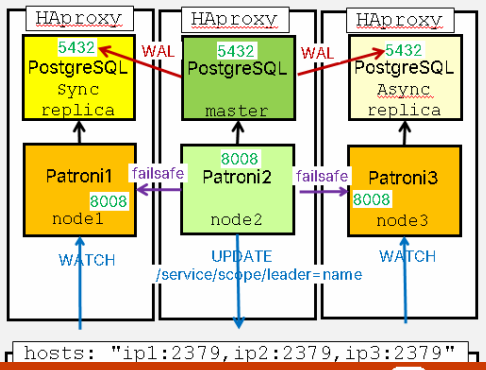
5

Передача PostgreSQL под управление Patroni

Patroni

- Patroni - программа, которая автоматизирует создание, пересоздание, активацию реплик кластера баз данных PostgreSQL
  - › использует внешнюю распределённую базу данных (Distributed Configuration Store, DCS)
  - › не стоит хранить в кластере etcd данные других приложений
- **Кластер Patroni** - мастер PostgreSQL и одна или более физических реплик



Patroni

Patroni - программа, которая автоматизирует создание, пересоздание, активацию реплик кластеров баз данных PostgreSQL. Является стандартом де-факто по созданию и обслуживанию высокодоступных (Highly Available, HA) конфигураций кластеров баз данных PostgreSQL. Patroni написан на языке Python. Разрабатывается с 2015 года, версия 1.0 вышла 5 июля 2016 года. Автор - Александр Кукушкин.

Кластер Patroni - это мастер PostgreSQL и одна или более физических реплик PostgreSQL, находящиеся под управлением Patroni. Кластер баз данных PostgreSQL можно ввести под обслуживание Patroni без простоя. Patroni:

1) перед продвижением реплики убеждается, что мастер остановлен

2) перед продвижением реплики убеждается, что транзакции не будут потеряны или продвигает наименее отставшую от мастера реплику

3) возвращает в строй или пересоздаёт бывшего мастера и сбойнувшие реплики.

Patroni наблюдает за доступностью мастера, гарантирует отсутствие split brain, минимизирует простой, немедленно продвигая одну из реплик до мастера.

Для создания кластера Patroni нужно иметь DCS - распределённую базу данных (**Distributed Configuration Store**, **DCS**). В Patroni версии 2.0 был встроен DCS (к нему относится скрипт `patroni_raft_controller`), позволяющий обойтись без внешнего DCS. В версии 3.0 его пометили как support deprecated, то есть он не будет дорабатываться, хотя его код постараются переносить на новые версии. Стандартом де-факто для DCS является etcd. Patroni не сильно нагружает etcd и один кластер etcd может обслуживать сотни и тысячи кластеров Patroni.

Один кластер etcd может хранить даннве многих приложний. Нпример, etcd может успешно хранить данные ~5000 узлов Kubernetes. Однако, желательно, чтобы для Patroni использовался выделенный кластер etcd, так как:

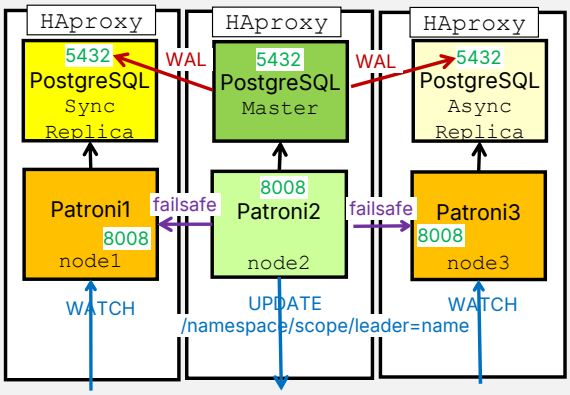
1) приложения могут создавать пиковые нагрузки и хранить ключи большого размера, уменьшая отзывчивость лидера etcd

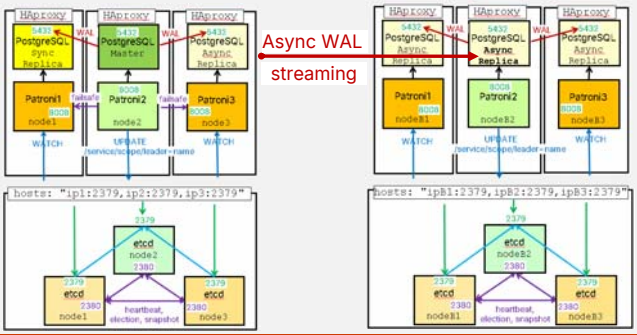
2) Patroni нечувствителен к потере всех данных в etcd. Можно пересоздать кластер etcd и Patroni автоматически загрузит в новый кластер etcd свои данные. Другие же приложения могут потребовать восстановления своих данных, а они восстанавливаются только целиком.

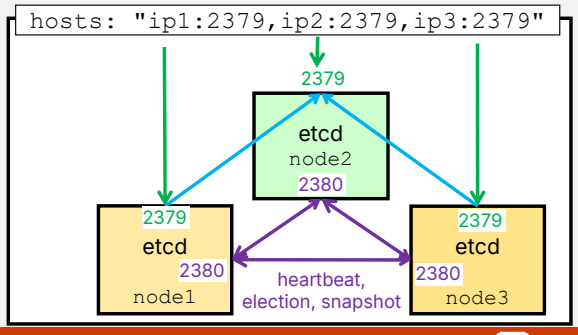
https://habr.com/ru/articles/504044/

Топология

- в одном DataCenter:
  - синхронная реплика
  - узлы Patroni + узлы etcd
- несколько DataCenters связаны:
  - асинхронной каскадной репликацией
  - переключение вручную







Топология

Узел, node, член кластера Patroni, local member, Patroni member - синонимы.

Primary, основной сервер, мастер - синонимы. Мастер всегда является лидером.

Standby, резервный сервер, реплика, secondary, follower Patroni - синонимы.

Лидер Patroni - экземпляр, владеющий ключом лидера в DCS. Обычно, ключом лидера владеет мастер, но в конфигурации с несколькими центрами обработки данных (multi datacenter, multi DC), в каждом datacenter может быть свой кластер DCS. В первом datacenter работает мастер, во втором (и остальных) datacenter работает реплика, которая обладает ключом лидера. Эта реплика может распространять WAL другим репликам в своём кластере Patroni (обычно, в своём datacenter), то есть использовать каскадную репликацию.

Каскадную репликацию можно настроить и в одном datacenter.

Продвижение реплики-лидера во втором (или других) datacenter выполняется вручную. Конфигурации Patroni+DCS в разных datacenters друг от друга изолированы. Patroni первого datacenter не знает о существовании других datacenters. При ручном продвижении реплики обязательно убедиться, что в первом datacenter нет мастера и он не появится в случае восстановления работоспособности первого datacenter, иначе получится split brain. Если есть более, чем два datacenters, то нужно будет вручную выбрать реплику-лидера в одном из запасных datacenters, которая меньше других отстала от мастера, то есть приняла более свежие журнальные записи.

Вместо конфигурации нескольких datacenters, в каждом из которых свой кластер etcd, можно расположить в нескольких datacenters (физически разнесённых местах) один кластер etcd. В этом случае, нужно иметь три datacenters, двух недостаточно для того, чтобы при выходе из строя одного из datacenter, во втором был кворум и кластер etcd продолжал работать. При такой конфигурации, реплики могут быть любые и в любой комбинации: синхронные и асинхронные.

Топология Patroni с несколькими datacenters описана в документации:
https://patroni.readthedocs.io/en/latest/ha_multi_dc.html

Настройка каскадирования описана в:
https://patroni.readthedocs.io/en/latest/standby_cluster.html

При создании реплики в другом datacenter утилитой `pg_basebackup` (`create_replica_methods: basebackup`), Patroni ожидает, что файл `postgresql.conf` или `postgresql.conf.backup` мастера находятся в `PGDATA` мастера. Если они находятся в другом месте, их нужно скопировать в `PGDATA`.

Установка

- С версиями Python 3.11 и новее, рекомендуется использовать Patroni 4.1.1 и более новые
- В пакет Patroni входят:
  - скрипт `patroni`
  - утилита `patronictl`, может использоваться для управления кластером Patroni вместо Patroni REST API
  - скрипт `patroni_raft_controller` - встроенный DCS, deprecated, вместо него рекомендуется использовать `etcd`
  - скрипты для работы с программами резервирования PostgreSQL: WAL-G, Barman, Amazon Web Services
  - файл службы для systemd: `/lib/systemd/system/patroni-tantor.service`

```
apt install libtk
dpkg -i python3-tantor-all_3.12*.deb
dpkg -i patroni-tantor-all_4.1.4*.deb
```


Установка

Установка Patroni:

apt install libtk

dpkg -i python3-tantor-all_3.12.13-1astra1.8-1_amd64.deb

Зависит от библиотек: `bz2, libc, crypt, ffi, gcc, gdbm-compat, gdbm, lzma, ncursesw6, nsl2, readline8, sqlite3, ssl, tcl, tinfo, tirpc, tk, uuid, zlib`

dpkg -i patroni-tantor-all_4.1.4-1astra1.8-1_all.deb

Пакет `patroni-tantor-all` зависит от пакета `python3-tantor-all`.

В Python, начиная с версии 3.11, есть ошибка, которая приводит к тому, что при отключённом оверкоммите `vm.overcommit_memory=2` (рекомендуемая настройка для PostgreSQL) Patroni REST API перестаёт отвечать. С версиями Python 3.11 и новее, рекомендуется использовать Patroni 4.1.1 и более новые. Новые версии запускают потоки заранее, при запуске скрипта, и вероятность столкнуться с нехваткой памяти для запуска потока меньше. Также рекомендуется, при работе с `vm.overcommit_memory=2`, запускать скрипт `patroni` с переменными окружения: `MALLOC_ARENA_MAX=1` и `PG_MALLOC_ARENA_MAX=`. (при звапуске экземпляра PostgreSQL устанавливает значение по умолчанию, а не наследует `MALLOC_ARENA_MAX`). В пакет Patroni входят:

1) `patroni` - скрипт Patroni

2) `patronictl` - утилита управления кластером Patroni через REST API

3) `patroni_raft_controller` - для запуска наблюдателя для встроенного DCS, если узлов с Patroni меньше 3. Вместо встроенного DCS рекомендуется использовать `etcd`

4) исторически написанные скрипты для параметра **`create_replica_methods`**: `patroni_wale_restore`, `patroni_barman`, `patroni_aws`.

5) файл службы `/lib/systemd/system/patroni-tantor.service`, запускающий скрипт под пользователем `postgres`. Таймаут на запуск 30 секунд не имеет значения, так как `Type=simple`. По умолчанию, рестарт скрипта в случае сбоя отключён. Служба использует конфигурационные файлы с названиями:
`/opt/tantor/etc/patroni_env.conf` и `/opt/tantor/etc/patroni/patroni.yml`

После установки, можно добавить в файл профиля пользователя `postgres` путь к исполняемым файлам:

sed -i 's|PATH=/opt/tantor/db|PATH=/opt/tantor/usr/bin:/opt/tantor/db|' /var/lib/postgresql/.bash_profile

Локальная конфигурация

- Локальная конфигурация:
  - Переменные окружения
    - имеют префикс `PATRONI_`
  - Файл в формате YAML
    - перекрывают динамическую конфигур
    - можно вставить в переменную окружения `PATRONI_CONFIGURATION` одной строкой
- Переменные окружения перекрывают и локальную и динамическую конфигурацию

```
export export PATRONI_NAME=tantor1
export PATRONI_SCOPE=a
export PATRONI_ETCD3_HOST="127.0.0.1:2379"
export PATRONI_RESTAPI_CONNECT_ADDRESS="127.0.1.1:8008"
patronictl list
patronictl topology
```


Локальная конфигурация

Конфигурация Patroni располагается в трёх местах:

1) "Динамическая конфигурация" (глобальная конфигурация) - в DCS, куда изначально (при `bootstrap`, то есть развёртывании Patroni) загружаются из раздела `bootstrap.dcs` файла `patroni.yml`

2) "Local configuration" - в файле в формате YAML или директории с файлами в формате YAML. Применяется к одному узлу, перекрывает динамическую конфигурацию.

Путь к локальной конфигурации передаётся Patroni как параметр:

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

Если вместо файла указана директория, то будут использоваться все YAML-файлы, в порядке сортировки по названиям файлов. Если ключ (параметр в YAML) присутствует в нескольких файлах, превалирует ключ из последнего файла. Использование директории усложняет конфигурирование, лучше использовать один файл параметров.

3) В переменных окружения так же, как это сделано в **etcd**.

Переменные окружения перекрывают и локальную и динамическую конфигурацию.

Переменные окружения имеют префикс `PATRONI_`.

Содержимое файла параметров YAML можно вставить в переменную окружения `PATRONI_CONFIGURATION` одной строкой. Если эта переменная окружения не пустая, то игнорируются все остальные переменные окружения, специфичные для Patroni.

Переменные окружения полезны, если часть параметров неизвестна и их удобнее передать переменными окружения. Например, внешний IP-адрес, при запуске Patroni внутри docker-контейнера нельзя узнать, но можно передать внутрь контейнера переменной окружения, установив её в файле `docker-compose.yml`.

Расположение файлов, по умолчанию, указывается в файле службы `/lib/systemd/system/patroni-tantor.service`, это:

`/opt/tantor/etc/patroni_env.conf` и `/opt/tantor/etc/patroni/patroni.yml`

Список переменных окружения Patroni:

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

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

Список открытых багов Patroni:

https://github.com/patroni/patroni/issues

Создание файла локальной конфигурации

- Добавить в PATH директорию со скриптами Patroni
- Создать директорию для файла конфигурации Patroni
- Patroni запускается под пользователем postgres
- Создать файл конфигурации Patroni, подсоединившись к работающему экземпляру PostgreSQL

```
echo $PATH /opt/tantor/usr/bin:/opt/tantor/db/18/bin:/usr/local/bin:/usr/bin:/bin
sudo mkdir /opt/tantor/etc/patroni
sudo chown postgres:postgres /opt/tantor/etc/patroni
patroni --generate-sample-config > /opt/tantor/etc/patroni/patroni-sample.yml
su - postgres -c 'PGPORT=5432 PGPASSWORD=postgres patroni --generate-config >
/opt/tantor/etc/patroni/patroni.yml'
```

- После редактирования файла конфигурации перечитать:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml reload a --force
+ Cluster: a (7676506443451906864) ------+----+-------------+-----+
...
Reload request received for member tantor1 and will be processed within 10 seconds
```

  - › применятся параметры, кроме раздела `bootstrap.dcs`

Создание файла локальной конфигурации

Проверить, что в пути указана директория с исполняемыми скриптами Patroni:

```
echo $PATH
/opt/tantor/usr/bin:/opt/tantor/db/18/bin:/usr/local/bin:/usr/bin:/bin
```

Создать директорию для файла Patroni:

```
sudo mkdir /opt/tantor/etc/patroni
```

Дать разрешение пользователю postgres на директорию или сделать его владельцем:

```
sudo chown postgres:postgres /opt/tantor/etc/patroni
```

Можно создать заготовку файла локальной конфигурации Patroni командой:

```
patroni --generate-sample-config > /opt/tantor/etc/patroni/patroni.yml
```

или, если есть работающий экземпляр PostgreSQL, то лучше использовать команду:

```
su - postgres -c 'PGPORT=5432 PGPASSWORD=postgres patroni --generate-config >
/opt/tantor/etc/patroni/patroni.yml'
```

тогда будут добавлены параметры конфигурации работающего PostgreSQL.

Адрес экземпляра будет браться из стандартных переменных окружения libpq:
`PGHOSTADDR`, `PGHOST`, **`PGPORT`**, `PGDATABASE`, `PGUSER`, **`PGPASSWORD`**.

В будущем, после внесения изменений в файл, чтобы изменения были применены работающим Patroni, файл можно перечитать командой:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml reload a --force
+ Cluster: a (7676506443451906864) ------+----+-  +----------------------+
| Member  | Host       | Role   | State   | TL |...| Tags                 |
+---------+------------+--------+---------+----+-  +----------------------+
| tantor1 |127.0.0.1:  | Leader | running |  2 |...| clonefrom: true      |
|         |      5432  |        |         |    |   | failover_priority: 1 |
|         |            |        |         |    |   | sync_priority: 1     |
+---------+------------+--------+---------+----+-  +----------------------+
Reload request received for member tantor1 and will be processed within 10
seconds
```

10 секунд - это значение определяется `loop_wait`.

Разделы файла локальной конфигурации

- На корневом уровне файла `patroni.yml`:

`scope: 'b'` имя кластера Patroni, одинаковое на всех узлах
`name: tantor1` имя хоста, уникально среди узлов кластера Patroni
`namespace: '/service'` префикс названия ключей в DCS, если в DCS несколько кластеров
`thread_pool_size: 5`
`thread_stack_size: 524288`

- разделы файла:

`restapi:` параметры конечной точки, на которой Patroni отвечает и управляется по REST
`etcd3:` адрес ETCD
`bootstrap:` параметры для инициализации кластера
`postgresql:` параметры как работать с экземпляром PostgreSQL
`tags:` несколько общих параметров
`log:` параметры файла лога, его размера, ротации

- редко используются:

`ctl:` (опционально) раздел для `patronictl`, если в **`restapi`** заданы имя+пароль или TLS
`watchdog:` по умолчанию включится, если есть устройство `/dev/watchdog`
`raft:` параметры встроенного DCS вместо ETCD. Лучше использовать ETCD

Разделы файла локальной конфигурации

Пример файла https://github.com/patroni/patroni/blob/master/postgres0.yml
Параметры описаны в https://patroni.readthedocs.io/en/latest/yaml_configuration.html
Параметров больше сотни, они перечислены в документации в алфавитном порядке, а не в порядке важности.
Полный список параметров и допустимые значения можно посмотреть в исходном коде: https://github.com/patroni/patroni/blob/master/patroni/validator.py
На корневом уровне файла YAML есть общие параметры (Global/Universal):
`scope: 'a'` - имя кластера Patroni, одинаковое на всех узлах
`namespace: '/service'` - префикс (путь, namespace) имени ключа в DCS
`name: tantor1` - имя хоста, на котором будет работать Patroni, должно быть уникально в пределах кластера Patroni
`thread_pool_size : 5` - число потоков, используемых Patroni для асинхронного обращения к Patroni на других узлах по протоколу REST. Рекомендуется увеличить, при увеличении числа узлов Patroni.
`thread_stack_size: 512kB` размер стека потока. Значение должно быть кратно `64kB`. До **версии 4.1.5** был баг, который не позволял устанавливать значение параметра - проверялось, что значение равно 64Кб-1, а не 64Кб. При уменьшении значения, уменьшается использование памяти, но увеличивается вероятность падения Patroni с ошибкой нехватки памяти в стеке потока.
В раздел `tags:` можно добавить произвольные `ключ: "значение"`, они будут выдаваться в `patronictl list` и по REST, смысла использовать нет.
Утилите `patronictl` нужны параметры для подсоединения по REST к Patroni. Утилита использует параметры трёх разделов: `restapi`, `etcd3`, `ctl`. Если есть раздел `ctl`, то параметры аутентификации и TLS берутся из него. Если раздел `ctl` отсутствует, то параметры берутся из раздела `restapi`. **Если есть раздел `restapi`, раздел `ctl` не нужен**.

Динамическая конфигурация

- хранится в DCS (etcd) в ключе **/service/a/config**

```
etcdctl --endpoints=:2379 get '/service/a/config' --print-value-only | jq
```

- загружается в DCS при создании кластера из раздела `bootstrap.dcs` локальной конфигурации
  - › после загрузки, раздел `bootstrap.dcs` не используется
- После загрузки, параметры меняются утилитой:

```
patronictl -c patroni.yml edit-config --set CONFIG="VALUE" --force
```

- Если `VALUE=null`, то параметр удаляется:

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

- Просмотр параметров динамической конфигурации:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml show-config
```


Динамическая конфигурация

"Динамическая конфигурация" (синоним "глобальная конфигурация") - параметры, применяющиеся ко всем членам кластера Patroni. Эти параметры хранятся в DCS.

Изначально, параметры динамической конфигурации загружаются в DCS из раздела `bootstrap.dcs` файла конфигурации лидером кластера Patroni. В дальнейшем, они могут меняться через REST API (`patronictl`)и восстанавливаться лидером кластера Patroni.

Просмотр параметров в ETCD в ключе `bootstrap.dcs` в формате YAML:

patronictl -c /opt/tantor/etc/patroni/patroni.yml show-config

`-c` указывает путь конфигурационному файлу, где указан адрес DCS.

Просмотр параметров в ETCD в ключе `bootstrap.dcs` в формате JSON:

etcdctl --endpoints=:2379 get '/service/a/config' --print-value-only | jq

Можно интерактивно отредактировать параметры в текстовом редакторе `nano`:

patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config

Сохранить **F2** или **<ctrl+x>**, подтвердить название временного файла **y <ENTER> y**.

Можно интерактивно отредактировать параметры в текстовом редакторе `mcedit`:

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

Сохранить **F2 <ENTER> F10 y <ENTER>**.

Можно поменять один или несколько параметров:

patronictl -c patroni.yml edit-config --set CONFIG="VALUE" [, ...]

`CONFIG` - имя параметра в YAML, путь к параметру задаётся точкой. Пример:

patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config --set postgresql.parameters.logging_collector=true

Если **CONFIG=null**, то параметр удаляется. Пример:

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

Опция **--force** не запрашивает подтверждения, удобно для скриптов.

Также есть опции **edit-config -apply файл.yml** и **-replace файл.yml** или **edit-config --force -apply - < файл.yml**. Файл должен быть в формате, который выдаёт **show-config**, иначе загрузится мусор. Этот мусор можно будет удалить, указав файл с правильным содержимым опцией **edit-config -replace файл.yml**. Перед изменением DCS, утилита показывает в формате `diff`, какие параметры она добавит или удалит и запросит подтверждение.

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

Параметры динамической конфигурации

- Три параметра зависят друг от друга:
  - **loop_wait + 2 * retry_timeout <= ttl**

```
bootstrap:
 dcs:
  ttl: 20 #min=20 #время жизни ключа лидера в DCS
  loop_wait: 1 #min=1 интервал циклов Patroni в секундах
  retry_timeout: 3 #min=3 таймаут на ответ от DCS или PostgreSQL
  check_timeline: true #реплика с более старым timeline не станет кандидатом
  failsafe_mode: true #мастер не остановится при сбое DCS
  maximum_lag_on_failover: 1048576 #байт отставания реплики при failover на неё
  maximum_lag_on_syncnode: -1
  max_timelines_history: 100 #история переключений (timelines), хранимая в DCS
  member_slots_ttl: 30min #сохранять слоты недоступных узлов, единицы: min h d
  primary_start_timeout: 300 #время на запуск мастера после сбоя до выполнения failover
  primary_stop_timeout: 300 #только для синхронного режима
  pause: true #режим работы после ввода под обслуживание Patroni
  synchronous_mode: quorum #включение синхронного режима
  synchronous_mode_strict: true #commit после получения репликой LSN
  synchronous_node_count: 1 #min=1 число реплик, где должны сохраниться LSN для commit
```


Параметры динамической конфигурации

Раздел `bootstrap` (развёртывания, инициализации) кластера Patroni.

`bootstrap:`

`dcs:` - основной раздел. В нем перечисляют все параметры, общие для экземпляров PostgreSQL. Их список: https://patroni.readthedocs.io/en/latest/dynamic_configuration.html

`ttl=30`, минимум **`20`**. Время жизни ключа лидера кластера Patroni в DCS. Через какое время начнутся выборы нового мастера и его продвижения (promote), если лизинг не будет продлён

`loop_wait=10`, минимум **`1`**, пауза между циклами активности Patroni.

`retry_timeout=10`, минимум **`3`**, таймаут на ответ от DCS или PostgreSQL. При превышении, мастер будет остановлен (если только не используется `failsafe_mode: true`), запущен в read only и начнётся переключение на реплику. К узлам etcd запросы отправляются последовательно с интервалом в секунду.

Значения трёх параметров должны удовлетворять условию:

`loop_wait + 2 * retry_timeout <= ttl`

`check_timeline: false`, если **`true`**, то реплика с более старым timeline, чем у прежнего лидера, не будет участвовать в выборах. Параметр появился в версии Patroni 1.5.4, но не был включён по умолчанию. Если реплика станет лидером, то экземпляры с более новым timeline не смогут запуститься и reinit не поможет. Поможет установка `remove_data_directory_on_rewind_failure: true` и `remove_data_directory_on_diverged_timelines: true` в разделе `postgresql`.

Однако, из-за мусора в DCS, **`true`** может не давать выполнить switchover даже, когда **`patronictl list`** не выдаёт ошибок. Чтобы выполнить switchover понадобится **убрать** параметр или установить его в **`false`**:

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

`failsafe_mode=false`, **рекомендуется установить `true`**. Доступен с **версии 3.0**. Если DCS неработоспособен (например, нет кворума), то мастер не сможет обновить лизинг и будет остановлен, что приведёт к недоступности PostgreSQL. `failsafe_mode=true` позволяет мастеру продолжать работать, но только пока Patroni узла мастера может связаться с Patroni реплик. В режиме `failsafe` нельзя выполнять переконфигурирование (unsafe endpoints), нельзя использовать `patronictl`. Можно использовать только чтение статуса через REST API. Переконфигурировать можно будет, когда DCS заработает или будет пересоздан.
https://patroni.readthedocs.io/en/latest/dcs_failsafe_mode.html

Параметры раздела `etcd3`

- раздел задаёт адрес внешнего DCS - etcd
- Один из трёх параметров:
- в `host`, `url` можно указать только один адрес
- в `hosts` один или несколько

```
etcd3:
 protocol: http
 hosts: "127.0.0.1:2379,127.0.0.2:2379,127.0.0.3:2379"
или
 hosts:
  - 127.0.0.1:2379
  - 127.0.0.2:2379
или
 host: 127.0.0.1:2379
или
 url: http://127.0.0.1:2379
```

- если в etcd нужно аутентифицироваться, то указать:

```
 username: u
 password: p
```


Параметры раздела `etcd3`

В файле параметров нужно указать адрес DCS. Для этого нужно добавить один из разделов, задающих адрес - **`etcd3`** или **`raft`** или другие (zookeeper, kubernetes, consul).

Основной параметр это адрес узла или узлов etcd. Адрес может быть задан: `host`, `url`, в которых можно указать только один адрес, что не удобно. Поэтому, лучше использовать `hosts`, в котором можно задать один или несколько адресов `etcd`.

```
etcd3:
 hosts: "127.0.0.1:2379,127.0.0.2:2379"
```
или
```
 hosts:
  - 127.0.0.1:2379
  - 127.0.0.2:2379
```
или
```
 host: 127.0.0.1:2379
```
или
```
 url: http://127.0.0.1:2379
```

`protocol: http` - можно ещё указать `https`. параметр опционален. Вместе с `https` в этом разделе придётся указать параметры `cacert`, `cert`, `key`. **Смысла использовать `https` нет**, он увеличивает задержки и приводит к сбоям при проблемах с сертифкатами. Если в etcd нужно аутентифицироваться, то имя и пароль указываются параметрами:

```
 username: u
 password: p
```

Параметр retry_timeout и etcd3.hosts

- Ожидание ответа от одного узла - не меньше 1 секунды
- Число запросов к одному узлу etcd за **retry_timeout**:

```
max_retries = 4 - min(count(etcd.hosts), 3)
```

  - если в **etcd3.hosts** указан 1 узел, то 3 запроса к нему
  - если в **etcd3.hosts** 2 узла, то максимум 2 попытки на узел
- Если число узлов etcd3.hosts>=3, а retry_timeout>=count(etcd3.hosts), то каждому узлу будет по очереди посылаться один запрос с таймаутом на получение ответа retry_timeout/etcd3.hosts секунд
  - если число узлов 1 или 3, то будет 3 запроса с интервалом 3.3 секунды

```
etcd3:
 hosts: "127.0.0.1:2379,127.0.0.2:2379,127.0.0.3:2379"
bootstrap:
 dcs:
  retry_timeout: 3
```


Параметры раздела etcd3

Минимальное значение параметра **bootstrap.dcs.retry_timeout=3**, таймаут на ответ от DCS. Patroni ожидает ответ от одного узла etcd не меньше 1 секунды. Если в параметре **etcd3.hosts** указан 1 узел, то 3 запроса к нему за интервал **retry_timeout**. Если в параметре **etcd3.hosts** указано 2 узла, то 2 попытки на узел. Если указано 3 или больше узла, то 1 попытка на узел. Время ожидания ответа - минимум 1 секунда, поэтому, если указано 5 узлов, то нужно установить **retry_timeout** не меньше, чем в 5, чтобы Patroni последовательно обращался ко всем 5 узлам по одному разу.

Если число узлов **etcd3.hosts>=3**, а **retry_timeout>=etcd3.hosts**, то каждому узлу будет последовательно посылаться один запрос с таймаутом на получение ответа **retry_timeout**/**etcd3.hosts** секунд.

Если **retry_timeout=10**, а число **etcd3.hosts** 1 или 3, то будет 3 запроса с интервалом 3.3 секунды в течение 10 секунд. Пример сообщений о превышении таймаута:

```
 23:25:23 ERROR: Request to server http://127.0.0.1:2379 failed:
ReadTimeoutError("HTTPConnectionPool(host='127.0.0.1', port=2379): Read timed
out. (read timeout=3.332982451383335)")
 23:25:26 ERROR: Failed to get list of machines from http://127.0.0.1:2379/v3:
ReadTimeoutError("HTTPConnectionPool(host='127.0.0.1', port=2379): Read timed
out. (read timeout=3.332566755319325)")
```

Если число узлов 5, то будет 5 запросов с интервалом 2 секунды.

Если запрос попал на follower etcd, то follower обращается к лидеру etcd, получает ответ и возвращает его. Если запрос попал к лидеру etcd, то он отвечает самостоятельно.

Параметры встроенного DCS - раздела raft

- Раздел конфигурирует DCS, встроенный в Patroni
- Появился в версии 2.0, перестал поддерживаться в 3.0
- Для использования убрать разделы других DCS (etcd3:) и добавить раздел raft:

```
raft:
 data_dir: /opt/tantor/etc/patroni
 self_addr: "127.0.1.1:8000"
 bind_addr: "127.0.1.1:8000"
 partner_addrs: ['127.0.1.1:8000','127.0.1.2:8000','127.0.1.3:8000']
```

- Для 3 узлов кворум 2 узла
- Вместо третьего узла можно использовать скрипт:

```
patroni_raft_controller /opt/tantor/etc/patroni/raft.yml
 INFO: Using default value thread_stack_size = 524288
```

› в файле скрипту достаточно задать только раздел raft

Параметры встроенного DCS - раздела raft

В Patroni версии 2.0 был встроен DCS (к нему относится скрипт patroni_raft_controller), позволяющий обойтись без внешнего DCS. В версии 3.0 его пометили как support deprecated, то есть он не будет дорабатываться, хотя его код постараются переносить на новые версии.

Для использования встроенного DCS, в patroni.yml нужно закомментировать или убрать раздел etcd3 и добавить раздел:

```
raft:
 data_dir: /opt/tantor/etc/patroni
 self_addr: "127.0.1.1:8000"
 bind_addr: "127.0.1.1:8000"
 partner_addrs: ['127.0.1.1:8000','127.0.1.2:8000','127.0.1.3:8000']
```

или

```
 partner_addrs: ['127.0.1.2:8000','127.0.1.3:8000']
```

В директории data_dir будут созданы файлы DCS небольшого размера:

```
127.0.1.1:8000.dump  127.0.1.1:8000.journal  127.0.1.1:8000.journal.meta
```

Параметры self_addr и bind_addr задают адрес сетевого интерфейса для общения с другими узлами. Параметр bind_addr обязателен. Если не указать параметр self_addr, то узел не будет участвовать в консенсусе и файлы DCS не будут создаваться. Кворум определяется параметром partner_addrs на узле, который инициирует выборы. У такого узла должен быть указан параметр self_addr. Для кворума из 3 узлов консенсус составляют 2 узла. Чтобы не было ошибок, у узлов, у которых указан параметр self_addr значения partner_addrs должны быть одинаковы. В параметре partner_addrs можно не указывать адрес самого узла, достаточно указать только его "партнёрские" узлы, с которыми он образует кворум.

Если Patroni используется только на 2 узлах, то при остановке любого из процессов Patroni кворум теряется. Третьим узлом DCS может быть скрипт patroni_raft_controller.

В partner_addrs узла с установленным self_addr можно указать только его же (или не устанавливать partner_addrs), тогда кворум и консенсус будут состоять из одного узла. Остальные Patroni с не установленным self_addr, но с установленным partner_addrs, где перечислены все узлы, смогут обслуживать PostgreSQL, эти узлы и, даже смогут становиться лидерами и primary, но будут зависеть от доступности узла с установленным self_addr - он должен будет работать. Если он остановится, то после запуска, он станет лидером и primary. То есть кворум из 1 узла **для встроенного** DCS не практичен (для внешнего практичен).

Потери при failover и линии времени

- История переключений и линий времени сохраняется в DCS (не больше, чем **max_timelines_history**) и в файлах PGDATA/wal/*.history:

```
patronictl -c /opt/tantor/etc/patroni/patroni0.yml history
+----+----------+------------------------------+-------------+------------+
| TL |      LSN | Reason                       | Timestamp   | New Leader |
+----+----------+------------------------------+-------------+------------+
|  1 | 50661808 | no recovery target specified | 2026-09-... | tantor2    |
+----+----------+------------------------------+-------------+------------+
etcdctl --endpoints=:2379 get '/service/a/history'
/service/a/history
[[1,50661808,"no recovery target specified","2026-09-...","tantor2"]]
```

- При удалении истории в DCS, она восстанавливается лидером Patroni по содержимому последнего файла PGDATA/wal/*.history
- **maximum_lag_on_failover: 1048576** байт, отставание реплики, чтобы могла стать primary. Если меньше ~176, failover может не пройти даже на синхронную реплику

Потери при failover и линии времени

Основное предназначение Patroni - автоматическое переключение (failover) при сбое мастера (primary). Отключить автоматический failover можно включением режима паузы. Ручное переключение без сбоя мастера называется switchover.

История переключений и линий времени сохраняется в DCS и в файлах PGDATA/wal/*.history:

```
patronictl -c /opt/tantor/etc/patroni/patroni0.yml history
| TL |      LSN | Reason                       | Timestamp   | New Leader |
+----+----------+------------------------------+-------------+------------+
|  1 | 50661808 | no recovery target specified | 2026-09-... | tantor2    |
etcdctl --endpoints=:2379 get '/service/a/history'
/service/a/history
[[1,50661808,"no recovery target specified","2026-09-...","tantor2"]]
```

При удалении истории в DCS, она восстанавливается лидером Patroni по содержимому последнего файла PGDATA/wal/*.history. В файле нет Timestamp, поэтому это поле, при восстановлении из файла, останется пустым, кроме последнего **TL**, который выставится по времени обновления файла history. Это указывает на аккуратность и продуманность Patroni до мелочей.

maximum_lag_on_failover: 1048576 число байт в журнале, на которые может отстать любая реплика, чтобы могла стать primary. Так как позиция журнала отслеживается не в реальном времени, то к значению добавляется объем журнальных записей за ttl (в среднем, loop_wait/2). Если нет проблем с сетью, то задержка в передаче журнальных записей от walsender до walreceiver, обычно, меньше секунды. Если потери транзакций недопустимы, то нужно использовать synchronous_mode, synchronous_mode_strict. При установке **maximum_lag_on_failover** в значение, меньше ~176 failover может не пройти даже на синхронную реплику, так как мастер может писать в журнал (обновление управляющего файла, изменения в слотах репликации) без подтверждения приёма репликой, а Patroni мастера может успеть записать в ключ DCS "xlog_location", который ещё не был передан.

maximum_lag_on_syncnode: -1 - чтобы заменить отставшие реплики менее отстающими. При небольшом значении **walsenders** будут часто перезапускаться.

max_timelines_history: 100, Patroni хранит историю переключений primary (timelines) в DCS. По умолчанию, 0 - значит история не стирается и занимаемое место в DCS не освобождается. Лучше установить значение, чтобы файл базы etcd не разрастался.

Параметры динамической конфигурации

- **primary_start_timeout:** по умолчанию, 300 секунд. Максимальное время на перезапуск мастера, в случае его сбоя, перед началом failover
- **primary_stop_timeout:** если экземпляр не остановится за заданное время, то Patroni пошлёт сигнал SIGKILL процессу **postmaster**
  - › действует только, если включён synchronous_mode.
- **pause: false** - должен ли находиться Patroni на паузе (Maintenance mode) сразу после запуска
- **synchronous_mode: quorum** - все реплики смогут подтверждать транзакции
  - › по умолчанию, синхронный режим отключён

Параметры динамической конфигурации (продолжение)

primary_start_timeout: по умолчанию, 300 секунд. Максимальное время недоступности мастера, в случае его сбоя и перезапуска для того, чтобы не начался failover. Определяется длительностью запуска экземпляра мастера. Если установить 0, то failover на реплику начнётся в течение loop_wait. Если отлично от нуля, то максимальная задержка на переключение равна loop_wait+primary_start_timeout+loop_wait. Если мастер упал, то пока процесс **startup** не закончит восстановление, реплики к нему не подсоединятся, чтобы получить журналы. Если нет синхронной реплики и мастер не успеет открыться, то будет потеря транзакций, хотя кластер мастера может быть и не повреждён. Имеет смысл установить время primary_start_timeout=checkpoint_timeout. Если такое время простоя не допустимо, то нужны синхронные реплики (synchronous_mode=quorum) и synchronous_mode_strict=true, чтобы минимизировать потери.

primary_stop_timeout: действует только, если включён synchronous_mode. Если за заданное время экземпляр не остановится в режиме fast, то Patroni пошлёт сигнал SIGKILL процессу **postmaster**. Если значение не установлено или меньше или равно нулю, то таймаута нет. Начиная **с версии Patroni 4.1.4**, watchdog не отключается при установке этого параметра.

pause: false - должен ли находиться Patroni на паузе (Maintenance mode) сразу после запуска. Если мастер есть, то можно установить **true**, чтобы избежать failover при ошибочных действиях или нарушении порядка запуска (первым при bootstrap должен запускаться Patroni на мастере). После запуска всех Patroni можно отключить режим **pause**. Значение стоит установить одинаковым у всех узлов, чтобы не задумываться о том, какое значение установится. Если мастера нет и нужно, чтобы Patroni создал новый кластер баз данных PostgreSQL (то есть выполнил initdb), то нужно установить **pause: false**.

synchronous_mode: quorum - все реплики смогут подтверждать транзакции. Если установить **true** (**on**), то будет выбрано synchronous_node_count наименее отстающих реплик. По умолчанию **off**.

synchronous_mode_strict: false - если синхронные реплики не доступны, то мастер сам подтверждает транзакции. Если установить **true**, то мастер не будет подтверждать транзакции, пока не появится хотя бы одна синхронная реплика.

synchronous_node_count: 1 - Задаёт "кворум" - сколько реплик должно подтвердить транзакции. Не имеет смысла ставить больше, чем **1**.

postgresql.parameters.synchronous_commit: remote_write - по умолчанию **on**.

Параметры слотов репликации

- Patroni обновляет состояние всех *постоянных слотов репликации* каждые **loop_wait** секунд на всех экземплярах и в DCS:

```
etcdctl --endpoints=http://:2379 get '/service/a/members/tantor1' --prefix
/service/a/members/tantor1
{"conn_url":"postgres://127.0.0.1:5432/postgres","api_url":"http://127.0.1.1:8008/pa
troni","state":"running","role":"primary"},"xlog_location":72141040,"timeline":1}
psql -p 5433 -qc "select slot_name, active, restart_lsn, wal_status, restart_lsn-
'0/0' decimal from pg_replication_slots"
 slot_name | active | restart_lsn | wal_status | decimal
-----------+--------+-------------+------------+----------
 tantor1   | f      | 0/44CC8F0   | reserved   | 72141040
 tantor3   | f      | 0/44CC8F0   | reserved   | 72141040
psql -p 5433 -qc "select slot_name, active, restart_lsn, wal_status, restart_lsn-
'0/0' decimal from pg_replication_slots"
 slot_name | active | restart_lsn | wal_status | decimal
-----------+--------+-------------+------------+----------
 tantor2   | t      | 0/44CC8F0   | reserved   | 72141040
 tantor3   | t      | 0/44CC8F0   | reserved   | 72141040
```


Параметры слотов репликации

dcs.postgresql.use_slots: true, включает использование *постоянных слотов репликации*. Действует на весь кластер Patroni, а не на отдельные узлы. Слоты, которых нет, будут созданы Patroni. Физические слоты создаются на всех узлах, кроме тех, у которых **failover_priority < 1**. Слоты **сохраняются** (не удаляются и пересоздаются) Patroni при switchover/failover. При удалении слота, Patroni **его пересоздаёт и устанавливает актуальный restart_lsn**. Слот с именем текущего узла не создаётся, так как узел сам от себя WAL не получает. Patroni обновляет LSN всех слотов каждые `loop_wait` секунд в DCS:

```
etcdctl --endpoints=http://:2379 get '/service/a/members/tantor1' --prefix
/service/a/members/tantor1
{"conn_url":"postgres://127.0.0.1:5432/postgres","api_url":"http://127.0.1.1:
8008/patroni","state":"running","role":"primary","version":"4.1.4","tags":{"cl
onefrom":true,"failover_priority":1,"sync_priority":1,"key1":"abcd"},"xlog_loc
ation":72141040,"timeline":1}
```

а также в столбце restart_lsn представления pg_replication_slots всех реплик. На мастер значения обновляются процессом walsender. DCS для обновления **не нужен, обновление выполняется без DCS** и работает в режиме faisafe.

Пример для узла `tantor2` (слота с тем же именем нет):

```
psql -p 5433 -qc "select slot_name, active, restart_lsn, wal_status,
restart_lsn-'0/0' decimal from pg_replication_slots"
 slot_name | active | restart_lsn | wal_status | decimal
-----------+--------+-------------+------------+----------
 tantor1   | f      | 0/44CC8F0   | reserved   | 72141040
 tantor3   | f      | 0/44CC8F0   | reserved   | 72141040
```

если подключиться к экземпляру лидера (`tantor1`, слота с тем же именем нет), то будет `active=t`:

```
psql -p 5432 -qc "select slot_name, active, restart_lsn, wal_status,
restart_lsn-'0/0' decimal from pg_replication_slots"
 slot_name | active | restart_lsn | wal_status | decimal
-----------+--------+-------------+------------+----------
 tantor2   | t      | 0/44CC8F0   | reserved   | 72141040
 tantor3   | t      | 0/44CC8F0   | reserved   | 72141040
```

Постоянные слоты репликации

- **member_slots_ttl:** время до удаления физических слотов репликации для отсутствующих узлов Patroni
- слот удалится через это время, если он не используется экземпляром и Patroni узла не обновил свой ключ в DCS
- **slots:** имена и свойства *постоянных слотов репликации*
  - создаются на всех экземплярах и синхронизируются
  - не будут удаляться Patroni при switchover/failover
  - требуют **dcs.postgresql.use_slots=true**
  - могут быть физическими и логическими
  - если есть постоянные **логические** слоты, Patroni включит hot_standby_feedback

```
name: tantor1
bootstrap:
 dcs:
  slots:
   tantor1:
    type: physical
   tantor2:
    type: physical
   tantor3:
    type: physical
   subscriber1:
    type: logical
    database: postgres
    plugin: pgoutput
```


Постоянные слоты репликации

member_slots_ttl: по умолчанию, 30min. Если не указать min, h, d, то значение задаётся в секундах. Появился **в версии Patroni 4.0**. Слот удалится через это время, если он не используется экземпляром и Patroni не обновил свой ключ в DCS. Другими словами, если узел отсутствует. Если значение 0, то неактивный слот удаляется, если ключ узла не обновился в DCS.

До появления member_slots_ttl слоты узлов удалялись сразу (например, при failover) и, чтобы этого избежать, предлагалось использовать узел slots или архивы журналов или wal_keep_size. Начиная с версии Patroni 4.0, раздел slots нет смысла использовать. Логические - потому, что Patroni включит hot_standby_feedback, что приведёт к удержанию горизонта очистки. При использовании логических слотов репликации на PostgreSQL **до 19 версии**, нужно не забыть установить wal_level, иначе будут ошибки в логе:

```
 ERROR: Failed to create logical replication slot 'subscriber1'
plugin='pgoutput': ObjectNotInPrerequisiteState('logical decoding requires
"wal_level" >= "logical"\n')
```

slots: в этом разделе перечисляются **имена** и свойства *постоянных слотов репликации*. Они создаются на всех экземплярах (кроме тех, у которых **failover_priority < 1**) и синхронизируются, в соответствии с LSN, который был передан реплике **мастером**. Требуется установка **dcs.postgresql.use_slots=true**. Слоты не будут удаляться Patroni при switchover/failover. Если определены постоянные **логические** слоты репликации, Patroni автоматически включит hot_standby_feedback, **что нежелательно**. Более того, при failover, подписчикам логической репликации изменения могут передаваться повторно. Логические слоты копируются с лидера на standby с перезапуском экземпляра.

имя: имя *постоянного слота репликации*. Если совпадает с **именем текущего узла**, то не создаётся на экземпляре на этом узле. Если совпадает с **именем любого узла** Patroni, то Patroni не будет удалять слот, даже если узел станет неотзывчивым. Иначе, Patroni удалит слот с названием узла, который сам же создал, если этот узел стал неотзывчивым. Позволяет сохранить слоты узлов при сбоях узлов или, если хочется пересоздать кластер Patroni, то при передаче существующих узлов под управление нового кластера Patroni.

type: тип слота - physical или logical

database: только для логического слота: имя базы данных

plugin: pgoutput только логического слота: модуль логического декодирования.

Игнорируемые Patroni слоты

- **ignore_slots:** массив со свойствами слотов репликации, которые Patroni не должен менять
- Используется для логических слотов подписок, созданных с параметром `WITH (failover = true)`, управляемых параметрами PostgreSQL (начиная с 17 версии) `sync_replication_slots` и `synchronized_standby_slots`

```
bootstrap:
 dcs:
  ignore_slots:
  - name: subscriber_db02
    type: logical
    database: db02
#   plugin: pgoutput
  - name: replica4
    type: physical
```

- Свойства:
  - `- name` имя слота репликации
  - `- type` - `physical` или `logical`
  - `- database` для логических слотов
  - `- plugin: pgoutput` для логических слотов

Игнорируемые Patroni слоты

ignore_slots: массив со свойствами слотов репликации, которые Patroni должен игнорировать (не менять, не удалять). Используется для логических слотов подписок, созданных с параметром `WITH (failover = true)`, управляемых параметрами PostgreSQL (начиная с 17 версии) `sync_replication_slots` и `synchronized_standby_slots`. Любое подмножество совпадающих свойств (если не указать `name`, а указать `type=logical`, то распространится на все логические слоты во всех базах) приведёт к игнорированию слота.

- **name:** имя слота репликации
- **type:** тип слота - `physical` или `logical`
- **database:** имя базы данных для логического слота репликации
- **plugin: pgoutput** модуль логического декодирования логического слота репликации

Пример:

```
bootstrap:
 dcs:
  ignore_slots:
  - name: subscriber1
    type: logical
    database: postgres
#   plugin: pgoutput
  - name: replica4
    type: physical
```

Параметры PostgreSQL в DCS

- Параметры PostgreSQL, устанавливаемые **только** в динамической конфигурации. Изначально в разделе `bootstrap.dcs.postgresql`:
  - одинаковые на всех экземплярах PostgreSQL:

`max_connections:` 100, минимум 25
`max_locks_per_transaction:` 64, минимум 32
`max_worker_processes:` 8, минимум 2
`max_prepared_transactions:` 0
`wal_level:` replica, ещё может быть только `logical`
`track_commit_timestamp:` off

  - могут различаться:

`max_wal_senders:` *10*, минимум 3
`max_replication_slots:` 10, минимум 4
`wal_log_hints:` on (отличается от значения по умолчанию PostgreSQL, `off`)
Patroni передаёт следующие параметры в командной строке `postgres -c`:
`listen_addresses` - применяется значение из параметра Patoni `postgresql.listen`
`port` - применяется значение из параметра Patoni `postgresql.listen`
`cluster_name` - применяется значение из **`scope`**
`hot_standby:` on

Параметры PostgreSQL в DCS

Параметры **PostgreSQL**, которые могут быть установлены **только** в "Динамической конфигурации" и должны быть **одинаковыми** на всех экземплярах PostgreSQL.

Параметры, установленные в разделе **`postgresql`** файла `patroni.yml` или переменными окружения или в файлах `postgresql.conf` и `postgresql.auto.conf` не будут действовать.

Эти параметры Patroni передаёт при запуске экземпляра в командной строке `postmaster`, чтобы их значения нельзя было перекрыть в `postgresql.conf`, `postgresql.auto.conf` или командой `ALTER SYSTEM`.

`max_connections:` 100, минимум 25
`max_locks_per_transaction:` 64, минимум 32
`max_worker_processes:` 8, минимум 2
`max_prepared_transactions:` 0
`wal_level:` replica, может быть ещё только `logical`
`track_commit_timestamp:` off

Параметры PostgreSQL, которые могут быть установлены **только** в "Динамической конфигурации" и **могут различаться** для членов кластера:

`max_wal_senders:` ***10***, минимум 3 (отличается от значения по умолчанию PostgreSQL, 3)
`max_replication_slots:` 10, минимум 4
`wal_log_hints:` on (отличается от значения по умолчанию PostgreSQL, `off`)
Patroni передаёт следующие параметры в командной строке `postgres -c`, чтобы эти параметры не перекрывались файлами `postgresql.conf` и `postgresql.auto.conf`:
`listen_addresses` - применяется значение из параметра Patoni `postgresql.listen` (точкой обозначен путь в YAML файле: раздел `postgresql:`, ключ `listen:`) или переменной окружения **`PATRONI_POSTGRESQL_`**`LISTEN`
`port` - применяется значение из параметра Patoni `postgresql.listen` или переменной окружения **`PATRONI_POSTGRESQL_`**`LISTEN`
`cluster_name` - применяется значение из параметра Patoni `scope` или переменной окружения Patoni `PATRONI_SCOPE`
`hot_standby:` on
`wal_keep_size:` ***128MB*** (отличается от значения по умолчанию PostgreSQL - 0), до PostgreSQL 13 версии использовался `wal_keep_segments`
https://patroni.readthedocs.io/en/latest/patroni_configuration.html

Другие параметры раздела bootstrap

- В bootstrap.dcs.postgresql стоит указать:
  - use_slots: true и use_pg_rewind: true

```
bootstrap:
 dcs:
  postgresql:
   parameters:
    TimeZone: Europe/Moscow
    checkpoint_timeout: 1200
    log_timezone: Europe/Moscow #для записей в диагностическом логе вместо UTC
    logging_collector: on
    max_connections: 100
    max_locks_per_transaction: 64
    synchronous_commit: remote_write #быстрее, чем on, установленный по
умолчанию
    max_wal_size: 1GB
   use_slots: true #использовать потоковую репликацию
   use_pg_rewind: true #требует включённых включенных контрольных сумм
  initdb:
  - data-checksums #включение контрольных сумм. В 18 версии включено по умолчанию
  - encoding: UTF8
  - locale: UTF8
```


Другие параметры раздела bootstrap

Рекомендуется проверить, чтобы в этом разделе (bootstrap.dcs.postgresql) было указано use_slots: true и use_pg_rewind: true.

```
bootstrap:
  dcs:
   postgresql:
```

use_pg_rewind: true pg_rewind может использоваться, если на кластере PostgreSQL включены контрольные суммы (включаются по умолчанию, **с 18 версии** PostgreSQL).

```
    use_slots: true
    parameters:
     logging_collector: on
```

- стоит установить

```
 #  restore_command: 'wal-g wal-fetch %f %p >> $PGDATA/log/restore_command.log
2>&1 || cp $HOME/archivelog/%f %p || cp $HOME/archivelog/%f.partial %p'
```

- команды копирования WAL из архива, если он есть.

Не стоит устанавливать этот параметр, если нет архива. Если restore_command установлен, он будет использоваться вместо получения журнальных записей по протоколу репликации. Есть вероятность, что узел получит State=in archive recovery. Для возвращения в строй его придётся инициализировать (reinit).

method: имя - по умолчанию отсутствует, что эквивалентно initdb

имя: - имя скрипта для создания кластера PostgreSQL. Примеры скриптов и их параметров в документации: https://patroni.readthedocs.io/en/latest/replica_bootstrap.html

initdb: - (опционально) параметры для метода initdb

```
  - data-checksums
  - encoding: UTF8
  - locale: UTF8
```

post_bootstrap: или post_init:

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

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

Параметры раздела `restapi`

- Patroni принимает команды для управления, выдаёт статусы только по протоколу HTTP(s)-REST
  - `patronictl` работает с Patroni по HTTP(s)-REST
- `listen: 0.0.0.0:8008` - имя хоста или IP, где будет прослушивать Patroni
  - должен быть доступен HAProxy
- `connect_address: 127.0.1.1:8008` - имя хоста или IP с портом этого узла
- Интерфейсы `restapi`, обычно, **делают доступными только узлам Patroni и HAProxy** и изолируют от внешнего мира

```
curl http://127.0.1.1:8008/primary -I
HTTP/1.0 200 OK
curl http://127.0.1.1:8008/replica -I
HTTP/1.0 503 Service Unavailable
```


Параметры раздела `restapi`

Раздел `restapi:` - Patroni принимает команды для управления, выдаёт статусы только по протоколу HTTP(s)-REST. Утилита patronictl работает с Patroni по REST. У каждого Patroni свой адрес (`ip:порт`). Часть команд может передаваться любому Patroni на любом узле, часть команд только конкретному Patroni. Пример, если обратиться к узлу мастера:

```
curl http://127.0.1.1:8008/primary -I
HTTP/1.0 200 OK
curl http://127.0.1.1:8008/replica -I
HTTP/1.0 503 Service Unavailable
```

`200` выдаётся, если роль узла Patroni такая, как запрошено, иначе `503`.

`listen: 0.0.0.0:8008` - имя хоста или IP, где будет прослушивать Patrioni. Должен быть доступен HAProxy, так как он будет проверять статус экземпляра PostgreSQL на этом узле - реплика или мастер

`connect_address: 127.0.1.1:8008` - Записывается в DCS, задаёт имя хоста или IP с портом, по которому Patroni на других узлах будут слать запросы к этому узлу кластера Patroni. `listen` задает интерфейс, на котором Patroni слушает трафик, а `connect_address` говорит остальным миру, "как ко мне подключиться"

`thread_pool_size : 5` - число потоков

`request_queue_size : 5` - число запросов в очереди, при превышении будет выдаваться ошибка: `Connection denied`

Интерфейсы `restapi`, обычно, **делают доступными только узлам Patroni и HAProxy** и изолируют от внешнего мира, поэтому аутентификация, фильтрация по IP (`allowlist:`), TLS не нужны. Фильтрация по IP уменьшает гибкость конфигурирования и увеличивает ошибки (`allowlist_include_members:`), TLS (`certfile: keyfile: keyfile_password: cafile: ciphers: verify_client:`) увеличивает сетевые задержки. Поэтому, эти параметры опциональны.

Аутентификация и "unsafe endpoints"

- Подсоединения к "safe endpoints" не ограничиваются
- Доступ к "unsafe endpoints" (`PUT`, `POST`, `PATCH`, `DELETE`, изменение состояний узлов) можно ограничить:
  - аутентификацией HTTP BASE
  - по адресам подсетей

```
restapi:
 authentication:
  username: u
  password: p
 allowlist: ['192.168.0.0/24','127.0.1.1/16']
```

- `patronictl` использует параметры трёх разделов файла конфигурации: `restapi`, `etcd3`, `ctl`
- Пример "unsafe" запросов:

```
curl -X PATCH http:/127.0.1.1:8008/config
no auth header received
patronictl -c /opt/tantor/etc/patroni/patroni.yml pause
Failed: pause cluster management status code=401, (no auth header received)
```


Аутентификация и "unsafe endpoints"

Подсоединения к "safe endpoints" не ограничиваются, иначе, при ошибках конфигурирования, HAProxyне сможет получить статус узла.

Аутентификация возможна методом HTTP BASE для доступа к "unsafe endpoints" (`PUT`, `POST`, `PATCH`, `DELETE`, изменение состояний узлов).

```
restapi:
 authentication:
  username: u
  password: p
```

Имя и пароль можно заключить в апострофы или кавычки.

Пример unsafe команд:

```
curl -X PATCH http:/127.0.1.1:8008/config
no auth header received
patronictl -c /opt/tantor/etc/patroni/patroni.yml pause
Failed: pause cluster management status code=401, (no auth header received)
```

Утилита использует параметры трёх разделов: `restapi`, `etcd3`, `ctl`. Если есть раздел `ctl`, то параметры аутентификации и TLS берутся из него. Если раздел `ctl` отсутствует, то параметры аутентификации и TLS берутся из раздела `restapi`. Если есть раздел `restapi`, добавлять раздел `ctl` нет смысла. Пример успешной аутентификации:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml pause
Success: cluster management is paused
patronictl -c /opt/tantor/etc/patroni/patroni.yml resume
Success: cluster management is resumed
curl -X PATCH -d '{"pause": true}' http://127.0.1.1:8008/config -u u:p
{..."pause": true}
curl -X PATCH -d '{"pause": false}' http://127.0.1.1:8008/config -u u:p
{..."pause": false}
```

По умолчанию, `allowlist: не установлен, что` позволяет подсоединяться к "unsafe endpoints" с любых адресов. Можно указать список адресов в параметре:

```
restapi:
 allowlist: ['192.168.0.0/24','127.0.1.1/16']
```

в этом случае, будет отказ в доступе, даже если клиент передаёт имя и пароль. Адреса, не указанные в параметре не будут иметь доступа к "unsafe endpoints".

Параметры раздела postgresql

```
postgresql:
 authentication: #пользователи и их пароли для подсоединения Patroni
  replication: #пользователь для подсоединения реплик по протоколу репликации
   password: 'postgres'
   username: 'postgres'
  superuser: #пользователь под которым Patroni подсоединяется к экземпляру
   password: 'postgres'
   username: 'postgres'
  rewind: #пользователь для pg_rewind
   password: 'postgres'
   username: 'postgres'
 bin_dir: /opt/tantor/db/18/bin #если не установить используется $PATH
 data_dir: /var/lib/postgresql/tantor-se-18/data #PGDATA
#config_dir: /var/lib/postgresql/tantor-se-18/data #по умолчанию data_dir
#callbacks:
# on_role_change: /opt/tantor/etc/patroni/callback.sh
# on_reload: /opt/tantor/etc/patroni/callback.sh
 connect_address: 127.0.1.1:5432
 listen: 0.0.0.0,127.0.1.1:5432 #значения для listen_addresses и port
 proxy_address: IP_or_hostname:6432
```


Параметры раздела postgresql

postgresql:
authentication: - пользователи и их пароли для подсоединения Patroni
superuser: - пользователь под которым Patroni подсоединяется к экземпляру
username: postgres
password: 'postgres'
replication: - пользователь, для подсоединения реплик по протоколу репликации
rewind: - пользователь для pg_rewind
bin_dir: /opt/tantor/db/18/bin - если не установить, используется $PATH
data_dir: /var/lib/postgresql/tantor-se-18/data - PGDATA
#config_dir: /var/lib/postgresql/tantor-se-18/data - по умолчанию **data_dir**
connect_address: 127.0.1.1:5432 - IP и порт, по которому экземпляр доступен Patroni, работающих на **других** (и только других) хостах. По этому адресу будет подсоединяться pg_basebackup, запускаемый Patroni с других узлов для создания реплик. В разделе **pg_hba** нужно добавить правило для этого IP, чтобы Patroni других узлов смогли подсоединяться как по libpq, так и по репликационному протоколу. Patroni вставит этот адрес в DCS в ключ /service/a/members/tantor1 в виде параметра с названием **conn_url** и значением postgres://**127.0.1.1:5432**/postgres
listen: 0.0.0.0,127.0.1.1:5432 - адреса, которые Patroni передаст postmaster как **listen_addresses** и **port**:

```
 DEBUG: Starting postgres: /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=0.0.0.0,127.0.1.1 --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
```

IP можно перечислять через запятую, после них порт. Patroni, обслуживающий экземпляр, будет использовать unix socket, если **use_unix_socket: true** или **localhost**, если в **listen** есть что-либо из: *, localhost, **0.0.0.0**, 127.0.0.1 . Если нет, то первый адрес из параметра **listen**. Чтобы Patroni смог подсоединиться, для unix socket и localhost **нужно** указать разрешение на соединение в разделе **pg_hba**.

Параметры раздела postgresql (продолжение)

- Параметр proxy_address:

```
postgresql:
 connect_address: 127.0.1.1:5432
 listen: 0.0.0.0,127.0.1.1:5432 #значения для listen_addresses и port
 proxy_address: IP_or_hostname:6432
```

- В параметре **proxy_address** можно указать адрес, который прослушивает пулер pgbouncer, через который клиенты должны подсоединяться к экземпляру
- Параметр с названием proxy_url, будет добавлен в ключ /service/a/members/*name* в DCS:

```
etcdctl --endpoints=:2379 get '/service/a/members/tantor1' --prefix
/service/a/members/tantor1
{"conn_url":"postgres://127.0.1.1:5432/postgres","api_url":"http://127.0.1
.1:8008/patroni","state":"running","role":"primary","version":"4.1.4","pro
xy_url":"postgres://IP:6432/postgres","tags":{"clonefrom":true,"failover_p
riority":1,"sync_priority":1},"xlog_location":25071280,"timeline":2}
```


Параметры раздела postgresql (продолжение)

Patroni вставит этот адрес в DCS в ключ /service/a/members/tantor1 в виде параметра с названием **proxy_url** и значением postgres://*external:6432*/postgres **и не более того**.

При изменении параметра **proxy_address** в файле patroni.yml и перечитывании файла (команда **patronictl -c patroni.yml reload**) Patroni будет обновлять значение в DCS.

Пример:

```
postgresql:
  connect_address: 127.0.1.1:5432
  listen: 0.0.0.0,127.0.1.1:5432
  proxy_address: external:6432
```

- для записи в ключ /service/a/members/*name* DCS в виде параметра "proxy_url":"**postgres://***IP_или_hostname:порт***/postgres**"

Пример того, что Patroni запишет в DCS:

```
etcdctl --endpoints=:2379 get '/service/a/members/tantor1' --prefix
/service/a/members/tantor1
{"conn_url":"postgres://127.0.1.1:5432/postgres","api_url":"http://127.0.1.1:
8008/patroni","state":"running","role":"primary","version":"4.1.4","proxy_url"
:"postgres://external:6432/postgres","tags":{"clonefrom":true,"failover_priori
ty":1,"sync_priority":1},"xlog_location":25071280,"timeline":2}
```

В параметре **proxy_address** можно указать адрес, который прослушивает пулер pgbouncer, через который клиенты (приложения и/или балансировщик HAProxy) должны подсоединяться к экземпляру. Сконфигурировать pgbouncer можно не устанавливая параметр **proxy_address**. Также **external:6432** можно явно указать в файле haproxy.cfg.

Зачем тогда нужен этот параметр? **proxy_address** имеет смысл использовать в случае, если **external:6432** часто меняется в patroni.yml и pgbouncer.ini, выполняется reload, а возможности вносить изменения в файл haproxy.cfg нет.

Параметры раздела postgresql (продолжение)

- Параметры callbacks:

```
postgresql:
 callbacks:
  on_role_change: /opt/tantor/etc/patroni/callback.sh
  on_reload: /opt/tantor/etc/patroni/callback.sh
```

- Пример скрипта, создаваемого самостоятельно:

```
#!/usr/bin/bash
ACTION=$1; ROLE=$2; CLUSTER_NAME=$3; echo "ScripT" $ACTION $ROLE
```

- Сообщения в логе при запуске primary:

```
INFO: starting as a secondary
 LOG:  redirecting log output to logging collector process
localhost:5432 - accepting connections
INFO: establishing a new patroni heartbeat connection to postgres
INFO: promoted self to leader by acquiring session lock
server promoting
DEBUG: CallbackExecutor.call(['/opt/tantor/etc/patroni/callback.sh',
on_role_change, primary, 'a'])
ScripT on_role_change primary
INFO: Lock owner: tantor1; I am tantor1
```


Параметры раздела postgresql (продолжение)

Параметры в разделе **callbacks** - это скрипты, вызываемые Patroni при выполнении действий. Скрипту передаются параметры: **action, role, name**. Переменные окружения с префиксом PATRONI_ в скриптах не видны в целях безопасности. Скрипт может быть универсальным, так как параметр **action** принимает значения, соответствующие названиям параметров:

on_reload: on_restart: on_role_change: on_start: on_stop:

Пример скрипта /opt/tantor/etc/patroni/callback.sh, которым можно протестировать момент его вызова:

```
#!/usr/bin/bash
ACTION=$1; ROLE=$2; CLUSTER_NAME=$3; echo "ScripT" $ACTION $ROLE
```

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

```
postgresql:
 callbacks:
  on_role_change: /opt/tantor/etc/patroni/callback.sh
  on_reload: /opt/tantor/etc/patroni/callback.sh
```

Вызвать событие. Например, перечитать файлы patroni.yml процессами Patroni:

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

в логе будут сообщения:

```
INFO: Reloading PostgreSQL configuration.
server signaled
DEBUG: CallbackExecutor.call(['/opt/tantor/etc/patroni/callback.sh',
on_reload, primary, 'a'])
ScripT on_reload primary
```

При запуске экземпляра он сначала secondary, а потом становится мастером:

```
ScripT on_role_change primary
```

В скрипте смены роли можно указать:

1) команды переноса виртуального IP-адреса (VIP)

2) команды, которые будет искать журналы, переименовывать .partial файл, создаваемый pg_receivewal. Однако, сложно написать скрипт, который будет работать во всех случаях и лучше полагаться на стандартные опции Patroni.

Параметры раздела postgresql (продолжение)

```
postgresql:
 parameters: #параметры конфигурации PostgreSQL, при patronictl reload меняют
содержимое PGDATA/postgresql.conf
# config_file: /var/lib/postgresql/tantor-se-18/data/postgresql.conf
# hba_file:    /var/lib/postgresql/tantor-se-18/data/pg_hba.conf
# ident_file:  /var/lib/postgresql/tantor-se-18/data/pg_ident.conf
  checkpoint_timeout: 20min
  idle_in_transaction_session_timeout: 5min
 use_unix_socket: true
 use_unix_socket_repl: true
 parameters:
  config_file: /var/lib/postgresql/tantor-se-18/data/postgresql.conf
  hba_file:    /var/lib/postgresql/tantor-se-18/data/pg_hba.conf
  ident_file:  /var/lib/postgresql/tantor-se-18/data/pg_ident.conf
 pg_hba: #для генерации hba_file. Значения игнорируются, файл не создаётся, не
переименуется (при создании реплики и нового кластера методом initdb), если параметр
hba_file установлен в значение, отличное от значения по умолчанию (PGDATA/pg_hba.conf)
 - local   all             all                                     trust
 - host    all             all             127.0.0.1/16            trust
 - local   replication     all                                     trust
 - host    replication     all             127.0.0.1/16            trust
```


Параметры раздела postgresql (продолжение)

postgresql:

parameters: - параметры конфигурации PostgreSQL

 # config_file: /var/lib/postgresql/tantor-se-18/data/postgresql.conf

 # hba_file: /var/lib/postgresql/tantor-se-18/data/pg_hba.conf

 # ident_file: /var/lib/postgresql/tantor-se-18/data/pg_ident.conf

use_unix_socket: false - Стоит установить в **true**, чтобы сначала пробовать использовать unix socket. Если не удастся - подсоединяться по IP. Соединение с unix socket лучше IP хотя бы потому, что нет SSL, при использовании которого возможен внезапный отказ в соединении из-за использования сертификатов. Patroni должен подсоединяться только напрямую к postmaster, а не через балансировщик нагрузки. Для подсоединения Patroni будет использовать перебирать по порядку:

1) unix socket - если `use_unix_socket: true`
2) `localhost`, если в параметре **listen** есть что-либо из: `*`, `localhost`, `0.0.0.0`, `127.0.0.1`
3) будет использовать первый адрес из параметра **listen**.

Необходимо дать разрешение на подсоединение в **pg_hba**.

#unix_unix_directories: - можно переопределить директории, где искать unix sockets. Без этого параметра Patroni не передаёт параметр `host` в libpq, только порт.

use_unix_socket_repl: false - для подсоединений к **локальному** экземпляру пользователя, заданного в `postgresql.replication`.

```
pg_hba:
- local   all             all                                     trust
- host    all             all             127.0.0.1/16            trust
- local   replication     all                                     trust
- host    replication     all             127.0.0.1/16            trust
```

Если в `patroni.yml` задан параметр `pg_hba` или он уже присутствует в DCS, то Patroni генерирует файл `pg_hba.conf` из параметров `pg_hba`. Если установлен параметр `postgresql.parameters.`**hba_file** и файл уже есть, то файл `pg_hba.conf` из параметров `pg_hba` не генерируется - Patroni считает, что он файлом не управляет. Если установлен параметр `postgresql.parameters.`**hba_file** и он имеет значение, отличное от значения по умолчанию (`PGDATA/pg_hba.conf`) **что не стоит делать**, то раздел `pg_hba` игнорируется, файл не создаётся, не переименуется и экземпляр не стартует ни при создании реплики, ни при `initdb`. Для **ident_file** и **pg_ident** те же правила.

Параметры раздела `postgresql` (продолжение)

```
postgresql:
 parameters: #параметры конфигурации PostgreSQL, при patronictl reload меняют содержимое PGDATA/postgresql.conf
# config_file: /var/lib/postgresql/tantor-se-18/data/postgresql.conf
# hba_file:    /var/lib/postgresql/tantor-se-18/data/pg_hba.conf
# ident_file:  /var/lib/postgresql/tantor-se-18/data/pg_ident.conf
  checkpoint_timeout: 20min
  idle_in_transaction_session_timeout: 5min
  shared_buffers: 128MB
```

- В разделе **`postgresql.parameters`**, как и в **`bootstrap.dcs.postgresql.parameters`** **не могут** находиться параметры не известные Patroni
  - кроме параметров PostgreSQL, в названии которых есть точка
- Неизвестные параметры будут игнорироваться:

```
WARNING: Removing unexpected parameter=enable_temp_memory_catalog value=True from the config
```


Параметры раздела `postgresql` (продолжение)

Пример установки параметров конфигурации PostgreSQL:

```
postgresql:
 parameters:
  checkpoint_timeout: 20min
  idle_in_transaction_session_timeout: 5min
  shared_buffers: 128MB
```

Параметры устанавливаются в файле узла и действуют на экземпляр этого узла.

Для применения параметров достаточно перечитать файлы:

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

Будет обновлено содержимое `PGDATA/postgresql.conf` . Если файл в другом месте, то он не меняется. **Не стоит располагать файлы конфигурации в местах, отличных от мест по умолчанию,** то есть устанавливать параметры `config_file`, `hba_file`, `ident_file`.

При создании `patroni.yml` на основе существующего кластера PostgreSQL командой **`patroni --generate-config`** эти параметры могут быть не закомментированы. Стоит проверить, что файлы находятся в корне `PGDATA` и имеют названия по умолчанию.

В разделе **`postgresql.parameters`**, как и в **`bootstrap.dcs.postgresql.parameters`** **не могут** находиться параметры не известные Patroni. Это сделано, чтобы избежать ошибок: экземпляр не запустится, если параметр, в названии которого нет точки, не известен PostgreSQL.

Набор параметров зависит от версии PostgreSQL (кроме параметров PostgreSQL, в названии которых есть точка). Параметры будут игнорироваться с предупреждением:

```
WARNING: Removing unexpected parameter=enable_temp_memory_catalog value=True from the config
```

Поэтому, при выходе новой версии PostgreSQL придётся использовать новую версию Patroni, в которой есть параметры, появившиеся в новой версии PostgreSQL.

Параметры, не известне Patroni, можно установить командой:

```
alter system set enable_temp_memory_catalog = true;
```

Команда поменяет файл `PGDATA/postgresql.auto.conf`. Если команда выполнена на лидере, то файл будет копироваться в создаваемые реплики и параметр не будет потерян.

Параметры раздела `postgresql` (продолжение)

```
 create_replica_methods:
#  - walg
 - basebackup
#  walg:
#    command: "wal-g backup-fetch $PGDATA LATEST"
#    no_leader: 1 #call the replica creation method even if there is no
running leader or replicas
 basebackup:
 - checkpoint: 'fast'
#- verbose
#- slot: tantor1
#- waldir: /var/lib/postgresql/tantor-se-18/wal
 pg_ctl_timeout: 60 #время ожидания выполнения pg_ctl start, stop, restart
#pre_promote: #проверочный скрипт, при возврате ошибки promote не выполняется
#before_stop: #выполняется до остановки экземпляра, блокируя остановку
 remove_data_directory_on_rewind_failure: true
 remove_data_directory_on_diverged_timelines: true
 use_pg_rewind: true #использовать утилиту pg_rewind, что быстрее
```


Параметры раздела `postgresql` (продолжение)

`postgresql:`

`create_replica_methods: basebackup` - можно перечислить свои методы и воспользоваться теми, для которых исторически были написаны скрипты: basebackup, wal_e, barman, pgbackrest. Рекомендуется **`basebackup`**. wal-e это не wal-g.

`pg_ctl_timeout: 60` - время ожидания выполнения `pg_ctl start`, `stop`, `restart`.

`use_pg_rewind: true` - использовать утилиту `pg_rewind`, что быстрее

`remove_data_directory_on_rewind_failure: false` - не сможет создать реплику и она не запустится "`State: start failed`". Лучше установить **`true`**. Недостаток - будет делаться бэкап, на больших кластерах это долго.

`remove_data_directory_on_diverged_timelines: false` - не сможет создать реплику и она не запустится "`State: start failed`". Лучше установить **`true`**.

`pre_promote:` - проверочный скрипт, вызывается после получения ключа лидера, но до сигнала promote. Если скрипт вернёт статус, отличной от нуля, promote не выполняется, ключ лидера удаляется из DCS.

`before_stop:` - выполняется до остановки экземпляра, блокируя остановку. Возвращаемый скриптом результат игнорируется.

Параметры раздела `tags`

- В первую очередь, Patroni делает мастером узел с наибольшими значениями принятого LSN, применённого LSN. Параметр **`failover_priority`** задаёт приоритет, если есть несколько узлов с одинаковыми значениями принятого и применённого LSN
- **`replicatefrom: -`** имя реплики, с которой забирать WAL, для каскадной репликации, чтобы не нагружать мастер
- **`clonefrom:`** - делать ли бэкапы с этого узла для создания других узлов

```
tags:
 clonefrom: true #можно ли делать бэкапы для создания реплик с этого узла
 failover_priority: 1 #не работает с synchronous_mode: quorum
 noloadbalance: false #исключить из балансировки запросов HAProxy
 nostream: false #не использовать репликационный протокол, только restore_command
 sync_priority: 1
 replicatefrom: 'tantor2' #для каскадной репликации
```


Параметры раздела `tags`

Названия, значения по умолчанию и описание раздела **`tags:`**

`clonefrom: true` - можно ли с этого узла делать бэкапы для создания других узлов, используя `pg_basebackup`. Бэкапы создают нагрузку на хост. По умолчанию, **`false`**. Если у нескольких узлов **`true`**, то узел для снятия бэкапа будет выбран случайно.

`failover_priority: 1`. Если **`0`** или отрицательное число, то эквивалентно **`nofailover: true`**. Не работает с **`synchronous_mode: quorum`**. В первую очередь, Patroni делает мастером узел с наибольшими значениями принятого LSN, применённого LSN. Параметр задаёт приоритет, если есть несколько узлов с одинаковыми значениями принятого и применённого LSN. Чем больше число, тем приоритетнее.

`nofailover: false`. Устарел, вместо него **`failover_priority: 0`**. Если **`true`**, то запрещено становиться лидером кластера Patroni, слоты репликации на нём не создаются.

`noloadbalance: false`. При установке в `true` вернёт статус HTTP `503` на запрос `GET /replica` и будет исключён HAProxy из балансировки запросов. Используется для синхронной реплики, которой нежелательно отставание из-за конфликтов процесса startup с серверными процессами, обслуживающими запросы к этой реплике.

`nostream: false`. Если `true`, то экземпляр не будет получать WAL по репликационному протоколу, что нежелательно. Сможет использовать только архив журналов (`restore_command`). Также, отключает копирование и синхронизацию постоянных логических слотов репликации на этом узле и всех его каскадных репликах.

`sync_priority: 1`. Приоритет, при выборе кандидата на роль синхронной реплики. Если `0` или отрицательное число, то эквивалентно `nosync: true` - не делать синхронной репликой.

`nosync: false`. Устарел, вместо него `sync_priority`. Если `true`, то узел нельзя будет назначить синхронной репликой, то есть добавить в параметр `synchronous_standby_names`.

`replicatefrom: ''`. Используется в каскадной репликации. Имя реплики, с которой экземпляр реплики PostgreSQL, работающей на этом узле, будет забирать WAL.

В этот раздел можно (но не нужно) добавить свои ключи, они будут выдаваться командой: **`patronictl -c /opt/tantor/etc/patroni/patroni.yml list`**:

```
 tags:
  key1: true
  key2: 1.4
  key3: "value"
```

Необязательные разделы `log` и `watchdog`

- По умолчанию, файл лога не создаётся, сообщения передаются в stderr
- Просмотр журнала службы:

```
sudo journalctl -u patroni-tantor.service --since=-24h
```

- Параметры раздела `log`:

```
log:
 type: plain - формат логов, можно установить формат json
 dir: /opt/tantor/var/log
 file_num: 4 - удерживаемых ротацией файлов
 file_size: 26214400 - в байтах, по умолчанию, 25Мб
 level: INFO - сообщения какого уровня логируются
```

- Если устройство `/dev/watchdog` есть и доступно, то оно будет использоваться

```
watchdog:
 mode: automatic, off - запрещён, required - нет устройства, не станет лидером
 device: /dev/watchdog - путь к устройству
 safety_margin: 0 - через сколько секунд перегрузится после  истечения ttl лидера
```


Необязательные разделы `log` и `watchdog`

Разделы `log:` и `watchdog:` можно не указывать, их наличие необязательно.

По умолчанию, логирование Patroni выполняется в `stderr`, который systemd перенаправляет в журнал linux, посмотреть которые можно командой:

```
sudo journalctl -u patroni-tantor.service --since=-24h
```

Если удобнее направлять сообщения в файл, то достаточно указать в параметре `dir:` директорию. В ней будет создаваться ротируемый файл лога `patroni.log`:

```
log:
 type: plain - формат логов, можно установить формат json
 dir: /opt/tantor/var/log  - по умолчанию, не установлено
 file_num: 4 - удерживаемых ротацией файлов
 file_size: 26214400 - в байтах, по умолчанию, 25Мб
 level: INFO - сообщения какого уровня логируются: DEBUG, WARNING, ERROR, CRITICAL.
 deduplicate_heartbeat_logs: false - стоит установить в true, чтобы не замусоривать
```

логи каждые `loop_wait` секунд сообщением об удержании лидерства:

```
INFO: no action. I am (tantor1), the leader with the lock
```

Логирование экземпляров PostgreSQL устанавливается параметрами конфигурации PostgreSQL.

Параметры раздела `watchdog:`

```
watchdog: - сторожевое устройство, обычно, перегружающее linux
 mode: automatic - если есть, то используется, off - запрещён, required - узел не станет
```

лидером Patroni, если у узла отсутствует watchdog.

```
 device: /dev/watchdog - путь к сторожевому устройству (watchdog)
 safety_margin: 0 - секунд между истечением срока действия ключа лидера и
```

перезагрузкой хоста (срабатыванием watchdog).

Устройство добавляется командой:

```
sudo modprobe softdog
ls -l /dev/watchdog
crw------- 1 root root 10, 130 /dev/watchdog
```

По умолчанию, разрешения на использование `/dev/watchdog` имеет только `root`.

Файл устройства удаляется командой:

```
sudo rmmod softdog
```

Передача PostgreSQL под управление Patroni

- Экземпляры PostgreSQL при передаче не останавливаются и обслуживание не прерывается
- На хосте мастера PostgreSQL отключить службу

```
sudo systemctl disable tantor-se-server-18
Removed "/etc/systemd/system/multi-user.target.wants/tantor-se-server-18.service"
```

- Создать файл локальной конфигурации Patroni:

```
su - postgres -c 'PGPORT=5432 PGPASSWORD=postgres patroni --generate-config > /opt/tantor/etc/patroni/patroni.yml'
```

- Отредактировать файл, заменив `#FIXME`:

```
scope: 'a'
name: tantor
namespace: '/service'
postgresql:
 authentication:
  replication:
   password: '#FIXME'
   username: '#FIXME'
```


Передача PostgreSQL под управление Patroni

Шаги по передаче кластера PostgreSQL под управление Patroni описаны в документации: https://patroni.readthedocs.io/en/latest/existing_data.html

Экземпляры PostgreSQL при передаче не останавливаются и прерывания обслуживания нет.

Первым конфигурируют Patroni на хосте мастера PostgreSQL, затем на остальных хостах.

1) Если экземпляр запускается через `systemd`, то отключить службу:

```
sudo systemctl disable tantor-se-server-18
Removed "/etc/systemd/system/multi-user.target.wants/tantor-se-server-18.service"
```

2) Скопировать файлы `postgresql.conf`, `pg_hba.conf`, `pg_ident.conf` в `PGDATA`, если они находятся в другом месте или с другими названиями.

3) Создать файл локальной конфигурации Patroni:

```
su - postgres -c 'PGPORT=5432 PGPASSWORD=postgres patroni --generate-config > /opt/tantor/etc/patroni/patroni.yml'
```

Описание параметров: https://patroni.readthedocs.io/en/latest/yaml_configuration.html

4) Заменить в файле буквосочетание `#FIXME` на реальные значения:

```
scope: 'a'
name: tantor
namespace: '/service'
postgresql:
 authentication:
  replication:
   password: '#FIXME'
   username: '#FIXME'
  superuser:
   password: postgres
   username: postgres
```

Если в DCS планируется хранить конфигурацию нескольких кластеров Patroni, то рекомендуется добавить параметр `namespace:`, который задаёт префикс в названии ключей. Значение по умолчанию `'/service'`.

Параметр `name` - идентификатор узла, должен быть уникален в пределах кластера Patroni. Значение по умолчанию - имя хоста. Это значение с именем лидера `/service/a/leader`.

Передача PostgreSQL под управление Patroni

- Создать пользователей, если не достаточно `postgres`:

```
CREATE USER replicator LOGIN REPLICATION PASSWORD 'replicator';
CREATE USER rewinder LOGIN REPLICATION PASSWORD 'rewinder';
GRANT EXECUTE ON function pg_ls_dir(text,boolean,boolean),
pg_stat_file(text,boolean), pg_read_binary_file(text),
pg_read_binary_file(text,bigint,bigint,boolean) TO rewinder;
```

- Добавить параметры для использования функционала `pg_rewind`:

```
bootstrap:
 dcs:
  postgresql:
   use_pg_rewind: true
   use_slots: true
postgresql:
 authentication:
  rewind:
   username: rewinder
   password: 'rewinder'
```


Передача PostgreSQL под управление Patroni

5) Если нужно иметь разных пользователей для подключения по протоколу репликации и для подключения утилиты `pg_rewind`, то создать их. Пользователей создают, чтобы уменьшить число подсоединений с атрибутом `SUPERUSER`. Пример:

```
CREATE USER replicator LOGIN REPLICATION PASSWORD 'replicator';
CREATE USER rewinder LOGIN REPLICATION PASSWORD 'rewinder';
GRANT EXECUTE ON function pg_ls_dir(text, boolean, boolean),
pg_stat_file(text, boolean), pg_stat_file(text,
boolean),pg_read_binary_file(text), pg_read_binary_file(text, bigint, bigint,
boolean) TO rewinder;
```

6) В сгенерированном файле `patroni.yml` есть параметры в разделе `bootstrap`.

Значения из этого раздела будут загружены в DCS с префиксом в названии ключей: `namespace/scope/config/` и именно они станут динамической (глобальной) конфигурацией.

Если в DCS уже есть записи ("кластер Patroni инициализирован"), то значения из раздела не загружаются ("игнорируются").

Добавить параметры для использования функционала `pg_rewind` и режима `failsafe`:

```
bootstrap:
 dcs:
  postgresql:
   use_pg_rewind: true
   use_slots: true
postgresql:
 authentication:
  rewind:
   username: rewinder
   password: 'rewinder'
```

Проверить, чтобы в разделе `bootstrap.dcs.postgresql` было указано `use_slots: true`.

`pg_rewind` может использоваться, если на кластере PostgreSQL включены контрольные суммы (включаются по умолчанию, **с 18 версии** PostgreSQL).

Передача PostgreSQL под управление Patroni

- добавить в `patroni.yml` раздел с названием `etcd3`:

```
etcd3:
 protocol: http
 hosts: "127.0.0.1:2379,127.0.0.2:2379"
```

- Добавить раздел для удобства:

```
log:
 level: INFO #DEBUG INFO WARN WARNING ERROR FATAL CRITICAL
# dir: /opt/tantor/var/log
 deduplicate_heartbeat_logs: true
```

- Закомментировать строки:

```
#  config_file: /var/lib/postgresql/tantor-se-18/data/postgresql.conf
#  hba_file: /var/lib/postgresql/tantor-se-18/data/pg_hba.conf
#  ident_file: /var/lib/postgresql/tantor-se-18/data/pg_ident.conf
```

  - › позволит избежать ошибок в будущем, если поменяется путь к директории PGDATA

Передача PostgreSQL под управление Patroni

7) В сгенерированном файле полностью отсутствует раздел с параметрами соединения к DCS. Нужно добавить раздел с названием `etcd3`:

```
etcd3:
 protocol: http
 hosts:
  - 127.0.0.1:2379
  - 127.0.0.2:2379
```

или

```
 hosts: "127.0.0.1:2379,127.0.0.2:2379"
```

или

```
 host: 127.0.0.1:2379
```

или

```
 url: http://127.0.0.1:2379
```

8) Добавить раздел `log`. Наличие параметров в DCS со значениями по умолчанию, позволяет их видеть при редактировании. Если нужно логирование в файл, то проверить разрешения на директорию для пользователя `postgres` и раскомментировать параметр `dir`:

```
log:
 level: INFO #DEBUG INFO WARN WARNING ERROR FATAL CRITICAL
# dir: /opt/tantor/var/log
 deduplicate_heartbeat_logs: true
```

9) Закомментировать строки, чтобы их не было в DCS и избежать ошибок в будущем, если поменяются директории. Эти параметры должны иметь значения по умолчанию:

```
#  config_file: /var/lib/postgresql/tantor-se-18/data/postgresql.conf
#  hba_file: /var/lib/postgresql/tantor-se-18/data/pg_hba.conf
#  ident_file: /var/lib/postgresql/tantor-se-18/data/pg_ident.conf
```

Передача PostgreSQL под управление Patroni

- Добавить параметры:

```
bootstrap:
 dcs:
  failsafe_mode: true
  check_timeline: true
  max_timelines_history: 100
  maximum_lag_on_failover: 1000
  maximum_lag_on_syncnode: -1
  member_slots_ttl: 1h
  primary_start_timeout: 1200
  synchronous_mode: quorum
  synchronous_mode_strict: true
  pause: true
   parameters:
    log_timezone: Europe/Moscow
    logging_collector: on
    synchronous_commit: remote_write
#   synchronous_standby_names: ANY 1 (tantor2)
 remove_data_directory_on_rewind_failure: true
 remove_data_directory_on_diverged_timelines: true
 use_unix_socket: true
 use_unix_socket_repl: true
postgresql:
 create_replica_methods:
 - basebackup
 basebackup:
 - checkpoint: 'fast'
```

- Patroni будет создавать новую линию времени при каждом перезапуске лидера Patroni, если не включена пауза. Паузу можно включить и после запуска Patroni:

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


Передача PostgreSQL под управление Patroni

10) Добавить параметры:

```
bootstrap:
 dcs:
  failsafe_mode: true
  check_timeline: true
  max_timelines_history: 100 #ограничивает число линий времени, сохраняемых в DCS
  maximum_lag_on_failover: 1000 #в байтах, по умолчанию 1Мб
  maximum_lag_on_syncnode: -1
  member_slots_ttl: 1h
  primary_start_timeout: 1200
  synchronous_mode: quorum
  synchronous_mode_strict: true
  pause: true
   parameters:
    log_timezone: Europe/Moscow
    logging_collector: on
    synchronous_commit: remote_write
#убрать параметр, если он есть, его установит Patroni  synchronous_standby_names: ANY 1 (tantor2)
 remove_data_directory_on_rewind_failure: true
 remove_data_directory_on_diverged_timelines: true
 use_unix_socket: true
 use_unix_socket_repl: true
postgresql:
 create_replica_methods:
 - basebackup
 basebackup:
 - checkpoint: 'fast'
```

Patroni будет создавать новую линию времени при каждом перезапуске Patroni мастера, если не включить режим паузы. DCS может замусориться историей линий времени. К значению ключа будет каждый раз добавляться примерно **90 байт**.

Передача PostgreSQL под управление Patroni

- Параметры можно поменять в DCS после запуска Patroni:

```
patronictl -c patroni.yml edit-config -s failsafe_mode=true --force
```

- Для параметра `pause` есть отдельные команды:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml pause
patronictl -c /opt/tantor/etc/patroni/patroni.yml resume
```

  - которые эквивалентны изменению значения параметра:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config -s pause=true
patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config -s pause=null
```

- Проверить отредактированный файл:

```
patroni -i --validate-config /opt/tantor/etc/patroni/patroni.yml
postgresql.listen localhost:5432 didn't pass validation: Port 5432 is already in use.
postgresql.connect_address 127.0.0.1:5432 didn't pass validation: must not contain
"127.0.0.1", "0.0.0.0", "*", "::1", "localhost"
tags  Multiple of ('nosync', 'sync_priority') provided
```

  - `tags.sync_priority: 1` эквивалент `tags.nosync=true`
  - `-i` - не выдавать предупреждение, что экземпляр работает

Передача PostgreSQL под управление Patroni

Если какой-то параметр не установить, можно будет поменять параметры в DCS после после запуска первого процесса Patroni (он станет лидером), командой `patronictl edit-config --set параметр=значение`:

`patronictl -c patroni.yml edit-config -s failsafe_mode=true --force`

Для параметра `pause` есть отдельные команды:

`patronictl -c /opt/tantor/etc/patroni/patroni.yml pause`

`patronictl -c /opt/tantor/etc/patroni/patroni.yml resume`

которые эквивалентны изменению параметра:

`patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config -s pause=true`

`patronictl -c /opt/tantor/etc/patroni/patroni.yml edit-config -s pause=null`

11) Закомментировать или убрать строку, которая осталась для совместимости со старыми версиями Patroni:

```
tags:
#nosync: false
 sync_priority: 1 #0
```

эквивалент `nosync=true`

Начиная **с версии 4.1.0** используется параметр `sync_priority`.

Если не закомментировать, то при проверке файла будет выдаваться предупреждение:

```
tags Multiple of ('nosync', 'sync_priority') provided
```

12) Проверить отредактированный файл командой:

`patroni -i --validate-config /opt/tantor/etc/patroni/patroni.yml`

Параметр **`-i`** позволяет не выдавать предупреждение о том, что порт используется, так как экземпляр PostgreSQL не останавливался и работает: `postgresql.listen localhost:5432 didn't pass validation: Port 5432 is already in use.`

До версии 4.1.5 **`--validate-config`** ошибочно ругается на `synchronous_mode=quorum`

Может выдаваться предупреждение:

```
postgresql.connect_address 127.0.0.1:5432 didn't pass validation: must not
contain "127.0.0.1", "0.0.0.0", "*", "::1", "localhost"
```

Параметр задаёт имя хоста или IP с портом, по которому Patroni на других узлах будут подключаться к экземпляру на этом узле. Мастер и реплики должны работать на разных узлах, чтобы сбой одного узла не привёл к сбою нескольких членов кластера Patroni.

Первый запуск Patroni

- При запуске Patroni создаёт в DCS ключи
- В PGDATA меняются файлы параметров PostgreSQL

```
sudo systemctl start patroni-tantor
sudo journalctl -u patroni-tantor.service --since=-24h
INFO: Using default value thread_stack_size = 524288
INFO: Patroni global thread_pool_size = 5
INFO: Selected new etcd server http://127.0.0.1:2379
WARNING: postgresql parameter wal_keep_size=0 failed validation, defaulting to 128MB
INFO: No PostgreSQL configuration items changed, nothing to reload.
localhost:5432 - accepting connections
INFO: establishing a new patroni heartbeat connection to postgres
INFO: Changed cluster_name from '' to 'a' (restart might be required)
INFO: Changed wal_keep_size from '0' to '128MB'
INFO: Reloading PostgreSQL configuration.
server signaled
INFO: REST API thread_pool_size = 5
INFO: acquired session lock as a leader
INFO: no action. I am (tantor), the leader with the lock
```


Первый запуск Patroni

После успешной валидации конфигурационного файла можно запустить Patroni на узле с работающим экземпляром мастера PostgreSQL:

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

Patroni не остановит экземпляр, если включить паузу:

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

Patroni можно запускать как службу:

sudo systemctl start patroni-tantor

Команда просмотра журнала сообщений службы:

sudo journalctl -u patroni-tantor.service --since=-24h

При первом запуске Patroni создаст в DCS ключи:

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

```
/service/a/config          {"loop_wait":10,"retry_timeout":10,"ttl":30,...
/service/a/failsafe        {"tantor":"http://127.0.1.1:8008/patroni"}
/service/a/history         [[1,25066504,"no recovery target specified",...
/service/a/initialize      7672262683107335101
/service/a/leader          tantor
/service/a/members/tantor {"conn_url":"postgres://127.0.1.1:5432/postgres",.
/service/a/status          {"optime":25213016,"slots":{"tantor":25213016},...
```

Создаст в директории PGDATA файл patroni.dynamic.json. Это бэкап динамической конфигурации.

В директории PGDATA скопирует файлы:

1) postgresql.conf в postgresql.conf.backup и postgresql.base.conf

В начало файла postgresql.conf будет добавлена директива:

```
include 'postgresql.base.conf'.
```

Файл postgresql.base.conf - это копия исходного файла postgresql.conf, его можно редактировать, он не будет затираться Patroni.

2) pg_hba в pg_hba.conf.backup

3) pg_ident.conf в pg_ident.conf.backup и заменит их своими

В начале файлов postgresql.conf и pg_hba.conf будет комментарий:

```
# Do not edit this file manually!
# It will be overwritten by Patroni!
```

Первый запуск Patroni

- Идентификатором кластера устанавливается идентификатор (system identifier) кластера PostgreSQL

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml topology a
+ Cluster: a (7674732731860980557) ----------+----+-------------+-----+------------+----
| Member | Host           | Role   | State   | TL | Receive LSN | Lag | Replay LSN | Lag
+--------+----------------+--------+---------+----+-------------+-----+------------+----
| tantor | 127.0.0.1:5432 | Leader | running |  4 |             |     |            |
+--------+----------------+--------+---------+----+-------------+-----+------------+----
patronictl -c /opt/tantor/etc/patroni/patroni.yml reload a --force
+ Cluster: a (7674732731860980557) ----------+----+   -----+----------------------+
| Member | Host           | Role   | State   | TL |... Lag | Tags                 |
+--------+----------------+--------+---------+----+   -----+----------------------+
| tantor | 127.0.0.1:5432 | Leader | running |  4 |        | clonefrom: true      |
|        |                |        |         |    |        | failover_priority: 1 |
|        |                |        |         |    |        | sync_priority: 1     |
+--------+----------------+--------+---------+----+   -----+----------------------+
Reload request received for member tantor and will be processed within 10 seconds
```

- Leader продлевает ключ лидера через `loop_wait`. Если в течение `ttl` не будет продлён, мастер останавливается и начинаются выборы нового лидера ("leader race")

Первый запуск Patroni

Patroni проверяет наличие ключа `initialize`:

```
etcdctl --endpoints=:2379 get '/service/a/initialize'
/service/a/initialize
7676506443451906864
```

Если ключ `initialize` отсутствует, то он создаётся и выполняется **bootstrap** (инициализация) первого узла кластера Patroni по методу, указанному в разделе `bootstrap`. Если инициализация не удалась, то Patroni удаляет ключ, давая возможность другому узлу выполнить инициализацию. Если ключ присутствует, то узел присоединяется к кластеру Patroni и сможет создавать новые реплики.

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

После запуска Patroni можно проверить статус:

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

или

```
patronictl --dcs etcd3://127.0.0.1:2379/service topology a
```

или

```
patronictl --dcs etcd3://127.0.0.1:2379 topology a
+ Cluster: a (7674732731860980557) ----------+----+-------------+-----+----
| Member | Host           | Role   | State   | TL | Receive LSN | Lag | Rep
+--------+----------------+--------+---------+----+-------------+-----+----
| tantor | 127.0.0.1:5432 | Leader | running |  4 |             |     |
+--------+----------------+--------+---------+----+-------------+-----+----
```

По умолчанию, утилита ищет файл `~/.config/patroni/patronictl.yaml`

Получение адреса мастера:

```
patronictl --dcs etcd3://127.0.0.1:2379 dsn a -r primary
host=127.0.1.1 port=5432
```

Роль Leader (лидер кластера Patroni) - обладание ключом лидера и его продление через `loop_wait`. Если в течение `ttl` лизинг не будет продлён, то мастер останавливается и начинаются выборы нового лидера (гонка за лидерство, "leader race"). Выбирается сначала наименее отставшая синхронная реплика. Если таких несколько, то учитывается значение `failover_priority`. Отставание не должно быть больше, чем `maximum_lag_on_failover`.

Идентификатором кластера устанавливается идентификатор (system identifier) кластера PostgreSQL, который автоматически генерируется при создании утилитой `initdb`.

Добавление secondary Patroni

- Если уже есть реплика, то создать `patroni.yml`, как создавали для мастера, будут скопированы её параметры
- Если реплики нет, то скопировать `patroni.yml` с мастера
- Параметры, которые отредактировать:

```
name: tantor2
restapi:
 connect_address: 127.0.1.2:8008
 listen: 127.0.1.2:8008
postgresql:
 connect_address: 127.0.0.2:5433
 data_dir: /var/lib/postgresql/tantor-se-18/data1
 listen: 0.0.0.0,127.0.1.2:5433
```

- В режиме паузы реплика создаваться не будет:

```
patroni /opt/tantor/etc/patroni/patroni2.yml
Lock owner: tantor1; I am tantor3
PAUSE: running with empty data directory
patronictl -c /opt/tantor/etc/patroni/patroni.yml resume
Success: cluster management is resumed
```


Добавление secondary Patroni

Если уже есть реплика, то создать для неё `patroni.yml`, отредактировать точно так же, как это делалось для мастера и запустить Patroni. Существующая реплика продолжит работать и будет принята под управление Patroni даже в режиме паузы.

Если реплики не было, то её можно создать с помощью Patroni. Для этого скопировать `patroni.yml` с мастера и отредактировать. Затем запустить Patroni. В режиме паузы ни реплика (ни мастер, если его не было) создаваться не будут:

```
patroni /opt/tantor/etc/patroni/patroni2.yml
Lock owner: tantor1; I am tantor3
PAUSE: running with empty data directory
```

Статусы `tantor1` - мастер, `tantor2` - работавшая реплика, добавленная в режиме паузы, `tantor3` - Patroni запущен для создании реплики:

```
patronictl -c /opt/tantor/etc/patroni/patroni.yml list
+ Cluster: a (7683506657521664036) +-----------+----+-------------+-----+-----------
| Member  | Host  | Role            | State     | TL | Receive LSN | Lag | Replay LSN
+---------+-------+-----------------+-----------+----+-------------+-----+-----------
| tantor1 | :5432 | Leader          | running   |  2 |             |     |
| tantor2 | :5433 | Quorum Standby | streaming |  2 |   0/3050F60 |   0 |  0/3050F60
| tantor3 | :5434 | Replica         | stopped   |    |     unknown |     |    unknown
+---------+-------+-----------------+-----------+----+-------------+-----+-----------
 Maintenance mode: on
```

Если нет ошибок, выключить режим паузы:

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

Patroni создаст реплику:

```
INFO: bootstrapped from replica 'tantor2'
INFO: postmaster pid=495878
 LOG:  redirecting log output to logging collector process
/var/run/postgresql:5434 - accepting connections
INFO: Lock owner: tantor1; I am tantor3
INFO: establishing a new patroni heartbeat connection to postgres
INFO: no action. I am (tantor3), a secondary, and following a leader (tantor1)
```

Создание реплик процессом Patroni

- Patroni использует для создания реплик `pg_basebackup`

```
'/opt/tantor/db/18/bin/pg_basebackup',
'--pgdata=/var/lib/postgresql/tantor-se-18/data1', '-X', 'stream',
'--dbname=dbname=postgres user=postgres host=127.0.0.1 port=5432',
'--checkpoint=fast'
```

- Бэкап берётся с реплики или мастера, у которых `tags.clonefrom: true`
- Для использования других утилит резервирования, нужно указать их и их параметры в файле `patroni.yml`:

```
postgresql:
 create_replica_methods:
  - basebackup
 basebackup:
  - checkpoint: 'fast'
# - verbose
# - slot: tantor2
# - waldir: /var/lib/postgresql/tantor-se-18/wal1
```


Создание реплик процессом Patroni

По умолчанию, Patroni использует для создания реплик `pg_basebackup`, который бэкапит работающий экземпляр PostgreSQL. Бэкап берётся с реплики или мастера, у которых `tags.clonefrom: true`. Пример команды, выполняемой Patroni:

```
'/opt/tantor/db/18/bin/pg_basebackup', '--pgdata=/var/lib/postgresql/tantor-se-18/data1', '-X', 'stream', '--dbname=dbname=postgres user=postgres host=127.0.0.1 port=5432', '--checkpoint=fast'
```

Patroni может использовать и другие утилиты резервного копирования. Для их использования нужно перечислить в нужном порядке их и их параметры в разделе `postgresql.create_replica_methods`:

```
postgresql:
 create_replica_methods:
  - basebackup
 basebackup:
  - checkpoint: 'fast'
# - verbose
# - slot: tantor2
# - waldir: /var/lib/postgresql/tantor-se-18/wal1
```

Patroni остановится на первом методе, который вернёт 0. Если ни один из методов не завершился со статусом 0, то попытки будут повторяться по кругу через `loop_wait`. У каждого метода должен быть одноимённый раздел в файле конфигурации, с командой и параметрами, которые будут передаваться команде в формате `--name=value`.
Параметр `no_leader` позволяет Patroni вызывать метод создания реплики даже если нет работающего лидера или реплик. Это может использоваться для восстановления ранее работавшего кластера из предварительно созданного бэкапа.

Метод `basebackup` будет использован даже, если `create_replica_methods` пуст. Базовая резервная копия берется с лидера, если только нет реплик с `clonefrom`, в этом случае одна из таких реплик будет использована как источник для `pg_basebackup`. Валидация параметров, переданных в раздел `basebackup`, не выполняется. При использовании символических ссылок для `PGDATA/pg_wal` нужно указать в параметре `--waldir` путь к реальной директории, чтобы после создания реплики или повторной инициализации символическая ссылка сохранилась.