



7

HAProxy и PostgreSQL



HAProxy

- Реплика при promote не разрывает сессии
- Статус кластера PostgreSQL:

```
show transaction_read_only;  
on -> off  
select pg_is_in_recovery();  
true -> false
```

- HAProxy:
 - › проверяет статус экземпляров PostgreSQL и направляет сессии на подходящие экземпляры PostgreSQL
 - › разрывает сессии с репликой, ставшей мастером
 - › состоит из:
 - исполняемого файла: /opt/tantor/usr/sbin/haproxy
 - файла службы: /lib/systemd/system/haproxy-tantor.service
 - Конфигурационного файла: /opt/tantor/etc/haproxy/haproxy.cfg
 - › опция HAProxy pgsql-check не проверяет ни роль, ни пароль:

```
option pgsql-check user ИМЯ
```



HAProxy

Patroni может выполнять автоматический failover. Прежний экземпляр мастера останавливается и запускается в режиме чтения. При остановке экземпляра, сессии клиентов разрываются:

```
FATAL: terminating connection due to administrator command  
The connection to the server was lost.
```

Новый мастер не останавливается, он открывается на запись:

```
show transaction_read_only;  
on -> off  
select pg_is_in_recovery();  
true -> false
```

Нужно:

- 1) разорвать сессии с репликой, ставшей мастером
- 2) после switchover/failover направлять новые сессии на читающие или пишущий экземпляры.

Patroni и PostgreSQL это не делают и для этого можно использовать HAProxy. pgbouncer не имеет функционала распознавания роли мастер/реплика и перенаправления клиентов на экземпляры с нужной ролью мастера или реплики. Для этого приходится использовать HAProxy.

HAProxy написан на языке C, является высокопроизводительным балансировщиком нагрузки. В отличие от pgbouncer, HAProxy не работает на уровне протокола PostgreSQL, не является пулером соединений (несколько клиентов используют одну сессию с экземпляром PostgreSQL), не вмешивается в аутентификацию пользователей PostgreSQL.

HAProxy может использоваться совместно с pgbouncer.

HAProxy работает на уровне TCP, HTTP, TLS. Проверка доступности PostgreSQL может выполняться опцией pgsql-check в HAProxy, но эта опция не может отличить соединение к реплике от соединения с мастером. Пример директивы:

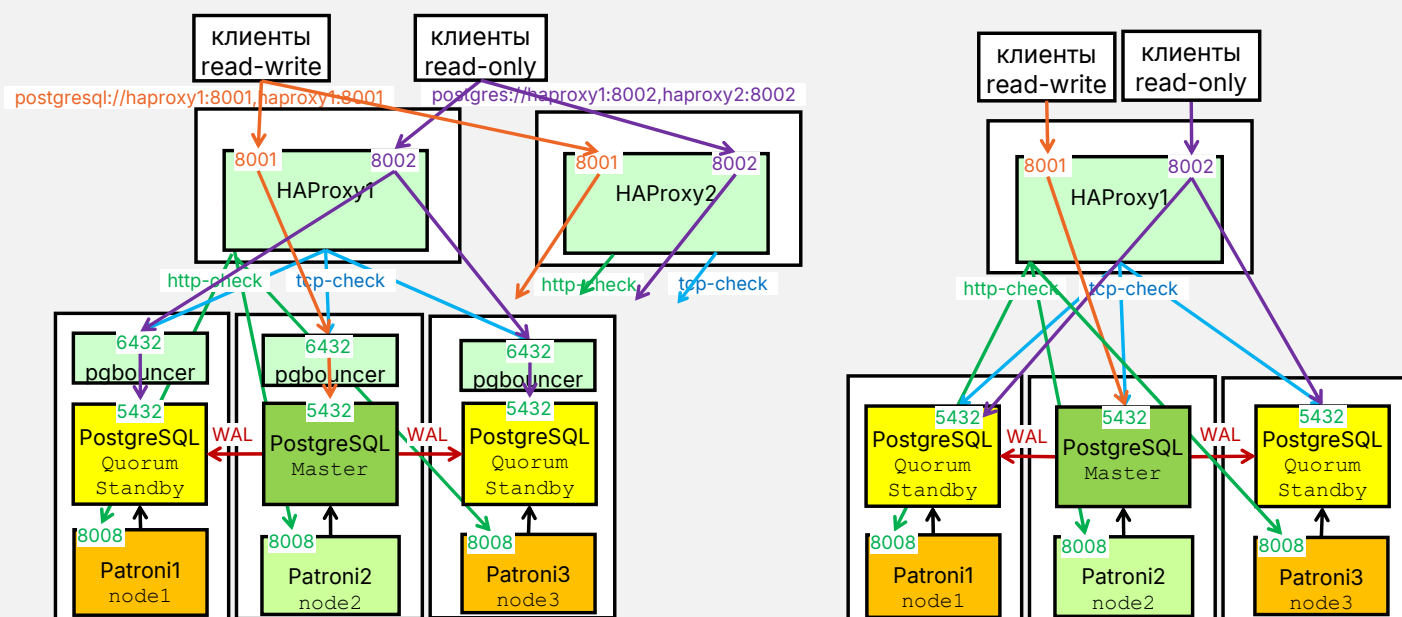
```
option pgsql-check user ИМЯ
```

Директива позволяет послать только пакет Startup и получить запрос на аутентификацию Authentication Request или ErrorResponse. Проверить роль сервера (мастер или реплика) или аутентифицироваться опция HAProxy pgsql-check не позволяет.

<https://www.haproxy.com/documentation/haproxy-configuration-tutorials/reliability/health-checks/#postgresql-health-checks>

Топология сетевых соединений с HAProxy

- HAProxy может выполнять проверки: `http-check` или `tcp-check`
- HAProxy может подсоединяться через `pgbouncer`



Топология сетевых соединений с HAProxy

HAProxy является точкой входа для клиентов. Клиентами могут быть сервера приложений, которые используют встроенные в них пулы соединений. Например, JDBC Connection Pools.

В HAProxy открывается порт или порты для клиентов, которые:

- 1) должны подсоединяться к мастеру и работать в режиме чтение-запись в базу данных;
- 2) должны подсоединяться к репликам и выполнять только читающие запросы. Перенос читающих запросов на реплики снимает нагрузку с мастера.

Порты для read-write и read-only разные.

HAProxy должен выполнять проверки, чтобы определить роль экземпляров PostgreSQL: мастер или реплика, на которые будет направлять (маршрутизировать) TCP-соединения.

Для проверок HAProxy может обращаться на порт Patroni REST API по протоколу HTTP или на порты экземпляров PostgreSQL. Проверки инициирует HAProxy, экземпляры не уведомляют HAProxy о смене своих ролей. Эти проверки должны повторяться через короткие интервалы времени, чтобы HAProxy мог быстро реагировать на смену роли.

Если выполнять проверки путём подсоединения к экземпляру, а не через Patroni REST API, то каждое создание сессии с экземпляром порождает серверный процесс, что трудоёмко и, при большой нагрузке на хост, серверный процесс может относительно долго запускаться. Для того, чтобы серверные процессы не порождались часто, можно использовать `pgbouncer`. `pgbouncer` имеет смысл запускать на том же хосте, что и экземпляр PostgreSQL.

Если планируется поставить Patroni на паузу, останавливать процессы Patroni, пересоздавать конфигурацию Patroni, при этом не останавливать экземпляры PostgreSQL или менять роли экземпляров, не останавливая обслуживание клиентов, то можно в HAProxy заменить проверки `http-check` на порт Patroni на прямые проверки `tcp-check` на порты экземпляров и перечитать конфигурацию HAProxy.

Конфигурация HAProxy

- Установка HAProxy:

```
dpkg -i haproxy-tantor-all_*.deb
```

- Проверить нет ли ошибок в конфигурационном файле:

```
/opt/tantor/usr/sbin/haproxy -c -f /opt/tantor/etc/haproxy/haproxy.cfg
```

- Интерактивный запуск с трассировкой в stderr:

```
/opt/tantor/usr/sbin/haproxy -f /opt/tantor/etc/haproxy/haproxy.cfg -d -dt -db
```

- Запуск службой:

```
systemctl start haproxy-tantor
```

- Путь к конфигурационному файлу:

```
Environment="CONFIG=/opt/tantor/etc/haproxy/haproxy.cfg" "PIDFILE=/run/haproxy-tantor.pid"
ExecStartPre=/opt/tantor/usr/sbin/haproxy -f $CONFIG -c -q $OPTIONS
ExecStart=/opt/tantor/usr/sbin/haproxy -Ws -f $CONFIG -p $PIDFILE $OPTIONS
ExecReload=/opt/tantor/usr/sbin/haproxy -f $CONFIG -c -q $OPTIONS
ExecReload=/bin/kill -USR2 $MAINPID
KillMode=mixed
SuccessExitStatus=143
Type=notify
```



Конфигурация HAProxy

Установка HAProxy:

```
dpkg -i haproxy-tantor-all_*.deb
```

Исполняемый файл: /opt/tantor/usr/sbin/haproxy

Файл службы: /lib/systemd/system/haproxy-tantor.service

Конфигурационный файл: /opt/tantor/etc/haproxy/haproxy.cfg

Документация HAProxy: <https://docs.haproxy.org/3.4/configuration.html>

Проверить нет ли ошибок в конфигурационном файле:

```
/opt/tantor/usr/sbin/haproxy -c -f /opt/tantor/etc/haproxy/haproxy.cfg
```

Интерактивный запуск с трассировкой в stderr:

```
/opt/tantor/usr/sbin/haproxy -f /opt/tantor/etc/haproxy/haproxy.cfg -dt -db
```

Запуск службой:

```
systemctl start haproxy-tantor
```

Путь к конфигурационному файлу:

```
cat /lib/systemd/system/haproxy-tantor.service | grep PID
```

```
Environment="CONFIG=/opt/tantor/etc/haproxy/haproxy.cfg"
```

```
"PIDFILE=/run/haproxy-tantor.pid"
```

```
ExecStartPre=/opt/tantor/usr/sbin/haproxy -f $CONFIG -c -q $OPTIONS
```

```
ExecStart=/opt/tantor/usr/sbin/haproxy -Ws -f $CONFIG -p $PIDFILE $OPTIONS
```

```
ExecReload=/opt/tantor/usr/sbin/haproxy -f $CONFIG -c -q $OPTIONS
```

```
ExecReload=/bin/kill -USR2 $MAINPID
```

```
KillMode=mixed
```

```
SuccessExitStatus=143
```

```
Type=notify
```

Параметр CFGDIR можно удалить из файла службы.

Параметр **-Ws** это запуск haproxy в режиме master-worker с уведомлением systemd о том, что haproxy готов обслуживать клиентов. Процесс master читает файл конфигурации и порождает (fork) процессы worker. Этот режим может использоваться и с foreground и с daemon режимами. В этом режиме можно поменять конфигурацию, перечитать её, прежние сессии дорабатывают на прежнем worker, а новые сессии направляются на новый worker. Максимальное время жизни старого worker можно ограничить параметром **hard-stop-after 30m**, но при работе с PostgreSQL, конфигурация меняется и перечитывается нечасто. Изменение конфигурации может делаться перед переводом Patroni в режим паузы.

Файл конфигурации HAProxy

- Секции **глобальных** параметров и параметров **по умолчанию**:

global

```
#log 127.0.0.1:514 local0 # отправлять логи на UDP порт 127.0.0.1, где слушает rsyslog
log stderr len 1024 format iso local0 err
chroot /opt/tantor/var/lib/haproxy # повышает безопасность, меняет корневой каталог
#pidfile /var/run/haproxy-tantor.pid # указан в файле службы параметром -p
maxconn 500 # общее число соединений с HAProxy, аналог параметра -n
# spread-checks 1 # сдвигать интервалы проверок в пределах 1%, полезно если сотни server
user haproxy # запускается под root, если нужно занять нижние порты и меняет пользователя
group haproxy
#daemon # запускать в фоновом режиме, а не интерактивно, аналог параметра -D.отключить: -db
stats socket /opt/tantor/var/lib/haproxy/stats mode 660 # Включает UNIX-сокеты для
управления. Через сокет можно динамически менять конфигурацию
defaults # конфигурация по умолчанию для секций frontend, backend, listen
mode tcp # для протокола PostgreSQL используется режим tcp
balance leastconn # направлять на сервер, где меньше соединений. по умолчанию : random
log global # использовать параметры директивы log из секции global
option tcplog # формат лог-записей для протокола TCP
#option log-health-checks # логировать проверки tcp-check, не нужно устанавливать
option dontlognull # не засорять лог пустыми запросами
```



Файл конфигурации HAProxy

У HAProxy больше тысячи параметров конфигурации, конфигурирование сложно, так как нужно знание работы протоколов на низком уровне.

HAProxy конфигурируется текстовым файлом с упорядоченным набором команд (ordered sequence of statements). Также можно использовать переменные окружения и ссылаться на них в файле конфигурации.

Разделители в строках файла - пробелы или символы табуляции.

Комментарии начинаются с символа #. Пустые строки и отступы не важны, однако, выравнивание (отступы) стараются использовать для удобства чтения файла.

В секции global указываются общие параметры, влияющие на процесс HAProxy.

Секция defaults сбрасывает все параметры в значения по умолчанию (имеет смысл, если секция defaults присутствует после секций frontend, backend, listen) и устанавливает значения, которые будут действовать дальше.

```
log куда [len <length>] [format <format>] [sample <ranges>:<sample_size>]
[profile <prof>] <facility> [<level> [<minlevel>]]
```

куда может быть: stdout, stderr, ip:UDPпорт, устройство, UNIX сокет, буфер и другие. Имя реального файла файловой системы указать нельзя, так как запись в него будет тормозить работу HAProxy.

len по умолчанию 1024. Диапазон от 80 до 65535.

format по умолчанию local. Может быть: iso, raw и другие.

facility одна из 24 syslog facilities: kern user mail daemon auth syslog lpr news uucp cron auth2 ftp ntp audit alert cron2 local0..local7. facility игнорируется для format raw и short, но указание facility обязательно и с этими format, обычно, указывают daemon.

level для фильтрации сообщений. По умолчанию посылаются все сообщения и фильтра нет. Можно указать в качестве фильтра: emerg alert crit err warning notice info debug, чтобы уменьшить число сообщений, если их не планируется читать.

Для отладки можно запускать HAProxy интерактивно с параметрами -dt (трассировка в stderr, с номерами строк в файлах *.c исходного кода) -d (отладка) -db (отключить параметр daemon, запускать интерактивно):

```
haproxy -f /opt/tantor/etc/haproxy/haproxy.cfg -d -dt -db
```

Таймауты HAProxy

- Устанавливаются в секции `defaults`:

defaults

```
retries 2 # если сервер не ответил, повторить ещё 2 раза перед выдачей ошибки клиенту
option redispatch # направлять запрос на другой сервер после исчерпания retries
timeout queue 1m # длительность ожидания клиента, перед выдачей ему ошибки
timeout connect 1s # таймаут соединения HAProxy к серверам
timeout client 3h # время неактивности клиента
timeout server 3h # время неактивности со стороны сервера
timeout check 2s # таймаут на проверку
maxconn 100 # для каждого фронтэнда или бэкэнда перед постановкой запросов в очередь
```

- параметры PostgreSQL, которые инициируют проверки:

`client_connection_check_interval`,
`tcp_keepalives_interval`

> по умолчанию, 0 (не установлены) и действует таймаут linux:

```
root@tantor:~# sysctl -a | grep keepalive_time
net.ipv4.tcp_keepalive_time = 7200
```



Файл конфигурации HAProxy

Единицы измерения: `us` - микросекунды (1/1000000 секунды), `ms` - миллисекунды (1/1000 секунды), `s`, `m`, `h`, `d`.

Параметры:

```
timeout client 3h # время неактивности клиента
timeout server 3h # время неактивности со стороны сервера
```

стоит установить в значения, превышающие значения параметров PostgreSQL:

`idle_in_transaction_session_timeout`, `idle_session_timeout`, `statement_timeout`, `transaction_timeout`. Для определения длительности запросов можно использовать параметр `log_min_duration_statement`.

Либо установить значения параметрам, которые инициирует посылку сетевого пакета со стороны сервера или клиента: `client_connection_check_interval`, `tcp_keepalives_count`, `tcp_keepalives_idle`, `tcp_keepalives_interval`. Параметр `tcp_user_timeout` переопределяет таймаут подтверждения пакетов для обнаружение зависших TCP-соединений и имеет приоритет над параметрами:

`net.ipv4.tcp_syn_retries=6`, `net.ipv4.tcp_keepalive_probes=9`, `net.ipv4.tcp_retries2=15`. Пример, как выглядит проблема зависших TCP-соединений со стороны сервера приложений до балансировщика нагрузки (HAProxy или pgbouncer), если между ними теряются (drop) пакеты: <https://tech.instacart.com/the-vanishing-thread-and-postgresql-tcp-connection-parameters-93afc0e1208c>

По умолчанию, параметры PostgreSQL установлены в нули и таймаут берётся из параметра linux: `net.ipv4.tcp_keepalive_time`, который, обычно, установлен в **два часа**:

```
root@tantor:~# sysctl -a | grep keepalive_time
net.ipv4.tcp_keepalive_time = 7200
```

Независимо от того, какая сторона послала пакет, другая сторона должна ответить и пакеты пройдут в обе стороны так, чтобы соединение не было разорвано по отсутствию пакетов со стороны клиента или сервера: `timeout client` и `timeout server`.

Таймауты серверов

- Можно установить в секции `defaults` одной строкой:

`defaults`

```
default-server inter 2s fastinter 50ms downinter 10s fall 3 rise 2 init-state down on-marked-down shutdown-sessions
```

> или в других секциях, перед директивами `server`

- Описание параметров:

`inter 2s` частота выполнения проверок, по умолчанию `2000ms`
`fastinter 50ms` ускоренный интервал проверок, если сервер был доступен и выдал первую ошибку или был недоступен и выдал первый успешный ответ
`downinter 10s` интервал проверок после того, как сервер получил статус DOWN
`fall 3` число подряд не пройденных проверок для получения статуса DOWN
`rise 2` число успешных проверок подряд для возврата в строй (статус UP)
`init-state down` при запуске или перезапуске HAProxy, считать сервера DOWN, пока успешно не пройдёт ПЕРВАЯ попытка health-check
`on-marked-down shutdown-sessions` если сервер становится DOWN, HAProxy немедленно разрывает все текущие сессии клиентов с этим сервером. **Нужно при продвижении реплики в кластере PostgreSQL, чтобы клиенты переподключились на другой экземпляр**



Файл конфигурации HAProxy

Адреса серверов, к которым подсоединяет HAProxy и доступность которых проверяет, устанавливаются в директивах `server`:

```
server wtantor1 127.0.0.1:5432 maxconn 100 check addr 127.0.1.1 port 8008  
server r1 127.0.0.1:5432 check
```

Таймауты проверок можно установить в директиве `default-server` одной строкой:

`default-server inter 2s fastinter 50ms downinter 10s fall 3 rise 2 init-state down on-marked-down shutdown-sessions`

Её можно указать в секции `defaults` или в других секциях, перед директивами `server`.

Параметры, которые устанавливаются:

`inter 2s` частота выполнения проверок, по умолчанию `2000ms`
`fastinter 50ms` ускоренный интервал проверок, если сервер был доступен и выдал первую ошибку или был недоступен и выдал первый успешный ответ
`downinter 10s` интервал проверок после того, как сервер получил статус DOWN
`fall 3` число подряд не пройденных проверок для получения статуса DOWN
`rise 2` число успешных проверок подряд для возврата в строй (статус UP)
`init-state down` при запуске или перезапуске HAProxy, считать сервера DOWN, пока успешно не пройдёт ПЕРВАЯ попытка health-check
`on-marked-down shutdown-sessions` если сервер становится DOWN, HAProxy немедленно разрывает все текущие сессии клиентов с этим сервером. **Нужно при продвижении реплики до мастера, чтобы клиенты переподключились на другой экземпляр, так как при продвижении реплики до мастера сессии с ней не разрываются.**

Небольшое значение `fastinter 50ms` позволяет обнаруживать смену ролей мастер-реплика не больше, чем за 2,125 секунды: `inter + fall*fastinter`. Значение `50ms` подходит для сети с низкой сетевой задержкой между HAProxy<->PostgreSQL или HAProxy<->pgbouncer и отзывчивыми серверными процессами.

В пределах этого времени (2.125 секунд) сессии, которые были подключены к реплике, получают возможность вносить изменения в новом мастере, после чего разрываются.

Страница статистики HAProxy

- Параметры статуса через **Patroni REST API**:

```
listen stats
mode http
bind *:8000
stats enable
stats uri /
stats refresh 10s
stats show-legends
#stats realm Haproxy
#stats auth admin:ad
```

HAProxy version 3.4.0, released 2026/06/03

Statistics Report for pid 2657218

> General process information

pid = 2657218 (process #1, nbproc = 1, nbthread = 4)
uptime = 0d 0h 03m 24s; warnings = 18
system limits: memmax = unlimited; ulimit-n = 1074
maxsock = 1074; maxconn = 500; reached = 0; maxpipes = 0
current conns = 1; current pipes = 0/0; conn rate = 0/sec; bit rate = 0.815 kbps
Running tasks: 0/99 (0 nice); idle = 100 %

active UP backup UP
active UP, going down backup UP, going down
active DOWN, going up backup DOWN, going up
active or backup DOWN not checked
active or backup DOWN for maintenance (MAINT)
active or backup SOFT STOPPED for maintenance
Note: "NO LB/DRAIN" = UP with load-balancing disabled.

Display option:

- Scope:
- Hide DOWN servers
- Disable refresh
- Refresh now
- CSV export
- JSON export (schema)

External resources:

- [Primary site](#)
- [Updates \(v3.4\)](#)
- [Online manual](#)

stats																																	
	Queue			Session rate			Sessions					Bytes				Denied		Errors		Warnings		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwrtme	Thrtle			
Frontend				0	1	-	1	1	100	1			8 718	1 019 140	0	0	0					OPEN											
Backend	0	0		0	0		0	0	10	0	0	0s	8 718	1 019 140	0	0	0	0	0	0	0	3m25s UP		0/0	0	0			0				

primary																																	
	Queue			Session rate			Sessions					Bytes				Denied		Errors		Warnings		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwrtme	Thrtle			
Frontend				0	0	-	0	0	100	0			0	0	0	0	0					OPEN											
wfantor1	0	0	-	0	0		0	0	100	0	0	?	0	0	0	0	0	0	0	0	0	3m25s UP	L7OK/200 in 4ms	1/1	Y	-	0	0	0s	-			
wfantor2	0	0	-	0	0		0	0	100	0	0	?	0	0	0	0	0	0	0	0	0	3m25s DOWN	L7STS/503 in 2ms	1/1	Y	-	1	0	3m25s	-			
wfantor3	0	0	-	0	0		0	0	100	0	0	?	0	0	0	0	0	0	0	0	0	3m25s DOWN	L7STS/503 in 2ms	1/1	Y	-	1	0	3m25s	-			
Backend	0	0		0	0		0	0	10	0	0	?	0	0	0	0	0	0	0	0	0	3m25s UP		1/1	1	0			0s				

replica																																	
	Queue			Session rate			Sessions					Bytes				Denied		Errors		Warnings		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwrtme	Thrtle			
Frontend				0	0	-	0	0	100	0			0	0	0	0	0					OPEN											
rfantor1	0	0	-	0	0		0	0	100	0	0	?	0	0	0	0	0	0	0	0	0	3m25s DOWN	L7STS/503 in 1ms	1/1	Y	-	1	0	3m25s	-			
rfantor2	0	0	-	0	0		0	0	100	0	0	?	0	0	0	0	0	0	0	0	0	3m25s UP	L7OK/200 in 1ms	1/1	Y	-	0	0	0s	-			
rfantor3	0	0	-	0	0		0	0	100	0	0	?	0	0	0	0	0	0	0	0	0	3m25s UP	L7OK/200 in 1ms	1/1	Y	-	0	0	0s	-			
Backend	0	0		0	0		0	0	10	0	0	?	0	0	0	0	0	0	0	0	0	3m25s UP		2/2	2	0			0s				



Учебный курс Tantor DBA-18 "Администрирование PostgreSQL 18. Часть 2" Тантор Лабс © 2026

236

Страница статистики HAProxy

HAProxy может выдавать статистику в виде HTML-страницы. Параметры для выдачи страницы:

```
listen stats
mode http
bind *:8000
stats enable
stats uri /
stats refresh 10s
stats show-legends
#stats realm Haproxy\ stats #для окна аутентификации типа HTTP BASE
#stats auth admin:admin #логин и пароль (admin / admin) для авторизации на веб-
странице статистики. Можно указать несколько строк с разными именами и паролями.
```

Проверка роли экземпляра через Patroni REST

```
listen primary
bind *:8001
option httpchk OPTIONS /primary
http-check expect status 200
default-server inter 2s fastinter 50ms downinter 10s fall 3 rise 2 init-state down on-
marked-down shutdown-sessions
server wtantor1 127.0.0.1:5432 maxconn 100 check addr 127.0.1.1 port 8008
server wtantor2 127.0.0.1:5433 maxconn 100 check addr 127.0.1.2 port 8008
server wtantor3 127.0.0.1:5434 maxconn 100 check addr 127.0.1.3 port 8008
listen replica
bind *:8002
option httpchk OPTIONS /replica
http-check expect status 200
default-server inter 2s fastinter 50ms downinter 10s fall 3 rise 2 init-state down on-
marked-down shutdown-sessions
server rtantor1 127.0.0.1:5432 maxconn 100 check addr 127.0.1.1 port 8008
server rtantor2 127.0.0.1:5433 maxconn 100 check addr 127.0.1.2 port 8008
server rtantor3 127.0.0.1:5434 maxconn 100 check addr 127.0.1.3 port 8008
```

- пример ответов, которые получит HAProxy при проверках:

```
curl -i --http1.0 -H "User-Agent:" -H "Host:" -H "Accept:" -X OPTIONS http://127.0.1.1:8008/primary
HTTP/1.0 200 OK
curl -i --http1.0 -H "User-Agent:" -H "Host:" -H "Accept:" -X OPTIONS http://127.0.1.1:8008/replica
HTTP/1.0 503 Service Unavailable
```



Проверка роли экземпляра через Patroni REST

Если используется Patroni, то HAProxy может проверять роль экземпляров PostgreSQL через Patroni REST. В конфигурации HAProxy создаётся две секции `listen` для мастера и реплик. Параметром `bind` указывается по какому адресу клиенты будут подсоединяться к экземплярам через HAProxy.

Адрес экземпляра PostgreSQL указывается после имени сервера: `server wtantor1 127.0.0.1:5432`

В директивах `server` указывается адрес Patroni REST API, который обслуживает сервер: `check addr 127.0.1.1 port 8008`

Параметром `maxconn 100` можно ограничить число сессий, которые будет подсоединять HAProxy к конкретному серверу.

Для корректной работы нужно, чтобы были установлены параметры `init-state=down` и, как минимум, у реплик `on-marked-down=shutdown-sessions`:

```
default-server inter 2s fastinter 50ms downinter 10s fall 3 rise 2 init-state
down on-marked-down shutdown-sessions
```

Директиву можно указать в секции `defaults` или в других секциях, перед директивами `server`. По умолчанию, директива `option httpchk` использует HTTP/1.0, Patroni всегда отвечает в версии HTTP/1.0.

Метод HTTP `OPTIONS`, используемый вместо метода `GET` позволяет Patroni не передавать тело ответа в виде JSON.

URI `/primary` и `/replica` подходят для определения мастера и реплик. Также есть `/read-writ` `/read-only` `/standby-leader` `/leader` (primary или standby-leader) `/synchronous` `/replica?lag=10MB` / и другие. Описание: https://patroni.readthedocs.io/en/latest/rest_api.html

Проверить, какие ответы получит HAProxy можно командами:

```
curl -i --http1.0 -H "User-Agent:" -H "Host:" -H "Accept:" -X OPTIONS
http://127.0.1.1:8008/primary
```

```
HTTP/1.0 200 OK
```

```
curl -i --http1.0 -H "User-Agent:" -H "Host:" -H "Accept:" -X OPTIONS
http://127.0.1.1:8008/replica
```

```
HTTP/1.0 503 Service Unavailable
```

Директива `listen` более проста, чем связка из `frontend+backend`.

Пример директив файла `haproxy.cfg` есть в репозитории Patroni:

<https://github.com/patroni/patroni/blob/master/haproxy.cfg>

Проверка роли экземпляра по свойству сессии

- Проверка свойства `in_hot_standby` в секции `listen`:

```
listen master1
mode tcp
bind *:8001
option tcp-check
tcp-check connect linger
tcp-check send-binary
00000041000300007573657200706f7374677265730064617461626173650074656d706c61746531006c6f6
75f646973636f6e6e656374696f6e73006f666660000 #передаются слова, записанные в HEX-формате:
user00postgres00database00template100log_disconnections00off00
tcp-check expect binary 696e5f686f745f7374616e646279006f666600 # in_hot_standby
off=6f6666 on=6f6e
tcp-check send-binary 5800000004 # 'X' - завершение сессии, длина 4 байта
server r1 127.0.0.1:5432 check
server r2 127.0.0.1:5433 check
server r3 127.0.0.1:5434 check
listen replical
mode tcp
balance leastconn
bind *:8002
...здесь те же строки, что в секции master1
tcp-check expect binary 696e5f686f745f7374616e646279006f6e00 # off=6f6666 on=6f6e
...здесь те же строки, что в секции master1
```



Проверка роли экземпляра по свойству сессии

Можно проверять роль экземпляра без обращения к REST API Patroni, путём проверки значения параметра `in_hot_standby`, который возвращает экземпляр PostgreSQL в пакете TCP, сразу после аутентификации. В этом пакете PostgreSQL передаёт сообщения: **'R'** - аутентификация пройдена, 15 (в 18 версии) сообщений **'S'** со значениями параметров, среди которых `in_hot_standby`, **'Z'** - сообщение о готовности к приёму SQL-команд.

Аутентификация без пароля (`trust`). При аутентификации с паролем, нужно больше сетевых пакетов и выполнение расчёта контрольных сумм, которые HAProxy не может делать.

Проверки можно проводить непривилегированным пользователем. Для изменения имени пользователя нужно поменять **имя** пользователя и **число байт** в пакете, если длина имени изменится. В строке передаются слова, записанные в HEX-формате:

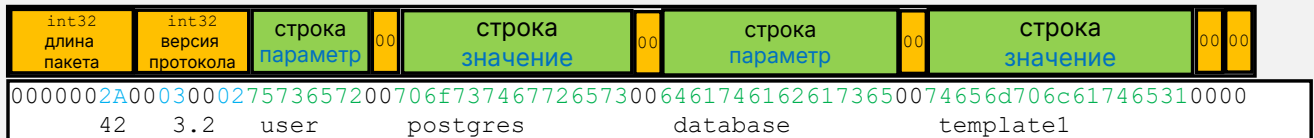
```
user00postgres00database00template100log_disconnections00off00
```

Пример конфигурации:

```
listen master1
mode tcp
bind *:8001
option tcp-check
#option log-health-checks # можно использовать при отладке
tcp-check connect linger
tcp-check send-binary
00000041000300007573657200706f7374677265730064617461626173650074656d706c617465
31006c6f675f646973636f6e6e656374696f6e73006f666660000
tcp-check expect binary 696e5f686f745f7374616e646279006f666600
tcp-check send-binary 5800000004 # 'X' - завершение сессии
server r1 127.0.0.1:5432 check
server r2 127.0.0.1:5433 check
server r3 127.0.0.1:5434 check
listen replical
mode tcp
bind *:8002
...здесь те же строки, что в секции master1
tcp-check expect binary 696e5f686f745f7374616e646279006f6e00
...здесь те же строки, что в секции master1
```

Сетевой протокол PostgreSQL

- Формат пакета Startup:



› в пакете Startup нет типа пакета (tag)

- Формат остальных пакетов:



- в одном TCP-frame может передаваться один или несколько пакетов (messages) сетевого протокола PostgreSQL
- tcp-check send-binary посылает полный пакет
- tcp-check expect binary проверяет наличие последовательности байт в ответе сервера



Сетевой протокол PostgreSQL

Первый байт пакета сетевого протокола PostgreSQL (кроме пакета Startup) это символ (обычно, буква латинского алфавита в верхнем регистре) в кодировке ASCII, обозначающий тип пакета (метку, tag). Затем идут 4 байта с длиной пакета, включая первый байт.

Пакет Startup посылается клиентом на порт процесса postmaster и инициирует создание сессии. В этом пакете первыми идут 4 байта длины, затем версия протокола. В PostgreSQL 18 используется версия 3.0 и 3.2, и принимается значение с версией 3.1. Затем идут строки в [кодировке клиента](#). Кодировка опционально может передаваться в том же пакете параметром `client_encoding` `UTF8`, в HEX-формате это `636c69656e745f655665636f64696e670055544638`, в котором принимает данные директива `NAProxy`:

```
tcp-check send-binary <hexstring> [comment <msg>]
```

Если кодировка клиента не передаётся, то используется кодировка сервера.

Обязательным в Startup пакет является только параметры: `user`. Если его не передать, то будет возвращён пакет с ошибкой:

```
FATAL 28000 no PostgreSQL user name specified in startup packet
```

Ни имя базы, ни кодировка не являются обязательными. Если не указать имя базы, то пользователь будет подключён к базе с его именем. Пример минимального Startup пакета:

```
tcp-check send-binary 0000001700003000007573657200706f7374677265730000
```

Если аутентификация не требуется (trust), то сервер возвращает пакеты, первый из которых типа 'R' (Authentication Request) с содержимым из [4 нулей](#), которые означают, что аутентификация успешна (Authentication OK): `520000000800000000`.

Если требуется аутентификация, то вместо нулей будет 4 байта с методом аутентификации: 02 - kerberosV5, 03 - пароль открытым текстом, 05 - MD5, 07 - GSSAPI, 09 - SSPI, 0A - SASL (SCRAM). Для методов MD5, SASL после 4 байтами метода аутентификации идут данные.

Если аутентификация успешна, то сервер передаёт 15 пакетов 'S' (ParameterStatus), затем пакет 'K' (BackendKeyData) с PID и ключом для завершения сессии, 'Z' (ReadyForQuery).

Также в протоколе есть пакеты: 'E' Error, 'Q' Query, 'P' Parse, 'B' Bind, 'C' CommandComplete, 'D' DataRow, 'T' RowDescription, 'N' NoticeResponse, 'A' NotificationResponse, 'H' Flush, 'X' Terminate и другие.

Текстовое описание пакетов есть в документации:

https://docs.tantorlabs.ru/tdb/ru/18_6/se/protocol-message-formats.html

Описание пакета, инициирующего соединение

- Пример пакета Startup, посылаемого клиентом:

int32 длина пакета	int32 версия протокола	строка параметр	00	строка значение	00	строка параметр	00	строка значение	00	00
0000002A	000300	027573657200	706f73746772657300	646174616261736500	74656d706c6174653100	0000				
42	3.2	user		postgres		database		template1		

- передача пакета сетевого протокола PostgreSQL:

```
tcp-check send-binary 000000410003000007573657200706f7374677265730064617461626173650074656d706c61746531006c6f675f646973636f6e6e656374696f6e73006f66660000
```

- дамп TCP-пакета:

```
tcpdump -i lo -X -s 0 port 5432
IP localhost.58712 > localhost.5432: Flags [P.], seq 1:43, ack 1, win 260,
options [nop,nop,TS val 2147231413 ecr 2147231413], length 42
0x0030: ba9b 5572 0000 0041 0003 0000 7573 6572 ..Ur...A....user
0x0040: 0070 6f73 7467 7265 7300 6461 7461 6261 .postgres.databa
0x0050: 7365 0074 656d 706c 6174 6531 006c 6f67 se.template1.log
0x0060: 5f64 6973 636f 6e6e 6563 7469 6f6e 7300 _disconnections.
0x0070: 6f66 6600 00 off..
```



Описание пакета, инициирующего соединение

Описание пакета, инициирующего соединение (StartupMessage):

00000041 - длина пакета (сообщения протокола PostgreSQL) 65 байт в шестнадцатеричном виде (HEX). Длина пакета обязательна, начиная с версии протокола 3.0. Длина занимает 4 байта и следует за одним байтом типа пакета. Все пакеты имеют тип, кроме пакетов Startup, в которых нет байта типа.

00030000 - версия протокола 3.0, которая используется начиная с PostgreSQL 7.4 (2003г.).

В 18 версии появилась версия протокола 3.2, разница с 3.0 только в том, что сервер может передавать ключ длиной от 4 до 256 байт (в 18 версии 32 байта), а протокол 3.0 использует ключ длиной 4 байт. Ключ используется для прерывания сессии. Короткий ключ позволяет организовывать атаки на отказ в обслуживании, прерывая сессии. Версии 3.1 не существовало, но PostgreSQL принимает версию 3.1 00030001. Для протокола версии 3.2 значение будет: 00030002. Клиент версии 18 указывает версию 3.0. PostgreSQL 18 поддерживает протоколы от 3.0 до 3.2. Протокол 2.0 перестал поддерживаться в версии PostgreSQL 14. При указании неподдерживаемой версии протокола PostgreSQL возвращает пакет 45 = E (ErrorResponse) с ошибкой:

FATAL: unsupported frontend protocol 2.0: server supports 3.0 to 3.2

7573657200 - строка со словом user, завершающаяся нулевым байтом, обозначающим завершение строки. Это слово и имя пользователя передавать обязательно.

706f73746772657300 - слово postgres для аутентификации. Можно указать другое имя.

646174616261736500 - слово database. Это слово и имя базы передавать не обязательно.

74656d706c6174653100 - слово template1. Можно указать имя существующей базы данных, например, postgres. Если не указать, то используется имя пользователя. База данных template1 есть во всех кластерах баз данных PostgreSQL.

6c6f675f646973636f6e6e656374696f6e73006f666600 - log_disconnections00off00

Передача этого параметра отключит логирование в момент отсоединения, если логирование было включено на экземпляре PostgreSQL. Логирование в момент соединения параметром log_disconnection не отключается и передавать этот параметр бесполезно. Утилита psql передаёт параметры application_name00psql и client_encoding00UTF8, их не обязательно передавать, кодировка будет установлена в кодировку сервера.

00 - пакет Startup завешается нулевым байтом после нулевого байта, которым заканчивается последнее переданное слово.

Описание пакета, возвращаемого сервером

- На пакет, инициирующий соединение, сервер возвращает:
- пакет 'R' аутентификация успешна (AuthenticationOk)
- 15 пакетов 'S' ParameterStatus со значениями параметров
- пакет 'K' BackendKeyData с PID и ключом CancelKey
- пакет 'Z' ReadyForQuery, который означает готовность сервера принимать команды

```
localhost.5432 > localhost.58712: Flags [P.]
0x0030: 7ffc 26b5 5200 0000 0800 0000 0053 0000 ..&.R.....S..
0x0040: 001b 496e 7465 7276 616c 5374 796c 6500 ..IntervalStyle.
0x0050: 706f 7374 6772 6573 0053 0000 0020 7365 postgres.S....se
0x0060: 6172 6368 5f70 6174 6800 2224 7573 6572 arch_path."$user
0x0070: 222c 2070 7562 6c69 6300 5300 0000 1469 ",.public.S....i

0x0120: 5446 3800 5300 0000 1769 6e5f 686f 745f TF8.S....in_hot_
0x0130: 7374 616e 6462 7900 6f66 6600 5300 0000 standby.off.S...

0x01e0: 4459 004b 0000 000c 0015 6c42 7fa8 4d15 DY.K.....lB..M.
0x01f0: 5a00 0000 0549 Z....I
```



Описание пакета, возвращаемого сервером

Сервер возвращает одним TCP-пакетом сообщения:

520000000800000000 - байт типа пакета 52 = 'R' (Authentication Request). Затем идёт 4 байта с длиной оставшейся части пакета - 8 байт. Четыре нуля - аутентификация успешна (AuthenticationOk).

530000001b496e74657276616c5374796c6500706f73746772657300 - затем сразу идёт пакет 53 = 'S' (ParameterStatus) и его длина, - примерно 27 байт. В пакете IntervalStyle00postgres.

Таких пакетов 15 - по числу параметров из списка PQparameterStatus.

4b0000000c00156c427fa84d15 - пакет типа 4b = 'K' (BackendKeyData). Для протокола версии 3.0 длина пакета 0c = 12 байт. Пакет содержит номер процесса PID = 00156c42 (4 байта) и ключ Cancel Key 7fa84d15 (4 байта для протокола версии 3.0) которые может использовать клиент, чтобы прервать сессию. Для протокола версии 3.2 длина ключа 32 байта и может варьироваться от 4 до 256 байт.

5a0000000549 - пакет типа 5a = 'Z' (ReadyForQuery), который означает готовность сервера принимать команды. Длина пакета 5 байт. 49 = 'I' (Idle), что означает, что сессия простаивает и транзакция не открыта. Пакеты типа 'S' не передаются в открытой транзакции. Пример, директивы НАProху, которой можно проверить, что аутентификация успешно пройдена:

```
tcp-check expect binary 520000000800000000
```

Директива проверяет в ответе сервера наличие последовательности байт, соответствующей пакету 52 = 'R' (Authentication Request) с payload об успешной аутентификации.

Однако, эта проверка излишняя, можно сразу проверять значение in_hot_standby, если нужно определить роль экземпляра PostgreSQL.

Для проверки наличия пакета 'S' с in_hot_standby=off (мастер):

```
tcp-check expect binary 5300000017696e5f686f745f7374616e646279006f666600
```

для проверки наличия пакета 'S' с in_hot_standby=on (реплика):

```
tcp-check expect binary 5300000016696e5f686f745f7374616e646279006f6e00
```

Префикс 5300000017 или 5300000016 можно не указывать, так как оставшаяся часть сообщения вряд ли встретится в каком-то другом пакете.

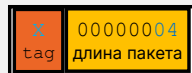
Содержимое сетевых пакетов можно получить утилитой tcpdump, перехватывая сетевые пакеты на интерфейс и порт экземпляра PostgreSQL:

```
tcpdump -i lo -X -s 0 port 5432
```

Корректное закрытие сессии проверки

- После аутентификации возвращается 15 параметров
 - › также они передаются, при изменении в течение сессии
 - › `in_hot_standby` передаётся, начиная с PostgreSQL 14
 - › `in_hot_standby` позволяет отличить мастер от реплики
- если проверка `expect` не прошла, то следующие за ней директивы, попавшие в буфер, без `linger` не выполняются:

```
tcp-check connect linger
tcp-check send-binary 00000017000300007573657200706f7374677265730000
tcp-check expect binary 696e5f686f745f7374616e646279006f666600 # in_hot_standby off, on=6f6e
tcp-check send-binary 5800000004 # завершение сессии 'X' = 58, длина 4 байта
```



- без передачи пакета закрытия сессии в диагностическом логе PostgreSQL будут ошибки:

```
LOG: could not receive data from client: Connection reset by peer
```



Корректное закрытие сессии проверки

Сразу после аутентификации серверный процесс возвращает параметры текущего статуса сервера https://docs.tantorlabs.ru/tdb/ru/18_4/se/libpq-status.html#LIBPQ-PQPARAMETERSTATUS. Также параметры передаются, если они меняются в течение сессии. Параметры `default_transaction_read_only` и `in_hot_standby` передаются, начиная с PostgreSQL версии 14. По значению параметра `in_hot_standby` можно отличить мастер от реплики.

Для HAProxy обязательно использование `linger`, который позволяет HAProxy передать пакет, закрывающий соединение (`send-binary 5800000004`), без него в логе PostgreSQL будут ошибки:

```
LOG: could not receive data from client: Connection reset by peer
```

Пример конфигурационного файла:

```
listen replica1
mode tcp
bind *:8002
tcp-check connect linger
tcp-check send-binary
0000002a000300007573657200706f7374677265730064617461626173650074656d706c617465310000
user00postgres00database00template100 (без log_disconnections00off) и закрытие
сообщения 00. Если разбить на отдельные части, то эти части будут передаваться
отдельными TCP-пакетами, поэтому передаётся одной длинной строкой.
tcp-check expect binary 696e5f686f745f7374616e646279006f666600
# in_hot_standby00on00 off=6f6666 on=6f6e
tcp-check send-binary 5800000004 # 'X' - завершение сессии , длина 4 байта
server r1 127.0.0.1:5432 check
server r2 127.0.0.1:5433 check
server r3 127.0.0.1:5434 check
```

Формат пакетов протокола PostgreSQL описан в документации:

https://docs.tantorlabs.ru/tdb/ru/18_4/se/protocol-message-formats.html

Проверка роли экземпляра SQL-запросом

- Проверка запросом `select pg_is_in_recovery();`

```
tcp-check connect linger
tcp-check send-binary
000000041000300007573657200706f73746772657300646174616261736500
74656d706c61746531006c6f675f646973636f6e6e656374696f6e73006f666660000 # пакет Startup
tcp-check expect binary 5a000000 # или 5a0000000549, где Z = 5a I = 49
tcp-check send-binary
510000002073656c6563742070675f69735f696e5f7265636f7665727928293b00 # 'Q', длина
запроса 0x20 (32 байта), запрос select pg_is_in_recovery();00
tcp-check send-binary 5800000004 # 'x' - завершение сессии
tcp-check expect binary 66430000000d # fC - false Complete или 74430000000d = tC
```

- Преимущества:
 - > результат не кэшируется pgbouncer и актуален
 - > проверяется возможность выполнять запросы
 - > в запрос можно вставить более сложные проверки
- Пакет закрытия сессии рекомендуется передать до проверки
 - > в этом случае, `linger` необязателен



Проверка роли экземпляра SQL-запросом

Проверка запросом `select pg_is_in_recovery();` Недостаток - лишней сетевой roundtrip, преимущество - проверяется возможность выполнения SQL-запросов. В протоколе есть пакеты 'F' (FunctionCall), которые позволяют вызывать функции ядра напрямую. Они не практичны, так как для вызова нужно указать OID функции, а он не фиксирован. Использование пакетов с запросами 'Q' (Query) более универсально.

```
listen master1
mode tcp
bind *:8001
option tcp-check
tcp-check connect linger
tcp-check send-binary
000000041000300007573657200706f7374677265730064617461626173650074656d706c617465
31006c6f675f646973636f6e6e656374696f6e73006f666660000
tcp-check expect binary 5a000000 # готов принимать команды 'Z' = 5a
tcp-check send-binary
510000002073656c6563742070675f69735f696e5f7265636f7665727928293b00
# запрос 'Q' = 51, select pg_is_in_recovery();00
tcp-check send-binary 5800000004 # 'x' - завершение сессии
tcp-check expect binary 66430000000d # fC - false и CommandComplete
server r1 127.0.0.1:5432 check
server r2 127.0.0.1:5433 check
server r3 127.0.0.1:5434 check
listen replica1
mode tcp
bind *:8002
...здесь те же строки, что в секции master1
tcp-check expect binary 74430000000d # tC - true Complete или 66430000000d = fC
...здесь те же строки, что в секции master1
```

Описание пакетов проверки SQL запросом

пакет с SQL-запросом

```
IP localhost.58712 > localhost.postgresql: Flags [P.], seq 43:76
 0x0000: 4500 0055 6ad6 4000 4006 d1ca 7f00 0001 E..Uj. @. @.....
 0x0010: 7f00 0001 e558 1538 8075 a4ca d858 0f81 .....X.8.u...X..
 0x0020: 8018 0104 fe49 0000 0101 080a 7ffc 26b9 .....I.....&.
 0x0030: 7ffc 26b8 5100 0000 2073 656c 6563 7420 ..&.Q....select.
 0x0040: 7067 5f69 735f 696e 5f72 6563 6f76 6572 pg_is_in_recover
 0x0050: 7928 293b 00                                y());.
```

Ответ с результатом выполнения запроса:

```
IP localhost.postgresql > localhost.58712: Flags [P.], seq 451:526
 0x0000: 4500 007f b426 4000 4006 8850 7f00 0001 E....&@. @..P....
 0x0010: 7f00 0001 1538 e558 d858 0f81 8075 a4eb .....8.X.X...u..
 0x0020: 8018 0104 fe73 0000 0101 080a 7ffc 26b9 .....s.....&.
 0x0030: 7ffc 26b9 5400 0000 2a00 0170 675f 6973 ..&.T....*.pg_is
 0x0040: 5f69 6e5f 7265 636f 7665 7279 0000 0000 _in_recovery....
 0x0050: 0000 0000 0000 1000 01ff ffff ff00 0044 .....D
 0x0060: 0000 000b 0001 0000 0001 6643 0000 000d .....fC....
 0x0070: 5345 4c45 4354 2031 005a 0000 0005 49 SELECT.1.Z....I
```

Команда 'X' закрытия соединения:

```
IP localhost.58712 > localhost.postgresql: Flags [P.], seq 76:81
 0x0000: 4500 0039 6ad7 4000 4006 d1e5 7f00 0001 E..9j. @. @.....
 0x0010: 7f00 0001 e558 1538 8075 a4eb d858 0fcc .....X.8.u...X..
 0x0020: 8018 0104 fe2d 0000 0101 080a 7ffc 26b9 .....-.....&.
 0x0030: 7ffc 26b9 5800 0000 04                                ..&.X....
```



Описание пакетов проверки SQL запросом

Передаваемый пакет с SQL-запросом:

tcp-check send-binary 510000002073656c6563742070675f69735f696e5f7265636f7665727928293b005100000020 - вначале тип (tag) пакета 51 = 'Q' (Query); затем длина пакета 0x20 = 32 байта; затем строка с запросом: `select pg_is_in_recovery();` и нулевой байт 00.

TCP-пакет с ответом сервера содержит пакеты сетевого протокола PostgreSQL:

540000002a000170675f69735f696e5f7265636f76657279000000000000000000000100001ffffffffff0000440000000b00010000000166430000000d5345 4c4543542031005a00000005495400000020 - пакет типа 54 = 'T' (RowDescription). Длина 2a = 42 байта.

0001 - число столбцов в ответе.

70675f69735f696e5f7265636f7665727900 - имя столбца `pg_is_in_recovery`.

Затем идут: OID таблицы (4 нуля, так как результат функции, а не таблица); номер столбца в таблице (2 нуля); 00000010 - OID типа данных столбца = 16 = bool, логический тип данных;

0001 - длина типа данных, для boolean 1 байт; ffffffff = -1 (не известен), модификатор типа данных столбца (typmod); формат кодирования данных - 2 нуля (текстовый формат, Text mode). Затем идёт сообщение:

440000000b - пакет типа 44 = 'D' (DataRow). Длина 0b = 11 байт; 0001 - число значений;

00000001 - длина значения 1 байт; 66 - само значение f. Затем идёт сообщение:

430000000d - пакет типа 43 = 'C' (CommandComplete). Длина 0d = 13 байт; строка с тэгом команды. Для команд типа select выдаёт число возвращённых строк в виде строки (тэга) `SELECT число`. Так как строка одна, то число = 1. Завершается нулевым байтом (00).

Для команды insert тэг: `INSERT 0 число`. Раньше, вместо 0 мог выдаваться OID таблицы. Затем идёт сообщение:

5a0000000549 - пакет типа 5a = 'Z' (ReadyForQuery). Длина 5 байт. 49 - статус транзакции I (Idle) транзакция не открыта.

Пакет завершения сессии, посылаемый клиентом:

5800000004 - пакет типа 58 = 'X' (Terminate) корректного закрытия сессии. Длина 4 байта. После этого пакета клиент может закрывать TCP-сокеты.

В конфигурации HAProxy, команда отсылки пакета завершения сессии (tcp-check send-binary 5800000004) должен идти до команды проверки (tcp-check expect), которая может завершиться неудачно, так как HAProxy не выполняет команды, которые идут после неудачной проверки.

Защита от split brain средствами HAProxy

- Проверка на единственность мастера:

```
frontend masterf
mode tcp
bind *:8001
acl single_master nbsrv(master1) eq 1
tcp-request connection reject if !single_master
default_backend master1
backend master1
mode tcp
option tcp-check
... проверки на мастера
server r1 127.0.0.1:5432 check
server r2 127.0.0.1:5433 check
server r3 127.0.0.1:5434 check
```

```
listen replicaf
mode tcp
balance leastconn
bind *:8002
option tcp-check
... проверки на реплику
server r1 127.0.0.1:5432 check
server r2 127.0.0.1:5433 check
server r3 127.0.0.1:5434 check
```

или

```
frontend replicaf
mode tcp
bind *:8002
use_backend replicaf
backend replicaf
mode tcp
balance leastconn
option tcp-check
... проверки на реплику
server r1 127.0.0.1:5432 check
server r2 127.0.0.1:5433 check
server r3 127.0.0.1:5434 check
```



Защита от split brain средствами HAProxy

Проверки на мастера и реплику напрямую (без Patroni) полезны, если Patroni переводится в режим паузы или вообще не используется. Конфигурация HAProxy может быть изменена и "бесшовно" перечитана: старые сессии доработают на старой конфигурации, новые - на новой конфигурации.

Одна из задач Patroni - не допускать работы двух мастеров (split brain). Если Patroni не используется или переведён в режим паузы, то нужно проверить, что мастер остановлен перед promote одной из реплик. HAProxy позволяет проверять, что среди серверов, на которые он направляет клиентов, только один сервер является мастером. Проверка может использоваться и с Patroni, как дополнительная защита от split brain.

Конфигурация HAProxy немного усложняется: вместо секции listen мастера нужно использовать связку из двух секций frontend и backend. Содержимое listen копируется в секцию backend, за исключением директивы bind, которая переходит в секцию frontend.

В секцию frontend мастера добавляется проверка на то, что мастер единственный из всех серверов, иначе отказывать клиентам в соединении с серверами. Выбирается "CP" - защита от расщепления (partition tolerance) вместо доступности (availability), как и в Patroni.

Конфигурация мастера:

```
frontend masterf
mode tcp
bind *:8001
acl single_master nbsrv(master1) eq 1
tcp-request connection reject if !single_master
default_backend master1
```

Для реплики можно не менять директиву listen, но можно и заменить на эквивалент из двух директив frontend и backend:

```
frontend replicaf
mode tcp
bind *:8002
use_backend replicaf
backend replicaf
mode tcp
balance leastconn
...
```

pgbouncer и параметр `in_hot_standby`

- При создании и изменении 15 параметров, PostgreSQL посылает клиенту пакет со значениями этих параметров
- pgbouncer не отслеживает такие пакеты и кэширует значения, которые получил при создании сессии
- эти кэшированные значения pgbouncer выдаёт своим клиентам, при создании ими сессии с pgbouncer независимо от значений своих опций

`server_reset_query_always=1` И `server_reset_query = DISCARD ALL`.

- Пример ошибки, при указании `target_session_attrs`:

```
psql "postgres://127.0.0.1:6432/postgres?target_session_attrs=primary"
psql: error: connection to server at "127.0.0.1", port 5434 failed: server is in hot standby mode
psql -q "postgres://127.0.0.1:6432/postgres?target_session_attrs=any" -tc "select
pg_is_in_recovery()"
f
```



pgbouncer и параметр `in_hot_standby`

При изменении значения 15 параметров, PostgreSQL 18 версии, без запроса клиента, посылает клиенту пакет со значениями этих параметров (https://docs.tantorlabs.ru/tdb/ru/18_4/se/protocol-flow.html#PROTOCOL-ASYNC). Также, такой пакет посылается при установке соединения после аутентификации, до сообщения `ReadyForQuery` ('Z' = 5a). Пакет не посылается, если клиент открыл транзакцию. Пакет посылается только вне транзакции. В 14 версии PostgreSQL в список отслеживаемых параметров, были добавлены параметры `in_hot_standby` и `default_transaction_read_only`, в 16 версии был добавлен параметр `scram_iterations`, в 18 версии был добавлен параметр `search_path`. Параметр `transaction_read_only` не входит в список отслеживаемых PostgreSQL.

pgbouncer может отслеживать параметры из списка `PQparameterStatus`, но только те, которые могут быть изменены клиентом (<https://www.pgbouncer.org/config.html> раздел, описывающий опцию `track_extra_parameters`) - 10 параметров из 15. В этот список pgbouncer не входит параметр `in_hot_standby`, он не может меняться клиентом. Параметр `default_transaction_read_only` не меняется при смене ролей мастер-реплика. Если перечислить параметры из этого списка (10 штук) в опции `track_extra_parameters`, то pgbouncer будет кэшировать значения перечисленных в опции параметров и будет восстанавливать их значения в сессиях для клиентов. По умолчанию, `track_extra_parameters` пуст и ни один из параметров не кэшируется. Значения остальных параметров будут сбрасываться в значения по умолчанию, если используется опция `server_reset_query_always=1`.

pgbouncer и параметр in_hot_standby

- При соединении клиента к pgbouncer после того, как реплика стала мастером, выдаются прежние значения:

```
psql -h localhost -p 6432
IP localhost.6432 > localhost.39742: Flags [P.], seq 2:455
0x00d0: 6e5f 686f 745f 7374 616e 6462 7900 6f6e n_hot_standby.on
0x01f0: 174a 515a 0000 0005 49 .JQZ....I
```

- при этом запрос выдаёт правильное значение **f**:

```
postgres=# select pg_is_in_recovery();
IP localhost.39742 > localhost.6432: Flags [P.], seq 93:126
0x0030: ad97 8c42 5100 0000 2073 656c 6563 7420 ..&.Q....select.
0x0040: 7067 5f69 735f 696e 5f72 6563 6f76 6572 pg_is_in_recover
0x0050: 7928 293b 00 y();.
IP localhost.6432 > localhost.39742: Flags [P.], seq 455:530
0x0030: ad97 ff8b 5400 0000 2a00 0170 675f 6973 ..&.T...*.pg_is
0x0040: 5f69 6e5f 7265 636f 7665 7279 0000 0000 _in_recovery....
0x0050: 0000 0000 0000 1000 01ff ffff ff00 0044 .....D
0x0060: 0000 000b 0001 0000 0001 6643 0000 000d .....fC....
0x0070: 5345 4c45 4354 2031 005a 0000 0005 49 SELECT.1.Z....I
```



pgbouncer и параметр in_hot_standby

Если реплика становится мастером, то in_hot_standby меняет значение с true на false, но проблема в том, что созданные pgbouncer сессии не разрываются, и **pgbouncer выдаёт своим клиентам значение in_hot_standby=true, хотя в реальности in_hot_standby=false**. Из-за этого, клиенты, которые подсоединяются к pgbouncer получают в пакете PQparameterStatus ложные значения и могут разорвать соединение. Например, если клиент указывает

target_session_attrs=primary или target_session_attrs=read-write. Пример:

```
psql "postgres://127.0.0.1:5432/postgres?target_session_attrs=primary"
```

```
psql: error: connection to server at "127.0.0.1", port 5434 failed: server is in hot standby mode
```

Такие клиенты могут использовать значение target_session_attrs=any. или при продвижении реплики нужно в pgbouncer нужно разорвать сессии с PostgreSQL. Это можно сделать командой командой pgbouncer RECONNECT. Для отправки этой команды, нужно сконфигурировать Patroni, чтобы при продвижении реплики до мастера подсоединялся к HAProxy и выполнял команду RECONNECT. Для этого создать файл

```
/opt/tantor/etc/patroni/callback.sh:
```

```
#!/usr/bin/bash
```

```
ACTION=$1; ROLE=$2; CLUSTER_NAME=$3;
```

```
[ "$ACTION" = "on_role_change" ] && [ "$ROLE" = "primary" ] && psql -h localhost -p 6432 -d pgbouncer -c "RECONNECT"
```

Установить параметры вызова скрипта в /opt/tantor/etc/patroni/patroni.yml:

```
postgresql:
```

```
callbacks:
```

```
on_role_change: /opt/tantor/etc/patroni/callback.sh
```

Проблема описана в: <https://github.com/pgbouncer/pgbouncer/issues/859>

Как выглядит проблема с соединениями через pgbouncer, если клиент запрашивает targetServerType=primary:

<https://github.com/keycloak/keycloak/issues/49515>

Кроме значения primary есть any, standby, prefer-standby, read-write, read-only (включает в себя случаи, когда параметр конфигурации default_transaction_read_only=on).