6

Управление кластером Patroni

Перезапуск процессов Patroni

- Перезапуск процессов Patroni
  - не увеличивает Timeline
  - не перезапускает экземпляры PostgreSQL

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml restart a --role any --force
+ Cluster: a (7682673136434780655) -----+----+-------------+-----+------------+-----+
| Member  | Host  | Role    | State     | TL | Receive LSN | Lag | Replay LSN | Lag |
+---------+-------+---------+-----------+----+-------------+-----+------------+-----+
| tantor1 | :5432 | Replica | streaming |  7 |  0/1D827370 |   0 | 0/1D827370 |   0 |
+---------+-------+---------+-----------+----+-------------+-----+------------+-----+
| tantor2 | :5433 | Leader  | running   |  7 |             |     |            |     |
+---------+-------+---------+-----------+----+-------------+-----+------------+-----+
Success: restart on member tantor1
Success: restart on member tantor2
```

  - можно перезапустить только с указанными ролями:
    `--role [leader | primary | standby-leader | replica | standby | any]`
- Для перечитывания файлов `patroni.yml`, с которыми были запущены процессы Patroni выполняется `reload`:

```
patronictl -c /opt/tantor/etc/patroni/patroni0.yml reload a --force
patronictl -c /opt/tantor/etc/patroni/patroni0.yml reload a tantor1 tantor2 --force
```


switchover

Перезапуск процессов Patroni:

```
patronictl -c /opt/tantor/etc/patroni/patroni0.yml restart a --force
+ Cluster: a (7682673136434780655) -----+----+-------------+-----+------------+-----+
| Member  | Host  | Role    | State     | TL | Receive LSN | Lag | Replay LSN | Lag |
+---------+-------+---------+-----------+----+-------------+-----+------------+-----+
| tantor1 | :5432 | Replica | streaming |  7 |  0/1D827370 |   0 | 0/1D827370 |   0 |
+---------+-------+---------+-----------+----+-------------+-----+------------+-----+
| tantor2 | :5433 | Leader  | running   |  7 |             |     |            |     |
+---------+-------+---------+-----------+----+-------------+-----+------------+-----+
Success: restart on member tantor1
Success: restart on member tantor2
```

Экземпляры PostgreSQL при этом не перезапускаются.

Перезапуск процессов Patroni нужен, если были изменены исполняемые файлы Patroni, например, при обновлении версии Patroni. При изменении файлов `patroni.yml` достаточно выполнить **`reload`**.

Перечитывание файлов `patroni.yml`, с которыми были запущены процессы.

Файл в параметре **`-c`** используется не процессами, а утилитой **`patronictl`** для того, чтобы подсоединиться к одному из процессов Patroni и к DCS и дать команду на перечитывание.

```
patronictl -c /opt/tantor/etc/patroni/patroni0.yml reload a --force
+ Cluster: a (7682905431918487538) --+-----------+----+-------------+----
| Member  | Host            | Role    | State     | TL | Receive LSN | Lag
+---------+-----------------+---------+-----------+----+-------------+----
| tantor1 | 127.0.0.1:5432  | Leader  | running   |  4 |             |
+---------+-----------------+---------+-----------+----+-------------+----
| tantor2 | 127.0.0.1:5433  | Replica | streaming |  4 |   0/3000000 |   0
+---------+-----------------+---------+-----------+----+-------------+----
Reload request received for member tantor1 and will be processed within 10 seconds
Reload request received for member tantor2 and will be processed within 10 seconds
```

Для перечитывания файла только одним процессом, нужно указать имя (или несколько имён через пробел) члена (членов) кластера:

```
patronictl -c /opt/tantor/etc/patroni/patroni0.yml reload a tantor1 --force
```

Режим паузы Patroni

- В режиме паузы Patroni не меняет состояние (запущен, остановлен, in recovery) экземпляров PostgreSQL
- остановка Patroni не остановит экземпляры PostgreSQL
- В режиме паузы **можно вручную** выполнять любые действия с экземплярами PostgreSQL, в том числе утилитой `patronictl`

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml pause --wait
'pause' request sent, waiting until it is recognized by all nodes
Success: cluster management is paused
patronictl -c /opt/tantor/etc/patroni/patroni.yml list
+ Cluster: a (7679758543612388307) --+-----------+----+-------------+----...
| Member  | Host           | Role    | State     | TL | Receive LSN | Lag
+---------+----------------+---------+-----------+----+-------------+----...
| tantor1 | 127.0.0.1:5432 | Leader  | running   |  1 |             |
| tantor2 | 127.0.0.1:5433 | Replica | streaming |  1 |   0/3050A68 |   0
+---------+----------------+---------+-----------+----+-------------+----...
patronictl -c /opt/tantor/etc/patroni/patroni.yml resume
Success: cluster management is resumed
```


Режим паузы Patroni

После передачи мастера и реплик под управление Patroni, процессы Patroni проверяют состояние экземпляров PostgreSQL, которыми они управляют; через DCS проверяют наличие лидера и могут автоматически останавливать, запускать экземпляры PostgreSQL, продвигать, пересоздавать реплики и другие действия. Если нужно, чтобы Patroni не выполнял автоматически действий с экземплярами PostgreSQL можно включить режим паузы (синоним `Maintenance mode`):

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml pause
Success: cluster management is paused
patronictl -c /opt/tantor/etc/patroni/patroni.yml list
+ Cluster: a (7679758543612388307) --+-----------+----+-------------+----...
| Member  | Host           | Role    | State     | TL | Receive LSN | Lag
+---------+----------------+---------+-----------+----+-------------+----...
| tantor1 | 127.0.0.1:5432 | Leader  | running   |  1 |             |
| tantor2 | 127.0.0.1:5433 | Replica | streaming |  1 |   0/3050A68 |   0
+---------+----------------+---------+-----------+----+-------------+----...
 Maintenance mode: on
```

Если пауза включена, то команды `list` и `topology` покажут в конце строку о том, что кластер Patroni находится в `Maintenance mode`. Снять с паузы:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml resume
Success: cluster management is resumed
```

В этом режиме паузы Patroni не выполняет автоматически failover, не запускает и не останавливает автоматически экземпляры PostgreSQL. Однако, на ручные действия ограничений нет: утилитой `patronictl` **можно** выполнять `switchover`, `failover`, `reinit`. Также, Patroni может читать и писать в DCS.

В этом режиме можно останавливать процессы (службы) Patroni, экземпляры PostgreSQL продолжат работать. При запуске процесса Patroni, он не будет запускать остановленный экземпляр PostgreSQL.

Если паузу не включить, то, при остановке Patroni, он остановит свой экземпляр PostgreSQL (в режиме `fast`). Если останавливается лидер Patroni, то он остановит мастера PostgreSQL и это приведёт к failover. Если даже нет реплик, Patroni остановит мастер, а потом запустит его сначала в режиме recovery, потом выполнит promote с увеличением линии времени.

switchover

- Получить название лидера и выбрать кандидата с наименьшими Lag:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml list
+ Cluster: a (7682673136434780655) --+-----------+----+-------------+-----+------------+-----+
| Member  | Host           | Role    | State     | TL | Receive LSN | Lag | Replay LSN | Lag |
+---------+----------------+---------+-----------+----+-------------+-----+------------+-----+
| tantor1 | 127.0.0.1:5432 | Leader  | running   |  6 |             |     |            |     |
+---------+----------------+---------+-----------+----+-------------+-----+------------+-----+
| tantor2 | 127.0.0.1:5433 | Replica | streaming |  6 |  0/1D827048 |   0 | 0/1D827048 |   0 |
```

- Переключиться на реплику:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml switchover --leader tantor1 --candidate
tantor2 --scheduled now --force
+ Cluster: a (7682673136434780655) --+-----------+----+-------------+-----+------------+-----+
| Member  | Host           | Role    | State     | TL | Receive LSN | Lag | Replay LSN | Lag |
+---------+----------------+---------+-----------+----+-------------+-----+------------+-----+
| tantor1 | 127.0.0.1:5432 | Replica | stopped   |  6 |     unknown |     |    unknown |     |
+---------+----------------+---------+-----------+----+-------------+-----+------------+-----+
| tantor2 | 127.0.0.1:5433 | Leader  | running   |  6 |             |     |            |     |
```

  - после команды статус переходный

switchover

После switchover можно запросить статус, он изменится:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml list
+ Cluster: a (7682673136434780655) -----+----+-------------+-----+------------+-----+
| Member  | Host  | Role    | State     | TL | Receive LSN | Lag | Replay LSN | Lag |
+---------+-------+---------+-----------+----+-------------+-----+------------+-----+
| tantor1 | :5432 | Replica | streaming |  7 |  0/1D827370 |   0 | 0/1D827370 |   0 |
+---------+-------+---------+-----------+----+-------------+-----+------------+-----+
| tantor2 | :5433 | Leader  | running   |  7 |             |     |            |     |
```

Переключать можно даже, если кластер на паузе.

Вывод PostgreSQL из под обслуживания Patroni

- Поставить кластер Patroni на паузу
- Остановить процессы Patroni и запретить службы
- Удалить записи в DCS:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml remove a
+ Cluster: a (7679758543612388307) --+-----------+----+-----...
...
 Maintenance mode: on
Please confirm the cluster name to remove: a
You are about to remove all information in DCS for a, please type: "Yes I am aware":
Yes I am aware
This cluster currently is healthy. Please specify the leader name to continue:
tantor1
```

- ИЛИ

```
etcdctl --endpoints=http://127.0.0.1:2379 del '/service/a' --prefix
4
```


Вывод PostgreSQL из под обслуживания Patroni

Сначала нужно поставить кластер Patroni на паузу.

Затем, остановить процессы (службы) Patroni и запретить их автоматический запуск.

Чтобы экземпляры PostgreSQL автоматически запускались при перезапуске операционной системы, включить (или создать и включить) службы.

Удалить файлы, созданные Patroni в директориях `PGDATA`:

```
patroni.dynamic.json *.backup
```

Удалить записи о кластере из DCS:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml remove a
+ Cluster: a (7679758543612388307) --+-----------+----+-----...
...
 Maintenance mode: on
Please confirm the cluster name to remove: a
You are about to remove all information in DCS for a, please type: "Yes I am aware": Yes I am aware
```

или

```
etcdctl --endpoints=http://127.0.0.1:2379 del '/service/a' --prefix
4
```

где `/service` - namespace, `a` - имя кластера.

Проверить, что данные из etcd удалены можно командой:

```
etcdctl --endpoints=http://127.0.0.1:2379 get '/service/a' --prefix
```

Возврат экземпляра под обслуживание Patroni

- Чтобы вернуть Patroni, достаточно запустить процессы Patroni на узлах с экземплярами PostgreSQL
- Patroni запускает **postmaster** делая его родителем systemd (**Parent PID=1**):

```
ps -eLo ppid,pid,cmd | egrep 'PPID|postgres'
   PPID     PID CMD
      1   80290 /opt/tantor/db/18/bin/postgres -D /var/lib/postgresql/tantor-se-
18/data --config-file=/var/lib/postgresql/tantor-se-18/data/postgresql.conf --
listen_addresses=127.0.0.1 --port=5432 --cluster_name=a --wal_level=replica --
hot_standby=on --max_connections=100 --max_wal_senders=10 --
max_prepared_transactions=0 --max_locks_per_transaction=64 --
track_commit_timestamp=False --max_replication_slots=10 --max_worker_processes=8
--wal_log_hints=on
  80290   80291 postgres: a: logger
  80290   80295 postgres: a: checkpointer
...
```

  - это позволяет Patroni останавливаться, не останавливая экземпляр PostgreSQL

Возврат экземпляра под обслуживание Patroni

Чтобы снова ввести экземпляр кластера PostgreSQL под обслуживание Patroni, достаточно запустить Patroni на узле (узлах), где работают экземпляры PostgreSQL, которые хочется ввести под обслуживание Patroni. У каждого экземпляра свой файл параметров. Если Patroni не работает ни на одном узле, то первым нужно запускать Patroni на узле мастера:

```
patroni /opt/tantor/etc/patroni/patroni.yml
```

Записи DCS будут восстановлены из PGDATA/**patroni.dynamic.json** - выполняется процедура восстановления повреждённого DCS (туда записываются значения при их изменении и они актуальнее patroni.yml) или patroni.yml - выполняется процедура инициализации кластера Patroni (bootstrap). Будет ли сразу включена пауза, определяется параметром:

```
dcs:
 pause: true
```

Patroni запускает postmaster делая его родителем systemd (**Parent PID=1**):

```
ps -eLo ppid,pid,cmd | egrep 'PPID|postgres'
   PPID     PID CMD
      1   80290 /opt/tantor/db/18/bin/postgres -D /var/lib/postgresql/tantor-
se-18/data --config-file=/var/lib/postgresql/tantor-se-18/data/postgresql.conf
--listen_addresses=127.0.0.1 --port=5432 --cluster_name=a --wal_level=replica
--hot_standby=on --max_connections=100 --max_wal_senders=10
--max_prepared_transactions=0 --max_locks_per_transaction=64
--track_commit_timestamp=False --max_replication_slots=10
--max_worker_processes=8 --wal_log_hints=on
  80290   80291 postgres: a: logger
  80290   80295 postgres: a: checkpointer
...
```

Это позволяет Patroni останавливаться, не останавливая экземпляр PostgreSQL, а также принимать под управление экземпляр, который был ранее запущен самим Patroni или через службы (systemctl start tantor-se-server-18).

При запуске через службу, родительским процессом для postmaster также является systemd (**Parent PID=1**).

Patroni передаёт в командной строке процессу **postgres** параметры, которые не должны перекрываться файлами postgres.conf и postgresql.auto.conf.

Подтверждение транзакций в PostgreSQL

- определяется параметрами PostgreSQL на мастере:
  - `synchronous_commit:` **`remote_write`** или **`on`**
  - `synchronous_standby_names`: по умолчанию пусто. Если установить непустое значение, то транзакции будут подтверждаться после:
    - получения репликами записи о коммите (**`remote_write`**)
    - получения и сохранении репликами записи о коммите (**`on`**)
- В параметре `synchronous_standby_names` указывается:
  - `ANY` *`n`* `(имя, имя,...)` - получить подтверждение у любых из перечисленных реплик, которые первыми ответили и доступны
  - `ANY` *`n`* `(*)` - получить подтверждение у *`n`* любых реплик
  - `[FIRST]` *`n`* `(имя, имя, ...)` - получить подтверждение у первых по списку, которые доступны

Подтверждение транзакций в PostgreSQL

Подтверждение транзакций определяется параметрами конфигурации PostgreSQL на мастере:

`synchronous_commit: on` по умолчанию. Возможные значения: `remote_apply`, `on`, **`remote_write`**, `local`, `off`. Разумный выбор сводится к двум значениям: **`remote_write`** или `on`.

`synchronous_standby_names`: по умолчанию пусто. Если установить непустое значение, то транзакции будут подтверждаться после:

1) получения репликами и передачей в страничный кэш операционной системы реплик (**`remote_write`**)

2) получения и гарантированном сохранении в WAL репликами (`on`)

3) получения, сохранения, применения репликами (`remote_apply`)

журнальной записи о фиксации транзакции и предыдущих журнальных записей.

В параметре можно указать значения:

1) `[FIRST] число (имя, имя, ...)` - получить подтверждение у первых из перечисленных в списке реплик, которые доступны. Этот способ не удобен тем, что одна из реплик может тормозить, что сильно увеличит время фиксации транзакций.

2) `ANY число (имя, имя,...)` - получить подтверждение у любых из перечисленных реплик, которые первыми ответили и доступны. Это называется подтверждение "кворумом". Неудобство этого способа в том, что при добавлении новой реплики, её нужно не забыть добавить в этот параметр. Обычно, в список добавляют реплики, которые имеют меньшую сетевую задержку между ними и мастером.

3) **`ANY число (*)`** - получить подтверждение у заданного `числа` любых реплик, которые первыми ответили. Этот способ позволяет не следить за именами реплик. Этот способ позволяет не следить за сетевой задержкой и её колебанием (jitter) до всех реплик - принимается ответ от наиболее быстро ответивших.

4) **`*`** - достаточно подтверждения одной репликой, эквивалент **`ANY 1 (*)`**.

При использовании Patroni, параметр `parameters.synchronous_standby_names` устанавливать нет смысла, так как Patroni сам сформирует значение для параметра PostgreSQL `synchronous_standby_names` при запуске экземпляра PostgreSQL и будет его обновлять.

https://patroni.readthedocs.io/en/latest/replication_modes.html

Режимы репликации: синхронный и асинхронный

- Настраиваются параметрами в DCS:
- `synchronous_mode: off` - реплики не подтверждают транзакции, значение по умолчанию
  - `quorum` - все реплики смогут подтверждать транзакции
  - `true` (`on`) - будет выбрано `synchronous_node_count` реплик, которые будут добавлены в `synchronous_standby_names`
- `synchronous_mode_strict: false` - если синхронные реплики не доступны, то мастер работает без них
- `synchronous_node_count: 1` - число подтверждающих реплик
- `postgresql.parameters.synchronous_commit: remote_write` или `on`

Режимы репликации: синхронный и асинхронный

По умолчанию, Patroni использует "асинхронный режим", при котором реплики не участвуют в подтверждении транзакций. Параметры, которые настраивают режим репликации:

```
bootstrap:
  dcs:
```

`synchronous_mode: quorum` - все реплики смогут подтверждать транзакции. Если установить `true` (`on`), то будет выбрано `synchronous_node_count` наименее отстающих реплик, которые будут добавлены в `synchronous_standby_names`, остальные реплики не будут добавлены. По умолчанию `off`. Имеет смысл использовать `quorum`.

`synchronous_mode_strict: false` - если синхронные реплики не доступны, то Patroni убирает параметр `synchronous_standby_names` и мастер сам подтверждает транзакции. Если установить `true`, то будет установлен `synchronous_standby_names='ANY 1 (*)'` и мастер не будет подтверждать транзакции, пока не появится хотя бы одна синхронная реплика.

`synchronous_node_count: 1` - устанавливает `число` реплик в параметре `synchronous_standby_names`, уменьшая его до числа работающих реплик, если установленное `число` больше. Задаёт "кворум" - сколько реплик должно подтвердить транзакции. Не имеет смысла ставить больше, чем `1`.

`maximum_lag_on_syncnode: -1` - чтобы заменить отставших менее отстающими. При небольшом значении walsenders будут часто перезапускаться.

```
    postgresql:
      parameters:
```

`synchronous_commit: remote_write` - по умолчанию `on`.

С такими параметрами Patroni установит в `postgresql.conf`:

```
synchronous_commit = 'remote_write'
```

у лидера установит и будет обновлять значение параметра:

```
synchronous_standby_names = 'ANY 1 (tantor2,tantor3)'
```

имена будут браться из параметров, которые Patroni установит у реплик:

```
primary_conninfo = 'dbname=postgres user=postgres  host=127.0.0.1 port=5432
passfile=/var/lib/postgresql/pgpass application_name=tantor2 gssencmode=prefer
channel_binding=prefer sslnegotiation=postgres'
primary_slot_name = 'tantor2'
```

Редактирование параметров Patroni

- Параметры можно редактировать в редакторе:

```
EDITOR=mcedit patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config
```

- Параметры можно менять без перезапуска экземпляров:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config --set
synchronous_mode=quorum --force
+synchronous_mode: quorum
Configuration changed
```

- Просмотр текущей конфигурации в DCS:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml show-config | grep sync
maximum_lag_on_syncnode: -1
    synchronous_commit: remote_write
synchronous_mode: quorum
synchronous_mode_strict: true
synchronous_node_count: 1
```

- Patroni вносит изменения в основной файл:

```
cat /var/lib/postgresql/tantor-se-18/data/postgresql.conf | grep sync
synchronous_commit = 'remote_write'
synchronous_standby_names = 'ANY 1 (tantor2)'
```


Редактирование параметров Patroni

Параметры можно менять без перезапуска экземпляров:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config --set
synchronous_mode_strict=true --force
```

Для изучения, что делают команды можно просматривать конфигурацию:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml show-config | grep sync
maximum_lag_on_syncnode: -1
    synchronous_commit: remote_write
synchronous_mode: true
synchronous_mode_strict: true
synchronous_node_count: 1
patronictl -c /opt/tantor/etc/patroni/patroni.yml list
tantor1 Role=Leader, tantor2 Role=Sync Standby, tantor3 Role=Replica
cat /var/lib/postgresql/tantor-se-18/data/postgresql.conf | grep sync
synchronous_commit = 'remote_write'
synchronous_standby_names = 'tantor2'
```

После изменения параметра:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config --set
synchronous_mode=quorum --force
```

или остановке Patroni на одном из узлов теми же командами посмотреть, что изменилось:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml show-config | grep sync
synchronous_mode: quorum
patronictl -c /opt/tantor/etc/patroni/patroni.yml list
tantor1 Role=Leader, tantor2 Role=Quorum Standby, tantor3 Role=Quorum Standby
```

Значение берётся из столбца `sync_state` представления `pg_stat_replication` мастера.

```
cat /var/lib/postgresql/tantor-se-18/data/postgresql.conf | grep sync
synchronous_commit = 'remote_write'
synchronous_standby_names = 'ANY 1 (tantor2,tantor3)'
```

Если остановить Patroni реплик `tantor2,tantor3`, то команды выдадут:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml list
tantor1 Role=Leader
cat /var/lib/postgresql/tantor-se-18/data/postgresql.conf | grep names
synchronous_standby_names = 'ANY 1 (*)'
```

Гарантии отсутствия потерь транзакций

- Параметр Patroni `synchronous_mode_strict` не позволяет Patroni отключить последнюю синхронную реплику
  - используется с параметром Patroni `synchronous_mode`
  - обеспечивает сохранение журнальных записей как минимум в двух местах: на мастере и реплике
  - с параметром или без него возможна потеря транзакций при переключении на реплику
- Фиксация транзакции в PostgreSQL неатомарна
  - запись о фиксации надёжно сохраняется в WAL мастера
  - запись отправляется на реплики, ждёт подтверждения
  - изменения становятся видны сессиям мастера
  - клиенту отправляется подтверждение о фиксации

Гарантии отсутствия потерь транзакций

В абзаце "Note:" страницы документации Patroni https://patroni.readthedocs.io/en/latest/replication_modes.html#synchronous-mode написано:

"Из-за особенностей реализации синхронной репликации в PostgreSQL, возможна потеря транзакций даже при использовании `synchronous_mode_strict`. Если работа серверного процесса PostgreSQL прерывается во время ожидания подтверждения от реплики (по любой причине: прерывание запроса, сессии, тайм-аута клиента, сбоя процесса), результат транзакции становится видимым для других сессий мастера. Запись же о фиксации транзакции может быть ещё реплицирована и, если реплика станет мастером, результат транзакции на новом мастере будет потерян."

Это поведение PostgreSQL и нет приемлемого способа устранить это средствами Patroni.

Фиксация транзакции физически многошаговая, а не атомарная, а с синхронными репликами, ещё и протяжённая во времени.

1) Журнальная запись, содержащая `COMMIT`, сохраняется в WAL-файл (writeback, fdatasync)

3) Устанавливается бит о фиксации транзакции в журнале транзакций `pg_xact` (`CLOG`)

4) WALSender отправляют содержимое WAL файлов репликам и ждут подтверждения о приёме (`synchronous_commit=remote_write`) или сохранении (`on`)

5) Транзакция помечается как видимая другим сессиям

6) Клиенту возвращается подтверждение о фиксации транзакции.

Эти действия неатомарны и могут быть прерваны в любой момент.

Более того, при задержке в передаче журнальных записей синхронной реплике может пройти много времени и вероятность прерывания на шагах 4-6 увеличивается.

В результате есть вероятность, что:

1) изменения записаны на диск, но не видны клиентам

2) транзакция реплицирована и видна сессиям реплики, но не видна сессиям мастера

3) результат транзакции виден сессиям мастера до того, как сессия, выполнившая транзакцию, получит подтверждение о фиксации своей транзакции

https://habr.com/ru/companies/tantor/articles/1070686/

Фиксация транзакции с предупреждением

- Изменения станут видны другим сессиям мастера, если:
  - клиенты прервут запросы или сессии

```
insert into t1 values (1);
WARNING:  canceling wait for synchronous replication due to user request
DETAIL:  The transaction has already committed locally, but might not have
been replicated to the standby.
INSERT 0 1
```

  - сессия прервётся по таймаутам
  - серверный процесс сбойнёт или экземпляр перезапустится
- При этом реплики могут не получить журнальные записи
- Изменения в сессиях, увидевших изменения не переданной транзакции, на новом мастере будут отсутствовать
  - последовательность фиксации транзакций не нарушается

Фиксация транзакции с предупреждением

Клиенты могут:
1) прервать собственные запросы, послав сетевой пакет `CancelRequest(pid, secret)`. psql посылает такой пакет при нажатии комбинации клавиш `ctrl+c`.
2) В других сессиях клиенты могут вызвать функцию `pg_cancel_backend()`
3) Суперпользователи и члены роли `pg_signal_backend` могут вызвать функцию `pg_terminate_backend()`
4) Сессия прервётся по таймауту `transaction_timeout` или таймауту на клиенте (серверный процесс получит уведомление о закрытии сетевого сокета).
4) экземпляр может перезапуститься после сбоя или пропадания питания.

В этих случаях, в локальном журнале транзакция, ожидающая подтверждения (`SyncRepWaitForLSN`) от синхронной реплики (реплик), будет зафиксирована, а синхронные реплики могут не получить LSN с фиксацией транзакции.

Пример:

При прерывании запроса в режиме автофиксации, серверный процесс, находящиеся в ожидании подтверждения от синхронной реплики (`SyncRep`) выдаст сообщение о том, что транзакция зафиксирована:

```
insert into t1 values (1);
WARNING:  canceling wait for synchronous replication due to user request
DETAIL:  The transaction has already committed locally, but might not have
been replicated to the standby.
INSERT 0 1
```

Изменения тут же станут **видны** в других сессиях мастера, при том, что реплика, из-за медленной работы сети, могла не получить изменения. Если мастер сбойнёт и реплика станет мастером, изменений прерванной транзакции на реплике не будет. Что хорошо, увидев изменения, транзакции не смогут поменять данные так, чтобы изменения (сделанные на основе увиденных данных) были доступны на новом мастере. Запись в WAL последовательная - если запись о `COMMIT` не принята репликой, то не приняты и все последующие записи.

Может быть и **обратная** ситуация, когда изменения переданы репликам, экземпляр мастера упал и клиент не получил подтверждение о том, что транзакция была зафиксирована:

```
server closed the connection unexpectedly
```

При этом, после promote реплики, вставленная строка будет присутствовать.

PostgreSQL Transaction Guard

- Рекомендации:
  - Не прерывать подвисшие запросы и сессии
  - Для критичных транзакций использовать `txid_current()`+`txid_status()` для проверки статуса транзакции после разрыва сессии:

```
insert into t1 values (1) returning txid_current();
 txid_current
--------------
          800
WARNING:  canceling wait for synchronous replication due to user request
INSERT 0 1
\connect
select txid_status(800);
 txid_status
-------------
committed
```

- Вероятность потери транзакций низка, если не прерывать сессии и не замедлять работу сети

PostgreSQL Transaction Guard

Рекомендации:

1) Не прерывать подвисшие запросы и сессии. В случае, если мастер не продлит лизинг, он будет остановлен и реплика станет новым мастером. При этом параллельные сессии не увидят данных неподтверждённых синхронной репликой транзакций. То есть не прерывать запросы и не устанавливать `transaction_timeout` в значения, меньшие 30 секунд (Patroni TTL, время на распознавание кто сбойнул - мастер или реплика). В случае, если сбойнула реплика или сеть, реплика станет отставшей и Patroni её не сделает её мастером. После восстановления связи, реплика получит недостающие журнальные записи.

2) Для критичных транзакций использовать `txid_current()` для получения номера транзакции, результат которой важен. В случае разрыва сессии, которое произойдёт при переключении на нового мастера, проверять статус транзакции функцией `txid_status()`:

```
insert into t1 values (1) returning txid_current();
 txid_current
--------------
          800
INSERT 0 1
```

После исчезновения мастера, разрыва сессии, в новой сессии с новым мастером запросить статус транзакции:

```
select txid_status(800);
 txid_status
-------------
committed
```

Такой способ используется в опции Oracle Transaction Guard, появившийся в 12 версии Oracle Database - приложение получает логический номер транзакции и проверяет статус, в случае разрыва сессии и переподсоединения к резервной базе данных.

Проблему сложно воспроизвести, вероятность её возникновения низка, если не прерывать сессии. Вероятность увеличивает замедление работы сети, большой поток коротких транзакций, перед тем, как реплика станет мастером. Если пакеты проходят по сети без замедления или в транзакции несколько команд, вероятность проблемы невелика. Вероятность возрастает при использовании огромного числа кластеров PostgreSQL, которые (или контейнеры, в которых они работают) часто перезапускаются, что приводит к частым продвижениям реплик до мастера.

Обновление Patroni

- Включить паузу (Maintenance mode)

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml pause --wait
'pause' request sent, waiting until it is recognized by all nodes
Success: cluster management is paused
```

- Обновить программное обеспечение Patroni

```
dpkg -i patroni-tantor-all*.deb
```

- Перезапустить службы Patroni на всех узлах

```
systemctl restart patroni-tantor --wait
```

- Выключить паузу

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml resume --wait
Success: cluster management is resumed
```

- В разделах **bootstrap.dcs.postgresql.parameters** и **postgresql.parameters** файла patroni.yml **не могут** находиться параметры PostgreSQL, не известные Patroni
- в новых версиях PostgreSQL добавляются параметры

Обновление Patroni

В разделе **postgresql.parameters**, как и в **bootstrap.dcs.postgresql.parameters** **не могут** находиться параметры не известные Patroni, что такие параметры есть в конкретной версии PostgreSQL (кроме параметров PostgreSQL, в названии которых есть точка). Параметры будут игнорироваться с предупреждением:

```
WARNING: Removing unexpected parameter=имя value=значение from the config
```

Поэтому, при выходе новой версии PostgreSQL придётся использовать новую версию Patroni, в которой есть параметры, появившиеся в новой версии PostgreSQL. В этом основная причина обновления Patroni.

Также, Patroni обновляют чтобы устранить ошибки, которые исправляются в новых версиях.

Обновление Patroni просто. Достаточно обновить программное обеспечение - пакет deb или rpm и перезапустить службу Patroni на всех узлах кластера Patroni:

1) Прочесть Release notes и проверить нет ли особенностей при обновлении:
https://patroni.readthedocs.io/en/latest/releases.html

1) Включить паузу:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml pause --wait
Success: cluster management is paused
```

2) Обновить программное обеспечение на всех узлах:

```
apt-get update && apt-get install patroni-tantor-all
```

или

```
dpkg -i patroni-tantor-all*.deb
patroni-tantor.service is a disabled or a static unit not running, not starting it.
```

3) Перезапустить службы на всех узлах:

```
systemctl restart patroni-tantor --wait
```

Обычно, Patroni может работать на разных узлах на разных версиях. В Release notes к 4 версии написано: Обновление до 4+ версии безопасно при переходе с версии 3.1.+. Обновление с более ранней версии возможно, но может привести к неожиданному поведению, если мастер выйдет из строя.

4) Выключить паузу:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml resume --wait
Success: cluster management is resumed
```

https://patroni.readthedocs.io/en/latest/installation.html#upgrading

Обновление основной версии PostgreSQL

- Отключить Patroni, удалить конфигурацию и повторно инициализировать кластер Patroni
- Удаление конфигурации из DCS:
  - остановить Patroni на мастере и репликах
  - удалить конфигурацию в DCS:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml remove a
+ Cluster: a (7676506443451906864) --+---------+----+
| Member  | Host            | Role    | State   | TL |
+---------+-----------------+---------+---------+----+
| tantor1 | 127.0.0.1:5432 | Replica | stopped |  6 |
Please confirm the cluster name to remove: a
You are about to remove all information in DCS for a, please type:
"Yes I am aware": Yes I am aware
```

  - или только ключ `initialize`:

```
etcdctl --endpoints=:2379 del '/service/a/initialize'
1
```


Обновление основной версии PostgreSQL

Единственный документированный способ выполнить мажорное обновление PostgreSQL, управляемый Patroni это отключить Patroni, удалить конфигурацию и повторно инициализировать кластер Patroni. Шаги:

1) Перевести Patroni в режим обслуживания. Остановить Patroni на репликах и мастере.

2) Установить новую версию PostgreSQL и обновить мастер до новой версии. Для обновления мастера можно использовать `pg_upgrade` или любой другой способ.

3) Обновить файл `patroni.yml`, указав новые пути и удалив устаревшие параметры, если такие есть

4) Удалить ключ **`initialize`** или все ключи из DCS. Это можно сделать командой:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml remove a
+ Cluster: a (7676506443451906864) --+---------+----+
| Member  | Host            | Role    | State   | TL |
+---------+-----------------+---------+---------+----+
| tantor1 | 127.0.0.1:5432 | Replica | stopped |  6 |

Please confirm the cluster name to remove: a
You are about to remove all information in DCS for a, please type: "Yes I am aware": Yes I am aware
```

Это нужно сделать, так как в процессе обновления утилитой `pg_upgrade` создаётся новый кластер баз данных PostgreSQL с новым идентификатором, который используется Patroni как идентификатор кластера:

```
pg_controldata | grep system
Database system identifier:           7676506443451906864
```

5) Скопировать файл из старой директории `PGDATA/patroni.dynamic.json` в новую директорию `PGDATA`. Если из DCS были удалены все данные кластера, а не ключ **`initialize`**, то это поможет сохранить часть параметров конфигурации PostgreSQL.

6) Запустить Patroni на обновлённом мастере с обновлённым файлом `patroni.yml`

7) Установить новую версию PostgreSQL на узлах реплик, обновить файлы `patroni.yml`, стереть директории `PGDATA` на репликах.

8) Запустить Patroni на узлах реплик и ждать, когда Patroni пересоздаст реплики.

https://patroni.readthedocs.io/en/latest/existing_data.html#major-upgrade-of-postgresql-version

Режим DCS failsafe

- включается `bootstrap.dcs.failsafe_mode: true`
- позволяет не останавливать экземпляр мастера PostgreSQL при отсутствии связи с DCS

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml show-config | egrep
"(ttl|loop|failsafe|timeout)"
failsafe_mode: true
loop_wait: 1
retry_timeout: 3
ttl: 20
```

- Включение `bootstrap.dcs.failsafe_mode`:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config -s failsafe_mode=true
--force
-failsafe_mode: false
+failsafe_mode: true
 loop_wait: 1
 max_timelines_history: 100
 maximum_lag_on_failover: 1048576
Configuration changed
```


Режим DCS failsafe

Patroni использует DCS для хранения параметров динамической конфигурации, обмена межу узлами данными о лидерстве. Лидер продлевает лизинг ключа, а реплики наблюдают (watch) за изменениями ключа. Patroni может открывать на чтение-запись экземпляр PostgreSQL только, если он обладает ключом лидера и лизинг не истёк. Если истёк строк лизинга (`ttl`), то Patroni останавливает экземпляр PostgreSQL и запускает его в режиме только для чтения. Вероятность недоступности etcd, **если etcd используется только Patroni**, а другие приложения ничего не хранят в etcd близка к нулю. Причины необновления ключа лидера:

1) расщепление сети - узлы etcd или Patroni не могут общаться

2) неотзывчивость (отсутствие ответа) узла etcd, к которому подсоединяется лидер Patroni.

Для защиты от непродления ключа лидера **стоит** использовать `failsafe_mode=true`:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config -s
failsafe_mode=true --force
-failsafe_mode: false
+failsafe_mode: true
 loop_wait: 1
 max_timelines_history: 100
 maximum_lag_on_failover: 1048576
Configuration changed
```

Свойство хранится в ключе DCS `/config`. Если обновление ключа не удалось по причине, отличной от несоответствия версий Patroni на узлах, неверного значения ключа `/leader` (другое имя узла-лидера), то Patroni не остановит экземпляр мастера, пока Patroni лидера может получить ответ от других узлов по **restapi**. Запросы отправляются с частотой `loop_wait`:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml show-config | egrep
"(ttl|loop|failsafe|timeout)"
failsafe_mode: true
loop_wait: 1
retry_timeout: 3
ttl: 20
```

https://patroni.readthedocs.io/en/latest/dcs_failsafe_mode.html

Режим DCS failsafe

- В режиме failsafe **все** Patroni из ключа failsafe должны быть доступны лидеру, иначе лидер остановит экземпляр мастера и запустит его в read only (как реплику)

```
etcdctl --endpoints=http://127.0.0.1:2379 get '/service/a/failsafe' --prefix
/service/a/failsafe
{"tantor1":"http://127.0.1.1:8008/patroni"}
```

- Реплики используют входящие запросы `POST /failsafe` как индикатор того, что лидер активен и кэшируют результат на время, заданное в `ttl`:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config -s ttl=20 --force
     wal_log_hints: 'off'
   use_slots: true
 retry_timeout: 3
-ttl: 30
+ttl: 20
Configuration changed
```


Режим DCS failsafe

Если **хоть один из узлов** Patroni не ответит, мастер перейдёт в режим только для чтения, **выборы лидера НЕ начнутся, мастер НЕ восстановится, пока не станет доступен DCS**.

Так сделано для защиты от split brain в случае расщепления сети, когда часть узлов Patroni и ETCD, в произвольной комбинации, не имеют связи друг с другом. **Источник истины должен быть единственный и это DCS**. Выбирающие узлы могут не иметь связи с DCS. DCS восстановится, его увидит лидер и сделает свой экземпляр мастером. Если бы выборы в отсутствие DCS были возможны, то реплики, не видя DCS из-за сетевого сбоя, выберут нового лидера и он сделает свой экземпляр мастером. Возникнет split brain.

По той же причине защиты от split brain, лидеру Patroni должны быть видны все узлы кластера, он не может полагаться на доступность большинства узлов. В режиме failsafe нет понятия "кворума". Термин "кворум" используется **etcd** и PostgreSQL. У PostgreSQL он обозначает число синхронных реплик, которые должны подтвердить транзакцию. Для подтверждения транзакции нужна одна реплика, не стоит использовать две и более реплики, это не улучшит отказоустойчивость.

В режиме failsafe узел может создавать реплику из бэкапа даже в отсутствие лидера.

Список узлов Patroni поддерживается лидером в ключе `/failsafe`:

```
etcdctl --endpoints=http://127.0.0.1:2379 get '/service/a/failsafe' --prefix
/service/a/failsafe
{"tantor1":"http://127.0.1.1:8008/patroni"}
```

Узел может участвовать в выборах лидера и становиться новым лидером только, если он присутствует в ключе `/failsafe`. Если кластер состоит из одного узла, ключ `/failsafe` будет содержать один узел. Реплики используют входящие запросы на **restapi** `POST /failsafe` как индикатор того, что лидер всё еще активен. Эта информация кэшируется на время, заданное в параметре динамической конфигурации `ttl`:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config -s ttl=30 --
force
     wal_log_hints: 'off'
   use_slots: true
 retry_timeout: 3
-ttl: 20
+ttl: 30
Configuration changed
```

Восстановление при потере данных в DCS

- Режим failsafe работает даже, если у мастера нет реплик
- Если failsafe не был включён или не был включён режим паузы, то Patroni остановит мастера в режиме `fast` и запустит в режиме standby, увеличив линию времени

```
LOG:  received fast shutdown request
WARNING:  specified neither "primary_conninfo" nor "restore_command"
HINT:  The database server will regularly poll the pg_wal subdirectory to
check for files placed there.
LOG:  entering standby mode
```

- В режиме failsafe нельзя переконфигурировать Patroni
  - добавлять или удалять узлы, выполнять switchover и другие unsafe операции
  - можно только читать состояние узлов по REST API
- `patronictl` не работает при недоступном DCS

Восстановление при потере данных в DCS

Patroni выгружает динамическую конфигурацию из DCS в файл `PGDATA/patroni.dynamic.json` **при каждом изменении конфигурации**.

Лидер Patroni (только он) автоматически восстанавливает динамическую конфигурацию из `PGDATA/patroni.dynamic.json` когда динамическая конфигурация полностью отсутствуют в DCS или она не распознаётся (invalid).

При потере DCS, то есть его полной недоступности, Patroni перейдёт в режим failsafe, если режим включён: https://patroni.readthedocs.io/en/latest/dcs_failsafe_mode.html. По умолчанию, режим выключен.

Режим failsafe работает даже, если у мастера нет реплик. Если failsafe не был включён, то мастер перезапустится и откроется в режиме только для чтения:

```
ERROR: Error communicating with DCS
INFO: demoting self because DCS is not accessible and I was a leader
INFO: Demoting self (offline)
LOG:  received fast shutdown request
INFO: closed patroni connections to postgres
WARNING:  specified neither "primary_conninfo" nor "restore_command"
HINT:  The database server will regularly poll the pg_wal subdirectory to
check for files placed there.
LOG:  entering standby mode
```

В режиме failsafe нельзя переконфигурировать Patroni: добавлять или удалять узлы, выполнять switchover и т.п. unsafe операции, можно только читать состояние узлов по REST API. `patronictl` не работает при недоступном DCS, так как `patronictl` пытается подсоединиться к DCS при выполнении любых команд.

Восстановление при потере данных в DCS

- Пример команд удаления конфигурации Patroni из etcd:

```
etcdctl --endpoints=http://127.0.0.1:2379 del '/service/a/config' --prefix
etcdctl --endpoints=http://127.0.0.1:2379 del '/service/a' --prefix
etcdctl --endpoints=http://127.0.0.1:2379 get '' --prefix
patronictl -c /opt/tantor/etc/patroni/patroni.yml remove a
```

- Процессы Patroni пересоздают записи в DCS из:
  - своей памяти
  - файла `PGDATA/patroni.dynamic.json`
  - раздела `bootstrap.dcs` файла `patroni.yml`
- Лидер Patroni пересоздаёт записи о конфигурации кластера Patroni, как только DCS станет доступен
- Процессы Patroni читают и пишут в DCS в процессе выборов лидера

Восстановление при потере данных в DCS

Пример удаления конфигурации Patroni, хранящейся в виде ключей:

```
etcdctl --endpoints=http://127.0.0.1:2379 del '/service/a/config' --prefix
etcdctl --endpoints=http://127.0.0.1:2379 del '/service/a' --prefix
etcdctl --endpoints=http://127.0.0.1:2379 get '' --prefix
```

также можно удалить ключи командой:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml remove a
```

После того, как DCS станет доступен, независимо от того, сохранились ли в нём данные (ключи) или ключи были полностью стёрты, прежний или выбранный лидер (в случае исчезновения DCS, перехода в failover, остановки лидера - будут работать только реплики, появления чистого DCS, инициализацией одной из реплик DCS, выборов оставшимися без лидера репликами нового лидера) Patroni пересоздаст динамическую конфигурацию из:

1) данных из своей памяти
2) если не работает, то **при запуске** на основе файла `PGDATA/patroni.dynamic.json`
3) если файла нет, то на основе раздела `bootstrap.dcs` файла `patroni.yml`.

При этом, обновятся файлы `PGDATA/postgresql.conf` и `PGDATA/postgresql.conf.backup`.

Период лидерства RaftTerm

- Patroni при работе с etcd проверяет **Cluster ID** и **RaftTerm**
- Как посмотреть **ClusterID** и **RaftTerm**:

```
etcdctl --endpoints=http://127.0.0.1:2379 endpoint status -w fields | egrep
'RaftTerm|ClusterID'
"ClusterID" : 11470682518379718621
"RaftTerm" : 3
"RaftTerm" : 3
```

- Пока **RaftTerm** не увеличится больше, чем видел Patroni, Patroni не взаимодействует с etcd:

```
ERROR: KVCache.run EtcdConnectionFailed('No more machines in the cluster')
WARNING: Connected to Etcd node with term 2. Old known term 4. Switching to another node.
ERROR: Failed to get list of machines from http://127.0.0.3:2379/v3: StaleEtcdNode()
WARNING: Connected to Etcd node with term 2. Old known term 4. Switching to another node.
ERROR: Failed to get list of machines from http://127.0.0.2:2379/v3: StaleEtcdNode()
WARNING: Connected to Etcd node with term 2. Old known term 4. Switching to another node.
ERROR: Failed to get list of machines from http://127.0.0.1:2379/v3: StaleEtcdNode()
ERROR: KVCache.run EtcdConnectionFailed('No more machines in the cluster')
```


Период лидерства RaftTerm

RaftTerm - период лидерства от начала выборов до прекращения лидерства. Это счётчик, увеличивающийся на 1 в начале выбора лидера.

RaftTerm начинает отсчёт с нуля и его нельзя переустановить или поменять. Увеличивают значение только выборы лидера. Например, выборы начнутся при перезапуске лидера. При долгой работе кластера etcd значение **RaftTerm** может равняться сотням или тысячам.

Начиная с версий 3.3.7 и 4.0.6, Patroni сохраняет **в памяти процесса** последний наибольший номер "**raft_term**", который он получал в ответах от etcd и, при посылке запросов в etcd, сравнивает "**raft_term**", возвращаемый узлом etcd с тем, что запомнил. Это позволяет не взаимодействовать с узлами etcd, которые работают со старым лидером.

Если после пересоздания кластера etcd, у него сохранится прежний **Cluster ID**, то Patroni, видя тот же **Cluster ID** и меньший **RaftTerm**, чем он видел, **не будет взаимодействовать с узлами такого кластера etcd** до тех пор, пока **RaftTerm** не превысит значение, которое видел Patroni. Таймаута на ожидание нет, Patroni считает, что DCS нет. Без DCS Patroni может находиться в режиме failsafe, если Patroni работал и не останавливался в то время, как кластер etcd был пересоздан с тем же **Cluster ID**.

Опасность пересоздания кластера без смены **Cluster ID** в том, что если один из процессов Patroni перезапустится, то память с хранящемся в ней **RaftTerm** очистится и перезапустившийся процесс Patroni начнёт взаимодействовать с кластером etcd, при том, что остальные процессы Patroni не будут взаимодействовать с etcd, до тех пор, пока они не перезапустятся. Перезапустившийся процесс, если заранее не был установлен режим паузы, выполнит failover и сделает свой экземпляр мастером. Однако, ситуации split brain не возникнет, так в режиме failsafe, при недоступности любого из процессов Patroni, лидер останавливает свой экземпляр мастера PostgreSQL и запускает его в режим реплики. В худшем случае, если на лидере кластер PostgreSQL с другим system ID (создан заново `initdb`), процесс Patroni выдаст ошибку уровня `CRITICAL` и **остановится**:

```
 WARNING: Etcd Cluster ID changed from 11470682518379718621 to
17572659380995478511
 INFO: Demoting self (immediate-nolock)
 INFO: starting after demotion in progress # запуск экземпляра PostgreSQL
 CRITICAL: system ID mismatch, node tantor1 belongs to a different cluster:
7680106658646956423 != 7680105675647060161
```

Если перезапуск в службе systemd не установлен, то Patroni не запустится.

Восстановление при потере кластера etcd

- Пример удаления и создания нового кластера etcd:

```
sudo killall etcd
rm -rf /opt/tantor/var/lib/etcd/a.etcd
etcd --name=a --data-dir=/opt/tantor/var/lib/etcd/a.etcd --listen-client-
urls http://127.0.0.1:2379 --advertise-client-urls http://127.0.0.1:2379 --
listen-peer-urls http://127.0.0.1:2380 --initial-advertise-peer-urls
http://127.0.0.1:2380 --initial-cluster=a=http://127.0.0.1:2380 --heartbeat-
interval=1000 --election-timeout=10000 --snapshot-catchup-entries=100 --
auto-compaction-retention=10s --quota-backend-bytes=1000000000 --log-format
console --initial-cluster-token NEW
```

- При пересоздании кластера etcd нужно обязательно сменить идентификатор кластера
  - использовать новое значение `initial-cluster-token`
  - изменённое значение в списке `initial-cluster`

Восстановление при потере кластера etcd

Кластер etcd может быть полностью потерян, пересоздан или восстановлен из снэпшота.
Пример удаления и создания нового кластера `etcd`:

```
sudo killall etcd
rm -rf /opt/tantor/var/lib/etcd/a.etcd
etcd --name=a --data-dir=/opt/tantor/var/lib/etcd/a.etcd --listen-client-urls
http://127.0.0.1:2379 --advertise-client-urls http://127.0.0.1:2379 --listen-
peer-urls http://127.0.0.1:2380 --initial-advertise-peer-urls
http://127.0.0.1:2380 --initial-cluster=a=http://127.0.0.1:2380 --heartbeat-
interval=1000 --election-timeout=10000 --snapshot-catchup-entries=100 --auto-
compaction-retention=10s --quota-backend-bytes=1000000000 --log-format console
--initial-cluster-token NEW
```

NEW - любое значение, не использовавшееся ранее. Это значение **должно быть одинаковым у всех узлов кластера etcd**, иначе узел не сможет присоединиться к кластеру etcd и в логе будет выдавать предупреждения:

```
warn rafthttp/http.go:512   request cluster ID mismatch
warn rafthttp/stream.go:658 request sent was ignored by remote peer due to
cluster ID mismatch
```

Если **Cluster ID** поменялся, то **RaftTerm** игнорируется и Patroni взаимодействует с таким кластером etcd.

Поэтому, при восстановлении кластера etcd, нужно, чтобы Cluster ID стал другим, а не был тем, который был у существовавшего ранее кластера etcd.

Восстанавливать данные ключей в кластере etcd необязательно и не нужно, так как лидер Patroni их сам восстановит из своей памяти или из файла `PGDATA/patroni.dynamic.json` кластера PostgreSQL, которым он управляет, или раздела `bootstrap.dcs` файла параметров `patroni.yml`, с которым Patroni был запущен.

Standby кластер Patroni

- Кластер Patroni не с мастером, а с репликой, которая будет получать WAL из другого кластера Patroni и распространять их своим followers
- Используется, если:
  - второй центр обработки данных
  - нужен кластер PostgreSQL с задержкой в применении WAL
- Для создания клона (лидера резервного кластера) нужно:
  - в `PGDATA` основного кластера присутствовал файл `postgresql.conf` или `postgresql.conf.backup`
  - в конфигурации основного кластера нужно определить постоянный слот репликации:

```
bootstrap:
 dcs:
  slots:
   standby1:
    type: physical
```


Standby кластер Patroni

Можно создать кластер Patroni не с мастером, а с репликой, которая будет получать WAL из другого кластера Patroni. Это используется, если:

1) есть второй центр обработки данных

2) нужен кластер PostgreSQL с задержкой в применении WAL.

Лидер резервного кластера ведёт себя так же, как и обычный лидер основного кластера, за исключением того, что лидер резервного кластера получает журналы с узла другого кластера. Лидер резервного кластера имеет followers, которым передаёт журналы. Получается каскадная репликация.

Лидер резервного кластера может реплицировать журналы не только с мастера основного кластера, но и с follower основного кластера или даже узла другого резервного кластера. В этом случае, на резервном кластере не сможет работать `pg_rewind`. Хост, с которого лидер резервного кластера получает журналы, задаётся в ключе `standby_cluster.host`.

Лидер резервного кластера удерживает и обновляет лизинг ключа лидера в DCS. Если лизинг истёк, то followers резервного кластера выбирают нового лидера.

У резервного и основного кластера свои записи в DCS (более того, скорее всего это два разных DCS) и через DCS резервный и основной кластер не общаются. Связь идёт только по репликационному протоколу от лидера резервного кластера до лидера основного кластера.

На основном кластере резервный кластер не отображается в выводе команд `patronictl list` и `patronictl topology`.

Для успешного создания клона (лидера резервного кластера) нужно, чтобы в `PGDATA` узла основного кластера, с которого будет создаваться клон, присутствовал файл `postgresql.conf` или `postgresql.conf.backup`.

В конфигурации основного кластера обязательно определить постоянный слот репликации:

```
bootstrap:
 dcs:
  slots:
   standby1:
    type: physical
```

Standby кластер не хранит историю переключений (**`patronictl history`** должна быть пуста). На standby кластере стоит отключить проверку timeline:

`patronictl -c patroni.yml edit-config --set check_timeline=null`

так как если история не пуста (мусор в DCS), её наличие не даст выполнить switchover.

Standby кластер Patroni

- Кластер Patroni не с мастером, а с репликой, которая будет получать WAL из другого кластера Patroni
- Используется, если:
  - второй центр обработки данных
  - нужен кластер PostgreSQL с задержкой в применении WAL

```
name: standby1 #должно быть уникально во всех кластерах: основном и резервном
bootstrap:
 dcs:
  standby_cluster: #если есть раздел, то инициализировать standby кластер
  check_timeline: false
   host: 127.0.0.1 #можно указать несколько узлов основного кластера
   port: 5432
   primary_slot_name: standby1 #по умолчанию, берётся из ключа name
   create_replica_methods: basebackup #как клонировать мастера из другого
кластера Patroni
#    restore_command: #если есть архив журналов
   archive_cleanup_command: #если нужно очищать архив журналов
   recovery_min_apply_delay:   #задержка в применении WAL
```


Standby кластер Patroni

Пример раздела `standby_cluster` в `patroni.yml` лидера резервного кластера:

```
name: standby1 #должно быть уникально во всех кластерах: основном и резервном
bootstrap:
 dcs:
  standby_cluster: #инициализировать standby кластер, а не основной
   host: 127.0.0.1 #можно указать несколько узлов основного кластера через запятую,
```

тогда Patroni добавляет в параметр лидера standby свойство `target_session_attrs=read-write`, которое указывает `libpq` standby лидера установить соединение с текущим мастером. Мастер может меняться, standby лидер подсоединится к новому мастеру.

```
   port: 5432
   primary_slot_name: standby1 #по умолчанию, берётся из ключа name
   create_replica_methods: #порядок, в котором Patroni будет инициализировать
```

реплику-лидера (создавать клон) из другого кластера Patroni. Список может отличаться от списка, который устанавливается в одноимённом ключе раздела `postgresql`. Если раздел отсутствует, то используется `basebackup`. В списке можно ссылаться на записи из раздела `postgresql`. Также можно определить методы в этом разделе.

```
#   - walg
   - basebackup #можно не указывать, он всегда есть и используется, если другие
```

методы не смогли создать работающую реплику

```
#   walg:
#   command: "wal-g backup-fetch $PGDATA LATEST"
#   no_leader: 1 #клонировать даже, если ни один экземпляр не работает. Patroni
```

определяет узел, с которого будет снят бэкап по `tags.clonefrom: true`

```
   basebackup:
   - checkpoint: 'fast'
#   - verbose
#   - slot: tantor1
#   - waldir: /var/lib/postgresql/tantor-se-18/wal
   restore_command: 'команды' #только, если есть архив журналов
   archive_cleanup_command: #если нужно очищать архив журналов
   recovery_min_apply_delay: 86400000 #задержка в применении WAL, миллисекунды
```

Promote standby кластера Patroni

- Автоматический failover и switchover на узлы standby кластера не выполняются
- проверить Lag лидера standby кластера

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml list
+ Cluster: a (7682673136434780655) -+------------+-----+-----------+-----+
| Member  | Role    | State     | TL | Receive LSN| Lag | Replay LSN| Lag |
+---------+---------+-----------+----+------------+-----+-----------+-----+
| tantor2 | Replica | streaming|  4 |  0/C761490 |   9 | 0/A51B1D8 |  43 |
+---------+---------+-----------+----+------------+-----+-----------+-----+
```

- или запросом на мастере:

```
select application_name name, state, sent_lsn, write_lsn, replay_lsn, now()-
reply_time receive_lag from pg_stat_replication;
 name   |   state   |  sent_lsn  | write_lsn  | replay_lsn |   receive_lag
-------+-----------+------------+------------+------------+-----------------
tantor2| streaming | 0/19E3DC78 | 0/19594B80 | 0/15449300 | 00:00:23.937738
```

- убедиться, что основной кластер не доступен клиентам

Promote standby кластера Patroni

Автоматический failover и switchover на узлы standby кластера не выполняются. Их нужно выполнять вручную.

Сначала проверить не отстаёт ли лидер standby кластера от мастера. Кластеры с отложенным применением журналов будут отставать на `recovery_min_apply_delay`.

Для проверки на основном и резервном кластерах можно выполнить команды:

```
patronictl -c patroni.yml list
```

и сравнить значения в столбцах `Receive LSN`, **`Lag`**, `Replay LSN`, **`Lag`**.

Одинаковые названия столбцов **`Lag`** сделаны для более компактного отображения.

Если мастер обслуживает приложения, то различие будет, но отставание в столбце `Lag` не должно быть большим:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml list
+ Cluster: a (7682673136434780655) -+------------+-----+-----------+-----+
| Member  | Role    | State     | TL | Receive LSN| Lag | Replay LSN| Lag |
+---------+---------+-----------+----+------------+-----+-----------+-----+
| tantor1 | Leader  | running   |  4 |            |     |           |     |
+---------+---------+-----------+----+------------+-----+-----------+-----+
| tantor2 | Replica | streaming|  4 |  0/C761490 |   9 | 0/A51B1D8 |  43 |
+---------+---------+-----------+----+------------+-----+-----------+-----+
```

Значения **`Lag`** выдаются в секундах и обновляются с частотой цикла Patroni `loop_wait`.

Также можно получить **`Lag`** на основном мастере можно выполнить запрос:

```
select application_name name, state, sent_lsn, write_lsn, replay_lsn, now()-
reply_time receive_lag from pg_stat_replication;
 name   |   state   |  sent_lsn  | write_lsn  | replay_lsn |   receive_lag
-------+-----------+------------+------------+------------+-----------------
tantor2| streaming | 0/19E3DC78 | 0/19594B80 | 0/15449300 | 00:00:23.937738
```

Лидер standby может не успевать принимать или накатывать журналы из-за большого объема изменений и можно остановить обслуживание клиентов мастером.

Затем убедиться, что основной кластер не доступен для клиентов и не станет доступен (например, остановлен и не запустится), иначе возникнет split brain.

Promote standby кластера Patroni

Способы:

- убрать свойство `standby_cluster`:

```
patronictl -c patroni.yml edit-config --set standby_cluster=null --force
 use_slots: true
primary_start_timeout: 1200
retry_timeout: 3
-standby_cluster:
- host: 127.0.0.1
- port: 5432
- primary_slot_name: standby1
ttl: 20
```

- Выполнить команду:

```
patronictl -c patroni.yml promote-cluster --force
Current cluster topology
```
Покажет таблицу в формате list
```
cluster is unlocked: False, leader role: standby_leader, leader state: running
2026-09-07 17:26:01 cluster is successfully promoted
```
Покажет таблицу, в которой изменятся атрибуты: `TL -> TL+1`, `Standby Leader -> Leader`

- Лидер standby кластера станет мастером

Promote standby кластера Patroni (продолжение)

После проверок, выполнить promote (сделать из реплики мастер) можно одним из двух способов:

1) убрать свойство **`standby_cluster`**:

`patronictl -c patroni.yml edit-config --set standby_cluster=null --force`

Лидер standby кластера станет мастером.

2) Выполнить команду:

`patronictl -c patroni.yml promote-cluster --force`

```
Current cluster topology
```
Покажет таблицу в формате команды `list`
```
2026-09-07 17:25:59 cluster is unlocked: False, leader role: standby_leader,
leader state: running
2026-09-07 17:26:01 cluster is successfully promoted
```
Покажет таблицу, в которой изменятся атрибуты:

`TL -> TL+1 Standby, Leader -> Leader`

Выполнить demote можно командой:

`patronictl -c patroni.yml demote-cluster --host 127.0.0.1 --port 5431 --force`

```
Current cluster topology
2026-09-07 17:25:23 cluster is successfully demoted
```

В команде обязательно указать имена хостов основного кластера и порт. Вместо них можно указать `--restore-command`, но это для передачи журналов командой, обычно, из архива журналов.