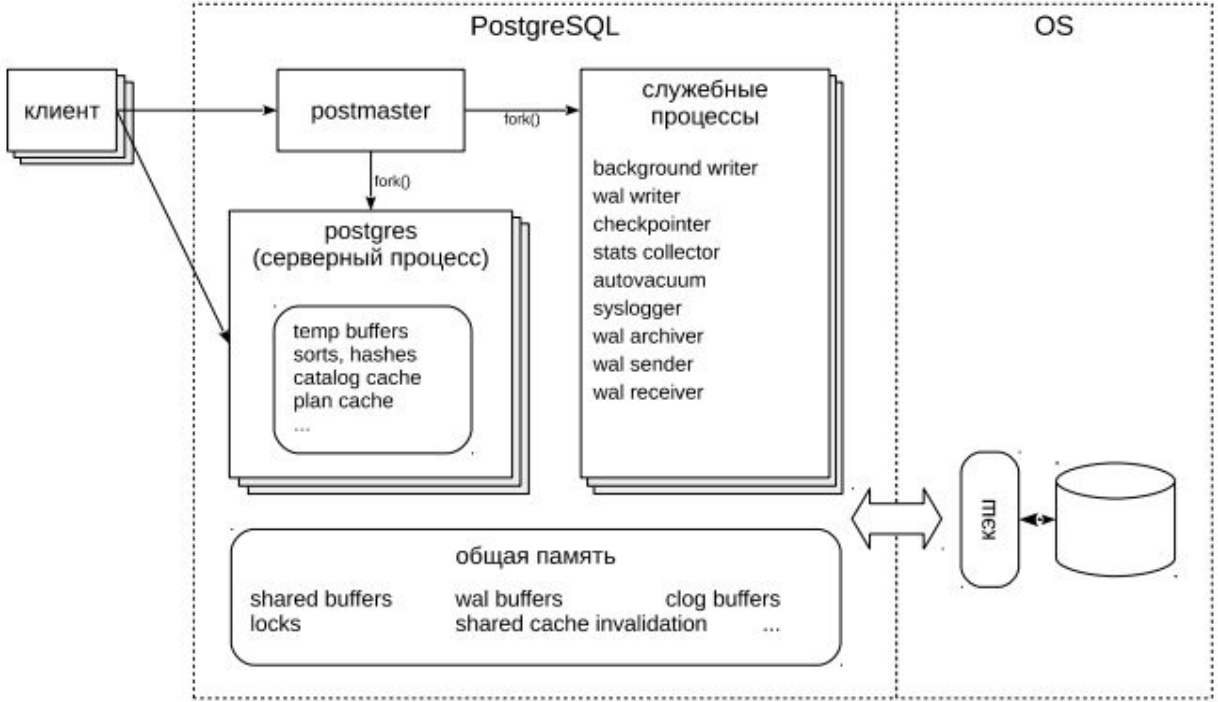
Кроме буферного кеша в общей памяти находится информация о блокировках и многое другое. Через неё же серверные процессы общаются друг с другом.

У каждого серверного процесса есть своя локальная память.

В ней находится:

- кеш каталога (часто используемая информация о базе данных)
- планы запросов
- рабочее пространство для выполнения запросов и другое

Архитектура изнутри



Источник

Балансировщик нагрузки PgBouncer

3

PgBouncer

PgBouncer — это лёгкий менеджер соединений для Postgres.

- PostgreSQL для каждого соединения создаёт новый процесс
- Чтобы «удешевить» соединение с БД, современные библиотеки используют пул соединений
- Библиотеки один раз соединяются с БД, а потом многократно используют это соединение. Если в библиотеке работы с БД нет возможности «пулить» соединения, то здесь приходит на помощь PgBouncer
- Вместо того, чтобы каждый раз создавать «дорогое» соединение с Postgres, создаётся лёгкое соединение с PgBouncer, который пользуется уже существующим соединением с БД

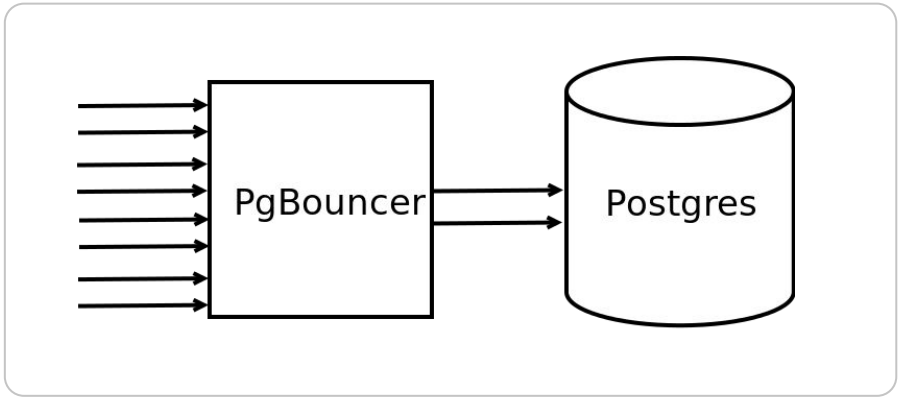
PgBouncer

- Сотни процессов PostgreSQL, запущенные на одной, пусть даже мощной машине, могут впустую тратить память и процессорное время. Тогда приходит на помощь PgBouncer
- Есть специальный режим `pool_mode=transaction`. В этом случае приложение будет думать, что оно держит настоящее соединение, но на самом деле оно будет держать его только на время одной транзакции
- Есть свои минусы: если меняются какие-то параметры внутри сессии (соединения), то приложению в следующий раз может достаться другое реальное соединение с БД, и это будет очень неожиданно

PgBouncer

Помимо экономии ресурсов, при использовании PgBouncer возможна незаметная перезагрузка или переключение БД. То есть можно настроить репликацию на новый сервер, дождаться, пока он синхронизируется со старым и переключить сервер в PgBouncer.

При этом для приложения соединения «повиснут» на некоторое время. То же самое при рестарте Postgres, если нужно поменять какой-то параметр в конфиге, который не подхватывается на лету



Источник

Настройка доступа при работе в сети

4

Настройка доступа возможна на 3 уровнях

1

На сетевом уровне

2

На транспортном уровне

3

При помощи конфигурационного файла pg_hba.conf

Безопасность на сетевом уровне. Firewall

В идеале ваш сервер PostgreSQL должен быть полностью изолированным и не допускать входящие подключения — SSH или psql.

К сожалению, PostgreSQL не поддерживает стандартную настройку такого типа

Безопасность на сетевом уровне. Firewall

Способ повысить безопасность сервера базы данных — заблокировать доступ к узлу, на котором работает база данных, на уровне порта с помощью брандмауэра

Безопасность на сетевом уровне. Firewall

По умолчанию PostgreSQL прослушивает TCP-порт 5432

Безопасность на сетевом уровне. Firewall

В зависимости от операционной системы существуют разные способы блокировки других портов. В Linux можно использовать `iptables` — наиболее доступную утилиту для управления файрволом

Прослушивание адресов

Хорошая практика — ограничение адресов, которые прослушивает сервер для клиентских подключений, с помощью параметра `listen_addresses` файла конфигурации

Прослушивание адресов

Если узел, на котором работает PostgreSQL, имеет несколько сетевых интерфейсов, используйте этот параметр, чтобы убедиться, что сервер прослушивает только те интерфейсы, через которые клиенты будут подключаться к нему:

```
listen_addresses = 'localhost, 192.168.0.1'
```

Прослушивание адресов

Если клиенты, подключающиеся к базе данных, всегда находятся на одном и том же узле (или совместно расположены в одном поде Kubernetes с PostgreSQL, работающем в качестве дополнительного контейнера), отключение прослушивания сокетов TCP может полностью исключить сеть из рассматриваемой картины

Прослушивание адресов

Запись пустой строки в параметр прослушиваемых адресов заставляет сервер принимать только соединения сокетов домена Unix:

```
listen_addresses = ''
```

Безопасность на транспортном уровне

Поскольку большая часть всемирной паутины переходит на HTTPS, найдётся мало оправданий тому, чтобы не использовать надёжное шифрование для соединений с базой данных.

PostgreSQL поддерживает TLS (который по-прежнему называется SSL в документации, конфигурации и CLI по legacy-причинам) и позволяет использовать его для аутентификации как сервера, так и клиента

Серверный TLS

Для аутентификации сервера сначала необходимо получить сертификат, который сервер будет предоставлять подключающимся клиентам.

Let's Encrypt упрощает получение бесплатных сертификатов X.509, например, с помощью инструмента `CLI certbot`:

```
certbot certonly --standalone -d postgres.example.com
```

По умолчанию certbot использует вызов `HTTP-01` ACME для проверки запроса сертификата, который требует действительного DNS для запрошенного домена, указывающего на узел и порт 80, который должен быть открыт

Серверный TLS

Если по какой-то причине вы не можете использовать Let's Encrypt и хотите сгенерировать все сертификаты локально, то можно использовать `openssl CLI`:

```
# Make a self-signed server CA.
openssl req -sha256 -new -x509 -days 365 -nodes \
    -out server-ca.crt \
    -keyout server-ca.key
```

Клиентский TLS

Аутентификация клиента по сертификату позволяет серверу проверить личность подключающегося, подтверждая, что сертификат X.509, представленный клиентом, подписан доверенным центром сертификации (CA).

Рекомендуется использовать разные центры сертификации для выдачи сертификатов клиенту и серверу. Поэтому давайте создадим клиентский CA и воспользуемся им для подписи сертификата клиента:

```
# Make a self-signed client CA.
openssl req -sha256 -new -x509 -days 365 -nodes \
    -out client-ca.crt \
    -keyout client-ca.key
```

Клиентский TLS

Последняя часть настройки — обновить файл host-based-аутентификации сервера PostgreSQL (`pg_hba.conf`), чтобы требовать TLS для всех подключений и аутентифицировать клиентов с помощью сертификатов X.509:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
hostssl all             all             ::/0                    cert
hostssl all             all             0.0.0.0/0               cert
```

Клиентский TLS

Теперь пользователи, подключающиеся к серверу, должны будут предоставить действительный сертификат, подписанный клиентским CA.

По умолчанию psql не будет выполнять проверку сертификата сервера. Поэтому для параметра sslmode необходимо установить значение `verify-full` или `verify-ca` — в зависимости от того, подключаетесь ли вы к серверу PostgreSQL, используя то же имя хоста, которое закодировано в поле CN его сертификата X.509

Клиентский TLS

Чтобы уменьшить размер команд и не вводить пути к сертификатам TLS каждый раз, когда вы хотите подключиться к базе данных, вы можете использовать файл службы подключения PostgreSQL.

Он позволяет группировать параметры подключения в «службы», на которые затем можно ссылаться в строке подключения через параметр «служба».

После этого при подключении к базе данных вам нужно будет только указать имя службы и имя базы данных, к которой вы хотите подключиться:

```
psql "service=example dbname=postgres"
```

Pg_hba.conf

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

HBA расшифровывается как `host-based authentication` — аутентификации по имени узла

Pg_hba.conf

Файл `pg_hba.conf` со стандартным содержимым создаётся командой `initdb` при инициализации каталога с данными. Однако его можно разместить в любом другом месте

Pg_hba.conf

- Формат файла `pg_hba.conf` представляет собой набор записей, по одной в строке
- Пустые строки игнорируются, как и любой текст комментария после знака #
- Записи не продолжаются на следующей строке
- Записи состоят из некоторого количества полей, разделённых между собой пробелом и/или tabs

Pg_hba.conf

В полях могут быть использованы пробелы, если они взяты в кавычки.

Если в кавычки берётся какое-либо зарезервированное 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

Pg_hba.conf

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

Pg_hba.conf

Первая запись с соответствующим типом соединения, адресом клиента, указанной базой данных и именем пользователя применяется для аутентификации

Pg_hba.conf

Процедуры fall-through или backup не предусмотрены: если выбрана запись и аутентификация не прошла, последующие записи не рассматриваются

Построение отказоустойчивого кластера

5

Практика построения кластера

На сегодняшний день процедура реализации отказоустойчивости в PostgreSQL — одна из самых простых и интуитивно понятных.

Чтобы её реализовать, необходимо определиться со сценариями отказоустойчивости — это залог успешной работы кластера — и протестировать его работу

Практика построения кластера

Настраивается репликация, чаще всего асинхронная, и в случае отказа текущего мастера другая нода (standby) становится текущим «мастером», другие ноды standby начинают следовать за новым мастером

Практика построения кластера

На сегодняшний день `repmgr` поддерживает сценарий автоматического failover (отказоустойчивость) — autofailover. Это позволяет поддерживать кластер в рабочем состоянии после выхода из строя ноды-мастера без мгновенного вмешательства сотрудника, что немаловажно, так как не происходит большого падения uptime

Практика построения кластера

В связи с развитием внутренних сервисов появилась необходимость реализовать систему хранения БД на PostgreSQL + репликацию + балансировку + failover

Практика построения кластера

1. Необходимо прописать названия всех нод в `/etc/hosts`, например так:

```
10.1.1.195 — pghost195
10.1.1.196 — pghost196
10.1.1.197 — pghost197
```

Практика построения кластера

2 Необходимо установить PostgreSQL и pgbouncer haproxy repmgr на все ноды

Практика построения кластера

3 Отключаем автозапуск PostgreSQL при старте системы — всеми процессами будет управлять пользователь postgres. Также это необходимо для того, чтобы бы не было ситуаций, когда у нас может оказаться две мастер-ноды после восстановления одной после сбоя питания

Практика построения кластера

4 Настраиваем SSH-соединение без пароля — между всеми нодами (делаем на всех серверах). Настроим подключения между всеми серверами и к самому себе через пользователя postgres (через пользователя postgres подключается также repmgr)

Практика построения кластера

Настройка сервера в качестве мастера (pghost195):

etc/postgresql/15.2/main/postgresql.conf

```
listen_addresses='*'
wal_level = 'hot_standby'
archive_mode = on
wal_log_hints = on
wal_keep_segments = 0
archive_command = 'cd .'
max_replication_slots = 5 # максимальное количество standby-нод, подключённых к мастеру.
hot_standby = on
shared_preload_libraries = 'repmgr_funcs, pg_stat_statements' ####подключаемая библиотека
repmgr и статистики postgres
max_connections = 800
max_wal_senders = 10#максимальное количество одновременных подключений от резервных
серверов или клиентов потокового резервного копирования
port = 5433
pg_stat_statements.max = 10000
pg_stat_statements.track = all
```

Практика построения кластера

Настройка слейвов (standby) — pghost196, pghost197. Конфигурируем repmgr на slave1 (pghost196), nano /etc/repmgr.conf:

```
cluster=test
node=2
node_name=node2
use_replication_slots=1
conninfo='host=pghost196 port=5433  user=repmgr dbname=repmgr'
pg_bindir=/usr/lib/postgresql/15.2/bin
```

Практика построения кластера

Регистрируем сервер как standby:

```
su postgres
cd ~/15.2/
rm -rf main/*
repmgr -h pghost1 -p 5433 -U repmgr -d repmgr -D main -f /etc/repmgr.conf
--copy-external-config-files=pgdata --verbose standby clone
pg_ctl -D /var/lib/postgresql/15.2/main
--log=/var/log/postgresql/postgres_screen.log start
```

Практика построения кластера

Регистрируем сервер в кластере:

```
su postgres
repmgr -f /etc/repmgr.conf standby register; repmgr -f /etc/repmgr.conf cluster
show
Просмотр состояния кластера
repmgr -f /etc/repmgr.conf cluster show
Видим

<spoiler title="">
<source lang="bash">
Role      | Name  | Upstream | Connection String
----------+-------|----------|----------------------------------------------
* master  | node195 |          | host=pghost195 port=5433  user=repmgr
dbname=repmgr
  standby | node196 | node1    | host=pghost196 port=5433  user=repmgr
dbname=repmgr
```

Практика построения кластера

Настраиваем автоматический failover. Для этого необходимо добавить новые секции в файл /etc/repmgr.conf на standby-серверах. На мастере этого быть не должно:

```
#######АВТОМАТИЧЕСКИЙ FAILOVER#######ТОЛЬКО НА STANDBY##################
master_response_timeout=20
reconnect_attempts=5
reconnect_interval=5
failover=automatic
promote_command='sh /etc/postgresql/failover_promote.sh'
follow_command='sh /etc/postgresql/failover_follow.sh'
#loglevel=NOTICE
#logfacility=STDERR
#logfile='/var/log/postgresql/repmgr-15.2.log'
priority=90 # a value of zero or less prevents the node being promoted to
master
```

Практика построения кластера

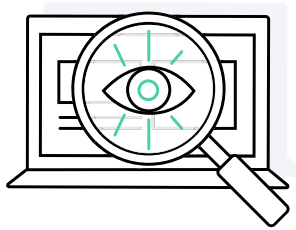
Все настройки autofailover идентичны, разница только в priority.

Если priority 0, то этот standby никогда не станет master.

Данный параметр будет определять очерёдность срабатывания failover. Меньшее число говорит о большем приоритете, значит, после отказа master-сервера его функции на себя возьмёт pghost197

Демонстрация работы

Построение отказоустойчивого кластера



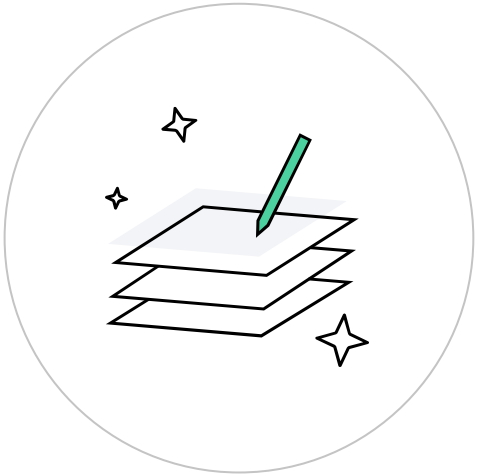
Сегодня мы:

1. Вспомнили архитектуру PostgreSQL
2. Познакомились с балансировщиком нагрузки PgBouncer
3. Узнали, как настраивать доступ и безопасность PostgreSQL
4. Настроили отказоустойчивый кластер

Домашнее задание

Давайте посмотрим ваше домашнее задание

1. Вопросы по домашней работе задавайте в чате группы
2. Задачи можно сдавать по частям
3. Зачёт по домашней работе ставят после того, как приняты все задачи



Задавайте вопросы и пишите отзыв о лекции

Роман Гордиенко
Старший инженер-программист в Factory5