4

4

etcd: распределённое хранилище конфигурации

etcd - Distributed Configuration Store

- etcd - строго согласованная распределенная база данных, хранящая ключ-значение, обеспечивающая надежность хранения данных
- распределённые базы данных в один момент времени могут реализовать два из трёх свойств:
  - согласованность (Consistency)
  - доступность (Availability)
  - устойчивость к разрыву связи между узлами (Partition tolerance)
- etcd реализует свойства: **CP**
- Название etcd - "распределённая директория `/etc`"
- **Кластер etcd** - набор взаимодействующих по сети процессов
- Процессы называются **узлами** кластера

etcd - Distributed Configuration Store

etcd - **строго согласованная** распределяемая по нескольким хостам база данных, хранящая данные в формате ключ=значение, обеспечивающая надежность хранения данных.

По теореме **CAP**, распределённые базы данных в один момент времени могут реализовать два из трёх свойств:

1) согласованность (**C**onsistency)
2) доступность (**A**vailability)
3) устойчивость к разрыву связи между хостами, на которых работает (**P**artition tolerance).

etcd реализует свойства **CP**. Это значит, что при разрыве связи между хостами (сетевое разделение, network partitioning) etcd выбирает согласованность вместо доступности.

Для Patroni недоступность etcd на короткое время допустима.

Название etcd - "распределённая директория `/etc`". Смысл названия - распределённое хранилище конфигурации. В директории `/etc` операционная система linux хранит свою конфигурацию. etcd создан в 2013 году CoreOS (Container Linux - легковесная OS для создания кластеров) для координации узлов в кластере. Написан на языке Go. В 2014 году Kubernetes выбрал etcd как хранилище конфигурации. В 2018 году RedHat приобрела CoreOS. В 2019 году IBM приобрела RedHat.

Кластер etcd - набор взаимодействующих по сети процессов etcd. Процессы, обычно, запускаются systemd как службы в linux. Процессы называются "узлами кластера etcd" или, сокращённо, **узлами**.

Начиная версии 3, для взаимодействия узлов используется протокол **gRPC**, работающий поверх протокола **HTTP/2**.

До **версии 3.4**, по умолчанию, использовались протоколы `HTTP/1.x-REST-JSON` и называлась эта связка протоколом "v2", по номеру версии etcd 2.x.

Начиная с **версии 3.6** протокол HTTP/1 ("v2") удалён и используется только gRPC (протокол "v3"). При смене версий клиентов кластера etcd (Patroni) и софт etcd надо было обновлять в пределах нескольких версий, чтобы версии могли совместно работать.

Для совместимости с клиентами, которые не работают по gRPC, есть отдельная программа HTTP/1.1-gRPC "шлюз" (https://github.com/grpc-ecosystem/grpc-gateway). "Шлюз" легко спутать с обратным gRPC "прокси" (https://etcd.io/docs/v3.7/op-guide/grpc_proxy/), который тоже запускается отдельно, но не требует отдельной программы. Прокси используется при большом числе клиентов etcd.

Результат аудита etcd компанией Jepsen: https://etcd.io/blog/2020/jepsen-343-results/

Роли узлов: лидер, followers, learners

- Один узлов является **лидером**
  - остальные - последователи (followers) или неголосующие (learners)
  - learner позволяет быстро сделать из него follower на замену сбойнувшему follower
  - кандидаты появляются в процессе выбора лидера
- Лидер выбирается и перевыбирается автоматически голосованием узлов по алгоритму Raft
- Только лидер отправляет изменения на другие узлы и получает от них подтверждение, что изменение надёжно сохранено на диск
- Узлы не имеют общих файлов
- Начиная с etcd 3.6 используется только протокол gRPC

Роли узлов: лидер, followers, learners

Один узлов является **лидером** (leader, ведущий). Остальные узлы работают как **последователи** (followers) или не голосующие (learners).

Лидер и последователи являются голосующими узлами и учитываются в определении числа узлов, образующих большинство (**кворум**). Learner принимает изменения от лидера, но не голосует, не учитывается в определении числа узлов для кворума, не обслуживает запросы (ни на чтение, ни на запись, не перенаправляет запросы лидеру). Learners появились в **версии 3.4**. Цель существования learner: из него можно быстро сделать follower на замену сбойнувшему follower. Сбойнувший лидер или последователь не может голосовать, но он учитывается в кворуме до его удаления из кластера.

Кластер etcd корректно обрабатывает сетевые сбои и может выдерживать отказы хостов, на которых работают узлы, в том числе лидер. Лидер выбирается и перевыбирается автоматически голосованием узлов по алгоритму Raft. Если узел долго не получал от лидера сообщений по сети, он становится **кандидатом** в лидеры и начинает голосование. Кандидаты появляются в процессе выбора лидера.

Узлы не имеют общих файлов. Только лидер отправляет изменения на другие узлы и получает от них подтверждение, что изменение надёжно (используя fsync) сохранено в WAL. Остальные узлы не взаимодействуют друг с другом, за исключением выбора и перевыборов лидера. Когда лидер получает запрос (чтение или запись ключа), он записывает в WAL запрос на запись, но не фиксирует её. Затем лидер передаёт запись followers. Если большинство (кворум) узлов согласны записать или выдают те же данные, лидер подтверждает запрос и запрашивает у последователей подтверждение о приёме запроса.

Алгоритм Raft позволяет легко добавлять и удалять узлы, не останавливая обслуживания.

https://raft.github.io/

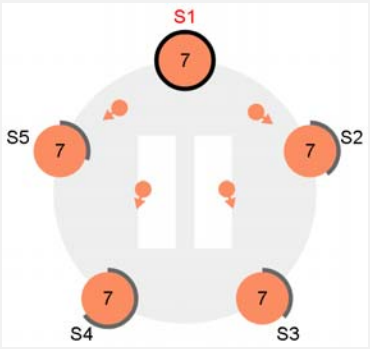
Клиент etcd может отправить запрос не только лидеру, но и последователям. **Последователи перенаправляют запросы лидеру и возвращают ответ клиенту**. Это называется обработкой запроса в режиме `l`inearizable. Клиент может запросить, чтобы ответил сам последователь - это режим `s`erializable: `etcdctl get ключ --consistency s`

В режиме `s`erializable **согласованность не гарантирована**, этот режим не стоит использовать. В режиме `s`erializable ответы на запросы на чтение быстрее на сетевой roundtrip до лидера.

https://jepsen.io/consistency/models/linearizable

Число узлов в кластере etcd

- Типичное число узлов etcd: 3, 5, реже 7
  - › больше 7 - задержки существенно возрастают, так как запросы на запись должны передаваться на все узлы
  - › для 3 узлов может отказать 1 узел, для 5 - 2, для 7 - 3
  - › для 3 узлов кворум - 2 узла, для 5 - 3, для 7 - 4
- По двум площадкам распределить узлы etcd, сохранив доступность при отказе одной площадки, нельзя
  - › нужно три площадкам, имеющие друг с другом связь



Число узлов в кластере etcd

Если лидер станет недоступен, проводится новое голосование и выбирается новый лидер. Лидера должно выбрать большинство узлов - **кворум**. Для 5 членов кластера кворум это 3 узла. Для 3 членов кластера, кворум это 2 узла. Основные вопросы:

1) сколько узлов должно быть в etcd?

Процесс etcd запускается и работает на нечётном числе хостов: 1,3,5,7.

Единственный узел etcd - точка отказа. При отказе единственного узла или кластера etcd, Patroni имеет режим failsafe: Primary работает, пока доступны все Standby.

Для 3 узлов может отказать 1 узел, для 5 - 2, для 7 - 3. Больше 7 - отказоустойчивость не улучшается, а задержки существенно возрастают, запросы на запись передаются на все узлы, а вероятность задержек и сбоев возрастает при большем числе узлов. В документации написано https://etcd.io/docs/v3.7/faq/ :

"Вероятно, кластер etcd, **не должен** состоять более чем из семи узлов. Сервис блокировок Google Chubby, аналогичный etcd и используемый в Google на протяжении многих лет, использует пять узлов.

Кластер etcd из 5 узлов может выдержать отказ двух узлов, чего достаточно в большинстве случаев. Хотя более крупные кластеры обеспечивают лучшую отказоустойчивость, производительность записи снижается, поскольку данные должны реплицироваться на большем количестве машин."

7 узлов используют, если их нужно расположить больше, чем на 5 площадках.

2) как располагать узлы, если центры обработки данных территориально разнесены по нескольким площадкам (центрам обработки данных)?

Если узлы территориально разнесены, то двух площадок не достаточно, нужна третья, имеющая связь с двумя, иначе высокой доступности не удастся достичь, так как на одной из двух площадок не будет кворума (при любом числе узлов) и на ней не удастся выбрать лидера.

Если всего две площадки, большее число узлов (кворум или все узлы) стоит располагать на той площадке, где работает primary Patroni. Тогда, в случае исчезновения площадки со standby, primary продолжит работу. На площадке с primary расположить синхронный standby, а на второй площадке асинхронный standby, так как, если расположить синхронный standby на второй площадке, это создаст задержки при фиксации транзакций.

3) использовать физические узлы или виртуальные машины или контейнеры докер?

Работа в докере возможна и документирована https://etcd.io/docs/v3.7/op-guide/container/

Объем хранимых данных

- etcd не предназначен для хранения ключей большого размера
- По умолчанию, размер ключа не больше `1572864` байт (параметр `max-request-bytes`)
- По умолчанию, оперативной памяти должно быть достаточно для хранения `5000` последних ключей (параметр `snapshot-catchup-entries`)
  - лидер хранит ключи в памяти в виде скользящего окна для передачи отстающим последователям
- etcd не предназначен для работы на хостах с жёстким ограничением по использованию оперативной памяти
- Не стоит хранить в кластере etcd что-либо, кроме данных одного или нескольких кластеров Patroni

Объем хранимых данных

etcd использует диск и хранит файлы журналов и снимки на диске. Если кластер etcd выходит из строя или останавливается из-за сбоя, он может восстановить данные (ключи и их значения) из файлов WAL и снимков и перезапустить службу.

Физически etcd хранит данные в виде пар ключ-значение. Ключ пары содержит ревизию, а значение пары - дельту от предыдущей версии.

Ключ и значение представлены в виде бинарного массива. Для ключа и значения нет фиксированного ограничения по размеру, но поскольку существует ограничение на размер запроса (параметр `max-request-bytes`, по умолчанию, `1572864` байт), допустимый размер ключа плюс значение определяется этим ограничением. Следствие этого ограничения - размер ConfigMap и Secret в K8s, ограничен 1Мб, они хранятся в etcd.

В Google Chubby (аналоге etcd с долгой историей использования) один из клиентов довёл размер ключей до 1.5 мегабайт. Общий объем, используемый сервисом превысил место, занимаемое ключами всех остальных сервисов. Это посчитали экстремальным использованием, **ввели ограничение на размер значения ключа в 256Кб** и в течение года перевели клиента на базу данных другого типа.

etcd не предназначен для хранения значений большого размера. **Не стоит хранить в кластере etcd что-то совместно с данными Patroni.** Причина: другие клиенты хранить большие размеры ключей или большое число ключей или часто менять значения ключей. Это увеличивает потребление памяти, приводит к подвисанию процесса, перервыборам лидера и задержкам обслуживания Patroni.

Вместе с ключом хранятся метаданные: `create_revision`, `mod_revision`, `version` и `lease`. `create_revision` - ревизия когда ключ создан, а `mod_revision`, - ревизия когда ключ обновили. Ревизия работает как глобальный счетчик, который увеличивается при изменении любых данных. Прежнее значение ключа можно получить, указав ревизию (**`etcdctl get ключ --rev`**), если она не была очищена (compacted). `Lease` используется для данных, имеющих определенный срок жизни, и по истечении этого срока данные удаляются и становятся недоступными.

etcd сначала сохраняет запрос в WAL, а затем обновляет файл базы (`db`), чтобы избежать увеличения размера WAL-файла. Файл базы содержит пары ключ-значение, организованные по структуре btree и ключи хранятся в отсортированном виде, что позволяет их быстро находить. Работа с файлом базы идёт через отображение (mmap) файла в виртуальную память процесса etcd.

Установка etcd

- Устанавливается из пакета:

```
root@tantor:~# dpkg -i etcd-tantor-all*
Setting up etcd-tantor-all (3.7.1-1astra1.8-1) ...
etcd-tantor.service is a disabled or a static unit not running, not starting it.
```

- в директории **/opt/tantor/usr/bin** 3 программы:
  - `etcd` - исполняемый файл, запускает процесс, который станет узлом (членом кластера etcd)
  - `etcdctl` - клиент для взаимодействия с `etcd` по gRPC
  - `etcdutl` - утилита для операций с файлами
- Добавить директорию в путь поиска:

```
root@tantor:~# sed -i 's|PATH=|PATH=/opt/tantor/usr/bin:|' ~/.bash_profile
root@tantor:~# source ~/.bash_profile
```

- etcd реагирует на переменные окружения:

```
ExecStart=/bin/bash -c "GOMEMLIMIT=256MiB GOGC=50 GOMAXPROCS=$(nproc)
/opt/tantor/usr/bin/etcd"
```


Установка etcd

Команда установки: **dpkg -i etcd-tantor-all***

Инсталляционный пакет содержит 3 исполняемых файла:

`etcd` - исполняемый файл, запускает процесс, который станет членом кластера etcd

`etcdctl` - клиент для взаимодействия с сервером по протоколу gRPC

`etcdutl` - утилита для операций с файлами, например, резервного копирования.

После установки пакета стоит добавить в путь директорию с этими файлами:

export PATH=/opt/tantor/usr/bin:/opt/tantor/db/18/bin:$PATH

Например, добавить путь в переменную окружения **PATH** можно отредактировав файл `/root/.bashrc` для работы через `sudo bash` и в файл `/root/.bash_profile` для работы через `su -` и файл `/home/astra/.bash_profile` для работы под пользователем `astra`.

При установке создаётся файл службы:

`/usr/lib/systemd/system/etcd-tantor.service`

Команда запуска процесса в файле службы устанавливает переменную **GOMAXPROCS**, ограничивающую число потоков процесса **etcd** числом ядер процессоров на хосте:

ExecStart=/bin/bash -c "GOMAXPROCS=$(nproc) /opt/tantor/usr/bin/etcd"

etcd написан на языке Go и программы на этом языке реагируют на переменные окружения:

`GOMEMLIMIT=256MiB GOGC=50 GOMAXPROCS=$(nproc)`

etcd написан на языке Go. Программы Go используют сборщик мусора и, в текущей версии Go, реагируют на **три** переменные окружения. Переменной `GOMEMLIMIT` можно установить объем виртуальной памяти, приближаясь к которому, сборщик будет чаще и агрессивнее освобождать память.

По умолчанию, значение `GOGC=100`, которое означает, что сборка мусора запустится, когда память удвоится (`1+GOGC/100`) по сравнению с объемом не освобождённой памяти после предыдущего цикла сборки мусора. При меньшем значении сборка мусора будет запускаться чаще. Эту переменную можно не менять, достаточно установить `GOMEMLIMIT`, по достижении которой сборщик мусора будет работать более активно. Параметр не устанавливает "лимит" - объем выделенной памяти может существенно выходить за заданное значение.

Параметры etcd

- В файле переменных окружения службы `EnvironmentFile=-/opt/tantor/etc/etcd/etcd.conf` заданы переменные окружения, которые соответствуют параметрам (flags) исполняемого файла etcd
  - переменные приводятся к верхнему регистру, получают префикс **ETCD_**, тире заменяются подчеркиванием
  - пример: **--max-wals '5'** аналогичен **ETCD_MAX_WALS='5'**
- Файл службы это не конфигурационный файл etcd

Параметры etcd

В файле `/opt/tantor/etc/etcd/etcd.conf` заданы переменные окружения, которые соответствуют параметрам (flags) исполняемого файла `etcd`. Список параметров:

/opt/tantor/usr/bin/etcd --help

Переменные окружения в верхнем регистре и имеют префикс **ETCD_**. Тире заменяются подчёркиванием. Пример:

параметр **--max-wals '5'** соответствует переменной **ETCD_MAX_WALS='5'**

Если etcd заданы параметры, то они превалируют над переменными окружения.

Файл переменных окружения службы не является конфигурационным файлом `etcd`. Конфигурационный файл имеет формат YAML и содержит параметры, который соответствуют параметрам `etcd`, только без префикса "--".

Файл переменных окружения службы содержит переменные окружения, которые устанавливаются директивой:

`EnvironmentFile=-/opt/tantor/etc/etcd/etcd.conf`

Значок "-" перед названием файла значит, что если файл отсутствует, `systemd` проигнорирует его отсутствие без вывода ошибок и предупреждений в лог операционной системы и запустит службу без установки переменных окружения.

Директива находится в файле службы:

/usr/lib/systemd/system/etcd-tantor.service

Команды запуска и остановки службы:

```
systemctl start etcd-tantor
systemctl stop etcd-tantor
```

Если нужно запустить процесс etcd не через systemd, то можно использовать команды:

```
set -a
source /opt/tantor/etc/etcd/etcd.conf
set a+
/opt/tantor/usr/bin/etcd
```

Один из форматов вывода утилит это JSON, чтобы выбирать значения из строк в формате JSON удобно использовать утилиту **jq**, которую можно установить командой:

```
sudo apt install jq -y
```

Основные параметры etcd

- Можно запустить узел командой: `etcd`
  - директория с файлами узла: `~/default.etcd`
  - `http://localhost:2379` и `http://localhost:2380`
- Основные параметры:
  - `ETCD_NAME=a1` - название узла, должно быть уникально
  - `ETCD_DATA_DIR=/opt/tantor/var/lib/etcd/a1.etcd` - директория с файлами
  - `ETCD_LISTEN_CLIENT_URLS` - локальные адреса (можно `0.0.0.0`) для входящих соединений клиентов
  - `ETCD_LISTEN_PEER_URLS` - локальные адреса (можно `0.0.0.0`) для входящих соединений от узлов
  - `ETCD_ADVERTISE_CLIENT_URLS` - адрес при начальном создании кластера, добавлении узла
  - `ETCD_INITIAL_ADVERTISE_PEER_URLS` - для общения узлов при начальном создании кластера, добавлении узла

Основные параметры etcd

Параметры `etcd` имеют значения по умолчанию. Можно запустить узел командой:
`etcd`
Директорией с файлами узла будет: `~/default.etcd`
Название кластера: `default`
Один узел в кластере, прослушивает `http://localhost:2379` и `http://localhost:2380`.
Если нужно запустить `etcd` как службу, то перед запуском службы нужно отредактировать файл переменных окружения `/opt/tantor/etc/etcd/etcd.conf`
и отредактировать значения:
`ETCD_NAME=a1` - название узла, обычно, какой-нибудь уникальный идентификатор или имя хоста. Значение должно быть уникально в пределах кластера etcd.
`ETCD_DATA_DIR=/opt/tantor/var/lib/etcd/a1.etcd` - название директории с файлами узла
`ETCD_LISTEN_CLIENT_URLS` - локальные адреса для приема входящих соединений клиентов, в том числе, для подсоединения утилиты `etcdctl`. `0.0.0.0` в качестве IP-адреса - прослушивать все сетевые интерфейсы.
`ETCD_LISTEN_PEER_URLS` - локальные адреса для приема входящих соединений от узлов кластера. `0.0.0.0` в качестве IP-адреса - прослушивать все сетевые интерфейсы.
`ETCD_ADVERTISE_CLIENT_URLS` - адрес узла для клиентов, в том числе, для подсоединения утилиты `etcdctl` при начальном создании кластера, инициализации, добавлении узла.
`ETCD_INITIAL_ADVERTISE_PEER_URLS` - адрес для обращения других узлов к этому узлу (при начальном создании кластера, инициализации и добавлении узла).

Параметры логирования etcd

- По-умолчанию логирование в формате `ETCD_LOG_FORMAT=json`, направлено в `ETCD_LOG_OUTPUTS=stdout`
- Пример для `/opt/tantor/etc/etcd/etcd.conf`:

```
ETCD_LOG_LEVEL=warn
ETCD_LOG_FORMAT=console
ETCD_LOG_OUTPUTS=/opt/tantor/var/log/etcd/a1.log
ETCD_ENABLE_LOG_ROTATION=true
ETCD_LOG_ROTATION_CONFIG_JSON='{"maxsize": 1, "maxage": 1, "maxbackups": 1,
"localtime": true, "compress": true}'
```

- Пример для `/opt/tantor/etc/etcd/etcd.yml`:

```
log-level: warn
log-format: console
log-outputs: ['stdout','/opt/tantor/var/log/etcd/a1.log']
log-rotation: true
log-rotation-config-json: '{"maxsize": 1, "maxage": 1, "maxbackups": 1,
"localtime": true, "compress": true}'
```


Параметры логирования etcd

По-умолчанию, логирование выполняется в формате json и направлено в stdout. Это удобно подходят для запуска узлов в докере или через Kubernetes, в остальных случаях не удобно для просмотра лога. Можно настроить логирование параметрами:

```
ETCD_LOG_LEVEL=warn
ETCD_LOG_FORMAT=console
ETCD_LOG_OUTPUTS=/opt/tantor/var/log/etcd/a1.log
ETCD_ENABLE_LOG_ROTATION=true
ETCD_LOG_ROTATION_CONFIG_JSON='{"maxsize": 1, "maxage": 1, "maxbackups": 1,
"localtime": true, "compress": true}'
```

Описание:

`ETCD_LOG_LEVEL=info` - возможные значения `debug`, `info`, `warn`, `error`, `panic`, `fatal`.

`ETCD_LOG_FORMAT=json` - возможные значения `json`, `console`

`ETCD_LOG_OUTPUTS=stdout` - можно перечислить **через запятую** несколько мест для логирования. Можно указать `stdout`, `stderr`, пути к файлам.

`ETCD_ENABLE_LOG_ROTATION=false` - включить ротацию файлов.

`ETCD_LOG_ROTATION_CONFIG_JSON='{"maxsize": 100, "maxage": 0, "maxbackups": 0, "localtime": false, "compress": false}'` - строка в формате JSON, где `maxsize` - в мегабайтах, `maxage` - в днях, 0 - без ограничений, `maxbackups` число файлов, 0 - без ограничений, `localtime` - использовать локальное время хоста, `compress` - сжимать файлы логов алгоритмом gzip.

Аналогичные параметры можено передать в параметрах. Пример:

```
etcd --log-format console
```

Сетевые параметры etcd

- Лидер периодически отправляет пакеты (heartbeat) последователям
- Если в течение `election-timeout` последователь не получит `heartbeat`, он начнёт выборы нового лидера
- Если лидер не сможет послать `heartbeat` в течение двух интервалов `heartbeat-interval`, то он выдаст предупреждение "`failed to send out heartbeat on time`"
- `heartbeat-interval` устанавливается в интервале **0.5-1.5** от `peer_round_trip_time`
- Просмотр метрик:

```
curl -s http://127.0.0.1:2379/metrics | grep fsync
...
etcd_snap_fsync_duration_seconds_bucket{le="0.512"} 0
```


Сетевые параметры etcd

Если лидер не сможет послать heartbeat в течение двух интервалов `heartbeat-interval`, то он выдаст предупреждение "`failed to send out heartbeat on time`".

Лидер периодически отправляет пакеты (heartbeat) последователям. Если в течение `election-timeout` последователь не получит heartbeat, он инициируют выборы нового лидера. Если лидер не отправляет сигналы подтверждения вовремя, но продолжает работу, выборы являются ложными и, вероятно, вызваны торможением процесса-лидера или сети.

Причины:

1) Медленный диск. Перед отсылкой `heartbeat` лидер сохраняет свое состояние на диск. Чтобы исключить проблемы со скоростью диска, наблюдайте за метрикой `wal_fsync_duration_seconds` (https://etcd.io/docs/v3.7/metrics/#disk), которая должна быть меньше 10мс в 99% измерений. Метрику можно измерить утилитами тестирования диска, например, **fio**.

2) Если сетевая задержка высокая или есть потери сетевых пакетов или дефицит пропускной способности стоит увеличить значения параметров. `heartbeat-interval` увеличивают так, чтобы он был примерно равен метрике `peer_round_trip_time_seconds` (двойной ping) и установить `election-timeout`, как минимум, в `5*heartbeat-interval` и, **не меньше** `10*peer_round_trip_time`. Обычно, `heartbeat-interval` устанавливается в интервале **0.5-1.5** от `peer_round_trip_time`. Слишком больше значение увеличивает время обнаружения сбоя лидера. Самый простой способ измерить `peer_round_trip_time` это использовать утилиту `ping`. По умолчанию, `election-timeout=1000` миллисекунд (максимально 50 секунд), `heartbeat-interval=100` миллисекунд (практический максимум 5 секунд).

Параметры `election-timeout` **и** `heartbeat-interval` **должны быть одинаковы на всех узлах. Установка разных значений нарушит стабильность кластера etcd.**

Рекомендуется, чтобы клиенты выдерживали недоступность etcd до 30 секунд, исходя из опыта Chabby, где длительность большинства сбоев была меньше 15 секунд, почти все сбои длились меньше 30 секунд. 9 более длительных сбоев были вызваны техобслуживанием сети (4 сбоя), проблемами с подключением к сети (2), ошибками софта (2), чрезмерной нагрузкой (1). Причины сбоев Chabby: проблемы с congestion у протокола TCP, техническое обслуживание, чрезмерная нагрузка, ошибки администраторов, сбои программ и железа. https://etcd.io/docs/v3.7/tuning/

Создание кластера etcd

- Static bootstrapping (создание кластера по списку узлов)
  - `ETCD_INITIAL_CLUSTER_TOKEN` - одинаков у узлов
  - `ETCD_INITIAL_CLUSTER_STATE=new`, а не `existing`
  - `ETCD_INITIAL_CLUSTER=$ETCD_NAME=$ETCD_INITIAL_ADVERTISE_PEER_URLS,...` - адреса узлов, чтобы узел нашёл их при запуске
- Custom etcd discovery - нужен существующий кластер
  - используется параметр `ETCD_DISCOVERY`
- Public etcd discovery - используется `discovery.etcd.io`
  - используется параметр `ETCD_DISCOVERY`
- DNS discovery - в DNS вставить `SRV` записи `_etcd-server-суффикс._tcp.домен` или `_etcd-server-ssl-суффикс._tcp.домен`

Создание кластера etcd

Если список и адреса начальных узлов известен, то можно инициализировать кластер способом, который называется **static bootstrapping**. Для этого установить параметры:

1) `ETCD_INITIAL_CLUSTER_TOKEN` - строка, **одинаковая** для всех узлов кластера, чтобы чужие узлы не подсоединялись. Используется при генерации идентификатора кластера (cluster ID).

Параметры `--enable-v2=true` (и соответствующая ему переменная окружения `ETCD_ENABLE_V2`) могли использоваться только до версии 3.6. В версии 3.4, по умолчанию, стал использоваться gRPC и`--enable-v2` был установлен в значение `false`, а `ETCDCTL_API` был установлен в значение 3.

2) `ETCD_INITIAL_CLUSTER_STATE=new`

3) `ETCD_INITIAL_CLUSTER=$ETCD_NAME=$ETCD_INITIAL_ADVERTISE_PEER_URLS,...` - адреса узлов, чтобы узел нашёл их при запуске. Эти два параметра используются только, если узел не инициализирован. Если инициализирован (есть файлы данных), то параметр `ETCD_INITIAL_CLUSTER_STATE` игнорируется. Если кластер не создан, то нужно установить "`new`" и перечислить все узлы, которые должны быть запущены при создании кластера. Для создания кластера нужно запустить процессы etcd на всех перечисленных узлах в любом порядке (параллельно). Узлы выберут лидера и кластер будет создан.

Значения этого параметра должны быть одинаковы на всех узлах кластера. В значении должны быть перечислены **`$ETCD_NAME=$ETCD_INITIAL_ADVERTISE_PEER_URLS`**. узлов кластера. Заменять IP на имя хоста нельзя, нужно указывать в точности - либо IP, либо имя хоста. Иначе будет ошибка Cluster ID mismatch.

Для случаев, когда список узлов неизвестен или IP-адреса неизвестны, так как узлы будут получать IP-адреса по DHCP, предназначены способы:

1) Custom etcd discovery - нужен существующий кластер etcd, используется параметр `ETCD_DISCOVERY`

2) Public etcd discovery - нужно получить адрес командой `curl https://discovery.etcd.io/new?size=3` и указать в параметре `ETCD_DISCOVERY`

3) DNS discovery - нужны `SRV` записи `_etcd-server-суффикс._tcp.домен` или `_etcd-server-ssl-суффикс._tcp.домен` в DNS, которые указывают на A-записи имён хостов-членов кластера etcd. Используется параметр `ETCD_DISCOVERY_SRV=домен` и `ETCD_DISCOVERY_SRV_NAME=суффикс`. Суффикс может быть пустым.

Способы описаны в документации: https://etcd.io/docs/v3.7/op-guide/clustering/

Cluster ID

- При static bootstrapping, Cluster ID генерируется из `initial-cluster`, отсортированного по названию узлов и `initial-cluster-token`
- `initial-advertise-peer-urls` должен включать точную строку узла из `initial-cluster`
- все узлы независимо друг от друга должны вычислить одинаковый Cluster ID
- По умолчанию, `initial-cluster-token=etcd-cluster`

```
Для --initial-
cluster=a1=http://localhost:2380,a2=http://127.0.0.2:2380,a3=http://127.0.0.3:2380
--initial-cluster-token etcd-cluster
"cluster_id":1525466304649484645
Для --initial-
cluster=a1=http://127.0.0.1:2380,a2=http://127.0.0.2:2380,a3=http://127.0.0.3:2380
--initial-cluster-token etcd-cluster
"cluster_id" : 7855965530579937127
```


Cluster ID

При запуске и начальной инициализации способом static bootstrapping каждый узел рассчитывает Custer ID. Значения рассчитываются независимо и должны совпасть, иначе узлы не объединятся в кластер.

При использовании etcd discovery Cluster ID рассчитывается на основе списка узлов.

cluster_id генерируется из параметра `initial-cluster` отсортированного по алфавиту имён узлов и `initial-cluster-token`. Сортировка выполняется для обеспечения нечувствительности к порядку перечисления узлов. `initial-cluster` должен включать точное значение для каждого узла из их `initial-advertise-peer-urls`: если IP, то IP в обоих параметрах, если имя то такое же имя в обоих параметрах:

`initial-cluster=name=initial-advertise-peer-urls,...`

У `initial-cluster-token` значение по умолчанию `etcd-cluster`.

Каждый узел нового кластера запускается с одинаковыми значениями `initial-cluster` и `initial-cluster-token`, все узлы независимо друг от друга должны вычислить одинаковый Cluster ID.

Если создаются кластеры с одинаковыми значениями IP-адресов или имён/портов, то стоит использовать разные `initial-cluster-token`, чтобы их Cluster ID отличались.

При выполнении восстановления с помощью команды `etcdutl snapshot restore` создается новый кластер. Cluster ID генерируется по тем же правилам: на основе `initial-cluster` и `initial-cluster-token`.

Значение Cluster ID может проверяться клиентами etcd для защиты от подмены кластера etcd. В дополнение ещё может проверяться RaftTerm - счётчик выбора лидера при запуске узлов кластера etcd для защиты от взаимодействия со старым лидером или follower, отдающим данные от старого лидера.

Patroni видя тот же Cluster ID, но меньший RaftTerm, чем он видел не взаимодействует с узлами такого кластера etcd до тех пор, пока RaftTerm не превысит виденное Patroni значение. Если Cluster ID поменялся, то RaftTerm игнорируется и Patroni взаимодействует с таким кластером etcd.

Файл с переменными окружения

- Пример начальной конфигурации для **первого** узла из трёх в файле службы `/opt/tantor/etc/etcd/etcd.conf`:

```
ETCD_NAME=a1
ETCD_DATA_DIR=/opt/tantor/var/lib/etcd/a1.etcd
ETCD_LISTEN_PEER_URLS=http://127.0.0.1:2380
ETCD_LISTEN_CLIENT_URLS=http://127.0.0.1:2379
ETCD_INITIAL_ADVERTISE_PEER_URLS=http://127.0.0.1:2380
ETCD_INITIAL_CLUSTER=a1=http://127.0.0.1:2380,a2=http://127.0.0.2:2380,a3
=http://127.0.0.3:2380
ETCD_ADVERTISE_CLIENT_URLS=http://127.0.0.1:2379
ETCD_HEARTBEAT_INTERVAL=1000
ETCD_ELECTION_TIMEOUT=10000
ETCD_AUTO_COMPACTION_RETENTION=15s
ETCD_SNAPSHOT_COUNT=100
ETCD_SNAPSHOT_CATCHUP_ENTRIES=100
ETCD_QUOTA_BACKEND_BYTES=100000000
```

- После запуска кворума (для 3 узлов кворум составляет 2 узла), кластер инициализируется, выберет лидера и начнёт принимать запросы

Файл с переменными окружения

Пример начальной конфигурации для **первого** узла из трёх:

```
ETCD_NAME=a1
ETCD_DATA_DIR=/opt/tantor/var/lib/etcd/a1.etcd
ETCD_LISTEN_PEER_URLS=http://127.0.0.1:2380
ETCD_LISTEN_CLIENT_URLS=http://127.0.0.1:2379
ETCD_INITIAL_ADVERTISE_PEER_URLS=http://127.0.0.1:2380
ETCD_INITIAL_CLUSTER=a1=http://127.0.0.1:2380,a2=http://127.0.0.2:2380,a3=htt
p://127.0.0.3:2380
ETCD_ADVERTISE_CLIENT_URLS=http://127.0.0.1:2379
ETCD_HEARTBEAT_INTERVAL=1000
ETCD_ELECTION_TIMEOUT=10000
ETCD_AUTO_COMPACTION_RETENTION=15s
ETCD_SNAPSHOT_COUNT=100
ETCD_SNAPSHOT_CATCHUP_ENTRIES=100
ETCD_QUOTA_BACKEND_BYTES=100000000
```

Пример параметров конфигурации для **второго узла** из трёх:

```
ETCD_NAME=a2
ETCD_DATA_DIR=/opt/tantor/var/lib/etcd/a2.etcd
ETCD_LISTEN_PEER_URLS=http://127.0.0.2:2380
ETCD_LISTEN_CLIENT_URLS=http://127.0.0.2:2379
ETCD_INITIAL_ADVERTISE_PEER_URLS=http://127.0.0.2:2380
ETCD_INITIAL_CLUSTER=a1=http://127.0.0.1:2380,a2=http://127.0.0.2:2380,a3=htt
p://127.0.0.3:2380
ETCD_ADVERTISE_CLIENT_URLS=http://127.0.0.2:2379
...
```

В примере, `127.0.0.1` ... `127.0.0.3` - IP адреса интерфейсов для общения узлов между собой. `127.0.0.1` ... `127.0.0.3` - IP адреса интерфейсов для общения узлов с клиентами.

`ETCD_NAME` и `ETCD_DATA_DIR` должны быть уникальными. Если не указать `ETCD_DATA_DIR`, то директория с данными узла будет создана в домашнем каталоге пользователя, под которым запустится процесс `etcd`.

После того, как будет запущен кворум (для 3 узлов кворум составляет 2 узла), кластер инициализируется, выберет лидера и начнет принимать запросы.

Конфигурационный файл etcd

- Формат YAML
- Запуск с конфигурационным файлом:

```
/opt/tantor/usr/bin/etcd --config-file=/opt/tantor/etc/etcd/etcd.yml
```

- или

```
ETCD_CONFIG_FILE=/opt/tantor/etc/etcd/etcd1.yml /opt/tantor/usr/bin/etcd
```

- Пример содержимого конфигурационного файла:

```
name: 'a1'
data-dir: /opt/tantor/var/lib/etcd/a1.etcd
listen-peer-urls: http://127.0.0.1:2380
listen-client-urls: http://127.0.0.1:2379
initial-advertise-peer-urls: http://127.0.0.1:2380
advertise-client-urls: http://127.0.0.1:2379
initial-cluster: a1=http://127.0.0.1:2380
initial-cluster-token: 'etcd-cluster'
auto-compaction-retention: "15s"
log-format: console
log-outputs: ['stdout','/opt/tantor/var/log/etcd/a1.log']
```


Конфигурационный файл etcd

Запуск с конфигурационным файлом:

```
export GOMEMLIMIT=256MiB
export GOGC=50
export GOMAXPROCS=2
/opt/tantor/usr/bin/etcd --config-file=/opt/tantor/etc/etcd/etcd.yml
```

или

```
ETCD_CONFIG_FILE=/opt/tantor/etc/etcd/etcd.yml /opt/tantor/usr/bin/etcd
```

Пример содержимого конфигурационного файла:

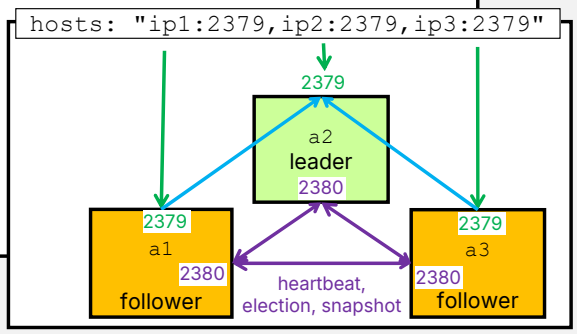
```
name: 'a1'
data-dir: /opt/tantor/var/lib/etcd/a1.etcd
listen-peer-urls: http://127.0.0.1:2380
listen-client-urls: http://127.0.0.1:2379
initial-advertise-peer-urls: http://127.0.0.1:2380
advertise-client-urls: http://127.0.0.1:2379
initial-cluster: a1=http://127.0.0.1:2380
initial-cluster-token: 'etcd-cluster'
initial-cluster-state: 'new'
heartbeat-interval: 1000
election-timeout: 10000
auto-compaction-retention: "15s"
snapshot-count: 100
snapshot-catchup-entries: 100
quota-backend-bytes: 1000000000
log-level: warn
log-format: console
log-outputs: ['stdout','/opt/tantor/var/log/etcd/a1.log']
log-rotation: true
log-rotation-config-json: '{"maxsize": 1, "maxage": 1, "maxbackups": 1,
"localtime": true, "compress": true}'
```

https://github.com/etcd-io/etcd/blob/main/etcd.conf.yml.sample

Запуск etcd с использованием параметров

- Пример параметров (флагов) для третьего узла из трёх:

```
/opt/tantor/usr/bin/etcd --name=a3
--data-dir=/opt/tantor/var/lib/etcd/a3.etcd
--listen-peer-urls http://127.0.0.3:2480
--listen-client-urls http://127.0.0.3:2479
--initial-advertise-peer-urls http://127.0.0.3:2380 \
--initial-cluster
a1=http://127.0.0.1:2380,a2=http://127.0.0.2:2380,a3=http://127.0.0.3:2380
--initial-cluster-token etcd-cluster
--advertise-client-urls http://127.0.0.3:2379
--heartbeat-interval=1000
--election-timeout=10000
--auto-compaction-retention=15s
--snapshot-count=100
--snapshot-catchup-entries=100
--quota-backend-bytes=1000000000
--log-level=warn
--log-format=console
--log-outputs=/opt/tantor/var/log/etcd/a1.log
```



Запуск etcd с использованием параметров

Пример параметров (флагов) вместо переменных окружения для третьего узла из трёх:

```
/opt/tantor/usr/bin/etcd --name=a3
--data-dir=/opt/tantor/var/lib/etcd/a3.etcd
--listen-peer-urls http://127.0.0.3:2480
--listen-client-urls http://127.0.0.3:2479
--initial-advertise-peer-urls http://127.0.0.3:2380
--initial-cluster
a1=http://127.0.0.1:2380,a2=http://127.0.0.2:2380,a3=http://127.0.0.3:2380
--initial-cluster-token etcd-cluster
--advertise-client-urls http://127.0.0.3:2379
--heartbeat-interval=1000
--election-timeout=10000
--auto-compaction-retention=15s
--snapshot-count=100
--snapshot-catchup-entries=100
--quota-backend-bytes=1000000000
--log-level=warn
--log-format=console
--log-outputs=/opt/tantor/var/log/etcd/a1.log
```

Создание, просмотр, удаление ключей

- Запись ключа и его значения:

```
etcdctl --endpoints=http://127.0.0.1:2379 put key1 value1
OK
```

- Чтение значения ключа в формате по умолчанию:

```
etcdctl --endpoints=http://127.0.0.1:2379 get key1 -w simple
key1
value1
```

- Чтение ключа в формате JSON:

```
etcdctl --endpoints=http://127.0.0.1:2379 get key1 -w json
{"header":{"cluster_id":2037210783374497686,"member_id":13195394291058371180,"revision":3002,"raft_term":2},"kvs":[{"key":"a2V5MQ==","create_revision":3002,"mod_revision":3002,"version":1,"value":"dmFsdWUx"}],"count":1}
```

- Список всех ключей, это опция **--prefix**:

```
etcdctl --endpoints=http://127.0.0.1:2379 get '' --prefix --keys-only
key
key1
```

- Удаление ключа:

```
etcdctl --endpoints=http://127.0.0.1:2379 del key1
1
```


Создание, просмотр, удаление ключей

Запись ключа и его значения (в **--endpoints** можно указывать несколько адресов через запятую):

```
etcdctl put key1 value1 --
endpoints=http://127.0.0.1:2379,http://127.0.0.2:2379,http://127.0.0.3:2379
```

Чтение значения ключа:

```
etcdctl --endpoints=http://127.0.0.1:2379 get key1
key1
value1
etcdctl --endpoints=http://:2379 get key1 -w json
{"header":{"cluster_id":12507812779319777084,"member_id":13195394291058371180,"revision":4,"raft_term":2},"kvs":[{"key":"a2V5MQ==","create_revision":2,"mod_revision":4,"version":3,"value":"dmFsdWUx"}],"count":1}
```

`version` - сколько раз менялось значение ключа

`create_revision` и `mod_revision` - ревизия создания и изменения ключа.

Описание остальных свойств ключа:

https://etcd.io/docs/v3.7/dev-guide/api_concurrency_reference_v3/

Список всех ключей это опция --prefix:

```
etcdctl --endpoints=:2379 get '' --prefix --keys-only
```

Удаление ключа:

```
etcdctl --endpoints=http://127.0.0.1:2379 del key1
```

Можно читать и удалять ключи по префиксу (начальным символам в имени ключа), удалять диапазоны ключей, получить уведомление при изменении значения ключа (`watch`), читать предыдущие значения ключа https://etcd.io/docs/v3.7/dev-guide/interacting_v3/

Можно наблюдать за ключами и диапазонами ключей, указав имя существующего или не существующего ключа:

```
etcdctl watch key1 key9
```

При изменении ключа наблюдатель немедленно получить уведомление о типе команды (`PUT` или `DELETE`) с новым значением. При удалении ключа, наблюдение не прекращается.

Клиенты, использующие **watch**, которые **медленно получают обновления ключей**, негативно влияют на etcd тем, что etcd должен сохранять историю изменений ключей, чтобы передать их "медленным активным watcheram". Очистка ключей удаляет старые версии ключей.

Кроме ключей, watches и их revisions есть "named lock", они редко используются.

Создание, просмотр, удаление ключей

- Создание лизинга на 60 секунд:

```
etcdctl lease grant 60
lease 6a6c9fe254648811 granted with TTL(60s)
```

- Добавление ключа в лизинг, по истечении времени лизинг и ключ удаляются:

```
etcdctl put key2 val2 --lease=6a6c9fe254648811
OK
```

- Проверка оставшегося времени лизинга:

```
etcdctl lease timetolive 6a6c9fe254648811
lease 6a6c9fe254648811 granted with TTL(60s), remaining(49s)
```

- Досрочное удаление лизинга с его ключами:

```
etcdctl lease revoke 6a6c9fe254648811
lease 6a6c9fe254648811 revoked
```

- Список параметров утилиты:

```
etcdctl get --help
Usage:
  etcdctl get [options] <key> [range end] [flags]
```

- Чтение диапазона ключей:

```
etcdctl get ke kez --keys-only --limit=10
```


Создание, просмотр, удаление ключей

Создание лизинга на 60 секунд:

```
etcdctl lease grant 60
lease 6a6c9fe254648811 granted with TTL(60s)
```

Добавление ключа в лизинг, что ограничивает время жизни (TTL) ключа:

```
etcdctl put key2 val2 --lease=6a6c9fe254648811
OK
```

В течение лизинга ключ доступен:

```
etcdctl get key2
key2
val2
```

По истечению времени лизинг и прикреплённые к нему ключи исчезают:

```
etcdctl get key2
```

В отдельной сессии можно запустить утилиту для продления лизинга пока утилита работает:

```
etcdctl lease keep-alive 6a6c9fe25464881e
lease 6a6c9fe254648811  keepalived with TTL(60)
lease 6a6c9fe254648811  keepalived with TTL(60)
lease 6a6c9fe254648811  keepalived with TTL(60)
^C
```

Проверка оставшегося времени лизинга:

```
etcdctl lease timetolive 6a6c9fe254648811
lease 6a6c9fe254648811 granted with TTL(60s), remaining(49s)
```

Если время лизинга не истекло, лизинг можно удалить досрочно:

```
etcdctl lease revoke 6a6c9fe254648811
lease 6a6c9fe254648811 revoked
```

Чтение диапазона ключей. Будут выданы ключи, попавшие в диапазон. Ключ с именем key 9 может отсутствовать, ошибки не будет. Сравнение имени ключа выполняется посимвольно.

```
etcdctl get ke kez --keys-only --limit=10
key1
```

Список опций утилиты для просмотра ключей:

```
etcdctl get --help
Usage:
  etcdctl get [options] <key> [range_end] [flags]
```

Проверка статуса узлов

- Список узлов:

```
etcdctl --endpoints=http://127.0.0.1:2379 member list -w table
+------------------+---------+------+-----------------------+-----------------------+------------+
|        ID        | STATUS  | NAME |      PEER ADDRS       |     CLIENT ADDRS      | IS LEARNER |
+------------------+---------+------+-----------------------+-----------------------+------------+
| 23284278a2d8a0e6 | started |   a2 | http://127.0.0.2:2380 | http://127.0.0.2:2379 |      false |
| b71f75320dc06a6c | started |   a1 | http://127.0.0.1:2380 | http://127.0.0.1:2379 |      false |
+------------------+---------+------+-----------------------+-----------------------+------------+
```

- Проверка доступности узлов:

```
etcdctl --endpoints=:2379 endpoint health --cluster -w table
+-----------------------+--------+-------------+-------+
|       ENDPOINT        | HEALTH |    TOOK     | ERROR |
+-----------------------+--------+-------------+-------+
| http://127.0.0.2:2379 |   true |  9.317972ms |       |
| http://127.0.0.1:2379 |   true | 19.352859ms |       |
+-----------------------+--------+-------------+-------+
```

- Названия параметров покажут `-w fields` или `-w table`:

```
etcdctl --endpoints=http://:2379 endpoint status -w fields
...
"DBSize" : 20480
"DBSizeInUse" : 16384
```


Проверка статуса узлов

Список узлов:

```
etcdctl --endpoints=http://127.0.0.1:2379 member list
a2d8a0e6, started, a2, http://127.0.0.2:2380, http://127.0.0.2:2379, false
0dc06a6c, started, a1, http://127.0.0.1:2380, http://127.0.0.1:2379, false
0001902d, started, a3, http://127.0.0.3:2380, http://127.0.0.3:2379, false
```

В параметре `endpoints` можно через запятую перечислить несколько узлов:

```
--endpoints=http://127.0.0.1:2379,http://127.0.0.1:2379,http://127.0.0.1:2379
```

Проверка доступности узлов:

```
etcdctl --endpoints=http://127.0.0.1:2379 endpoint health --cluster
http://127.0.0.2:2379 is healthy: successfully committed: took = 5.451879ms
http://127.0.0.1:2379 is healthy: successfully committed: took = 12.325891ms
http://127.0.0.3:2379 is unhealthy: failed to commit: deadline exceeded
Error: unhealthy cluster
```

Лидера и выбран ли лидер покажет команда:

```
etcdctl --endpoints=http://127.0.0.1:2379 endpoint status --cluster -w table
http://127.0.0.1:2379, b71f75320dc06a6c, 3.7.1, 3.7.0, 20 kB, 16 kB, 20%, 2.1
GB, false, false, 2, 28, 28, , , false
```

Посмотреть названия параметров можно опциями `-w fields` или `-w table`:

```
etcdctl --endpoints=http://127.0.0.1:2379 endpoint status -w fields
"ClusterID" : 2037210783374497686
"MemberID" : 13195394291058371180
"Version" : "3.7.1"
"DBSize" : 708595712
"DBSizeInUse" : 1069056
"DBSizeQuota" : 1000000000
"Leader" : 13195394291058371180
"IsLearner" : false
"RaftIndex" : 45
"RaftTerm" : 5
"RaftAppliedIndex" : 45
```

Табличный (`table`) вывод слишком широкий. В выводе типа `fields` можно **сопоставить** значения и выяснить, является ли узел лидером.

Добавление узлов в кластер etcd

- Для создания кластера из одного узла, указать этот узел:
  - `ETCD_INITIAL_CLUSTER=a1=http://127.0.0.1:2380`
- Затем по одному добавлять узлы:

```
etcdctl --endpoints=http://127.0.0.1:2379 member add a2 --peer-urls
http://127.0.0.2:2380
Member 63cdccba2302c4b7 added to cluster 4e4ebccffe2df54d
ETCD_NAME=a2
ETCD_INITIAL_CLUSTER="a2=http://127.0.0.2:2380,a1=http://127.0.0.1:2380"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://127.0.0.2:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

- Запустить процесс etcd с этими параметрами плюс:

```
ETCD_DATA_DIR=/opt/tantor/var/lib/etcd/a2.etcd
ETCD_LISTEN_PEER_URLS=http://127.0.0.2:2380
ETCD_LISTEN_CLIENT_URLS=http://127.0.0.2:2379
```

- Директория добавляемого узла должна быть пустой
  - если процесс запускался и выдал ошибку, он создаст директорию, её нужно удалить

Добавление узлов в кластер etcd

Если нужно создать кластер etcd из одного узла, который сразу после запуска станет лидером и затем вручную добавлять последователей, то нужно указать значение для параметра:

`ETCD_INITIAL_CLUSTER=a1=http://127.0.0.1:2380`

Сразу после запуска процесса `ETCD_NAME=a1` будет создан кластер etcd. Добавлять узлы по процедуре:

1) добавить новый узел в конфигурацию кластера подсоединившись по gRPC (HTTP мог использоваться до версии 3.6) или командой лидеру или работающему узлу:

`etcdctl --endpoints=http://127.0.0.1:2379 member add a2 --peer-urls http://127.0.0.2:2380`

Команда выдаст конфигурацию, которую нужно использовать при запуске нового узла:

```
Member 63cdccba2302c4b7 added to cluster 4e4ebccffe2df54d
ETCD_NAME=a2
ETCD_INITIAL_CLUSTER="a2=http://127.0.0.2:2380,a1=http://127.0.0.1:2380"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://127.0.0.2:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

Для добавляемых узлов должно быть установлено значение `"existing"`.

Директория данных добавляемого узла при запуске процесса etcd должна быть пустой.

`ETCD_INITIAL_CLUSTER` на добавляемом узле должен включать актуальный список узлов кластера и добавляемый узел. **Если какой-то узел не работает, то его надо исключить из кластера ещё до добавления нового узла.**

2) Запустить новый процесс etcd с параметрами, выданными утилитой плюс параметры:

```
ETCD_DATA_DIR=/opt/tantor/var/lib/etcd/a2.etcd
ETCD_LISTEN_PEER_URLS=http://127.0.0.2:2380
ETCD_LISTEN_CLIENT_URLS=http://127.0.0.2:2379
```

Параметры таймаутов `ETCD_HEARTBEAT_INTERVAL` и `ETCD_ELECTION_TIMEOUT` должны быть одинаковы на всех узлах.

Новые узлы нужно добавлять **последовательно, по одному узлу**.

При добавлении нового участника лидер реплицирует все свои данные на добавляемый узел. **Если объем данных велик, лидер может не успеть передать `heartbeats` последователям и они начнут перевыборы лидера.** Новый узел начнет отвечать клиентам после приёма снэпшота от лидера. **`etcd` не рассчитан на частое удаление и добавление узлов.**

Удаление узла из кластера etcd

- Узлы удаляются, если:
  - узел неработоспособен, его нужно быстро удалить
  - нужно уменьшить число узлов
  - нужно сменить адрес узла, перенести на другой хост
- Неработающие узлы влияют на кворум
- Для удаления узла из кластера:
  - Остановить процесс, проверить статус и адрес:

```
etcdctl --endpoints=http://127.0.0.1:2379 endpoint health --cluster -w table
+-----------------------+--------+-------------+--------------------------+
|       ENDPOINT        | HEALTH |     TOOK    |           ERROR          |
+-----------------------+--------+-------------+--------------------------+
| http://127.0.0.2:2379 |   true |   8.801032ms |                          |
| http://127.0.0.1:2379 |   true |   8.318939ms |                          |
| http://127.0.0.3:2379 |  false | 5.001322304s | context deadline exceeded |
+-----------------------+--------+-------------+--------------------------+
Error: unhealthy cluster
```


Удаление узла из кластера etcd

Узлы удаляются, если:
1) узлы неработоспособны
2) нужно уменьшить число узлов
3) нужно сменить IP-адрес узла или перенести узел на другой хост.

Неработоспособные узлы не должны присутствовать в кластере, так как они влияют на число узлов кворума. При наличии неработоспособных узлов нельзя добавлять новые узлы, нужно удалить неработоспособные. Если неработоспособными стало большое число узлов и оставшиеся узлы не образуют кворум, то такой кластер придётся удалить, создать новый кластер и импортировать снэпшот для восстановления ключей прежнего кластера. Технически можно было бы оставить кластер неработоспособным, но этого не сделано, так как это нарушает строгую согласованность.

Если директория узла повредилась, процесс etcd не стартует и цель запустить процесс, то можно не удалять узел из кластера. Достаточно удалить директорию с данным кластера, установить:

```
ETCD_INITIAL_CLUSTER_STATE=existing
```

и запустить процесс etcd. Процесс получит конфигурацию с лидера.

Для удаления узла из кластера:
1) Остановить процесс (службу) etcd узла, если процесс не был остановлен
2) Проверить доступность удаляемого узла:

```
etcdctl --endpoints=http://127.0.0.1:2379 endpoint health --cluster -w table
+-----------------------+--------+-------------+--------------------------+
|       ENDPOINT        | HEALTH |     TOOK    |           ERROR          |
+-----------------------+--------+-------------+--------------------------+
| http://127.0.0.2:2379 |   true |   8.801032ms |                          |
| http://127.0.0.1:2379 |   true |   8.318939ms |                          |
| http://127.0.0.3:2379 |  false | 5.001322304s | context deadline exceeded |
+-----------------------+--------+-------------+--------------------------+
Error: unhealthy cluster
```

Удаление узла из кластера etcd (продолжение)

- Найти идентификатор удаляемого узла:

```
etcdctl --endpoints=http://127.0.0.1:2379 member list
...
cc4a03f30001902d, started, a3, http://127.0.0.3:2380, http://127.0.0.3:2379, false
```

- Удалить из конфигурации кластера узел:

```
etcdctl --endpoints=http://127.0.0.1:2379 member remove cc4a03f30001902d
Member cc4a03f30001902d removed from cluster ad94ad741c72573c
```

- Удалить директорию удалённого из конфигурации узла

- Если неработоспособными стало большое число узлов и оставшиеся узлы не образуют кворум, то такой кластер придётся удалить, создать новый кластер и импортировать снэпшот для восстановления данных прежнего кластера

Удаление узла из кластера etcd (продолжение)

3) Найти идентификатор удаляемого узла:

```
etcdctl --endpoints=http://127.0.0.1:2379 member list
...
cc4a03f30001902d, started, a3, http://127.0.0.3:2380, http://127.0.0.3:2379, false
```

Статус узла будет ложно указан как **started**, несмотря на то, что узел не работает.

4) Удалить из конфигурации кластера узел:

```
etcdctl --endpoints=http://127.0.0.1:2379 member remove cc4a03f30001902d
Member cc4a03f30001902d removed from cluster ad94ad741c72573c
```

5) Удалить директорию удалённого из конфигурации узла:

```
rm -rf /opt/tantor/var/lib/etcd/a3.etcd
```

Смена лидера

- Команда, показывающая какой узел является лидером:

```
etcdctl --endpoints=http://127.0.0.1:2379 endpoint status -w table --command-timeout=5s
+-----------------------+------------------+---------+--------+-----------+
|       ENDPOINT        |        ID        | DB SIZE | IN USE | IS LEADER |
+-----------------------+------------------+---------+--------+-----------+
| http://127.0.0.1:2379 | b71f75320dc06a6c |  2.1 MB | 2.1 MB |      true |
+-----------------------+------------------+---------+--------+-----------+
```

  - по умолчанию, таймаут ожидания ответа у etcdctl 5 секунд

- Команда смены лидера:

```
etcdctl --endpoints=http://127.0.0.1:2379 move-leader b71f75320dc06a6c
Leadership transferred from b71f75320dc06a6c to b71f75320dc06a6c
```

- Команда выборов лидера с рекомендуемым лидером:

```
etcdctl --endpoints=http://127.0.0.1:2379 elect my a1
my/b71f75320dc06a6c
a1
```

  - нужно указать произвольное название выборов

Смена лидера

Смена лидера нужна, если нужно перезапустить процесс etcd или удалить узел из кластера. Можно положиться на автоматический выбор нового лидера, но на обнаружение исчезновения лидера тратится время, заданное `ETCD_ELECTION_TIMEOUT`, по умолчанию 1 секунда, но этот параметр связан (устанавливается в 5-10 раз больше) с параметром `ETCD_HEARTBEAT_INTERVAL` (по умолчанию, 100 миллисекунд), который зависит от пинга между узлами. Если узлы находятся далеко друг от друга, то значения увеличиваются.

Сначала нужно узнать идентификатор узла, которому передать лидерство командой:

```
etcdctl --endpoints=http://127.0.0.1:2379 endpoint status -w table --command-timeout=5s
+-----------------------+------------------+---------+--------+-----------+
|       ENDPOINT        |        ID        | DB SIZE | IN USE | IS LEADER |
+-----------------------+------------------+---------+--------+-----------+
| http://127.0.0.1:2379 | b71f75320dc06a6c |  2.1 MB | 2.1 MB |      true |
+-----------------------+------------------+---------+--------+-----------+
```

Часть столбцов не приведена, в реальном результате 16 столбцов или больше (зависит от версии etcd). В параметре `--endpoints` можно через запятую перечислить всех членов кластера или часть из них. Таймаут утилиты на ответ задаётся параметром: `etcdctl --command-timeout=5s`, по умолчанию 5 секунд. При превышении таймаута утилита выдаёт ошибку.

В столбце ID - идентификатор узла, в столбце `IS LEADER` - является ли узел лидером.

Команда смены лидера:

```
etcdctl --endpoints=:2379 move-leader b71f75320dc06a6c
Leadership transferred from b71f75320dc06a6c to b71f75320dc06a6c
```

Может случить, что лидер в кластере отсутствует - ни один из узлов не является лидером.

Второй способ - инициировать выборы. Команду **`elect`** можно выполнить вне зависимости есть лидер или его нет:

```
etcdctl --endpoints=http://127.0.0.1:2379 elect my a1
my/b71f75320dc06a6c
a1
```

В команде опционально можно указать рекомендуемого лидера. Обязательно указывается произвольно выбранное название выборов для логирования.

https://etcd.io/docs/v3.7/tasks/operator/how-to-conduct-elections/

Raft Term

- Term - срок лидерства
  - целое число, увеличивается кандидатом на 1
  - используется для защиты от Split Brain
- Если в течение `election-timeout` узел не получит сообщения от лидера, он инициирует выборы: становится кандидатом, увеличивает на 1 term

```
"index":3,"term":1,"confState":"voters:a1 voters:a2 voters:a3"
a1 became follower at term 1
a1 is starting a new election at term 1
a1 became pre-candidate at term 1
a1 [logterm: 1, index: 3] sent MsgPreVote request to a2 at term 1
a1 [logterm: 1, index: 3] sent MsgPreVote request to a3 at term 1
a1 has received 2 MsgPreVoteResp votes and 0 vote rejections
a1 became candidate at term 2
a1 [logterm: 1, index: 3] sent MsgVote request to a2 at term 2
a1 [logterm: 1, index: 3] sent MsgVote request to a3 at term 2
a1 has received 2 MsgVoteResp votes and 0 vote rejections
a1 became leader at term 2"
```


Raft Term

Term - срок лидерства от начала выборов до прекращения лидерства. Term - целое число, увеличивается на 1 в начале выбора лидера. При самом первом запуске узла Term=0 и узел считает себя `follower`. Если в течение `election-timeout` узел не получит сообщения от лидера, узел инициирует выборы: становится кандидатом, увеличивает на 1 Term, голосует сам за себя, посылает сообщение другим узлам, предлагая проголосовать за себя в новом Term. Другие узлы со статусом follower голосуют за кандидата, если они ещё не голосовали в предложенном Term и журнал кандидата старше журнала голосующего. Если кандидат получает большинство голосов, он становится лидером. Избранный лидер отправляет `heartbeat` всем узлам, это предотвращает новые выборы и сообщает о том, что в Term выбран лидер. Если узел не набирает кворум или истекает срок таймаута на выборы (`election-timeout`), то, по истечении таймаута, узел снова инициирует выборы и увеличивает Term на 1. Чтобы Term не накручивались, используется фаза pre-candidate: узел не станет кандидатом и не сможет увеличить Term, пока не будет кворума, чтобы подтвердить его переход в статус кандидата. Если узел получит `heartbeat` от лидера, то он становится follower в Term лидера. Term используется для защиты от split brain. Если лидер окажется изолирован от большинства узлов, они проведут выборы нового лидера в новом term. Когда связь восстановится, старый лидер получит от нового лидера `heartbeat` с новым Term. Увидев больший Term в сообщении от лидера, прежний лидер поймёт, что есть новый лидер, станет follower, откатит незафиксированные (если нет кворума кворума, то изменения не фиксируются) записи в журнале и синхронизирует записи с новым лидером.


```
"index":3,"term":1,"confState":"voters:a1 voters:a2 voters:a3"
a1 became follower at term 1
a1 is starting a new election at term 1
a1 became pre-candidate at term 1
a1 [logterm: 1, index: 3] sent MsgPreVote request to a2 at term 1
a1 [logterm: 1, index: 3] sent MsgPreVote request to a3 at term 1
a1 has received 2 MsgPreVoteResp votes and 0 vote rejections
a1 became candidate at term 2
a1 [logterm: 1, index: 3] sent MsgVote request to a2 at term 2
a1 [logterm: 1, index: 3] sent MsgVote request to a3 at term 2
a1 has received 2 MsgVoteResp votes and 0 vote rejections
a1 became leader at term 2"
```

Тестирование скорости работы etcd

- Увеличение числа версий ключа:

```
time for i in {1..1000}; do etcdctl --endpoints=http://127.0.0.1:2379 put
key1 value1 > /dev/null || break; done
```

- Пример информационных сообщений, если `ETCD_SNAPSHOT_COUNT=100`, снэпшоты создаются каждые 100 изменений ключа `key1`:

```
{"msg":"triggering snapshot","local-member-applied-index":808,"local-member-
snapshot-index":707,"local-member-snapshot-count":100,"snapshot-forced":false}
{"msg":"saved snapshot to disk","snapshot-index":808}
{"msg":"triggering snapshot","local-member-applied-index":909,"local-member-
snapshot-index":808,"local-member-snapshot-count":100,"snapshot-forced":false}
Раз в 30 секунд файлы снэпшотов удаляются:
{"msg":"purged","path":"/opt/tantor/var/lib/etcd/a1.etcd/member/snap/0000000000000
002-00000000000002c3.snap"}
```


Тестирование скорости работы etcd

Пример команды, увеличивающей число версий ключа:

```
time for i in {1..1000}; do etcdctl --endpoints=http://127.0.0.1:2379 put
key1 value1 > /dev/null || break; done
```

Пример информационных сообщений, если `ETCD_SNAPSHOT_COUNT=100`, снэпшоты создаются каждые 100 изменений ключа `key1`:

```
{"msg":"saved snapshot to disk","snapshot-index":707}
{"msg":"triggering snapshot","local-member-applied-index":808,"local-member-
snapshot-index":707,"local-member-snapshot-count":100,"snapshot-forced":false}
{"msg":"saved snapshot to disk","snapshot-index":808}
{"msg":"triggering snapshot","local-member-applied-index":909,"local-member-
snapshot-index":808,"local-member-snapshot-count":100,"snapshot-forced":false}
```

Раз в 30 секунд файлы снэпшотов удаляются:

```
{"msg":"purged","path":"/opt/tantor/var/lib/etcd/a1.etcd/member/snap/00000000
00000002-00000000000002c3.snap"}
```

Можно протестировать скорость работы при изменении числа узлов. Например, команда, обновляющая ключ 1000 раз, выполняется 23 секунды с 5 узлами, 19 секунд с 1 узлом.

Тестирование использования памяти

- Увеличение числа версий ключа размером 1Мб:

```
for i in {1..1000}; do echo "loop: $i"; dd if=/dev/urandom bs=1024 count=1024
2> /dev/null | etcdctl put key || break; done
```

- Параметры для запуска узла:

```
rm -rf /opt/tantor/var/lib/etcd/a.etcd
etcd --name=a --data-dir=/opt/tantor/var/lib/etcd/a.etcd --listen-client-urls
http://127.0.0.1:2379 --advertise-client-urls http://127.0.0.1:2379 --listen-
peer-urls http://127.0.0.1:2380 --initial-advertise-peer-urls
http://127.0.0.1:2380 --initial-cluster=a=http://127.0.0.1:2380 --initial-
cluster-token etcd-cluster  --log-format console
```

- Результат на одном узле с параметрами по умолчанию:

```
real    0m54.411s
top
PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
20   0   13.2g   2.0g  20224 S   2.3  70.8   0:17.48 etcd
ls -l /opt/tantor/var/lib/etcd/a.etcd/member/snap/db
1189355520
```


Тестирование использования памяти

На ключах маленького размера сложно протестировать использование памяти и места на диске. Нужно использовать ключи большого размера. Ограничение на размер ключа 1.5Мб, поэтому можно использовать ключи меньшего размера.

Пример команды, увеличивающей число версий ключа размером 1Мб:

```
time for i in {1..1000}; do echo "loop: $i"; dd if=/dev/urandom bs=1024
count=1024 2> /dev/null | etcdctl put key || break; done
```

Тестировать использование места на диске, памяти и длительности теста можно на одноузловом кластере. Запускать etcd можно любым способом. Например, с минимальным набором параметров и перед итерациями теста удалив директорию узла:

```
rm -rf /opt/tantor/etc/etcd/default.etcd && etcd
```

или указав адреса и директории:

```
rm -rf /opt/tantor/var/lib/etcd/a.etcd
etcd --name=a --data-dir=/opt/tantor/var/lib/etcd/a.etcd --listen-client-urls
http://127.0.0.1:2379 --advertise-client-urls http://127.0.0.1:2379 --listen-
peer-urls http://127.0.0.1:2380 --initial-advertise-peer-urls
http://127.0.0.1:2380 --initial-cluster=a=http://127.0.0.1:2380 --initial-
cluster-token etcd-cluster --log-format console
```

В отдельном окне терминала запустить команду `top` и наблюдать за процессом. После окончания теста остановить `etcd` набрав `ctrl+c`.

Результат с параметрами по умолчанию:

```
real    0m54.411s
PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
20   0   13.2g   2.0g  20224 S   2.3  70.8   0:17.48 etcd
ls -l /opt/tantor/var/lib/etcd/a.etcd/member/snap/db
1189355520
```

Используется 13.2Гб виртуальной памяти и 2Гб оперативной, что довольно много.

Использование памяти

- пример вывода команды `top`:

```
PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
20   0   13.2g   2.0g  20224 S   2.3  70.8   0:17.48 etcd
```

- реальное использование памяти это RES
- **VIRT** - включает в себя mmap для файла базы узла
  - уменьшается пропорционально уменьшению `snapshot-count`
  - по умолчанию, выделяет 10Гб

```
// initialMmapSize is the initial size of the mmapped region.
// Setting this larger than the potential max db size can prevent
// writer from blocking reader. This only works for linux.
// initialMmapSize = uint64(10 * 1024 * 1024 * 1024)
```

- Отключение overcommit:

```
sysctl -w vm.overcommit_ratio=100
sysctl -w vm.overcommit_memory=2
```


Использование памяти

При настройках оверкоммита linux, OOM killer не убивает процесс etcd, etcd просто подвешивает linux на какое-то время, пока не освободится немного памяти. Память не освобождается и узел работает с периодическими подвисаниями. Более оптимально было бы перезапустить процесс etcd, он всё равно на время подвисания неработоспособен. Для этого можно отключить overcommit:

```
sysctl -w vm.overcommit_ratio=100
sysctl -w vm.overcommit_memory=2
```

Чтобы параметры сохранились при перезапуске linux, нужно создать файл `/usr/lib/sysctl.d/51-overcommit.conf`

При отключенном overcommit процессу etcd не хватит памяти и он остановится. Служба systemd его запустит заново. В момент отключения оверкоммита, перерасхода памяти быть не должно, иначе операционная система перегзапустится.

При работе в контейнере docker, где, обычно, ограничивают память, etcd может выйти за пределы ограничений докера и контейнер перезапустится. Разработчики считают, что etcd не предназначен для работы там, где память ограничена. Утилитой top можно увидеть в столбце **VIRT**, что etcd резервирует **10Гб** виртуальной памяти:

```
PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
20   0   13.2g   2.0g  20224 S   2.3  70.8   0:17.48 etcd
```

VIRT - это mmap для файла `db`:

```
// initialMmapSize is the initial size of the mmapped region.
// Setting this larger than the potential max db size can prevent
// writer from blocking reader. This only works for linux.
// initialMmapSize = uint64(10 * 1024 * 1024 * 1024)
```

https://github.com/etcd-io/etcd/issues/10190

Рекомендации по использованию памяти и другого железа:
https://etcd.io/docs/v3.7/op-guide/hardware

Приводится пример, что для узла маленького кластера etcd достаточно: 2 ядра процессора, 8Гб оперативной памяти, скорость диска 25Мб/с, 1500 IOPS. Маленький означает 200 запросов в секунду, 100 клиентов, объем хранимых данных 100Мб, подходит для хранения данных Kubernetes о 50 контейнерах docker.

Параметр `snapshot-catchup-entries`

- соответствуют переменной окружения `ETCD_SNAPSHOT_CATCHUP_ENTRIES`
- По умолчанию, `snapshot-catchup-entries=5000`
- Большое значение `snapshot-catchup-entries`
  - используется существенно больше оперативной памяти
  - в его пределах последователи могут отстать так, чтобы лидеру не нужно было передавать им снэпшот
- Размер используемой памяти: суммарный размер последних версий ключей в количестве `snapshot-catchup-entries`

Параметр `snapshot-catchup-entries`

Соответствует переменной окружения `ETCD_SNAPSHOT_CATCHUP_ENTRIES`, по умолчанию 5000. Устанавливает сколько ключей и их значений сохраняется в оперативной памяти узла.

Узел может отстать на `ETCD_SNAPSHOT_CATCHUP_ENTRIES` и ему не нужно будет передавать снэпшот. Последователи, в пределах числа изменений, смогут догнать лидера без необходимости полной передачи снэпшота файла базы лидера.

Значение по умолчанию, выбрано из предположений:

```
// Number of entries for slow follower to catch-up after compacting
// the raft storage entries.
// We expect the follower has a millisecond level latency with the leader.
// The max throughput is around 10K. Keep a 5K entries is enough for helping
// follower to catch up.
```

Если последователь получает всю базу (снэпшот), за время передачи базы ключи на лидере могут обновляться. Последователь должен иметь возможность получить ключи, накопившиеся за время получения снэпшота, иначе последователю придётся получать снэпшот снова и снова. Значение параметра `snapshot-catchup-entries` зависит от размера базы. Размер базы ограничивается параметром `quota-backend-bytes`. Размер базы уменьшается после очистки и дефрагментации. Интервалы обновления устанавливаются параметром `auto-compaction-retention`.

Память, выделенная под хранение 5000 ключей зависит от их размера, а размер ключей ограничивается параметром `ETCD_MAX_REQUEST_BYTES`. Если размер каждого ключа или версии ключа 1Мб, то узел выделит 5Гб памяти, что много. При нехватке памяти в операционной системе с оверкоммитом (значения по умолчанию), процесс etcd подвиснет и перестанет отвечать.

При большом размере ключей существенно влияет на память.
https://github.com/etcd-io/etcd/issues/18382

Уменьшение использования памяти

- Добавим параметр `--snapshot-catchup-entries=100`

```
real    0m54.248s
PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
20   0   11.7g   1.2g  789248 S   3.0  40.8   0:17.77 etcd
```

- Добавим параметр `--quota-backend-bytes=1000000000`
  - Размер базы `1020346368`, что превышает `1000000000` и тест прерывается на 841 итерации с ошибкой

```
Error: etcdserver: mvcc: database space exceeded
etcdctl endpoint health --cluster
http://127.0.0.1:2379 is unhealthy: failed to commit proposal: Active
Alarm(s): NOSPACE
Error: unhealthy cluster
```

- Добавим параметр `--auto-compaction-retention=10s`

```
real    0m53.126s
PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
20   0 3844196 898632 469760 S   3.7  30.1   0:19.79 etcd
```

  - Существенно уменьшились `VIRT`, `RES` и размер базы

Уменьшение использования памяти

Добавим параметр `--snapshot-catchup-entries=100`. Получим:

```
real    0m54.248s
PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
20   0   11.7g   1.2g  789248 S   3.0  40.8   0:17.77 etcd
```

Использование памяти уменьшилось с 2Гб до 1.2Гб - это большой прогресс!

Размер файла базы не изменился: `1218764800`.

К предыдущему параметру добавим `--quota-backend-bytes=1000000000`.

Размер базы `1020346368`, что превышает `1000000000` и тест прерывается на 841 итерации с ошибкой:

```
Error: etcdserver: mvcc: database space exceeded
etcdctl endpoint health --cluster
http://127.0.0.1:2379 is unhealthy: failed to commit proposal: Active
Alarm(s): NOSPACE
Error: unhealthy cluster
```

Кластер перестанет принимать команды на запись и нужно будет выполнить compaction, defragmentation, убрать предупреждение:

```
etcdctl alarm disarm
```

Пока не будет убрано предупреждение, кластер не будет принимать команды `put`.

Подстраивать квоту нет смысла, так как при увеличении числа итераций база будет расти. Нужно очищать базу, чтобы она не разрасталась.

Добавим параметр `--auto-compaction-retention=10s`. Получим:

```
real    0m53.126s
PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
20   0 3844196 898632 469760 S   3.7  30.1   0:19.79 etcd
```

Размер базы `533282816`.

Существенно уменьшились `VIRT`, `RES` и размер базы. Замедлений при работе цикла не было.

Параметр `snapshot-count`

- соответствуют параметрам `ETCD_SNAPSHOT_COUNT` и `ETCD_SNAPSHOT_CATCHUP_ENTRIES`
- По умолчанию, `snapshot-catchup-entries=5000`
- По умолчанию, `snapshot-count=10000` (с версии 3.6)
- Большое значение `snapshot-count`
  - WAL хранит больше записей и занимает больше места
  - меньше нагрузка на диск по созданию снэпшотов
- Меньшее значение:
  - снэпшоты создаются более часто
  - при большом числе ключей размер снэпшота большой и узел становится неотзывчивым

Параметр `snapshot-count`

`snapshot-count` задаёт число транзакций, после которых данные из памяти сохраняются в файл базы. Соответствует параметру `ETCD_SNAPSHOT_COUNT`. Значение по умолчанию 10000, начиная **с версии 3.6**. Значение было увеличено в версии 3.2 с 10000 до 100000 из-за того, что создание снэпшотов v2 было трудоёмким. Создание снэпшотов v3 не такое трудоёмкое, поэтому, в версии 3.6 вернули значение 10000.

Изменения ключей записываются в WAL. WAL постоянно растёт и хранит историю изменений ключей. **Снэпшоты выполняются, чтобы можно было удалять старые файлы WAL.** Если снэпшот не будет делаться, то число WAL-файлов будет только увеличиваться, независимо от значения параметра `ETCD_MAX_WALS=5`.

Большое значение `snapshot-count`:

1) увеличивает использование места на диске под файлы WAL

2) позволяет хранить больше записей в WAL.

Слишком маленькое значение заставляет чаще формировать файл базы. При больших объемах файла возникают большие задержки.

Если установить только `--snapshot-count=100` и `--auto-compaction-retention=15s`, тест прерывается на 682 итерации. Это означает, что etcd не отвечал больше 5 секунд выполняя очистку, это неприемлемо долго. Значит за 15 секунд накопилось много изменений и надо уменьшить интервал. Если уменьшить `auto-compaction-retention` до 5 секунд: `--snapshot-count=100` и `--auto-compaction-retention=5s` , то начнутся затыки в течение ~1 секунды после 700 итерации. Использование памяти не изменится, но увеличится время теста:

```
real    1m4.736s
```

Размер файла базы уменьшится до 267Мб.

При размере базы в 6Гб, очистка займёт не меньше 15 секунд. На это время etcd не будет отвечать клиентам. Если другие параметры не помогут, можно попробовать уменьшить `compaction-batch-limit` и увеличить `compaction-sleep-interval`.

https://github.com/etcd-io/etcd/issues/18481

Уменьшение использования диска

- Добавим параметр `--snapshot-count=100`
  - параметр не повлиял ни на память, ни на время выполнения, ни на размер базы
  - параметр уменьшил число WAL файлов с 17 до 5:

```
ls -l /opt/tantor/var/lib/etcd/a.etcd/member/wal/
было:  total 1140936
стало: total  378984
```

- Добавим переменную окружения `GOMEMLIMIT=256MiB`

```
real    0m53.593s
PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
20   0 3713236 762040 507392 S   3.3  25.6   0:20.32 etcd
```

  - Использование памяти уменьшилось с **963776** до **762040**
  - Размер базы не изменился: `514379776`
- С использованием пяти приведённых параметров и переменной окружения `GOMEMLIMIT`, размер базы, WAL, памяти стабильны

Уменьшение использования диска

Добавим параметр `--snapshot-count=100`. Получим:

```
real    0m53.250s
PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
20   0 3844196 963776 538112 S   4.3  32.3   0:18.34 etcd
```

Размер базы `531177472`. Параметр `--snapshot-count=100` не повлиял ни на память, ни на время выполнения, ни на размер базы. Однако, он уменьшил число WAL файлов с 17 до 5:

```
ls -l /opt/tantor/var/lib/etcd/a.etcd/member/wal/
```

было: `total 1140936`

стало: `total 378984`

Каждый WAL файл не меньше `64000000` байт, разница существенная.

Добавим переменную окружения `GOMEMLIMIT=256MiB`:

```
GOMEMLIMIT=256MiB etcd --name=a --data-dir=/opt/tantor/var/lib/etcd/a.etcd --listen-client-urls http://127.0.0.1:2379 --advertise-client-urls http://127.0.0.1:2379 --listen-peer-urls http://127.0.0.1:2380 --initial-advertise-peer-urls http://127.0.0.1:2380 --initial-cluster=a=http://127.0.0.1:2380 --initial-cluster-token etcd-cluster --log-format console --snapshot-catchup-entries=100 --quota-backend-bytes=1000000000 --auto-compaction-retention=10s --snapshot-count=100
```

Получим:

```
real    0m53.593s
PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
20   0 3713236 762040 507392 S   3.3  25.6   0:20.32 etcd
```

Использование памяти уменьшилось с **963776** до **762040**. Размер базы не изменился: `514379776`.

Параметр `GOMEMLIMIT` уменьшает использование памяти на несколько десятков процентов.

С использованием пяти параметров и переменной окружения `GOMEMLIMIT`, размер базы, WAL, памяти стабильны даже при увеличении числа итераций цикла.

Файл базы $ETCD_DATA_DIR/member/snap/db

- Максимальный размер файла `--quota-backend-bytes`
- Если достигается на любом узле, то кластер принимает только запросы на чтение и удаление ключей (`get`, `del`)
- `DBSizeInUse` если дефрагментировать
- `DBSize` размер на диске
- Посмотреть метрики можно командами:

```
etcdctl --endpoints=http://127.0.0.1:2379 endpoint status --write-out=fields |
grep DBSize
"DBSize" : 1089536
"DBSizeInUse" : 1081344
"DBSizeQuota" : 2147483648
ls -sh /opt/tantor/var/lib/etcd/a1.etcd/member/snap/
total 1.1M
1.1M db
curl -s http://127.0.0.1:2379/metrics | egrep ^etcd_mvcc_db_total_size
etcd_mvcc_db_total_size_in_bytes 1.089536e+06
etcd_mvcc_db_total_size_in_use_in_bytes 1.081344e+06
```


Файл базы $ETCD_DATA_DIR/member/snap/db

По умолчанию, максимальный размер файла базы, 2Гб (параметр `--quota-backend-bytes`). Рекомендуемое максимальное значение - 8Гб, выше которого выдаются предупреждения при запуске узла. Лимиты: https://etcd.io/docs/v3.7/dev-guide/limit/

Ограничение действует на узел.

Если ограничение достигается любым из узлов, то кластер начинает принимает только запросы на чтение и удаление.

Для возврата к работоспособности, нужно выполнить compact, defrag и **убрать предупреждение**:

`etcdctl alarm disarm`

Если не убрать предупреждение кластер не принимает команды на запись.

Метрика `etcd_mvcc_db_total_size_in_use_in_bytes` (`"DBSizeInUse"`) показывает размер если дефрагментировать, `etcd_mvcc_db_total_size_in_bytes` (`"DBSize"`) показывает полный размер файла базы, включая пустое место, которое после дефрагментации вернется файловой системе. Обе метрики не должны приближаться к ограничению `--quota-backend-bytes` (`"DBSizeQuota"`)

Посмотреть метрики можно командами:

```
etcdctl --endpoints=http://127.0.0.1:2379 endpoint status --cluster --write-
out=fields | grep DBSize
"DBSize" : 1073152
"DBSizeInUse" : 1064960
"DBSizeQuota" : 2147483648
```

или

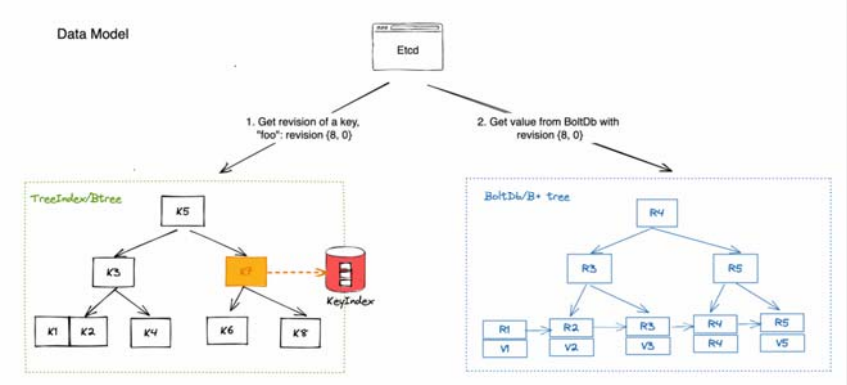
```
curl -s http://127.0.0.1:2379/metrics | egrep ^etcd_mvcc_db_total_size
etcd_mvcc_db_total_size_in_bytes 1.089536e+06
etcd_mvcc_db_total_size_in_use_in_bytes 1.081344e+06
```

Реальный размер файла:

```
ls -sh /opt/tantor/var/lib/etcd/a1.etcd/member/snap/
total 1.1M
1.1M db
ls -lh /opt/tantor/var/lib/etcd/a1.etcd/member/snap/
total 1.1M
-rw------- 1 root root 2.0M db
```

Файл базы $ETCD_DATA_DIR/member/snap/db

- Файл базы (bbolt, свалка) хранит:
- ключи и значения ключей, занимают наибольшее место
- данные об узлах: memberId, id, peerUrls, name, clientURLs
- аутентификационные данные: auth, authRoles, authUsers
- данные о кластере: clusterVersion, downgrade
- метаданные: scheduledCompactRev, finishedCompactRev, consistent_index
- уведомления: alarm
- лизинг (lease): идентификатор и TTL
- удалённые узлы: memberId



Файл базы $ETCD_DATA_DIR/member/snap/db

Файл базы лежит в директории:

```
ls -sh /opt/tantor/var/lib/etcd/a1.etcd/member/snap/
total 1.1M
1.1M db
```

Файл базы (bbolt, свалка) хранит:

1) данные ключей, занимают наибольшее место
2) уведомления (alarm) - если было предупреждение
3) аутентификационные данные (**auth, authRoles, authUsers**)
4) данные о кластере: **clusterVersion, downgrade** (разрешен даунгрейд и на какую версию)
5) лизинг (lease): идентификатор и TTL
6) данные об узлах (members): **memberId, id, peerUrls, name, clientURLs**
7) удалённые узлы: **memberId**
8) метаданные: **scheduledCompactRev, finishedCompactRev, consistent_index** (ссылка на запись WAL, которая была применена к файлу).

Файл содержит всё, что было перенесено из WAL. Формат хранения использует индекс типа b+tree.

Очистка (compation) убирает старые записи именно из этого файла, который хранит данные в структуре индекса. Дефрагментация создаёт новый файл и удаляет старый, аналог команды REINDEX в PostgreSQL.

Есть утилита для просмотра содержимого файла https://github.com/etcd-io/etcd/tree/main/tools/etcd-dump-db

Описание структуры хранения: https://etcd.io/docs/v3.7/learning/persistent-storage-files/

Файлы снэпшотов и временные файлы

- В `$ETCD_DATA_DIR/member/snap` могут создаваться:
- файл `*.snap.db`, содержащий базу, скачиваемую с лидера. Файл появляется при:
  - запуске узла (за время простоя узел мог отстать)
  - по команде лидера использовать переданный образ
- файлы `*.snap`, их число `ETCD_MAX_SNAPSHOTS`
- файлы `*.snap.broken`. Это файлы, которые не смогли загрузиться при запуске узла или при восстановлении из бэкапа или при обновлении на новую версию
- файлы `tmp*`. Это файлы базы в процессе скачивания с лидера. После успешного скачивания, файл переименуется в `*.snap.db`

Файлы снэпшотов и временные файлы

В директории `$ETCD_DATA_DIR/member/snap` могут создаваться:
1) файл `*.snap.db`, содержащий базу, скачиваемую с лидера. Файл появляется при:
a) запуске узла (за время простоя узел мог отстать)
b) по команде лидера использовать переданный им образ базы лидера при восстановлении и смене версий софта.
По завершению восстановления файл `*.snap.db` не удаляется, он должен удалиться в течение 30 секунд.
2) файлы `*.snap`, в количестве до `ETCD_MAX_SNAPSHOTS` (по умолчанию, `5`). Они являются копией (снэпшотом) базы узла. Параметр устарел в версии 3.6 и будет удалён в версии 3.8.
3) файлы `*.snap.broken`. Это файлы, которые не смогли загрузиться при запуске узла или при восстановлении из бэкапа или при смене версий софта
4) файлы `tmp*`. Это файлы базы в процессе скачивания с лидера. После успешного скачивания, файл переименуется в `*.snap.db`. Автоматически удаляются, начиная с версии 3.5.

Файлы WAL

- WAL (журнал) находится в директории каждого узла:

```
ls -Xsw 1 /opt/tantor/var/lib/etcd/a1.etcd/member/wal
total 441488
62500 0.tmp
62500 1.tmp
63500 0000000000000000-0000000000000000.wal
63496 0000000000000001-000000000000007e.wal
63496 0000000000000002-00000000000000bc.wal
63496 0000000000000003-00000000000000fa.wal
62500 0000000000000004-0000000000000146.wal
```

- В WAL файлах хранятся записи переменной длины (frames), защищённые контрольной суммой CRC32
- `0.tmp` и `1.tmp` создаются для резервирования места
- Если снэпшоты (перенос данных из WAL в файл данных) делаются редко, то число WAL файлов может превысить `ETCD_MAX_WALS` (`--max_wals`)

Файлы WAL

WAL (журнал) находится в директории каждого узла:

```
ls -Xsw 1 $ETCD_DATA_DIR/member/wal
ls -Xsw 1 /opt/tantor/var/lib/etcd/a1.etcd/member/wal
total 441488
62500 0.tmp
62500 1.tmp
63500 0000000000000000-0000000000000000.wal
63496 0000000000000001-000000000000007e.wal
63496 0000000000000002-00000000000000bc.wal
63496 0000000000000003-00000000000000fa.wal
62500 0000000000000004-0000000000000146.wal
```

В названии: порядковый номер, тире, указатель (index) на первую запись или снэпшоты ключей в файле.

В файлах хранятся записи переменной длины (frames), защищённые контрольной суммой CRC32.

Число файлов ограничено `ETCD_MAX_WALS` (`--max_wals`, по умолчанию, `5`). Размер каждого файла при создании 64000000 байт.

Файлы `0.tmp` и `1.tmp` создаются для резервирования места. В WAL-файл завершается запись, когда размер файла перешагивает за 64000000 байт, поэтому файлы могут превышать этот размер. Резервирование места файлами `0.tmp` и `1.tmp` не обеспечивает полной защиты от превышения дискового пространства.

В WAL записываются транзакции, принятые по протоколу Raft и снэпшоты ключей.

Если снэпшоты (перенос данных из WAL в файл данных) делаются редко (параметр `--snapshot-count`, по умолчанию `10000`), то число WAL файлов может превысить `ETCD_MAX_WALS`.

Есть утилита для просмотра содержимого WAL-файлов: https://github.com/etcd-io/etcd/tree/main/tools/etcd-dump-logs

Отказы etcd

- **Невозможна запись**. Возникает при превышения лимита на размер файла данных.
  - Выполнить очистку (compaction), дефрагментацию (defragmentation) и сбросить ошибку (disarm alert).
  - Дефрагментация и сброс ошибки выполняются вручную
- **Возможно только чтение**. Возникает при потере кворума
  - Создать снэпшот, подсоединившись, по возможности, к лидеру и создать новый кластер с использованием этого снэпшота
- **Невозможно чтение**. Возникает, если потерян кворум
  - Запустить один из узлов с параметром `--force-new-cluster`. К новому кластеру добавить узлы

Отказы etcd

1) **Невозможна запись**. Возникает при превышения лимита на размер файла данных. Кластер переходит в режим "обслуживания", в котором он принимает команды на чтение и удаление ключей, но не принимает команды добавления ключей и изменения значения ключей.

Нужно выполнить очистку (compaction), дефрагментацию (defragmentation) и сбросить ошибку (**`disarm alert`**). Дефрагментация и сброс ошибки выполняются вручную или вручную создать работу, выполняемую по расписанию.

2) **Возможно только чтение**. Возникает при потере кворума. То есть одновременно или последовательно перестают работать узлы так, что число работающих узлов не образует кворум. Нужно создать снэпшот, подсоединившись, по возможности, к лидеру и создать новый кластер с использованием этого снэпшота.

Чтобы минимизировать вероятность потери кворума нужно при исчезновении (неработоспособности) узла удалить его из кластера. Удаление узла из кластера уменьшает число узлов необходимых для кворума. В кластере не должно быть не работающих узлов.

3) **Невозможно чтение**. Возникает, если потерян кворум или узлы не работают. Восстановление возможно, но нет гарантий, что не были потеряны изменения ключей. Можно остановить все узлы и запустить один из узлов с параметром `--force-new-cluster`. Будет создан кластер из одного узла. Затем к новому кластеру можно добавить узлы, как новые, по процедуре добавления узла.

В процессе работы узлов возможны проблемы с тем, что узел перестаёт отвечать на длительное время. Причины:

1) нехватка памяти. При одинаково сконфигурированных хостах, в первую очередь, затрагивает лидера и начинаются перевыборы лидера. При нехватке памяти либо линукс, либо процесс etcd подвисают на время. Если был отключён оверкоммит, то процесс etcd будет убит OOM killer, что более благоприятно, так как уменьшает недоступность узла на время его подвисания. Процесс etcd будет перезапущен systemd.

2) Нагрузка на диск. Вызывается очисткой и дефрагментацией, если они выполняются редко.

3) Нагрузка на процессор. Вызывается большим потоком запросов клиентов и на чтение и на запись. При этом увеличивается использование памяти, выполняется сборка мусора, которая ещё больше нагружает процессор. Очистка и/или дефрагментация выполняются дольше обычного.

Очистка (compaction)

- etcd хранит историю изменений ключей (multiple revisions), чтобы можно было получать согласованные моментальные снимки (snapshots)
- Очистка удаляет историю до выбранного revision
- Рекомендуется включить автоматическую очистку:
  - `ETCD_AUTO_COMPACTION_RETENTION=1`
  - или параметром `--auto-compaction-retention=1`
- Нужно для дефрагментации
- Очистить вручную, оставив 100 версий:

```
REV=$(etcdctl --endpoints=http://127.0.0.1:2379 endpoint status --write-out=json
| egrep -o '"revision":[0-9]*' | egrep -o '[0-9].*') && echo $REV
668
etcdctl --endpoints=http://127.0.0.1:2379 compact $((REV-100))
compacted revision 568
```


Очистка (compaction)

Очистка (compaction) и дефрагментация (defragmentation) необходимы, они не дают разрастись файлу базы и уменьшить вероятность нехватки места на диске.

etcd хранит историю значений ключей - multiple revisions (многоверсионность), чтобы можно было получать согласованные моментальные снимки (snapshots) значений на один момент. По аналогии с вакуумированием PostgreSQL, старые версии надо вычищать. Очистка удаляет историю до задаваемой версии (revision). Освобожденное место в файле базы может использоваться для новых значений ключей. Очистка не уменьшает размер файла базы. Размер файла уменьшает дефрагментация.

Очистка ускоряет работу узла, так как уменьшается число версий ключей, снижаются накладные расходы при работе с ключами. **По умолчанию, очистка и дефрагментация отключены. Рекомендуется включить автоматическое очистка**. Для этого нужно установить переменную:

`ETCD_AUTO_COMPACTION_RETENTION=1` или параметр `--auto-compaction-retention=1`

Значение измеряется в часах, но единицу измерения можно задать явно: `1h`, `10m`, `15s`.

Если установить параметр `--auto-compaction-mode=revision`, то вместо времени (retention) параметр будет устанавливать число версий, которые надо удерживать, а предыдущие удалять, удаление будет происходить **каждые 5 минут**, а это долго.

Также, очищать можно вручную командой:

`etcdctl compact` *`число`*

где ***`число`*** - revision до которой версии будут очищены.

Текущая версия и очистка всех версий ключей, кроме `10` последних:

```
REV=$(etcdctl --endpoints=http://127.0.0.1:2379 endpoint status --write-
out=json | egrep -o '"revision":[0-9]*' | egrep -o '[0-9].*') && echo $REV
etcdctl --endpoints=http://127.0.0.1:2379 compact $((REV-10))
28
compacted revision 18
```

Увеличивает ревизию любое изменение данных, например, обновление ключа:

```
etcdctl --endpoints=http://127.0.0.1:2379 put key1 value1
```

Старые ревизии могут запрашиваться `etcdctl get`, а история изменений `etcdctl watch` с парметром `--rev=ревизия`. Если ревизия уполтнена, будет ошибка:

```
etcdserver: mvcc: required revision has been compacted
```

Дефрагментация

- Уменьшает размер файла данных узла
- Копирует содержимое файла в новый, а старый файл удаляется, поэтому при дефрагментации нужно дополнительное место в размере `--quota-backend-bytes`
- Дефрагментировать узлы по очереди
- Когда: если файл в 2-3 раза больше, чем объем удерживаемых данных или приближается к 2Гб

```
for ep in http://127.0.0.1:2379 http://127.0.0.2:2379 http://127.0.0.3:2379; do
 echo "Defragment $ep"
 etcdctl --endpoints=$ep defrag --command-timeout=90s
 sleep 30
done
Defragment http://127.0.0.1:2379
Finished defragmenting etcd member[http://127.0.0.1:2379]. took 24.139925ms
...
```


Дефрагментация

Максимальный интервал для очистки: от раза в сутки до раза в неделю.

После очистки в файле базы появляется свободное место, которое может использоваться ключами. Недостаток в том, что файл базы фрагментирован и, при разных размерах ключей, небольшие свободные фрагменты не смогут использоваться. Также очистка не уменьшает размер файла, то есть не освобождает место на диске. Поэтому, периодически, нужно выполнять дефрагментацию.

Дефрагментации освобождает пустое место, возвращая его файловой системе. Дефрагментация должна выполняется отдельно для каждого узла, по одному узлу за раз. Если дефрагментировать одновременно все узлы, то узлы начнут тормозить и увеличится вероятность пропусков heartbeat как на отправке, так и на приёме. Поэтому, дефрагментировать стоит последовательно каждый узел с паузами.

Не стоит использовать команду с опцией **`--cluster`**:

```
etcdctl --endpoints=http://127.0.0.1:2379 defrag --command-timeout=90s --cluster
```

Команда для дефрагментации, в которой нужно указать все узлы:

```
for ep in http://127.0.0.1:2379 http://127.0.0.2:2379 http://127.0.0.3:2379; do
   echo "Defragment $ep"
   etcdctl --endpoints=$ep defrag --command-timeout=90s
   sleep 30
done
Defragment http://127.0.0.1:2379
Finished defragmenting etcd member[http://127.0.0.1:2379]. took 14.594866ms
...
```

Интервалы между дефрагментациями. Без очистки дефрагментация бессмысленна. Планировать выполнение **`etcdctl defrag`** на время с наименьшей нагрузкой на кластер etcd. Ориентир для дефрагментации: если размер файла базы в 2-3 раза больше, чем объем удерживаемых данных или, если размер приближается к 2Гб. Обычно, это происходит не чаще, чем раз в несколько дней.

Дефрагментация копирует содержимое файла в новый, а старый файл удаляется, поэтому при дефрагментации нужно дополнительное место в размере `--quota-backend-bytes`.

https://etcd.io/docs/v3.7/op-guide/maintenance/

Дефрагментация по расписанию

- Файл `etcd-defrag.service` и `etcd-defrag.timer`:

```
[Unit]
Description=Run etcdctl defrag
After=network.target
[Service]
Type=oneshot
ExecStart=/opt/tantor/usr/bin/etcdctl defrag
[Install]
WantedBy=multi-user.target
```

```
[Unit]
Description=Run etcd-defrag
After=network.target
[Timer]
OnCalendar=*-*-* 01:00:0
[Install]
WantedBy=multi-user.target
```

- Разрешить таймер:

```
sudo systemctl daemon-reload
sudo systemctl enable --now etcd-
defrag.timer
sudo systemctl status etcd-defrag.timer
```

- Просмотр лога:

```
sudo journalctl -u etcd-defrag.service --since=-24h
```


Дефрагментация по расписанию

Создать в директории `/usr/lib/systemd/system` файлы:
Файл службы `etcd-defrag.service`:

```
[Unit]
Description=Run etcdctl defrag
After=network.target
[Service]
Type=oneshot
ExecStart=/opt/tantor/usr/bin/etcdctl defrag
[Install]
WantedBy=multi-user.target
```

Файл таймера `etcd-defrag.timer`:

```
[Unit]
Description=Run etcd-defrag.service every day
After=network.target
[Timer]
OnCalendar=*-*-* 01:00:0
[Install]
WantedBy=multi-user.target
```

Разрешить таймер:

```
systemctl daemon-reload
systemctl enable --now etcd-defrag.timer
systemctl status etcd-defrag.timer
```

Проверка того, что команда отрабатывает и просмотр лога:

```
systemctl start etcd-defrag.service
journalctl -u etcd-defrag.service --since=-24h
```

Резервирование кластера etcd

- Пока кворум не потерян, восстановление не нужно, нужно заменить потерянные узлы
- Восстановление кластера нужно при потере кворума
- Снэпшот - бэкап ключей и их значений
- По умолчанию, сохраняются последние версии ключей

```
etcdctl --endpoints=http://127.0.0.1:2379 snapshot save backup-$(date
+%F_%H%M%S).db
{"msg":"created temporary db file","path":"backup-2026-08-07_111658.db.part"}
{"msg":"opened snapshot stream; downloading"}
{"msg":"fetching snapshot","endpoint":"http://127.0.0.1:2379"}
{"msg":"completed snapshot read; closing"}
{"msg":"fetched snapshot","endpoint":"http://127.0.0.1:2379","size":"712
MB","took":"8.503098484s","etcd-version":"3.6.0"}
{"msg":"saved","path":"backup-2026-08-07_111658.db"}
Snapshot saved at backup-2026-08-07_111658.db
Server version 3.6.0
```


Резервирование кластера etcd

В кластере etcd хранятся данные: ключи и их значения. Для приложений (клиентов etcd) эти данные могут быть важны. Например, Kubernetes может хранить конфигурацию в ключах.

Patroni может пересоздавать ключи и нечувствителен к их полной потере, но он проверяет `cluster_id` и `raft_term` и не пересоздаст свои ключи, пока `raft_term` не станет равным или большим, чем тот, который видел Patroni. Для удаления увиденных `cluster_id` и `raft_term` придётся перезапустить Patroni, что запустит процедуру развёртывания (bootstrap).

Основная цель кластера etcd - обеспечивать строгую согласованность и надёжное хранение данных. То есть наличие нескольких узлов в кластере само по себе является резервированием и полной потери кластера не должно быть. Если возникает такая вероятность, то в кластер добавляются дополнительные узлы. Каждый узел получает от лидера всю базу - все ключи и их значения. Если узел или несколько узлов исчезают (повреждаются), то достаточно исключить их из кластера как можно быстрее (так как исчезнувшие влияют на число узлов в кворуме) и добавить новые узлы. Новые узлы при запуске получают от лидера всю базу данных (снэпшот).

Если узлы исчезли так, что оставшиеся узлы не образуют кворум, то возникает необходимость восстановления кластера. Как только кворум исчезает, кластер перестаёт принимать изменения и хранит только те изменения, по которым был достигнут консенсус, пока кворум существовал. Для восстановления работоспособности кластера нужно выгрузить базу (создать снэпшот), создать новый кластер (с новым cluster-id) на основе этого снэпшота. Можно было бы не пересоздавать кластер и добавить узлы, но такого не сделано намеренно.

Можно периодически делать снэпшоты на работающем в режиме чтения-записи кластере, но с таких снэпшотов (моментальный снимок) можно восстановиться только на момент создания снэпшота, то есть на момент времени в прошлом.

Снэпшот - бэкап (дамп) данных (ключей и их значений) из базы etcd. Сохраняется копия базы плюс изменения из WAL. **Чтобы снэпшот был меньшего размера, перед созданием снэпшота нужно выполнить не только compact, но и defrag.** Создание снэпшота:

```
etcdctl --endpoints=http://127.0.0.1:2379 snapshot save backup-$(date
+%F_%H%M%S).db
```

В примере команды файл снэпшота `backup-*` будет создан в текущей директории, но можно указать путь к файлу. https://etcd.io/docs/v3.7/op-guide/recovery/

Восстановление кластера etcd

- Восстановление на момент создания снэпшота
- Нужно восстановить тот же снэпшот на **всех** узлах
- Cluster ID зависит от `initial-cluster` и `initial-cluster-token`, он **может** быть прежним или стать новым
- Raft term **не** восстанавливается и принимает значение 2

```
etcdutl -w table snapshot status backup-*
+----------+----------+------------+------------+---------+
|   HASH   | REVISION | TOTAL KEYS | TOTAL SIZE | VERSION |
+----------+----------+------------+------------+---------+
| c7e44c6a |     2001 |          1 |     712 MB |   3.6.0 |
+----------+----------+------------+------------+---------+
etcdutl snapshot restore backup-* --data-dir /opt/tantor/var/lib/etcd/a1.etcd --
name a1 --initial-advertise-peer-urls http://127.0.0.1:2380 --initial-cluster
a1=http://127.0.0.1:2380,a2=http://127.0.0.2:2380,a3=http://127.0.0.3:2380 --
initial-cluster-token etcd-cluster
systemctl start etcd-tantor
etcdctl --endpoints=http://:2379 endpoint status -w table
```


Восстановление кластера etcd

Снэпшот позволяет восстановить данные кластера etcd только на момент сохранения снимка.

Проверить целостность файла снэпшота можно командой:

```
etcdutl -w table snapshot status backup-*
+----------+----------+------------+------------+---------+
|   HASH   | REVISION | TOTAL KEYS | TOTAL SIZE | VERSION |
+----------+----------+------------+------------+---------+
| c7e44c6a |     2001 |          1 |     712 MB |   3.6.0 |
+----------+----------+------------+------------+---------+
```

Можно скопировать файл базы `/opt/tantor/var/lib/etcd/a1.etcd/member/snap/db` с узла, но последние данные, хоть и находятся в WAL, могут быть не применены к файлу базы `member/snap/db`, а утилита `etcdctl` формирует свежий моментальный снимок. Поэтому, не стоит копировать файл вручную.

Начиная **с версии 3.6**, вместо команд `etcdctl snapshot status` и `etcdctl snapshot restore`, используются команды `etcdutl snapshot status` и `etcdutl snapshot restore`.

Для восстановления **на всех узлах** выполнить набор команд:

```
systemctl stop etcd-tantor
rm -rf /opt/tantor/var/lib/etcd/a1.etcd
etcdutl snapshot restore backup-* --data-dir /opt/tantor/var/lib/etcd/a1.etcd
--name a1 --initial-advertise-peer-urls https://127.0.0.1:2380 --initial-
cluster-token etcd-cluster --initial-cluster
a1=http://127.0.0.1:2380,a2=http://127.0.0.2:2380,a3=http://127.0.0.3:2380
systemctl start etcd-tantor
```

Если на каком-то узе не восстановить базу, она будет пустой, при этом узел будет в кластере. Поэтому, убедиться в работоспособности кластера: проверить статус **каждого** узла кластера и наличие лидера:

```
etcdctl --endpoints=http://127.0.0.1:2379 endpoint status -w table
```

Cluster ID генерируется на основе `initial-cluster` и `initial-cluster-token`.

У команды `etcdutl snapshot restore` есть опции `--mark-compacted` и `--bump-revision число`, последняя позволяет при восстановлении увеличить ревизию на указанное в опции значение. Это позволяет убедиться, что номер ревизии не уменьшится даже, если восстановиться с неактуального снэпшота.