PICODATA

Распределенный сервер приложений со встроенной распределенной базой данных

Руководство по эксплуатации

Оглавление

О данном руководстве

Документ «Руководство по эксплуатации» содержит сведения, которые должны помочь пользователям и системным администраторам запускать программное обеспечение Picodata и использовать его в своей работе.

Информация об инсталляции программного обеспечения приведена в отдельном документе «Руководство по установке».

Информация о внутреннем устройстве распределенной системы (кластера) приведена в отдельном документе «Руководство администратора».

В текущем документе содержится описание параметров запуска и последовательности действий, необходимой для развертывания и поддержания работоспособности распределенного кластера СУБД.

Сведения в данном документе относятся к текущей публично доступной версии ПО Picodata 22.07.0, вышедшей в июле 2022 г. Информация в этом руководстве будет обновляться для наиболее полного соответствия фактической функциональности ПО Picodata на момент публикации.

Дата публикации: 13.10.2022

Общее описание продукта

Данный раздел содержит общие сведения о продукте Picodata, его назначении, области применения и внутреннем устройстве.

Что такое Picodata?

Программное обеспечение Picodata — распределенная система промышленного уровня для управления базами данных, а также среда выполнения/сервер приложений. Программное обеспечение Picodata реализует хранение структурированных и неструктурированных данных, транзакционное управление данными, языки запросов SQL и GraphQL, а также среду выполнения приложений (хранимых процедур) на языках Rust и Lua.

Назначение

Основным назначением продукта Picodata является горизонтально масштабируемое хранение структурированных и неструктурированных данных, управление ими, предоставление среды вычислений внутри кластера, состоящего из реплицированных отдельных узлов. Такие узлы называют экземплярами Picodata или *инстансами*.

Задачи

Программное обеспечение Picodata решает следующие задачи:

- реализация общего линеаризуемого хранилища конфигурации, схемы данных и топологии кластера, встроенного в распределенную систему управления базами данных;
- предоставление интерфейса командной строки для управления топологией кластера;
- реализация runtime-библиотек по работе с сетью, файловому вводу-выводу, реализация кооперативной многозадачности и управления потоками, работа со встроенной СУБД средствами языка Rust;
- поддержка языка SQL для работы как с данными отдельного инстанса, так и с данными всего кластера;
- управление кластером;
- поддержка жизненного цикла приложения в кластере, включая версионирование, управление зависимостями, упаковку дистрибутива, развертывание и обновление запущенных приложений.

Область применения

Кластер Picodata обеспечивает быстрый доступ к данным внутри распределенного хранилища. Это позволяет использовать его в следующих областях:

- управление телекоммуникационным оборудованием;
- банковские и в целом финансовые услуги, биржевые торги, аукционы;
- формирование персональных маркетинговых предложений с привязкой ко времени и месту;
- обработка больших объемов данных в реальном времени для систем класса "интернет вещей" (IoT);
- игровые рейтинговые таблицы;
- и многое другое!

Особенности кластера Picodata

Кластер с СУБД Picodata обладает следующими свойствами:

- автоматическое горизонтальное масштабирование кластера;
- более простая настройка для запуска шардированного кластера. Требуется меньше файлов конфигурации;
- совместимость с любыми инструментами развертывания инстансов (Ansible, Chef, Puppet и др.);
- обеспечение высокой доступности данных без необходимости в кластере Etcd и дополнительных настройках;
- автоматическое определение активного инстанса в репликасетах любого размера;
- единая схема данных во всех репликасетах кластера;
- возможность обновлять схему данных и менять топологию работающего кластера, например добавлять новые инстансы. Picodata автоматически управляет версиями схемы;
- встроенные инструменты для создания и запуска приложений.

Принцип работы

Каждый экземпляр ПО Picodata, который также называют инстансом или узлом кластера, включает в себя два архитектурных слоя: один из них отвечает за хранение пользовательских данных, второй — за управление конфигурацией кластера и обеспечение его отказоустойчивости.

Узлы кластера Picodata представляют собой отдельные процессы (процессы в понимании операционных систем) на одной или нескольких независимых ЭВМ, соединенных между собой сетью передачи данных и представляющих для пользователя единое, линеаризуемое хранилище данных и среду выполнения приложений.

Каждый узел кластера представляет собой отдельную независимую СУБД и среду выполнения приложений со своим собственным хранилищем данных и журналом выполнения транзакций, отвечающий за фрагмент данных (партицию, шард) всего кластера.

Обработка локальных транзакций внутри одного процесса выполняется последовательно с полным (эксклюзивным) доступом транзакции к данным процесса, что позволяет исключить издержки на разделение такого доступа и откат транзакций в случае конфликта между ними по обращению к данным.

Слой управления конфигурацией — это реализация алгоритма Raft, который присутствует в каждом экземпляре ПО и является внутренним компонентом по отношению к кластеру. Raft хранит информацию об общей схеме данных, о топологии (составе участников) кластера, о роли каждого из узлов, расположении фрагментов (партиций, шардов) кластера. Также, Raft предоставляет пользователю интерфейсы (API) мониторинга и управления отдельными узлами кластера.

В кластере Raft каждый из узлов в любой момент времени находится в одном из трёх состояний:

- Leader (лидер) – обрабатывает все клиентские запросы, является эталоном всех данных в журнале, поддерживает журнал фоловеров.
- Follower (фоловер) – пассивный узел, который только принимает новые записи в журнал от лидера и перенаправляет все входящие запросы от клиентов на лидера. По сути, является репликой лидера в режиме ожидания.
- Candidate (кандидат) – специальное состояние узла, возможное только во время выбора нового лидера.

Во время нормальной работы в кластере только один сервер является лидером, все остальные – его фоловеры.

Более подробная информация о работе узлов Picodata приведена в разделе Архитектура кластера.

Гарантия сохранности данных

Отказоустойчивость системы обеспечивается за счет резервирования каждого из компонентов, при этом копии (реплики) процессов должны располагаться на разных физических компьютерах, а согласование состояния данных копий осуществляться с использованием технологии синхронной репликации.

В условиях потери сетевой связности между компонентами системы Picodata ставит доступность конфигурации превыше консистентности, в терминах теоремы CAP — режим AP. Пользователь имеет возможность управлять кластером пока выполняется условие доступности большинства узлов (2 реплики из 3, или 3 из 5). В части хранения пользовательских данных консистентность данных, напротив, имеет приоритет над доступностью.

Архитектура кластера

Составные части кластера

Архитектура кластера Picodata предполагает систему отдельных *инстансов* — программных узлов, входящих в состав кластера. Каждый такой узел может выполнять различные роли, например роль хранения данных, роль сервера приложения, или служебную роль координатора кластера. Все инстансы работают с единой схемой данных и кодом приложения. Каждый процесс базы данных выполняется на одном процессорном ядре и хранит используемый набор данных в оперативной памяти. Любой отдельный инстанс является частью набора реплик, который также называют *репликасетом*. Репликасет может состоять из одного или нескольких инстансов — дубликатов одного и того же набора данных. Внутри репликасета всегда есть *активный* инстанс и — если реплик больше 1 — то некоторое число *резервных* инстансов, обеспечивающих отказоустойчивость системы в случае выхода из строя или недоступности активного инстанса. Число реплик определяется *фактором репликации*, заданным в глобальных настройках Picodata.

На рисунке ниже показана схема простого кластера из двух репликасетов, каждый из которых состоит из двух инстансов (активного и резервного):

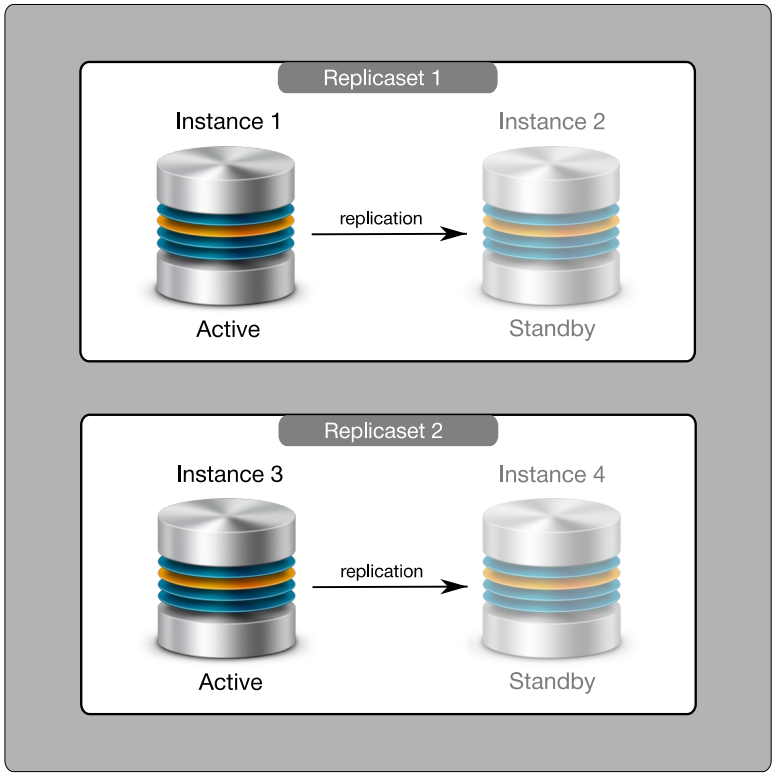

Рисунок 1: Устройство репликасета

Репликасеты являются единицами горизонтального масштабирования кластера. Данные балансируются между ними автоматически.

Хранение данных

Внутри каждого репликасета есть *бакет* (bucket*)* — виртуализированная неделимая единица хранения, обеспечивающая локальность данных (например, хранение нескольких связанных с клиентом записей на одном физическом узле сети). Сам по себе бакет не имеет ограничений по емкости и может содержать любой объем данных. Горизонтальное масштабирование позволяет распределить бакеты по разным шардам, оптимизировав производительность кластера путем добавления новых реплицированных инстансов. Чем больше репликасетов входит в состав кластера, тем меньше нагрузка на каждый из них. Бакет хранится физически на одном репликасете и является промежуточным звеном между данными и устройством хранения. В каждом репликасете может быть много бакетов (или не быть ни одного). Внутри бакета данные задублированы по всем инстансам в рамках репликасета в соответствии с фактором репликации. Общее количество бакетов в кластере задается при первоначальной настройке кластера.

На схеме ниже показан пример схемы хранения данных внутри репликасета:

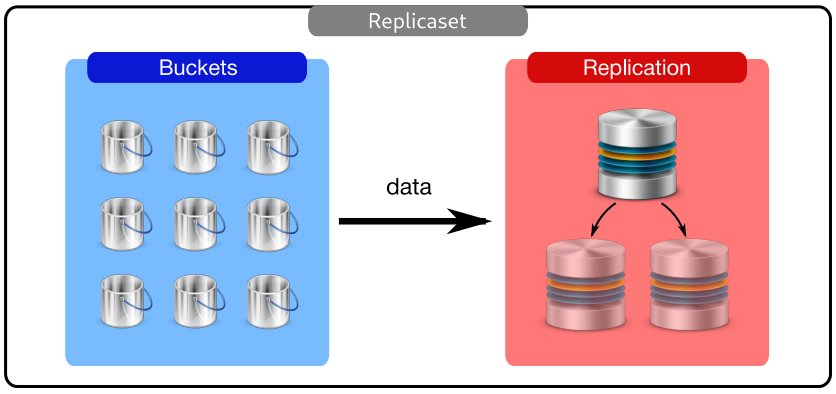

Рисунок 2: Принцип хранения данных в репликасете

Отказоустойчивость

Наличие нескольких реплик внутри репликасета обеспечивают его отказоустойчивость. Дополнительно для повышения надежности каждый инстанс кластера внутри репликасета находится на разных физических серверах, а в некоторых случаях — в удаленных друг от друга датацентрах. Таким образом, в случае недоступности датацентра в репликасете происходит переключение на резервную реплику/инстанс без прерывания работы.

Пример географического распределения репликасета показан на схеме ниже:

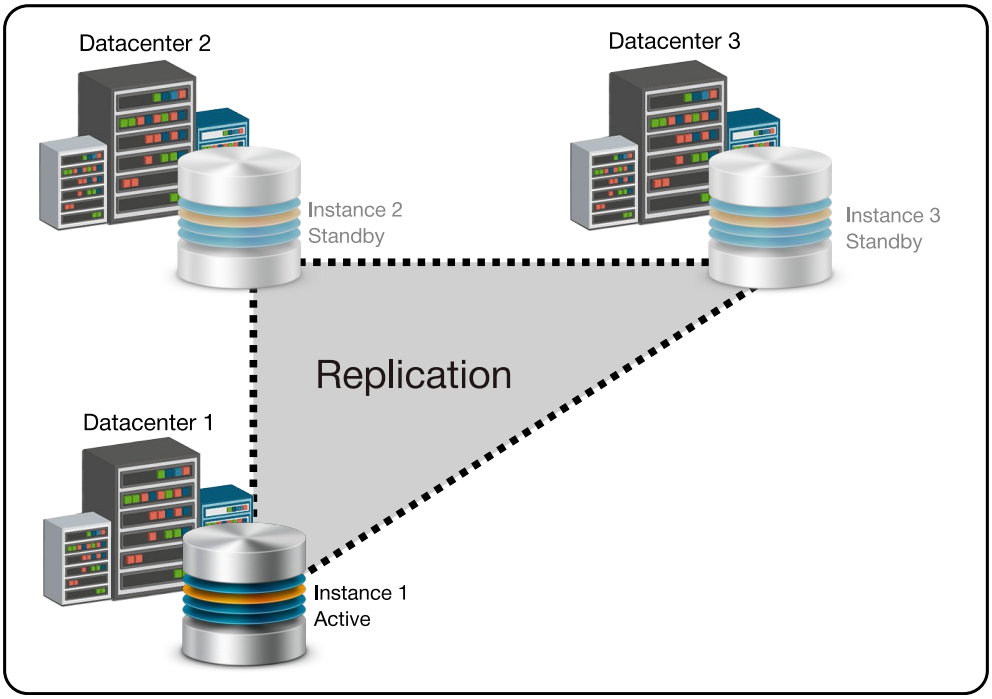

Рисунок 3: Распределение удаленных реплик

Шардирование

Шардирование — это распределение бакетов между различными репликасетами. В Picodata используется основанное на хэшах шардирование с хранением данных в виртуальных бакетах. Каждый репликасет является *шардом*, и чем больше репликасетов имеется в кластере, тем эффективнее данная функция может разделить массив данных на отдельные наборы данных меньшего размера. При добавлении новых инстансов в кластер и/или формировании новых репликасетов Picodata автоматически равномерно распределит бакеты с учетом новой конфигурации. Пример автоматического шардирования при добавлении в кластер новых инстансов показан на схеме ниже:

Таким образом, каждый инстанс (экземпляр Picodata) является *частью репликасета*, а каждый репликасет — *шардом*, а шарды распределены между несколькими серверами.

Описание параметров запуска

Picodata является консольным приложением, которое поддерживает различные параметры запуска в виде аргументов командной строки.

Полный список аргументов доступен с помощью следующей команды:

```
$ picodata run --help
```

Ниже приводится описание этих аргументов.

`--advertise <[host][:port]>`
Адрес, по которому другие инстансы смогут подключиться к данному инстансу. По умолчанию используется значение из аргумента `--listen`. Аналогичная переменная окружения: `PICODATA_ADVERTISE`.

`--cluster-id <name>`
Имя кластера. Инстанс не сможет стать частью кластера, если у него указано другое имя. Аналогичная переменная окружения: `PICODATA_CLUSTER_ID`.

`--data-dir <path>`
Директория, в которой инстанс будет сохранять свои данные для постоянного хранения. Аналогичная переменная окружения: `PICODATA_DATA_DIR`.

`-e, --tarantool-exec <expr>`
Данный аргумент позволяет выполнить Lua-скрипт на Tarantool

`--failure-domain <key=value>`
Список параметров географического расположения сервера (через запятую). Также этот аргумент называется *зоной доступности*. Каждый параметр должен быть в формате КЛЮЧ=ЗНАЧЕНИЕ. Также, следует помнить о том, что добавляемый инстанс должен обладать тем же набором доменов (т.е. ключей данного аргумента), которые уже есть в кластере. Picodata будет избегать помещения двух инстансов в один репликасет если хотя бы один параметр зоны доступности у них совпадает. Соответственно, инстансы будут формировать новые репликасеты. Аналогичная переменная окружения: `PICODATA_FAILURE_DOMAIN`.

`-h, --help`
Вывод справочной информации

`--init-replication-factor <INIT_REPLICATION_FACTOR>`
Число реплик (инстансов с одинаковым набором хранимых данных) для каждого репликасета. Аргумент используется только при начальном создании кластера и в дальнейшем игнорируется. Аналогичная переменная окружения: `PICODATA_INIT_REPLICATION_FACTOR`.

`--instance-id <name>`
Название инстанса. Если этот аргумент не указать, то название будет сгенерировано автоматически. Данный аргумент удобно использовать для явного указания инстанса при его

перезапуске (например, в случае его недоступности или при переносе в другую сеть). Аналогичная переменная окружения: `PICODATA_INSTANCE_ID`.

`-l, --listen <[host][:port]>`
Адрес и порт привязки инстанса. По умолчанию используется localhost:3301 Аналогичная переменная окружения: `PICODATA_LISTEN`.

`--log-level <LOG_LEVEL>`
Уровень регистрации событий. Возможные значения: fatal, system, error, crit, warn, info, verbose, debug. По умолчанию используется уровень info. Аналогичная переменная окружения: `PICODATA_LOG_LEVEL`.

`--peer <[host][:port]>`
Адрес другого инстанса. В данному аргументе можно передавать несколько значение через запятую. По умолчанию используеься значение localhost:3301, т.е. без связывания с каким-либо другим инстансом. Указание порта опционально. Аналогичная переменная окружения: `PICODATA_PEER`.

`--replicaset-id <name>`
Название целевого репликасета. Аналогичная переменная окружения: `PICODATA_REPLICASET_ID`

Развертывание кластера

В данном разделе рассматриваются различные сценарии работы с кластером. Все они основаны на одном и том же принципе: запуске и объединении отдельных экземпляров Picodata в распределенный кластер. При этом сложность развертывания и поддержания работоспособности кластера зависит только от сложности его топологии.

Минимальный вариант кластера

Picodata может создать кластер, состоящий всего из одного экземпляра/инстанса. Обязательных параметров у него нет, что позволяет свести запуск к выполнению всего одной простой команды:

```
picodata run
```

Можно добавлять сколько угодно последующих инстансов — все они будут подключаться к этому кластеру. Каждому инстансу следует задать отдельную рабочую директорию (параметр `--data-dir`), а также указать адрес и порт для приема соединений (параметр `--listen`) в формате `<HOST>:<PORT>`. Фактор репликации по умолчанию равен 1 — каждый инстанс образует отдельный репликасет. Если для `--listen` указать только порт, то будет использован IP-адрес по умолчанию (127.0.0.1):

```
picodata run --data-dir i1 --listen :3301
picodata run --data-dir i2 --listen :3302
picodata run --data-dir i3 --listen :3303
```

Кластер на нескольких серверах

Выше был показан запуск Picodata на одном сервере, что удобно для тестирования и отладки, но не отражает сценариев полноценного использования кластера. Поэтому ниже будет показан запуск Picodata на нескольких серверах. Предположим, что их два: `192.168.0.1` и `192.168.0.2`. Порядок действий будет следующим:

На `192.168.0.1`:

```
picodata run --listen 192.168.0.1:3301
```

На `192.168.0.2`:

```
picodata run --listen 192.168.0.2:3301 --peer 192.168.0.1:3301
```

На что нужно обратить внимание:

Во-первых, для параметра `--listen` вместо стандартного значения `127.0.0.1` надо указать конкретный адрес. Формат адреса допускает упрощения — можно указать только хост `192.168.0.1` (порт по умолчанию `:3301`), или только порт, но для наглядности лучше использовать полный формат `<HOST>:<PORT>`.

Значение параметра `--listen` не хранится в кластерной конфигурации и может меняться при перезапуске инстанса.

Во-вторых, надо дать инстансам возможность обнаружить друг друга для того чтобы механизм discovery правильно собрал все найденные экземпляры Picodata в один кластер. Для этого в параметре `--peer` нужно указать адрес какого-либо соседнего инстанса. По умолчанию значение параметра `--peer` установлено в `127.0.0.1:3301`. Параметр `--peer` не влияет больше ни на что, кроме механизма обнаружения других инстансов.

Параметр `--advertise` используется для установки публичного IP-адреса и порта инстанса. Параметр сообщает, по какому адресу остальные инстансы должны обращаться к текущему. По умолчанию он равен `--listen`, поэтому в примере выше не упоминается. Но, например, в случае `--listen 0.0.0.0` его придется указать явно:

```
picodata run --listen 0.0.0.0:3301 --advertise 192.168.0.1:3301
```

Значение параметра `--advertise` анонсируется кластеру при запуске инстанса. Его можно поменять при перезапуске инстанса или в процессе его работы командой `picodata set-advertise`.

Именование инстансов

Чтобы проще было отличать инстансы друг от друга, им можно давать имена:

```
picodata run --instance-id barsik
```

Если имя не дать, то оно будет сгенерировано автоматически в момент добавления в кластер. Имя инстанса задается один раз и не может быть изменено в дальнейшем (например, оно постоянно сохраняется в снапшотах инстанса). В кластере нельзя иметь два инстанса с одинаковым именем — пока инстанс живой, другой инстанс сразу после запуска получит ошибку при добавлении в кластер. Тем не менее, имя можно повторно использовать если предварительно исключить первый инстанс с таким именем из кластера.

Проверка работы кластера

Каждый инстанс Picodata — это отдельный процесс в ОС. Для его диагностики удобно воспользоваться встроенной консолью, которая автоматически открывается после запуска инстанса (`picodata run ...`). Для диагностики всей Raft-группы (например, для оценки количества инстансов в кластере) выполните следующую команду:

```
box.space.raft_group:fselect()
```

Дополнительно, можно добиться ответа инстансов с помощью такой команды:

```
picolib.raft_propose_info("Hello, Picodata!")
```

В журнале каждого инстанса (по умолчанию выводится в `stderr`) появится фраза "Hello, Picodata!"

Репликация и зоны доступности (failure domains)

Количество инстансов в репликасете определяется значением переменной `replication_factor`. Внутри кластера используется один и тот же `replication_factor`.

Управление количеством происходит через параметр `--init-replication-factor`, который используется только в момент запуска первого инстанса. При этом, значение из аргументов командной строки записывается в конфигурацию кластера. В дальнейшем значение параметра `--init-replication-factor` игнорируется.

По мере усложнения топологии возникает еще один вопрос — как не допустить объединения в репликасет инстансов из одного и того же датацентра. Для этого в Picodata имеется параметр `--failure-domain` — *зона доступности*, отражающая признак физического размещения сервера, на котором выполняется инстанс Picodata. Это может быть как датацентр, так и какое-либо другое обозначение расположения: регион (например, `eu-east`), стойка, сервер, или собственное обозначение (blue, green, yellow). Ниже показан пример запуска инстанса Picodata с указанием зоны доступности:

```
picodata run --init-replication-factor 2 --failure-domain region=us,zone=us-west-1
```

Добавление инстанса в репликасет происходит по следующим правилам:

- Если в каком-либо репликасете количество инстансов меньше необходимого фактора репликации, то новый инстанс добавляется в него при условии, что их параметры `--failure-domain` отличаются (регистр символов не учитывается).
- Если подходящих репликасетов нет, то Picodata создает новый репликасет.

Параметр `--failure-domain` играет роль только в момент добавления инстанса в кластер. Принадлежность инстанса репликасету впоследствии не меняется.

Как и параметр `--advertise`, значение параметра`--failure-domain` каждого инстанса можно редактировать, перезапустив инстанс с новыми параметрами.

Добавляемый инстанс должен обладать тем же набором параметров, которые уже есть в кластере. Например, инстанс `dc=msk` не сможет присоединиться к кластеру с `--failure-domain region=eu/us` и вернет ошибку.

Как было указано выше, сравнение зон доступности производится без учета регистра символов, поэтому, к примеру, два инстанса с аргументами `--failure-domain region=us` и `--failure-domain REGION=US` будут относиться к одному региону и, следовательно, не попадут в один репликасет.

Динамическое переключение голосующих узлов в Raft (Raft voter failover)

Все узлы Raft в кластере делятся на два типа: голосующие (`voter`) и неголосующие (`learner`). За консистентность Raft-группы отвечают только узлы первого типа. Для коммита каждой транзакции требуется собрать кворум из `N/2 + 1` голосующих узлов. Неголосующие узлы в кворуме не участвуют.

Чтобы сохранить баланс между надежностью кластера и удобством его эксплуатации, в Picodata предусмотрена удобная функция — динамическое переключение типа узлов. Если один из голосующих узлов становится недоступным или прекращает работу (что может нарушить кворум в Raft), то тип `voter` автоматически присваивается одному из доступных неголосующих узлов. Переключение происходит незаметно для пользователя.

Количество голосующих узлов в кластере не настраивается и зависит только от общего количества инстансов. Если инстансов 1 или 2, то голосующий узел один. Если инстансов 3 или 4, то таких узлов три. Для кластеров с 5 или более инстансами — пять голосующих узлов.

Удаление инстансов из кластера (expel)

Удаление — это принятие кластером решения, что некий инстанс больше не является участником кластера. После удаления кластер больше не будет ожидать присутствия инстанса в кворуме, а сам инстанс завершится. При удалении текущего лидера будет принудительно запущен выбор нового лидера.

Удаление инстанса с помощью консольной команды

```
picodata expel --instance-id <instance-id> [--cluster-id <cluster-id>] [--peer <peer>]
```

где `cluster-id` и `instance-id` — данные об удаляемом инстансе, `peer` — любой инстанс кластера.

Пример:

```
picodata expel --instance-id i3 --peer 192.168.100.123
```

В этом случае на адрес `192.168.100.123:3301` будет отправлена команда `expel` с `instance-id = "i3"` и стандартным значением `cluster-id`. Инстанс на `192.168.100.123:3301` найдет лидера и отправит ему команду `expel`. Лидер отметит, что указанный инстанс удален; остальные инстансы получат эту информацию через Raft. Если удаляемый инстанс запущен, он завершится, если не запущен — примет информацию о своем удалении при запуске и затем завершится. При последующих запусках удаленный инстанс будет сразу завершаться.

Удаление инстанса из консоли Picodata с помощью Lua API

В консоли запущенного инстанса введите следующее:

```
picolib.expel(<instance-id>)
```

например:

```
picolib.expel("i3")
```

Будет удален инстанс `i3`. Сам инстанс `i3` будет завершен. Если вы находитесь в консоли удаляемого инстанса — процесс завершится, консоль будет закрыта.