Tarantool Enterprise manual

Выпуск 1.10.4-1

Mail.Ru, Tarantool team

дек. 24, 2019

Оглавление

Данное руководство посвящено Enterprise-версии продукта Tarantool, который сочетает в себе сервер приложений Lua и отказоустойчивую распределенную СУБД.

Enterprise-версия предлагает дополнительные возможности по разработке и эксплуатации кластерных приложений:

- Статическая сборка для автономных Linux-систем.
- Модуль tracing для отладки производительности (на базе OpenTracing, Zipkin, Jaeger).
- Модуль task для работы с фоновыми задачами.
- Графический проводник для просмотра данных.
- Модуль интеграции с OpenLDAP.
- Журнал аудита безопасности.
- Подключения к корпоративным базам данных: Oracle и любым СУБД с интерфейсом ODBC (MySQL, Microsoft SQL Server и т.д.). (MySQL, Microsoft SQL Server, etc.).

Данное руководство описывает особенности и возможности именно Enterprise-версии Tarantool. Полная документация по Tarantool с открытым кодом доступна здесь.

ГЛАВА 1

Журнал изменений

1.1 1.10.4-1 - 2019-10-23

Главные новости:

- Фреймворк для разработки кластерных приложений (Tarantool Cartridge) стал частью Tarantool с открытым кодом. Теперь он представлен двумя модулями: cartridge (вместо модуля cluster) и cartridge-cli (вместо утилиты tarantoolapp).
- Новый открытый модуль luacheck – статический анализатор кода, предконфигурированный под Тарантул.
- Выложена документация по всем закрытым модулям.
- Обновленный дизайн веб-интерфейса.

Изменено:

- Базовая версия Tarantool Community обновлена до 1.10.4-21-g9349237.
- Обновлен модуль vshard-zookeeper.

Добавлено:

- Модуль quickfix v1.0.0.
- Модуль luarapidxml v2.0.0.
- Модуль cartridge v1.0.0 (вместо модуля cluster).
- Модуль cartridge-cli v1.0.0-1 (вместо утилиты tarantoolapp).
- Модуль task v0.4.0.
- Модуль membership v2.1.2, улучшения стабильности на больших кластерах.
- Модуль membership v2.1.4.
- Модуль odbc v0.3.0.
- Модуль cluster v0.10.0.

- Модуль task v0.3.0.
- Модуль odbc v0.4.0.
- Модуль kafka v1.0.2.
- Модуль frontend-core v6.0.1.
- Модуль space-explorer v1.1.0.
- Модуль oracle v1.2.2.
- Модуль http v1.1.0.
- Модуль avro-schema v3.0.3.
- Модуль vshard v0.1.12.
- Модуль icu-date v1.3.1.
- Модуль luatest v0.2.2.
- Модуль metrics v0.1.6.
- Модуль queue v1.0.4.

Удалено:

- Утилита tarantoolapp.

1.2 1.10.3-71 - 2019-07-12

Изменено:

- Базовая версия Tarantool Community обновлена до 1.10.3-89-g412e943.

Добавлено:

- Шаблоны юнит-тестов и интеграционных тестов на кластер Tarantool'a, доступные сразу через tarantoolctl rocks test.
- При сборке проекта можно задавать исключения с помощью файла .tarantoolapp.ignore.
- Модуль kafka v1.0.1 (коннектор Apache Kafka для Tarantool).
- Модель tracing v0.1.0 (для отладки производительности приложения; реализован на базе OpenTracing/Zipkin).
- Модуль task v0.2.0 (для работы с фоновыми задачами в кластерных приложениях).
- Модуль odbc v0.1.0 (исходящий ODBC-коннектор для Tarantool).
- Модуль luatest v0.2.0 (тестовый фреймворк для Tarantool).
- Модуль checks v3.0.1.
- Модуль errors v2.1.1.
- Модуль frontend-core v5.0.2.
- Модуль metrics v0.1.5.
- Модуль oracle v1.1.6.
- Модуль cluster v0.9.2 с новыми возможностями:

  - Users, новая вкладка веб-интерфейса для управления пользователями-администраторами кластера.
  - Для наборов реплик с серверами хранилищ можно задавать группы (например, группы hot и cold для независимой обработки горячих и холодных данных).
  - Хелперы для интеграционных тестов на luatest.

Удалено:

- Модуль cluster v0.2.0, v0.3.0.
- Модуль oracle v1.0.0, v1.1.0.
- Модуль vshard v0.1.5, v0.1.6.

1.3 1.10.3-29 - 2019-05-28

Изменено:

- Базовая версия Tarantool Community обновлена до 1.10.3-57-gd2efb0d.

Добавлено:

- Команда tarantoolapp pack теперь добавляет файл VERSION в запакованный проект.
- Возможность задать уровень логирования для каждого маршрута.
- Модуль http v0.1.6 с новой возможностью: задание уровня логирования для каждого маршрута.
- Модуль space-explorer v1.0.2.
- Модуль cron-parser v1.0.0.
- Модуль argon2 v3.0.1 с алгоритмами хеширования класса PCI DSS.
- Модуль metrics v0.1.3.
- Модуль vshard v0.1.9
- Модуль oracle v1.1.5.
- Модуль frontend-core v5.0.0, v5.0.1
- Модуль cluster v0.8.0 с исправленными дефектами и новыми возможностями:
  - зависимые роли;
  - кластерные cookies;
  - текстовые метки для серверов.

1.4 1.10.3-5 - 2019-04-05

- Базовая версия Tarantool Community обновлена до 1.10.3-6-gfbf53b9.
- Добавлены следующие функции:
  - Настройка приоритета восстановления после отказа через веб-интерфейс.
  - Модуль RPC (удаленные вызовы между экземплярами кластера).
  - Проводник в веб-интерфейсе.

  - Клиент OpenLDAP для Tarantool.
- Перезапуск экземпляра теперь запускает проверку конфигурации до инициализации ролей.
- Обновлен дизайн веб-интерфейса.

1.5 1.10.2-41 - 2019-02-08

- Базовая версия Tarantool Community обновлена до 1.10.2-131-g6f96bfe.
- Обновлен шаблон кластера, чтобы приложение могло работать на нескольких хостах (виртуальных машинах).
- Добавлены последние версии следующие модулей с закрытым исходным кодом: queue, front, errors, membership, cluster и oracle. Эти сторонние библиотеки включают в себя:
  - новое ядро фронтенда с косметическими изменениями;
  - индикатор активного мастера во время восстановления после отказа;
  - возможность отключить роль vshard-storage по окончании процесса балансировки;
  - обновление зависимостей и незначительные улучшения.

1.6 1.10.2-15 - 2018-12-13

- Базовая версия Tarantool Community обновлена до 1.10.2-84-g19d471bd4.
- Обновлен комплект модуля checks.
- В модуль cluster добавлены специализированные (пользовательские) API для кластерных ролей.
- Добавлена поддержка параметра веса для набора реплик в vshard.

1.7 1.10.2-4 — 2018-10-31

- Добавлены примеры приложений для демонстрации запуска Tarantool в Docker и использовании кэша со сквозной записью для PostgreSQL.
- В веб-интерфейс добавлена возможность переключения мастера в наборе реплик вручную, а также автоматического восстановления после отказа.
- Добавлена утилита tarantoolapp, которая помогает настроить среду разработки и упаковать приложение независимо от среды разработки.

1.8 1.10.1-29 — 2018-10-04

- Пример приложения теперь основан на кластере, а управление кластером осуществляется через веб-интерфейс.
- Релиз Tarantool Enterprise представляет собой архив, который включает в себя автономный репозиторий сторонних библиотек, из которого можно установить все необходимые модули.
- В репозиторий сторонних библиотек добавлен коннектор для Oracle. Теперь Lua-приложения могут получать доступ к базам данных Oracle.

1.9 1.10.1 — 2018-04-09

- Теперь у Tarantool статическая сборка: все зависимости скомпонованы в статический двоичный файл. Это упрощает развертывание Tarantool в среде Linux.
- Базовая версия для сообществ Tarantool Community обновлена до 1.10.1. Чтобы получить информацию обо всех новых возможностях и исправленных ошибках в версии для сообщества, см. https://github.com/tarantool/tarantool/releases.
- Модули, необходимые для интеграции с ZooKeeper и orchestrator, объявлены устаревшими и больше не поддерживаются.

ГЛАВА 2

Настройка

В этой главе объясняется, как загрузить и настроить Tarantool Enterprise, а также запустить пример приложения.

2.1 Системные требования

Ниже представлены рекомендуемые системные требования для запуска Tarantool Enterprise.

2.1.1 Требования к аппаратному обеспечению

Чтобы обеспечить полную отказоустойчивость системы распределенного хранения данных, необходимы как минимум три физических компьютера или виртуальных сервера.

Для целей тестирования или разработки систему можно развернуть, используя меньшее количество серверов. Тем не менее, не рекомендуется использовать такие конфигурации в производственной среде.

2.1.2 Требования к программному обеспечению

1. Tarantool Enterprise поддерживает операционные системы Red Hat Enterprise Linux и CentOS версии 7.5 и выше.

   Примечание: Tarantool Enterprise может работать на других дистрибутивах Linux на основе systemd, но тестирование на них не проводится, поэтому корректная работа не гарантирована.

   ---

2. Требуется glibc версии 2.17-260.el7_6.6 и выше. Необходимо проверить текущую версию и обновить в случае необходимости:

```
$ rpm -q glibc
glibc-2.17-196.el7_4.2
$ yum update glibc
```

2.1.3 Сетевые требования

Здесь и далее по тексту под серверами хранения данных или серверами Tarantool понимаются компьютеры, которые используются для хранения и обработки данных, а под сервером администрирования понимается компьютер, с помощью которого оператор устанавливает и настраивает систему.

Кластер Tarantool работает по принципам полносвязной топологии (full mesh topology), поэтому все серверы Tarantool должны поддерживать обмен и передачу данных с любого TCP-порта на TCP-порты 3000:4000.

Чтобы настроить удаленный мониторинг или подключиться по административной консоли, сервер администрирования должен иметь доступ к следующим TCP-портам на серверах Tarantool:

- 22 – чтобы использовать SSH-протокол;
- к портам, указанным в конфигурации экземпляра (параметр http_port), для мониторинга HTTP-метрик.

Кроме того, рекомендуется применить следующие настройки для sysctl на всех серверах Tarantool:

```
$ # TCP KeepAlive setting
$ sysctl -w net.ipv4.tcp_keepalive_time=60
$ sysctl -w net.ipv4.tcp_keepalive_intvl=5
$ sysctl -w net.ipv4.tcp_keepalive_probes=5
```

Эта необязательная настройка сетевого стека Linux помогает ускорить решение проблем с сетевым подключением при физическом отказе сервера. Для достижения максимальной производительности может также потребоваться настройка других параметров сетевого стека, которые не относятся к СУБД Tarantool. Для получения дополнительной информации обратитесь к разделу Руководство по оптимизации сетевой производительности (Network Performance Tuning Guide) в пользовательской документации по RHEL7.

2.2 Комплектация

Последние версии пакетов Tarantool Enterprise доступны в разделе для клиентов <https://www.tarantool.io/ru/accounts/customer_zone/packages/enterprise> _ на веб-сайте Tarantool. Для получения доступа обратитесь в support@tarantool.org.

Каждый пакет представляет собой архив tar + gzip и включает в себя следующие компоненты и функции:

- статический бинарный файл Tarantool, чтобы упростить развертывание в средах Linux,
- набор модулей с открытым и закрытым исходным кодом,
- пример приложения, чтобы рассмотреть все включенные модули.

Содержимое архива:

- tarantool – основной исполняемый файл Tarantool.
- tarantoolctl – служебный скрипт для установки вспомогательных модулей и подключения к административной консоли.

- cartridge – служебный скрипт, который помогает настроить среду разработки для приложений и упаковать их с целью упростить развертывание.
- examples/ – директория, которая содержит примеры приложений:
  - pg_writethrough_cache/ – приложение, которое наглядно показывает, как Tarantool может кэшировать данные, например, записанные в базу данных PostgreSQL;
  - ora_writebehind_cache/ – приложение, которое наглядно показывает, как Tarantool может кэшировать вставки и ставить их в очередь, например, в базу данных Oracle;
  - docker/ – приложение, предназначенное для простой упаковки в Docker-контейнер;
- rocks/ – директория, которая содержит набор дополнительных модулей с открытым и закрытым исходным кодом, включенных в дистрибутив в качестве автономного репозитория сторонних библиотек. Для получения более подробной информации см. справочник по сторонним библиотекам.
- templates/ – директория, которая содержит файлы шаблонов для среды разработки приложений.
- deprecated/ – это набор модулей, которые больше не поддерживаются:
  - vshard-zookeeper-orchestrator – это приложение на Python для запуска orchestrator,
  - файлы zookeeper-scm представляют собой модули для интеграции с ZooKeeper (требуются библиотеки usr/).

2.3 Установка

Готовый архив tar + gzip необходимо загрузить на сервер и распаковать:

```
$ tar xvf tarantool-enterprise-bundle-<version>.tar.gz
```

Дополнительная установка не требуется, поскольку распакованные бинарные файлы практически готовы к работе. Перейдите в каталог с бинарными файлами (tarantool-enterprise) и добавьте их в путь для поиска исполняемых файлов, запустив скрипт из дистрибутива:

```
$ source ./env.sh
```

Затем настройте среду разработки, как описано в руководстве для разработчика.

ГЛАВА 3

Руководство для разработчика

Для разработки приложений используйте среду Tarantool Cartridge, которая установлена как компонент Tarantool Enterprise.

Ниже приведен список необходимых команд:

1. Создайте приложение с поддержкой кластеров из шаблона:

```
$ cartridge create --name <app_name> /path/to
```

2. Разработайте приложение:

```
$ cd /path/to/<app_name>
$ ...
```

3. Упакуйте приложение:

```
$ cartridge pack [rpm|tgz] /path/to/<app_name>
```

4. Разверните приложение:

   - Для пакета rpm:

     1. Загрузите пакет на все серверы, выделенные для Tarantool.
     2. Установите пакет:

        ```
        $ yum install <app_name>-<version>.rpm
        ```

     3. Запустите приложение.

        ```
        $ systemctl start <app_name>
        ```

   - Для архива tgz:

     1. Загрузите архив на все серверы, выделенные для Tarantool.
     2. Распакуйте архив:

```
$ tar -xzvf <app_name>-<version>.tar.gz -C /home/<user>/apps
```

3. Запустите приложение

```
$ tarantool init.lua
```

Более подробную информацию с примерами см. в документации по Tarantool с открытым кодом:

- руководство для начинающих, в котором пошагово разбирается разработка и развертывание простого кластерного приложения с помощью Tarantool Cartridge,
- подробное руководство по созданию и управлению кластерными приложениями Tarantool с помощью Tarantool Cartridge.

Кроме того, в этом руководстве особое внимание уделяется функциям разработчика, специфичным для Enterprise-версии, которые доступны в дополнение к версии Tarantool с открытым исходным кодом в среде Tarantool Cartridge:

- авторизация с использованием LDAP в веб-интерфейсе,
- независимые от среды приложения,
- примеры приложений, которые подходят для Enterprise-версии.

3.1 Реализация авторизации с использованием LDAP в веб-интерфейсе

Если в вашей организации есть сервер LDAP, можно подключить к нему Tarantool Enterprise для обработки авторизации. В этом случае следуйте общим рекомендациям <https://www.tarantool.io/en/doc/1.10/book/cartridge/cartridge_dev/#implementing-authorization-in-the-web-interface>‘_ , где на первом шаге необходимо добавить модуль ldap в файл .rockspec в качестве зависимости; рекомендуется также реализовать функцию check_password следующим образом:

```
-- auth.lua
-- Require the LDAP module at the start of the file
local ldap = require('ldap')
...
-- Add a function to check the credentials
local function check_password(username, password)

    -- Configure the necessary LDAP parameters
    local user = string.format("cn=%s,ou=superheros,dc=glauth,dc=com", username)

    -- Connect to the LDAP server
    local ld, err = ldap.open("localhost:3893", user, password)

    -- Return an authentication success or failure
    if not ld then
        return false
    end
    return true
end
 ...
```

3.2 Формирование независимых от среды приложений

Tarantool Enterprise позволяет создавать независимые от среды приложения.

Независимое от среды приложение представляет собой сборку следующих компонентов (в одной директории):

- файлы с кодом на Lua,
- исполняемый файл tarantool,
- подключенные внешние модули (при необходимости).

Запущенное с помощью исполняемого файла tarantool приложение обеспечивает работу сервиса.

Модули – это сторонние библиотеки на Lua, установленные в виртуальную среду (в директории приложения), которая аналогична virtualenv в Python и bundler в Ruby.

Структура такого приложения остается неизменной как на стадии разработки, так и на стадии производственной эксплуатации. Весь связанный с приложением код находится в одном месте, готов к упаковке и копированию на любой сервер.

3.2.1 Упаковка приложений

После определения пользовательских кластерных ролей и разработки приложения упакуйте его со всеми зависимостями (бинарные файлы модулей) и с исполняемым файлом tarantool.

Это позволит вам легко загружать, устанавливать и запускать приложение на любом сервере.

Чтобы упаковать приложение, выполните команду:

```
$ cartridge pack [rpm|tgz] /path/to/<app_name>
```

где укажите путь к вашей среде разработки (репозиторию Git, который содержит код приложения), а также один из параметров сборки:

- rpm для сборки RPM-пакета (рекомендуется), или
- tgz для сборки архива tar + gz (выберите этот параметр, только если у вас нет прав уровня root на серверах, выделенных для Tarantool Enterprise).

В результате будет создан пакет (или сжатый архив) с именем <имя_приложения>-<версия_тег>-<количество_коммитов> (например, myapp-1.2.1-12.rpm), который будет хранить ваше приложение в не зависимом от среды виде.

Далее переходите к развертыванию пакетных (или же архивированных) приложений на серверах.

3.2.2 Развертывание пакетных приложений

Чтобы развернуть пакетное приложение, выполните следующие действия на каждом сервере, выделенном для Tarantool Enterprise:

1. Загрузите локально пакет, созданный на предыдущем шаге.
2. Установите приложение:

   ```
   $ yum install <app_name>-<version>.rpm
   ```

3. Запустите один или несколько экземпляров Tarantool с соответствующими сервисами, как описано ниже.

- Отдельный экземпляр:

```
$ systemctl start <app_name>
```

  Это запустит экземпляр сервиса systemd с прослушиванием по порту 3301.

- Несколько экземпляров на одном или нескольких серверах:

```
$ systemctl start <app_name>@instance_1
$ systemctl start <app_name>@instance_2
...
$ systemctl start <app_name>@instance_<number>
```

  где <app_name>@instance_<number> (<имя_приложения>@экземпляр_<число>) – это имя экземпляра сервиса systemd с инкрементным числом <number> (уникальным для каждого экземпляра), которое следует добавить к порту 3300 для настройки прослушивания (например, 3301, 3302 и т.д.).

4. Если это приложение с поддержкой кластеров, далее переходите к развертыванию кластера.

Чтобы остановить все сервисы на сервере, используйте команду systemctl stop и укажите имена экземпляров по одному. Например:

```
$ systemctl stop <app_name>@instance_1 <app_name>@instance_2 ... <app_name>@instance_<N>
```

3.2.3 Развертывание архивированных приложений

Тогда как пакет RPM по умолчанию помещает ваше приложение в /usr/share/tarantool/<имя_приложения> на вашем сервере, архив tar + gz не создает какую-либо структуру, помимо директории <имя_приложения>/, поэтому вы сами несете ответственность за правильность размещения приложения.

Примечание: Для развертывания рекомендуется использовать RPM-пакеты. Развертывайте архивы, только если у вас нет прав уровня root.

Чтобы разместить и развернуть приложение, выполните следующие действия на каждом сервере, выделенном для Tarantool Enterprise:

1. Загрузите архив, распакуйте его и извлеките содержимое в директорию /home/<user>/apps:

```
$ tar -xzvf <app_name>-<version>.tar.gz -C /home/<user>/apps
```

2. Запустите экземпляры Tarantool с соответствующими сервисами.

   Для управления и конфигурации экземпляров используйте такие средства, как ansible, systemd и supervisord.

3. Если это приложение с поддержкой кластеров, далее переходите к развертыванию кластера.

3.2.4 Обновление кода

Все экземпляры в кластере должны использовать один и тот же код. Это относится ко всем компонентам: пользовательским ролям, приложениям, бинарным файлам модулей, исполняемым файлам tarantool и tarantoolctl (при необходимости).

Обратите внимание на возможную обратную совместимость, которую может принести с собой любой компонент. Это поможет вам выбрать сценарий для обновления в эксплуатационной среде. Имейте в виду, что вы несете ответственность за совместимость кода и обработку конфликтов в случае возникновения несоответствий.

Чтобы обновить любой из компонентов, подготовьте новую версию пакета (архива):

1. Обновите необходимые файлы в вашей среде разработки (директории):
   - Ваш собственный исходный код: пользовательские роли и / или приложения.
   - Бинарные файлы модулей.
   - Исполняемые файлы. Замените их на файлы из нового комплекта.
2. Увеличьте версию, как описано в разделе по управлению версиями приложения.
3. Повторно упакуйте обновленные файлы, как описано в разделе по упаковке приложений.
4. Выберите сценарий обновления, как описано в разделе по обновлению в эксплуатационной среде.

3.3 Запуск примеров приложений

Дистрибутив Enterprise включает в себя примеры приложений в директории examples/, которые демонстрируют основные функциональные возможности Tarantool.

Примеры приложений:

- Приложение кэширования со сквозной записью в PostgreSQL
- Приложение кэширования с отложенной записью в Oracle
- Простейшее приложение в Docker

3.3.1 Приложение кэширования со сквозной записью в PostgreSQL

Пример в pg_writethrough_cache/ показывает, как Tarantool может кэшировать данные, записанные через него в базу данных PostgreSQL для ускорения запросов на чтение.

Для примера приложения требуется развернутая база данных PostgreSQL и следующие модули сторонних библиотек:

```
$ tarantoolctl rocks install http
$ tarantoolctl rocks install pg
$ tarantoolctl rocks install argparse
```

Просмотрите код в файлах, чтобы понять, что может делать приложение.

Чтобы запустить приложение для локальной базы данных PostgreSQL, выполните команду:

```
$ tarantool cachesrv.lua --binary-port 3333 --http-port 8888 --database postgresql://localhost/postgres
```

3.3.2 Приложение кэширования с отложенной записью в Oracle

Пример в ora-writebehind-cache/ показывает, как Tarantool может кэшировать записи и помещать их в базу данных Oracle для ускорения как записи, так и чтения.

Требования приложения

Для примера приложения необходимы:

- развернутая база данных Oracle;
- инструменты Oracle: Instant Client and SQL Plus, оба версии 12.2;

  Примечание: Если Oracle Instant Client выдает ошибки в файлах с расширением .so (динамические библиотеки Oracle), поместите их в какую-либо директорию и добавьте ее в переменную окружения LD_LIBRARY_PATH.

  Например: export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PWD/<путь_к_файлам_so>

- модули сторонних библиотек, перечисленные в файле rockspec.

Чтобы установить модули, выполните следующую команду в директории examples/ora_writebehind_cache:

```
$ tarantoolctl rocks make oracle_rb_cache-0.1.0-1.rockspec
```

Если у вас нет развернутого экземпляра Oracle под рукой, запустите пустой объект в контейнере Docker:

1. В браузере войдите в Реестр контейнеров Oracle, выберите Database (База данных) и примите «Корпоративные условия и ограничения Oracle».
2. В директории ora-writebehind-cache/ войдите в репозиторий под учетной записью Oracle, получите данные и запустите образ с помощью подготовленных скриптов

   ```
   $ docker login container-registry.oracle.com
   Login:
   Password:
   Login Succeeded
   $ docker pull container-registry.oracle.com/database/enterprise:12.2.0.1
   $ docker run -itd \
     -p 1521:1521 \
     -p 5500:5500 \
     --name oracle \
     -v "$(pwd)"/setupdb/configDB.sh:/home/oracle/setup/configDB.sh \
     -v "$(pwd)"/setupdb/runUserScripts.sh:/home/oracle/setup/runUserScripts.sh \
     -v "$(pwd)"/startupdb:/opt/oracle/scripts/startup \
     container-registry.oracle.com/database/enterprise:12.2.0.1
   ```

Когда всё будет готово, запустите пример приложения.

Запуск кэширования с отложенной записью

Чтобы запустить приложение, выполните следующую команду в директории examples/ora_writebehind_cache:

```
$ tarantool init.lua
```

Данное приложение поддерживает следующие запросы:

- Получение данных: GET http://<host>:<http_port>/account/id;

- Добавление: POST http://<host>:<http_port>/account/ со следующими данными:

```
{"clng_clng_id":1,"asut_asut_id":2,"creation_data":"01-JAN-19","navi_user":"userName"}
```

- Обновление: POST http://<host>:<http_port>/account/id с теми же данными, что и в запросе на добавление;
- Удаление: DELETE http://<host>:<http_port>/account/id,где id – это идентификатор учетной записи.

Взгляните на примеры скриптов CURL в директории examples/ora_writebehind_cache/testing и просмотрите файл README.md для получения дополнительной информации об их использовании.

3.3.3 Простейшее приложение в Docker

Пример в директории docker/ содержит простейшее приложение, которое можно упаковать в контейнер Docker и запустить на CentOS 7.

Файл hello.lua представляет собой элементарную точку входа в приложение, поэтому вы можете добавить туда собственный код.

1. Чтобы создать контейнер, выполните команду:

```
$ docker build -t tarantool-enterprise-docker -f Dockerfile ../..
```

2. Чтобы запустить его:

```
$ docker run --rm -t -i tarantool-enterprise-docker
```

ГЛАВА 4

Руководство администратора кластера

В этом руководстве особое внимание уделяется функциям разработчика, специфичным для Enterprise-версии, которые доступны в дополнение к версии Tarantool с открытым исходным кодом в среде Tarantool Cartridge:

- проводник спейсов
- обновление независимых от среды приложений на рабочих серверах

Либо обратитесь к документации по Tarantool с открытым исходным кодом, чтобы получить:

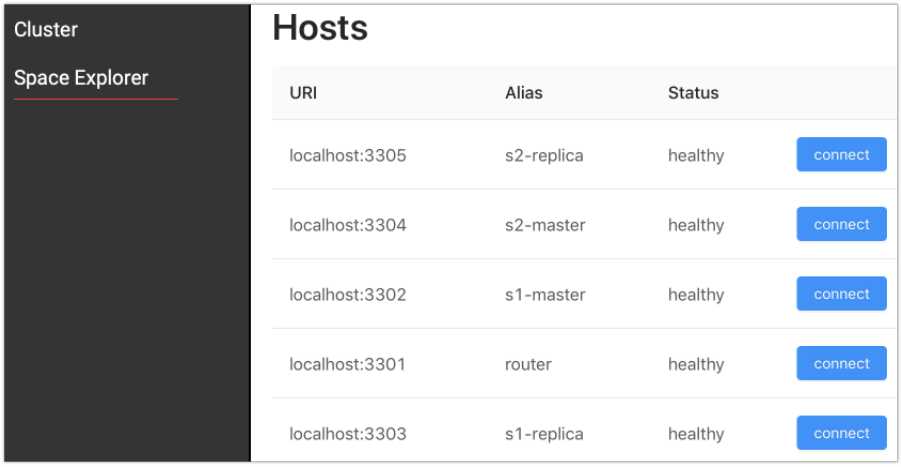
- базовую информацию о развертывании и управлении Tarantool-кластером
- дополнительную информацию об управлении экземплярами Tarantool.

4.1 Анализ спейсов

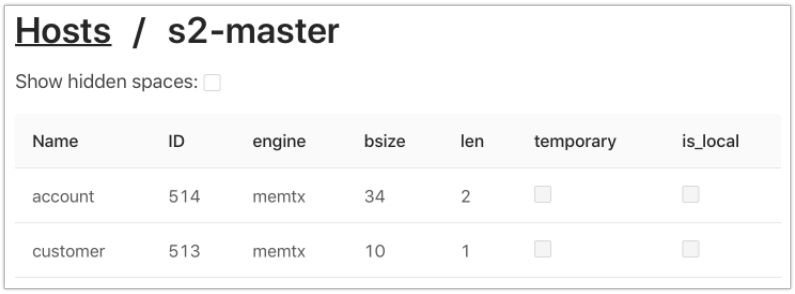
Веб-интерфейс позволяет подключиться к любому экземпляру в кластере в браузере, чтобы проверить, какие в нем хранятся спейсы (если они есть) и их содержимое.

Чтобы просмотреть спейсы:

1. Откройте вкладку Space Explorer (Проводник спейсов) в меню слева:

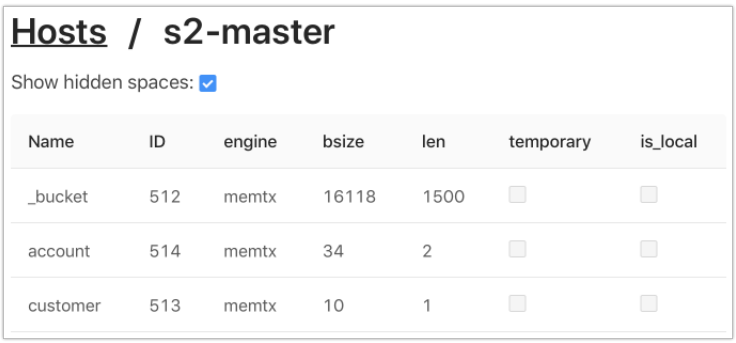


2. Нажмите connect (подключиться) рядом с экземпляром, в котором хранятся данные. Базовая проверка работоспособности (test.py) из примера приложения размещает образцы данных в одном наборе реплик (шарде), поэтому его мастер и реплика хранят данные в своих спейсах:

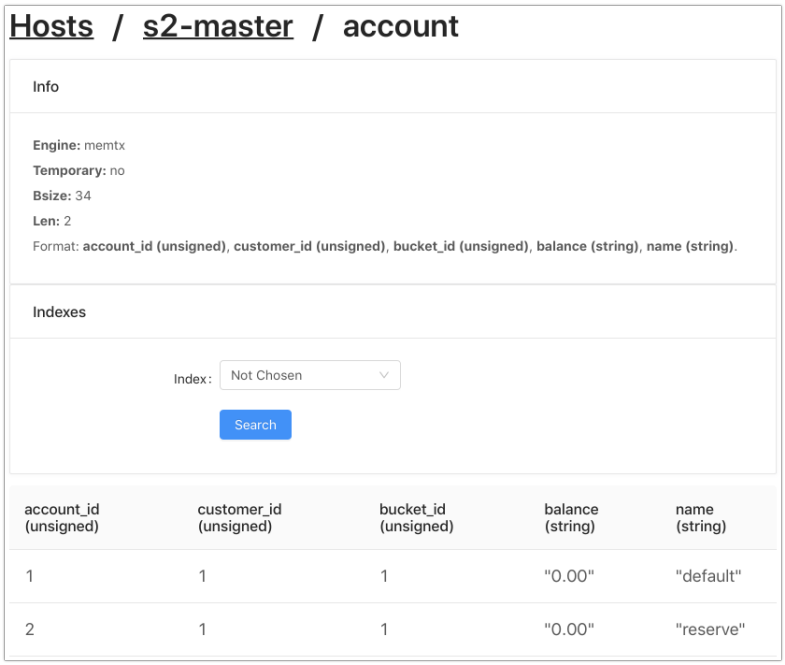


При подключении к экземпляру проводник спейсов отображает таблицу с базовой информацией по спейсам. Для получения дополнительной информации см. справочник по box.space.

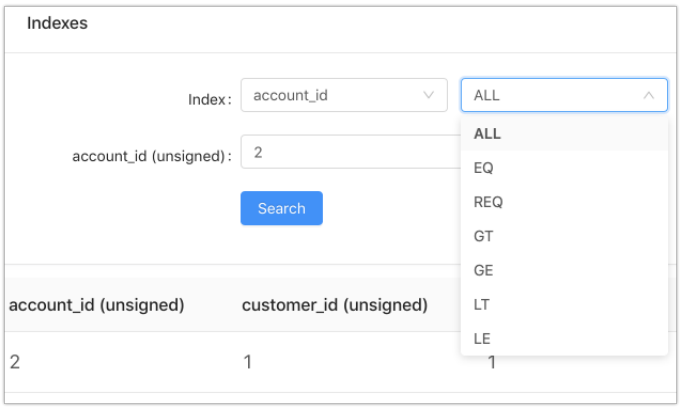
Чтобы просмотреть скрытые спейсы, отметьте соответствующий флажок:



3. Нажмите на имя спейса, чтобы увидеть информацию о его формате и содержимом:



Для поиска данных выберите индекс и (необязательно) его тип итерации из выпадающего списка и введите значение индекса:



4.2 Обновление в эксплуатационной среде

Чтобы выполнить обновление отдельного экземпляра или кластера, необходима новая версия пакетного (или архивированного) приложения.

Обновить отдельный экземпляр просто:

1. Загрузите пакет (архив) на сервер.
2. Остановите текущий экземпляр.
3. Разверните новый экземпляр, как описано в разделе о развертывании пакетных приложений (или архивированных).

4.2.1 Обновление кластера

Чтобы обновить кластер, выберите один из следующих сценариев:

- Завершение работы кластера. Рекомендуется для обновлений без обратной совместимости, требует простоя.
- Поочередное обновление. Рекомендуется для обновлений с обратной совместимостью, не требует простоя.

Чтобы обновить кластер, выполните следующие действия:

1. Запланируйте время простоя или поочередное обновление экземпляров.
2. Загрузите новый пакет (или архив) приложения на все серверы.

Далее действуйте по выбранному сценарию:

- Завершение работы кластера:
  1. Остановите все экземпляры на всех серверах.
  2. Разверните новый пакет (архив) на каждом сервере.
- Поочередное обновление. Выполните следующие действия для каждого набора реплик поочередно:
  1. Остановите реплику на любом сервере.
  2. Разверните новый пакет (архив) на месте старой реплики.
  3. Переведите новую реплику в статус мастера (см. раздел Смена мастера в наборе реплик в руководстве по Tarantool).
  4. Повторно разверните старый мастер и остальные экземпляры в наборе реплик.
  5. Будьте готовы решать возможные проблемы с логикой.

ГЛАВА 5

Инструкции по повышению безопасности

В данном руководстве объясняется, как обеспечить безопасность в кластере Tarantool Enterprise, используя встроенные функции, и даются общие рекомендации по повышению безопасности.

В Tarantool Enterprise нет выделенного API для контроля безопасности. Все необходимые настройки можно выполнить через административную консоль или код инициализации.

5.1 Встроенные средства безопасности

В Tarantool Enterprise есть следующие встроенные средства безопасности:

- аутентификация,
- управление доступом,
- журнал аудита;

А также функции резервного копирования:

- создание снимков (физическое),
- создание дампов (логическое).

В следующих разделах описаны эти функции и приведены ссылки на подробную документацию.

5.1.1 Аутентификация

Tarantool Enterprise поддерживает аутентификацию на основе паролей и допускает два типа соединений:

- через административную консоль и
- через двоичный порт для операций чтения / записи и вызова процедур.

Для получения дополнительной информации об аутентификации и типах соединений см. раздел по безопасности в руководстве по Tarantool.

Кроме того, Tarantool предоставляет следующие функциональные возможности:

- сеансы – состояния, которые связывают соединения с пользователями и предоставляют доступ к API Tarantool после аутентификации,
- триггеры аутентификации, которые выполняют действия при событиях аутентификации.
- сторонние (внешние) протоколы и службы аутентификации, такие как LDAP или Active Directory, поддерживаются в веб-интерфейсе, но недоступны на уровне бинарного протокола.

5.1.2 Управление доступом

Для администраторов Tarantool Enterprise предоставляет средства предотвращения несанкционированного доступа к базе данных и к определенным функциям.

Tarantool различает:

- разных пользователей (гостей и администраторов),
- права, связанные с пользователями,
- роли (контейнеры для ролей), выданные пользователям;

И хранит информацию о доступе в системных спейсах:

- спейс _user для хранения имен пользователей и хеш-паролей для аутентификации,
- спейс _priv для хранения прав доступа.

Для получения дополнительной информации см. раздел по управлению доступом в руководстве по Tarantool.

Пользователи, которые создают объекты (спейсы, индексы, пользователей, роли, последовательности и функции) в базе данных становятся их владельцами и автоматически получают права на то, что они создают. Для получения дополнительной информации см. раздел о владельцах и правах в руководстве по Tarantool.

5.1.3 Журнал аудита

В Tarantool Enterprise есть встроенный журнал аудита, в котором записываются такие события, как:

- пройденная и непройденная аутентификации;
- закрытие подключения;
- создание, удаление, включение и отключение пользователей;
- изменение паролей, прав и ролей;
- отказ в доступе к объектам базы данных;

В журнале аудита указываются:

- отметки времени,
- имена пользователей, которые выполняли действия,
- типы событий (например, user_create, user_enable, disconnect и т.д.),
- описание.

Есть два конфигурационных параметра для настройки журнала аудита:

- audit_log = <PATH_TO_FILE>, аналог параметра log; если данный параметр задан, то Tarantool записывает события, подлежащие аудиту, в указанный файл;
- audit_nonblock, аналог параметра log_nonblock.

Для получения дополнительной информации о журналировании см. следующие разделы:

- раздел о журналах в руководстве по Tarantool,
- раздел по журналированию в справочнике по настройке,
- приложение А данного документа.

Права доступа к файлам журнала можно настроить, как для любого другого объекта файловой системы Unix – через chmod.

5.2 Рекомендации по повышению безопасности

В этом разделе даны рекомендации, которые могут помочь вам повысить безопасность кластера.

5.2.1 Криптографическая защита трафика

Tarantool Enterprise не шифрует трафик, проходящий через двоичные соединения (т.е. между серверами в кластере). Чтобы защитить такие соединения, рекомендуется:

- настроить туннелирование соединения или
- зашифровать сами данные, которые хранятся в базе.

Для получения дополнительной информации см. справочник по модулю crypto.

Модуль HTTP-сервера из сторонних библиотек не поддерживает протокол HTTPS. Чтобы установить безопасное соединение для клиента (например, REST-сервис), рекомендуется скрыть экземпляр Tarantool (роутер, если это кластер экземпляров) за сервером Nginx и настроить для него сертификат SSL.

Чтобы убедиться, что никакая информация не может быть перехвачена извне, запустите Nginx на том же физическом сервере, что и экземпляр, и настройте их связь по Unix-сокету. Для получения дополнительной информации см. справочник по модулю socket.

5.2.2 Настройка брандмауэра

Чтобы защитить кластер от нежелательной сетевой активности извне, настройте брандмауэр на каждом сервере, чтобы разрешить трафик через порты, перечисленные в сетевых требованиях.

Если вы используете статические IP-адреса, повторно внесите их в белый список на каждом сервере, поскольку кластер работает на принципах полносвязной топологии (full mesh topology). Рекомендуется внести в черный список всех остальные адреса на всех серверах, кроме роутера (работающего за сервером Nginx).

Tarantool Enterprise не предоставляет защиту от DoS-атак или DDoS-атак. Для этих целей рекомендуется использовать сторонние программы.

5.2.3 Целостность данных

Tarantool Enterprise не хранит контрольные суммы и не предоставляет инструменты для контроля целостности данных. Тем не менее, он обеспечивает персистентность данных с помощью журнала упреждающей записи, регулярно делает снимок всего набора данных на диск и проверяет формат данных всякий раз, когда считывает данные с диска. Для получения дополнительной информации см. раздел о персистентности данных в руководстве по Tarantool.

ГЛАВА 6

Справочник по модулям

6.1 ldap

6.1.1 LDAP client library for tarantool

This library allows you to authenticate in a LDAP server and perform searches.

Usage example

First, download glauth, a simple Go-based LDAP server using the following command:

```
./download_glauth.sh
```

Then run glauth:

```
./glauth -c glauth_test.cfg
```

Then run the following tarantool script in a separate terminal

```
#!/usr/bin/env tarantool

local ldap = require('ldap')
local yaml = require('yaml')

local user = "cn=johndoe,ou=superheros,dc=glauth,dc=com"
local password = "dogood"

local ld = assert(ldap.open("localhost:3893", user, password))

local iter = assert(ldap.search(ld,
    {base="dc=glauth,dc=com",
     scope="subtree",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    sizelimit=10,
    filter="(objectclass=*)"}))

for entry in iter do
   print(yaml.encode(entry))
end
```

Usage ldap for authorization in the web interface

See this doc page

6.1.2 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

[Unreleased]

Added

- Gitlab CI testing

[1.0.0] - 2019-04-03

Added

- Basic functionality
- Luarock-based packaging
- Build without any dependencies but Tarantool Enterprise

6.2 task

6.2.1 Module *scheduler*

Task scheduler (a cartridge role).

Functions

init (opts)

Initialize the scheduler, start cron background fiber.

Parameters:

- opts:

  - runner: (string) name of the runner module name, default is task.runner.local
  - storage: (string) name of the storage module, default is task.storage.local

get_tasks ()

List registered tasks.

get_task_log (opts)

List task execution log, ordered by creation time.

Parameters:

- opts:
  - filter: (table) must contain either an id number, or an array of names
  - limit: (number) the maximum length of a single task log fetched from storage
  - created: (string) ISO 8601 timestamp, acts as offset for pagination

start (name)

Start a task.

Parameters:

- name: (string) name of the task

stop (id)

Stop a running or pending task.

Parameters:

- id: (string) name of the task

forget (id)

Remove task execution log record from storage.

Parameters:

- id: (string) name of the task

start_periodical_task (name)

Start a periodical task.

Parameters:

- name: (string) name of the task

register (tasks)

Register available tasks. Starts launching periodical and continuous tasks, allows to start single_shot tasks.

Parameters:

- tasks: (table) names of tasks

6.2.2 Module *roles.scheduler*

Task manager (a cartridge role).

Handles setting available tasks from the cluster config, handles scheduler fibers (including failover). In a basic case, it sets up a scheduler, a task storage, and a task runner on the same node.

6.2.3 Module *roles.runner*

Local task runner module, used by default.

You must provide the same interface if you want to write your own one. It is expected to take tasks from storage, run them, and complete them.

6.2.4 Module *roles.storage*

Local task storage module, used by default.

You must provide the same interface if you want to write your own one.

Functions

select (index, key, opts)

Select task log.

Parameters:

- index: (string) index to iterate over; default is id
- key: (string) index key value
- opts: (optional table) compatible with[vanilla Tarantool iterator parameters](https://www.tarantool.io/en/doc/1.10/book/box/box_index/#lua-function.index_object.select)

6.2.5 Task Manager for Tarantool Enterprise

@lookup README.md

Task manager module allows you to automate several types of background jobs:

- periodical, that need to be launched according to cron-like schedule;
- continuous, that need to be working at all times;
- single_shot, that are launched manually by the operations team.

You get the following features out-of-the-box:

- configurable schedule of periodic tasks;
- guarding and restarting continuous tasks;
- stored log of task launches;
- API and UI for launching tasks and observing launch results.

Task manager comes with several built-in cluster roles:

- task.roles.scheduler, a module which allows to configure launchable tasks;
- task.roles.runner, a cluster-aware stateless task runner;
- task.roles.storage, a cluster-aware dedicated task contents storage;
- plugin-based API which allows you to provide you own storage module (e. g. distributed, or external to Tarantool cluster), or your own runner module, providing more tooling for your needs.

Basic usage (single-node application)

1) Embed the following to the instance file:

```
...
local task = require('task')
...
cartridge.cfg({
  roles = {'my_role', ...}
})
task.init_webui()
```

2) Add to your role dependency on task scheduler, runner and storage roles:

```
return {
    ...
    dependencies = {
        'task.roles.storage',
        'task.roles.scheduler',
        'task.roles.runner'
    }
}
```

3) Add tasks section to your cluster configuration:

```
tasks:
  my_task:
    kind: periodical
    func_name: my_module.my_task
    schedule: "*/* 1 * * * *"
```

4) That's it! my_task function will be launched every minute.

Advanced usage (multi-node installation)

1) Embed the following to the instance file:

```
...

...
local task = require('task')
cartridge.cfg({
  roles = {
    ...
    'task.roles.scheduler',
    'task.roles.storage',
    'task.roles.runner'
  }
})

task.init_webui()
```

2) Enable the task scheduler role on a dedicated node in your cluster (after deployment). If you set up a big cluster, don't set up more than one replica set with the scheduler.

3) Enable the task storage role on a dedicated node in your cluster (after deployment), possibly on the same node as task scheduler. If you set up a big cluster, don't set up more than one replica set with the storage.

4) Enable the task runner role on dedicated stateless nodes in your cluster (after deployment) - as many as you may need.

Advanced usage (sharded storage)

1) Embed the following to the instance file:

```
...
local task = require('task')
cartridge.cfg({
  roles = {
    ...
    'task.roles.sharded.scheduler',
    'task.roles.sharded.storage',
    'task.roles.sharded.runner'
  }
})

task.init_webui()
```

2) Enable the task scheduler role on a dedicated node in your cluster (after deployment). If you set up a big cluster, don't set up more than one replica set with the scheduler.

3) Enable the task storage role on the nodes of some vshard group (or an all storage nodes). Set up cartridge built-in vshard-storage role on these nodes.

4) Enable the task runner role on dedicated stateless nodes in your cluster (after deployment) - as many as you may need.

Tasks configuration

Tasks are configured via the scheduler cluster role. An example of valid role configuration:

```
tasks:
  my_reload:
    kind: periodical
    func_name: my_module.cache_reload
    schedule: "*/* 1 * * * *"
    time_to_resolve: 180
  my_flush:
    kind: single_shot
    func_name: my_module.cache_flush
    args:

      - some_string1
      - some_string2
  push_metrics:
    kind: continuous
    func_name: my_module.push_metrics
    pause_sec: 30
```

- Every task must have a unique name (subsection name in config).
- Each task must have a kind: periodical, continuous, single_shot.
- Each task must have a func_name - name of the function (preceded by the name of the module) which will be invoked.
- Each task may have time_to_resolve - timeout after which a running task is considered lost (failed).
- Each task may have args - an array of arguments which will be passed to the function (to allow basic parametrization)
- Periodical tasks also must have a schedule, conforming with ccronexpr (basically, cron with seconds).
- Tasks may have a pause_sec - pause between launches (60 seconds by default).

You may set up default task config for your application in task.init() call:

```
task.init({
  default_config = {
    my_task = {
      kind = 'single_shot',
      func_name = 'dummy_task.dummy',
    }
  }
})
```

Default config will be applied if no tasks are set in clusterwide config. task.init() should be called prior to cluster.cfg().

Advanced usage

Running a task via API

Everything visible from the UI is available via the API. You may look up requests in the UI or in the cartridge graphql schema.

```
curl -w "\n" -X POST http://127.0.0.1:8080/admin/api --fail -d@- <<'QUERY'
{"query": "mutation { task { start(name: "my_reload") } }"}
QUERY
```

Supplying custom runner and storage

Embed the following to your instance file

```
task.init({
    runner = 'my_tasks.my_runner',
    storage = 'my_tasks.my_storage'
})
...
cartridge.cfg{...}
task.init_webui()
```

Be sure to call task.init() it prior to cartridge.cfg, so that custom options would be provided by the time role initialization starts.

You may set up then only task scheduler role, and handle storages and runners yourself.

Writing your own runner and storage

Runner module must expose api member with the following functions:

- stop_task

storage module must expose api member with the following functions:

- select
- get
- delete
- put
- complete
- take
- cancel
- wait

For more details refer to built-in runner and storage documentation

6.2.6 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

[Unreleased]

[0.6.1]

- Bugfixes and minor improvements

[0.6.0]

Added:

- Ability to specify task args in config
- several bugfixes

[0.5.0]

Added:

- task storages can now be set up on vshard storages
- default task configuration can be set up in task.init()

[0.4.0]

Changed:

- task now depends on cartridge instead of cluster
- cartridge dependency bumped to 1.0.0

[0.3.0]

Added: - Cluster-aware task storage role - Cluster-aware task runner role

Changed: - Module initialization API changed: now task scheduler role does not start runner and storage by default; you must handle it via role dependencies or via respective cluster roles

[0.2.0]

Added: - Basic functionality - scheduler, local task storage, local task runner, cluster role, cluster UI

6.3 tracing

6.3.1 Module *opentracing*

OpenTracing API module entrypoint

Functions

set_global_tracer (tracer)

Set global tracer

Parameters:

- tracer: (table)

get_global_tracer ()

Get global tracer

Returns:

(table) tracer

start_span (name, opts)

Start root span

Parameters:

- name: (string)
- opts: table specifying modifications to make to thenewly created span. The following parameters are supported: trace_id , references ,a list of referenced spans; start_time , the time to mark when the spanbegins (in microseconds since epoch); tags , a table of tags to add tothe created span.
  - trace_id: (optional string)
  - child_of: (optional table)
  - references: (optional table)
  - tags: (optional table)

Returns:

(table) span

start_span_from_context (context, name)

Start new child span from context

Parameters:

- context: (table)
- name: (string)

Returns:

(table) span

trace_with_context (name, ctx, fun, ...)

Trace function with context by global tracer This function starts span from global tracer and finishes it after execution

Parameters:

- name: (string) span name
- ctx: (table) context
- fun: (function) wrapped function
- ...: (vararg) function's arguments

Returns:

(vararg) result

Or

(nil) nil

(string) err error message

Usage:

```
-- Process HTTP request
local ctx = opentracing.extractors.http(req.headers)
local result, err = opentracing.trace_with_context('process_data', ctx, process_data, req.body)
-- Wrap functions. In example we create root span that generates two child spans
local span = opentracing.start_span()
local result, err = opentracing.trace_with_context('format_string', span:context(), format, str)
if not result ~= nil then
    print('Error: ', err)
end
opentracing.trace_with_context('print_string', span:context(), print, result)
span:finish()
```

trace (name, fun, ...)

Trace function by global tracer This function starts span from global tracer and finishes it after execution

Parameters:

- name: (string) span name
- fun: (function) wrapped function
- ...: (vararg) function's arguments

Returns:

(vararg) result

Or

(nil) nil

(string) err error message

Usage:

```
local result, err = opentracing.trace_with_context('process_data', process_data, req.body)
```

6.3.2 Module *opentracing.span*

Span represents a unit of work executed on behalf of a trace.

Examples of spans include a remote procedure call, or a in-process method call to a sub-component. Every span in a trace may have zero or more causal parents, and these relationships transitively form a DAG. It is

common for spans to have at most one parent, and thus most traces are merely tree structures. The internal data structure is modeled off the ZipKin Span JSON Structure This makes it cheaper to convert to JSON for submission to the ZipKin HTTP api, which Jaegar also implements.

You can find it documented in this OpenAPI spec: https://github.com/openzipkin/zipkin-api/blob/7e33e977/zipkin2-api.yaml#L280

Functions

new (tracer, context, name, start_timestamp)

Create new span

Parameters:

- tracer: (table)
- context: (table)
- name: (string)
- start_timestamp: (optional number)

Returns:

(table) span

context (self)

Provides access to the SpanContext associated with this Span The SpanContext contains state that propagates from Span to Span in a larger tracer.

Parameters:

- self: (table)

Returns:

(table) context

tracer (self)

Provides access to the Tracer that created this span.

Parameters:

- self: (table)

Returns:

(table) tracer the Tracer that created this span.

set_operation_name (self, name)

Changes the operation name

Parameters:

- self: (table)

- name: (string)

Returns:

(table) tracer

start_child_span (self, name, start_timestamp)

Start child span

Parameters:

- self: (table)
- name: (string)
- start_timestamp: (optional number)

Returns:

(table) child span

finish (self, opts)

Indicates the work represented by this Span has completed or terminated.

If finish is called a second time, it is guaranteed to do nothing.

Parameters:

- self: (table)
- opts:
  - finish_timestamp: (number) a timestamp represented by microsecondssince the epoch to mark when the span ended. If unspecified, the currenttime will be used.
  - error: (string) add error tag

Returns:

(boolean) true

Or

(boolean) false

(string) error

set_tag (self, key, value)

Attaches a key/value pair to the Span .

The value must be a string, bool, numeric type, or table of such values.

Parameters:

- self: (table)
- key: (string) key or name of the tag. Must be a string.
- value: (any) value of the tag

Returns:

(boolean) true

get_tag (self, key)

Get span's tag

Parameters:

- self: (table)
- key: (string)

Returns:

(any) tag value

each_tag (self)

Get tags iterator

Parameters:

- self: (table)

Returns:

(function) iterator

(table) tags

get_tags (self)

Get copy of span's tags

Parameters:

- self: (table)

Returns:

(table) tags

log (self, key, value, timestamp)

Log some action

Parameters:

- self: (table)
- key: (table)
- value: (table)
- timestamp: (optional number)

Returns:

(boolean) true

log_kv (self, key_values, timestamp)

Attaches a log record to the Span .

Parameters:

- self: (table)
- key_values: (table) a table of string keys and values of string, bool, ornumeric types
- timestamp: (optional number) an optional timestamp as a unix timestamp.defaults to the current time

Returns:

(boolean) true

Usage:

```
span:log_kv({
  ["event"] = "time to first byte",
  ["packet.size"] = packet:size()})
```

each_log (self)

Get span's logs iterator

Parameters:

- self: (table)

Returns:

(function) log iterator

(table) logs

set_baggage_item (self, key, value)

Stores a Baggage item in the Span as a key/value pair.

Enables powerful distributed context propagation functionality where arbitrary application data can be carried along the full path of request execution throughout the system.

Note 1: Baggage is only propagated to the future (recursive) children of this Span .

Note 2: Baggage is sent in-band with every subsequent local and remote calls, so this feature must be used with care.

Parameters:

- self: (table)
- key: (string) Baggage item key
- value: (string) Baggage item value

Returns:

(boolean) true

get_baggage_item (self, key)

Retrieves value of the baggage item with the given key.

Parameters:

- self: (table)
- key: (string)

Returns:

(string) value

each_baggage_item (self)

Returns an iterator over each attached baggage item

Parameters:

- self: (table)

Returns:

(function) iterator

(table) baggage

set_component (self, component)

Set component tag (The software package, framework, library, or module that generated the associated Span.)

Parameters:

- self: (table)
- component: (string)

set_http_method (self, method)

Set HTTP method of the request for the associated Span

Parameters:

- self: (table)
- method: (string)

set_http_status_code (self, status_code)

Set HTTP response status code for the associated Span

Parameters:

- self: (table)
- status_code: (number)

set_http_url (self, url)

Set URL of the request being handled in this segment of the trace, in standard URI format

Parameters:

- self: (table)
- url: (string)

set_http_host (self, host)

Set the domain portion of the URL or host header. Used to filter by host as opposed to ip address.

Parameters:

- self: (table)
- host: (string)

set_http_path (self, path)

Set the absolute http path, without any query parameters. Used as a filter or to clarify the request path for a given route. For example, the path for a route «/objects/:objectId» could be «/objects/abdc-ff». This does not limit cardinality like HTTP_ROUTE(«http.route») can, so is not a good input to a span name.

The Zipkin query api only supports equals filters. Dropping query parameters makes the number of distinct URIs less. For example, one can query for the same resource, regardless of signing parameters encoded in the query line. Dropping query parameters also limits the security impact of this tag.

Parameters:

- self: (table)
- path: (string)

set_http_route (self, route)

Set the route which a request matched or «» (empty string) if routing is supported, but there was no match. Unlike HTTP_PATH(«http.path»), this value is fixed cardinality, so is a safe input to a span name function or a metrics dimension. Different formats are possible. For example, the following are all valid route templates: «/users» «/users/:userId» «/users/*»

Route-based span name generation often uses other tags, such as HTTP_METHOD(«http.method») and HTTP_STATUS_CODE(«http.status_code»). Route-based names can look like «get /users/{userId}», «post /users», «get not_found» or «get redirected».

Parameters:

- self: (table)
- route: (string)

set_http_request_size (self, host)

Set the size of the non-empty HTTP request body, in bytes. Large uploads can exceed limits or contribute directly to latency.

Parameters:

- self: (table)
- host: (string)

set_response_size (self, host)

Set the size of the non-empty HTTP response body, in bytes. Large downloads can exceed limits or contribute directly to latency.

Parameters:

- self: (table)
- host: (string)

set_peer_address (self, address)

Set remote «address», suitable for use in a networking client library. This may be a «ip:port», a bare «hostname», a FQDN, or even a JDBC substring like «mysql://prod-db:3306»

Parameters:

- self: (table)
- address: (string)

set_peer_hostname (self, hostname)

Set remote hostname

Parameters:

- self: (table)
- hostname: (string)

set_peer_ipv4 (self, IPv4)

Set remote IPv4 address as a .-separated tuple

Parameters:

- self: (table)
- IPv4: (string)

set_peer_ipv6 (self, IPv6)

Set remote IPv6 address as a string of colon-separated 4-char hex tuples

Parameters:

- self: (table)
- IPv6: (string)

set_peer_port (self, port)

Set remote port

Parameters:

- self: (table)
- port: (number)

set_peer_service (self, service_name)

Set remote service name (for some unspecified definition of «service»)

Parameters:

- self: (table)
- service_name: (string)

set_sampling_priority (self, priority)

Set sampling priority If greater than 0, a hint to the Tracer to do its best to capture the trace. If 0, a hint to the trace to not-capture the trace. If absent, the Tracer should use its default sampling mechanism.

Parameters:

- self: (table)
- priority: (number)

set_kind (self, kind)

Set span's king Either «client» or «server» for the appropriate roles in an RPC, and «producer» or «consumer» for the appropriate roles in a messaging scenario.

Parameters:

- self: (table)
- kind: (string)

set_client_kind (self)

Set client kind to span

Parameters:

- self: (table)

set_server_kind (self)

Set server kind to span

Parameters:

- self: (table)

set_producer_kind (self)

Set producer kind to span

Parameters:

- self: (table)

set_consumer_kind (self)

Set consumer kind to span

Parameters:

- self: (table)

6.3.3 Module *opentracing.span_context*

SpanContext represents Span state that must propagate to descendant Span „s and across process boundaries.

SpanContext is logically divided into two pieces: the user-level «Baggage» (see Span.set_baggage_item and Span.get_baggage_item) that propagates across Span boundaries and any tracer-implementation-specific fields that are needed to identify or otherwise contextualize the associated Span (e.g., a (trace_id, span_id, sampled) tuple).

Functions

new (opts)

Create new span context

Parameters:

- opts: options
  - trace_id: (optional string)
  - span_id: (optional string)
  - parent_id: (optional string)

  - should_sample: (optional boolean)
  - baggage: (optional table)

Returns:

(table) span context

child ()

Create span child span context

Returns:

(table) child span context

clone_with_baggage_item (self, key, value)

New from existing but with an extra baggage item Clone context and add item to its baggage

Parameters:

- self: (table)
- key: (string)
- value: (string)

Returns:

(table) context

get_baggage_item (self, key)

Get item from baggage

Parameters:

- self: (table)
- key: (string)

Returns:

(string) value

each_baggage_item (self)

Get baggage item iterator

Parameters:

- self: (table)

Returns:

(function) iterator

(table) baggage

6.3.4 Module *opentracing.tracer*

Tracer is the entry point API between instrumentation code and the tracing implementation.

This implementation both defines the public Tracer API, and provides a default no-op behavior.

Functions

new (reporter, sampler)

Init new tracer

Parameters:

- reporter:
  - report: (function)
- sampler:
  - sample: (function)

Returns:

(table) tracer

start_span (self, name, opts)

Starts and returns a new Span representing a unit of work.

Example usage:

Create a root Span (a Span with no causal references):

```
tracer:start_span("op-name")
```

Create a child Span :

```
tracer:start_span(
      "op-name",
      {["references"] = {{"child_of", parent_span:context()}}})
```

Parameters:

- self: (table)
- name: (string) operation_name name of the operation represented by the new.. code-block:: lua Span from the perspective of the current service.
- opts: table specifying modifications to make to thenewly created span. The following parameters are supported: trace_id , references ,a list of referenced spans; start_time , the time to mark when the spanbegins (in microseconds since epoch); tags , a table of tags to add tothe created span.
  - trace_id: (optional string)
  - child_of: (optional table)
  - references: (optional table)
  - tags: (optional table)
  - start_timestamp: (optional number)

Returns:

(table) span a Span instance

register_injector (self, format, injector)

Register injector for tracer

Parameters:

- self: (table)
- format: (string)
- injector: (function)

Returns:

(boolean) true

register_extractor (self, format, extractor)

Register extractor for tracer

Parameters:

- self: (table)
- format: (string)
- extractor: (function)

Returns:

(boolean) true

inject (self, context, format, carrier)

Inject context into carrier with specified format. See https://opentracing.io/docs/overview/inject-extract/

Parameters:

- self: (table)
- context: (table)
- format: (string)
- carrier: (table)

Returns:

(table) carrier

Or

(nil)

(string) error

extract (self, format, carrier)

Extract context from carrier with specified format. See https://opentracing.io/docs/overview/inject-extract/

Parameters:

- self: (table)
- format: (string)
- carrier: (table)

Returns:

(table) context

Or

(nil)

(string) error

http_headers_inject (self, context, carrier)

Injects span_context into carrier using a format appropriate for HTTP headers.

Parameters:

- self: (table)
- context: (table) the SpanContext instance to inject
- carrier: (table)

Returns:

(table) context a table to contain the span context

Usage:

```
carrier = {}
tracer:http_headers_inject(span:context(), carrier)
```

text_map_inject (self, context, carrier)

Injects span_context into carrier .

Parameters:

- self: (table)
- context: (table) the SpanContext instance to inject
- carrier: (table)

Returns:

(table) context a table to contain the span context

Usage:

```
carrier = {}
tracer:text_map_inject(span:context(), carrier)
```

http_headers_extract (self, carrier)

Returns a SpanContext instance extracted from the carrier or nil if no such SpanContext could be found. http_headers_extract expects a format appropriate for HTTP headers and uses case-sensitive comparisons for the keys.

Parameters:

- self: (table)
- carrier: (table) the format-specific carrier object to extract from

Returns:

(table) context

Or

(nil)

(string) error

text_map_extract (self, carrier)

Returns a SpanContext instance extracted from the carrier or nil if no such SpanContext could be found.

Parameters:

- self: (table)
- carrier: (table) the format-specific carrier object to extract from

Returns:

(table) context

Or

(nil)

(string) error

6.3.5 Module *zipkin.tracer*

Client for Zipkin

Functions

new (config, sampler)

Init new Zipkin Tracer

Parameters:

- config: Table with Zipkin configuration
  - base_url: (table) Zipkin API base url
  - api_method: (table) API method to send spans to zipkin
  - report_interval: (table) Interval of reports to zipkin
  - spans_limit: (optional number) Limit of a spans buffer (1k by default)
  - on_error: (optional function) On error callback that apply error in string format
  - spans_limit: (optional number) Limit of a spans buffer (1k by default)
- sampler: (table) Table that contains function samplethat is apply span name and mark this span for further report

Returns:

(table) context

Or

(nil) nil

(string) error

6.3.6 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

[Unreleased]

[0.1.0] - 2019-06-05

- OpenTracing API implementation
- Zipkin client
- Lua API documentation, which you can read with tarantoolctl rocks doc tracing command

6.3.7 Tracing for Tarantool

Tracing module for Tarantool includes the following parts:

- OpenTracing API
- Zipkin tracer

Table of contents

OpenTracing

This library is a Tarantool platform API for OpenTracing.

Required Reading

To fully understand this platform API, it's helpful to be familiar with the OpenTracing project and terminology more specifically.

Conventions

- All timestamps are in microseconds

Span

> The "span" is the primary building block of a distributed trace, representing an individual unit of work done in a distributed system. Traces in OpenTracing are defined implicitly by their Spans. In particular, a Trace can be thought of as a directed acyclic graph (DAG) of Spans, where the edges between Spans are called References.

```
local opentracing_span = require('opentracing.span')
-- tracer - External tracer
-- context - Span context
-- name - Name of span
-- start_timestamp (optional) - Time of span's start in microseconds (by default current time)
local span = opentracing_span.new(tracer, context, name, start_timestamp)
```

SpanContext

> The SpanContext carries data across process boundaries.

```
local opentracing_span_context = require('opentracing.span_context')
-- trace_id (optional) - Trace ID (by default generates automatically)
-- span_id (optional) - Span ID (by default generates automatically)
-- parent_id (optional) - Span ID of parent span (by default is empty)
-- should_sample (optional) - Flag is enable collecting data of this span (by default false)
-- baggage (optional) - Table with trace baggage (by default is empty table)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
local context = opentracing_span_context.new({
                    tracer_id = trace_id,
                    span_id = span_id,
                    parent_id = parent_id,
                    should_sample = should_sample,
                    baggage = baggage,
                })
```

Tracer

> The Tracer interface creates Spans and understands how to Inject (serialize) and Extract (deserialize) their metadata across process boundaries.

An interface for custom tracers

```
local opentracing_tracer = require('opentracing.tracer')
-- reporter (optional) - Table with `report` method to process finished spans (by default no-op table)
-- sampler (optional) - Table with `sample` method to select traces to send to distributing tracing system (by
↪default random selection)
-- But you can implement your own sampler with appropriate sampling strategy
-- For more information see: https://www.jaegertracing.io/docs/1.11/sampling/
local tracer = opentracing_tracer.new(reporter, sampler)
```

Basic usage

```
local zipkin = require('zipkin.tracer')
local opentracing = require('opentracing')

-- Create client to Zipkin and set it global for easy access from any part of app
local tracer = zipkin.new(config)
opentracing.set_global_tracer(tracer)

-- Create and manage spans manually
local span = opentracing.start_span('root span')
-- ... your code ...
span:finish()

-- Simple wrappers via user's function

-- Creates span before function call and finishes it after
local result = opentracing.trace('one span', func, ...)

-- Wrappers with context passing
local span = opentracing.start_span('root span')

-- Pass your function as third argument and then its arguments
opentracing.trace_with_context('child span 1', span:context(), func1, ...)
opentracing.trace_with_context('child span 2', span:context(), func2, ...)
span:finish()
```

Zipkin

Zipkin is a distributed tracing system.

It helps gather timing data needed to troubleshoot latency problems in microservice architectures. It manages both the collection and lookup of this data.

This module allows you to instance Zipkin Tracer that can start spans and will report collected spans to Zipkin Server.

Basic usage

```
local zipkin = require('zipkin.tracer')
-- First argument is config that contains url of Zipkin API,
--  method to send collected traces and interval of reports in seconds
-- Second optional argument is Sampler (see OpenTracing API description), by default random sampler
local tracer = zipkin.new({
    base_url = 'localhost:9411/api/v2/spans',
    api_method = 'POST',
    report_interval = 0,
}, Sampler)

local span = tracer:start_span('example')
-- ...
span:finish()
```

Examples

HTTP

This example is a Lua port of Go OpenTracing tutorial.

Description

The example demonstrates trace propagation through two services: formatter that formats the source string to «Hello, world» and publisher that prints it in the console.

Add data to these services via HTTP; initially it sends client.

Note: example requires http rock (version >= 2.0.1) Install it using 'tarantoolctl rocks install http 2.0.1'

How to run

- Create docker-compose.zipkin.yml

```
---
version: '3.5'



============================================================================
Initially got from https://github.com/openzipkin/docker-zipkin/blob/master/docker-compose.yml
============================================================================
```

(continues on next page)

(продолжение с предыдущей страницы)

```
services:
  storage:
    image: openzipkin/zipkin-mysql
    container_name: mysql
    networks:
      - zipkin
    ports:
      - 3306:3306

  # The zipkin process services the UI, and also exposes a POST endpoint that
  # instrumentation can send trace data to. Scribe is disabled by default.
  zipkin:
    image: openzipkin/zipkin
    container_name: zipkin
    networks:
      - zipkin
    # Environment settings are defined here https://github.com/openzipkin/zipkin/tree/1.19.0/zipkin-server
↪#environment-variables
    environment:
      - STORAGE_TYPE=mysql
      # Point the zipkin at the storage backend
      - MYSQL_HOST=mysql
      # Enable debug logging
      - JAVA_OPTS=-Dlogging.level.zipkin=DEBUG -Dlogging.level.zipkin2=DEBUG
    ports:
      # Port used for the Zipkin UI and HTTP Api
      - 9411:9411
    depends_on:
      - storage

  # Adds a cron to process spans since midnight every hour, and all spans each day
  # This data is served by http://192.168.99.100:8080/dependency
  #
  # For more details, see https://github.com/openzipkin/docker-zipkin-dependencies
  dependencies:
    image: openzipkin/zipkin-dependencies
    container_name: dependencies
    entrypoint: crond -f
    networks:
      - zipkin
    environment:
      - STORAGE_TYPE=mysql
      - MYSQL_HOST=mysql
      # Add the baked-in username and password for the zipkin-mysql image
      - MYSQL_USER=zipkin
      - MYSQL_PASS=zipkin
      # Dependency processing logs
      - ZIPKIN_LOG_LEVEL=DEBUG
    depends_on:
      - storage

networks:
  zipkin:
```

- Start Zipkin docker-compose -f docker-compose.zipkin.yml up

- Run mock applications from separate consoles: consumer, formatter and client

Formatter HTTP server

```
#!/usr/bin/env tarantool

local http_server = require('http.server')
local http_router = require('http.router')
local fiber = require('fiber')
local log = require('log')
local zipkin = require('zipkin.tracer')
local opentracing = require('opentracing')

local app = {}

local Sampler = {
    sample = function() return true end,
}

local HOST = '0.0.0.0'
local PORT = '33302'

local function handler(req)

    -- Extract content from request's http headers
    local ctx, err = opentracing.http_extract(req:headers())
    if ctx == nil then
        local resp = req:render({ text = err })
        resp.status = 400
        return resp
    end

    local hello_to = req:query_param('helloto')
    -- Start new child span
    local span = opentracing.start_span_from_context(ctx, 'format_string')
    -- Set service type
    span:set_component('formatter')
    span:set_server_kind()
    span:set_http_method(req:method())
    span:set_http_path(req:path())
    local greeting = span:get_baggage_item('greeting')
    local result = ('%s, %s!'):format(greeting, hello_to)
    local resp = req:render({ text = result })

    -- Simulate long request processing
    fiber.sleep(2)
    span:log_kv({
        event = 'String format',
        value = result,
    })
    resp.status = 200
    span:set_http_status_code(resp.status)
    span:finish()
    return resp
end

function app.init()
    -- Initialize zipkin client that will be send spans every 5 seconds
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    local tracer = zipkin.new({
        base_url = 'localhost:9411/api/v2/spans',
        api_method = 'POST',
        report_interval = 5,
        on_error = function(err) log.error(err) end,
    }, Sampler)
    opentracing.set_global_tracer(tracer)

    local httpd = http_server.new(HOST, PORT)
    local router = http_router.new()
        :route({ path = '/format', method = 'GET' }, handler)
    httpd:set_router(router)
    httpd:start()
end

app.init()

return app
```

Publisher HTTP server

```
#!/usr/bin/env tarantool

local http_server = require('http.server')
local http_router = require('http.router')
local fiber = require('fiber')
local log = require('log')
local zipkin = require('zipkin.tracer')
local opentracing = require('opentracing')

local app = {}

local Sampler = {
    sample = function() return true end,
}

local HOST = '0.0.0.0'
local PORT = '33303'

local function handler(req)
    local ctx, err = opentracing.http_extract(req:headers())

    if ctx == nil then
        local resp = req:render({ text = err })
        resp.status = 400
        return resp
    end

    local hello = req:query_param('hello')
    local span = opentracing.start_span_from_context(ctx, 'print_string')
    span:set_component('publisher')
    span:set_server_kind()
    span:set_http_method(req:method())
    span:set_http_path(req:path())

    -- Simulate long request processing
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    fiber.sleep(3)

    io.write(hello, '\n')
    local resp = req:render({text = '' })
    resp.status = 200
    span:set_http_status_code(resp.status)
    span:finish()
    return resp
end

function app.init()
    local tracer = zipkin.new({
        base_url = 'localhost:9411/api/v2/spans',
        api_method = 'POST',
        report_interval = 5,
        on_error = function(err) log.error(err) end,
    }, Sampler)
    opentracing.set_global_tracer(tracer)

    local httpd = http_server.new(HOST, PORT)
    local router = http_router.new()
        :route({ path = '/print', method = 'GET' }, handler)
    httpd:set_router(router)
end

app.init()

return app
```

Client

```
#!/usr/bin/env tarantool

local http_client = require('http.client')
local json = require('json')
local log = require('log')
local fiber = require('fiber')
local zipkin = require('zipkin.tracer')
local opentracing = require('opentracing')

local app = {}

-- Process all requests
local Sampler = {
    sample = function() return true end,
}

local function url_encode(str)
    local res = string.gsub(str, '[^a-zA-Z0-9_]',
        function(c)
            return string.format('%%%02X', string.byte(c))
        end
    )
    return res
end
```

(continues on next page)

(продолжение с предыдущей страницы)

```
-- Client part to formatter
local formatter_url = 'http://localhost:33302/format'
local function format_string(ctx, str)
    local span = opentracing.start_span_from_context(ctx, 'format_string')
    local httpc = http_client.new()
    span:set_component('client')
    span:set_client_kind()
    span:set_http_method('GET')
    span:set_http_url(formatter_url)

    -- Use http headers as carrier
    local headers = {
        ['content-type'] = 'application/json'
    }
    opentracing.http_inject(span:context(), headers)

    -- Simulate problems with network
    fiber.sleep(1)
    local resp = httpc:get(formatter_url .. '?helloto=' .. url_encode(str),
            { headers = headers })
    fiber.sleep(1)

    span:set_http_status_code(resp.status)
    if resp.status ~= 200 then
        error('Format string error: ' .. json.encode(resp))
    end
    local result = resp.body
    -- Log result
    span:log_kv({
        event = 'String format',
        value = result
    })
    span:finish()
    return result
end

-- Client part to publisher
local printer_url = 'http://localhost:33303/print'
local function print_string(ctx, str)
    local span = opentracing.start_span_from_context(ctx, 'print_string')
    local httpc = http_client.new()
    span:set_component('client')
    span:set_client_kind()
    span:set_http_method('GET')
    span:set_http_url(printer_url)

    local headers = {
        ['content-type'] = 'application/json'
    }
    opentracing.http_inject(span:context(), headers)

    -- Simulate problems with network
    fiber.sleep(1)
    local resp = httpc:get(printer_url .. '?hello=' .. url_encode(str),
            { headers = headers })
    fiber.sleep(1)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    span:set_http_status_code(resp.status)
    if resp.status ~= 200 then
        error('Print string error: ' .. json.encode(resp))
    end
    span:finish()
end

function app.init()
    -- Initialize Zipkin tracer
    local tracer = zipkin.new({
        base_url = 'localhost:9411/api/v2/spans',
        api_method = 'POST',
        report_interval = 0,
        on_error = function(err) log.error(err) end,
    }, Sampler)
    opentracing.set_global_tracer(tracer)

    -- Initialize root span
    local span = opentracing.start_span('Hello-world')

    local hello_to = 'world'
    local greeting = 'my greeting'
    span:set_component('client')
    -- Set service type
    span:set_client_kind()
    -- Set tag with metadata
    span:set_tag('hello-to', hello_to)
    -- Add data to baggage
    span:set_baggage_item('greeting', greeting)

    local ctx = span:context()
    local formatted_string = format_string(ctx, hello_to)
    print_string(ctx, formatted_string)
    span:finish()
end

app.init()

os.exit(0)
```

- Check results on http://localhost:9411/zipkin

Tarantool Cartridge

Opentracing could be used with Tarantool Cartridge.

This example is pretty similar to previous. We will have several roles that communicate via rpc_call.

Basics

Before describing let's define some restrictions of «tracing in Tarantool». Remote communications between tarantools are made using net.box module. It allows to send only primitive types (except functions) and doesn't have containers for request context (as headers in HTTP). Then you should transfer span context explicitly as raw table as additional argument in your function.

```
-- Create span
local span = opentracing.start_span('span')

-- Create context carrier
local rpc_context = {}
opentracing.map_inject(span:context(), rpc_context)

-- Pass context explicitly as additional argument
local res, err = cartridge.rpc_call('role', 'fun', {rpc_context, ...})
```

Using inside roles

The logic of tracing fits into a separate role. Let's define it:

```
local opentracing = require('opentracing')
local zipkin = require('zipkin.tracer')

local log = require('log')

-- config = {
--     base_url = 'localhost:9411/api/v2/spans',
--     api_method = 'POST',
--     report_interval = 5,    -- in seconds
--     spans_limit = 1e4,      -- amount of spans that could be stored locally
-- }

local function apply_config(config)
    -- sample all requests
    local sampler = { sample = function() return true end }

    local tracer = zipkin.new({
        base_url = config.base_url,
        api_method = config.api_method,
        report_interval = config.report_interval,
        spans_limit = config.spans_limit,
        on_error = function(err) log.error('zipkin error: %s', err) end,
    }, sampler)

    -- Setup global tracer for easy access from another modules
    opentracing.set_global_tracer(tracer)

    return true
end

return {
    role_name = 'tracing',
    apply_config = apply_config,
    dependencies = {},
}
```

Then you can use this role as dependency:

```
local opentracing = require('opentracing')
local membership = require('membership')
```

(continues on next page)

(продолжение с предыдущей страницы)

```
local role_name = 'formatter'
local template = 'Hello, %s'

local service_uri = ('%s@%s'):format(role_name, membership.myself().uri)

local function format(ctx, input)
    -- Extract tracing context from request context
    local context = opentracing.map_extract(ctx)
    local span = opentracing.start_span_from_context(context, 'format')
    span:set_component(service_uri)

    local result, err
    if input == '' then
        err = 'Empty string'
        span:set_error(err)
    else
        result = template:format(input)
    end

    span:finish()

    return result, err
end

local function init(_)
    return true
end

local function stop()
end

local function validate_config(_, _)
    return true
end

local function apply_config(_, _)
    return true
end

return {
    format = format,

    role_name = role_name,
    init = init,
    stop = stop,
    validate_config = validate_config,
    apply_config = apply_config,
    -- Setup tracing role as dependency
    dependencies = {'app.roles.tracing'},
}
```

6.4 odbc

6.4.1 ODBC connector for Tarantool

Based on unixODBC

Examples

Use a single connection

```
local odbc = require 'odbc'
local yaml = require 'yaml'

local env, err = odbc.create_env()
local conn, err = env:connect("DSN=odbc_test")

local result, err = conn:execute("SELECT 1 as a, 2 as b")
print(yaml.encode(result))

conn:close()
```

Use ODBC transactions

```
local odbc = require 'odbc'
local yaml = require 'yaml'

local env, err = odbc.create_env()
local conn, err = env:connect("DSN=odbc_test")
conn:execute("CREATE TABLE t(id INT, value TEXT)")

conn:set_autocommit(false)
conn:execute("INSERT INTO t VALUES (1, 'one')")
conn:execute("INSERT INTO t VALUES (2, 'two')")
local result, err = conn:execute("SELECT * FROM t")
print(yaml.encode(result))

conn:commit()
conn:close()
```

Use connection pool

```
local odbc = require 'odbc'
local yaml = require 'yaml'

local pool, err = odbc.create_pool({
    size = 5
})
local _, err = pool:connect()

local conn = pool:get()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
local res, err = conn:execute("SELECT 1 as a, 2 as b")
print(yaml.encode(res))

pool:put(conn)

pool:close()
```

Use connection pool for ad-hoc requests

Pool implements :execute(), :drivers(), :datasources() and :tables() methods that acquire and release a connection object for you.

```
local odbc = require 'odbc'
local yaml = require 'yaml'

local pool, err = odbc.create_pool({
    size = 5
})
local _, err = pool:connect()

local res, err = pool:execute("SELECT 1 as a, 2 as b")
print(yaml.encode(res))

pool:close()
```

API Reference

Creates ODBC environment.

Options

date_as_table - configures behaviour of odbc package of how to deal with dates. If date_as_table is false then dates are represented in the default way for the driver (e.g. represents as strings for PostgreSQL). If date_as_table is true then dates are represented as tables compatible with os.date(„*t“).

Creates a connection pool.

Options

1. All options for odbc.create_env
2. dsn - connection string
3. size - number of connections in the pool

Environment methods

env:connect(dsn)

Parameters

1. dsn - Connection string (documentation).

Connection methods

conn:execute(query, params)

Executes an arbitrary SQL query

Parameters

1. query - SQL query
2. param - table with parameters binding

Example

```
conn:execute("SELECT * FROM t WHERE id > ? and value = ?", {1, "two"})
```

conn:set_autocommit(flag)

Sets autocommit of connection to a specified value. Used to achieve transaction behaviour. Set autocommit to false to execute multiple statements in one transactions.

Parameters

1. flag - true/false.

conn:commit()

Commit a transaction

conn:rollback()

Rollback a transaction

conn:set_isolation(level)

Sets isolation level of a transaction. Cannot be run in an active transaction.

Parameters

1. level - isolation level. One of the values defined in the odbc.isolation table.

Isolation levels

1. odbc.isolation.READ_UNCOMMITTED
2. odbc.isolation.READ_COMMITTED
3. odbc.isolation.REPEATABLE_READ
4. odbc.isolation.SERIALIZABLE

conn:is_connected()

Returns true if connection is active.

conn:state()

Returns an internal state of a connection.

conn:close()

Disconnect and close the connection.

conn:drivers()

Returns a list of drivers available in the system (contents of odbcinst.ini file).

Example

conn:datasources()

Returns a list of data sources available in the system (contents of odbc.ini file).

Example

conn:tables()

Returns a list of tables of a connected data source.

Example

conn:cursor(query, params)

Creates a cursor object for the specified query.

Parameters

1. query - SQL query
2. param - table with parameters binding

conn:prepare(query)

Create object and prepare query.

Parameters

1. query - SQL query

Cursor methods

cursor:fetchrow()

Fetch one row from the data frame and return as a single table value.

Example

cursor:fetch(n)

Fetch multiple rows.

Parameters 1. n - number of rows to fetch

Example

cursor:fetchall()

Fetch all available rows in the data frame.

cursor:is_open()

Returns true if cursor is open.

cursor:close()

Close cursor discarding available data.

Prepare methods

prepare:execute()

Execute prepared SQL query

Parameters 1. param - table with parameters binding

Example

prepare:is_open()

Returns true if prepare is open.

prepare:close()

Close prepare discarding prepared query.

Pool methods

pool:connect()

Connect to all size connections.

pool:acquire()

Acquire a connection. The connection must be either returned to pool with pool:release() method or closed.

pool:release(conn)

Release the connection.

pool:available()

Returns the number of available connections.

pool:close()

Close pool and all underlying connections.

pool:execute(query, params)

Acquires a connection, executes query on it and releases the connection.

1. query - SQL query
2. param - table with parameters binding

pool:tables()

Acquires a connection, executes tables() on it and releases.

pool:drivers()

Acquires a connection, executes drivers() on it and releases.

pool:datasources()

Acquires a connection, executes datasources() on it and releases.

Installation

Prerequisites:

1. unixODBC driver
2. Driver for the database of your choice. Currently this module is tested only with PostgreSQL, MySQL and MS SQL Server databases.
3. Datasource for the database in odbc.ini file

PostgreSQL

Linux (Ubuntu)

```
$ sudo apt-get install odbc-postgresql
```

Add to file /etc/odbcinst.ini:

```
[PostgreSQL ANSI]
Description=PostgreSQL ODBC driver (ANSI version)
Driver=psqlodbca.so
Setup=libodbcpsqlS.so
Debug=0
CommLog=1
UsageCount=1
```

Add to file /etc/odbc.ini:

```
[<dsn_name>]
Description=PostgreSQL
Driver=PostgreSQL ANSI
Trace=No
TraceFile=/tmp/psqlodbc.log
Database=<Database>
Servername=localhost
username=<username>
password=<password>
port=
readonly=no
rowversioning=no
showsystemtables=no
showoidcolumn=no
fakeoidindex=no
connsettings=
```

MacOS

Use brew to install:

```
$ brew install psqlodbc
```

/usr/local/etc/odbcinst.ini contents:

```
[PostgreSQL ANSI]
Description=PostgreSQL ODBC driver (ANSI version)
Driver=/usr/local/lib/psqlodbca.so
Debug=0
CommLog=1
UsageCount=1
```

/usr/local/etc/odbc.ini contents:

```
[<dsn_nam>]
Description=PostgreSQL
```

(continues on next page)

(продолжение с предыдущей страницы)

```
Driver=PostgreSQL ANSI
Trace=No
TraceFile=/tmp/psqlodbc.log
Database=<database>
Servername=<host>
UserName=<username>
Password=<password>
ReadOnly=No
RowVersioning=No
ShowSystemTables=No
ShowOidColumn=No
FakeOidIndex=No
```

MSSQL

Linux

Please follow to the official installation guide.

/etc/odbcinst.ini contents:

```
[ODBC Driver 17 for SQL Server]
Description=Microsoft ODBC Driver 17 for SQL Server
Driver=/opt/microsoft/msodbcsql17/lib64/libmsodbcsql-17.2.so.0.1
UsageCount=1
```

/etc/odbc.ini contents:

```
[<dsn_name>]
Driver=ODBC Driver 17 for SQL Server
Database=<Database>
Server=localhost
```

MacOS

For El Capitan, Sierra and High Sierra use brew to install:

```
$ brew tap microsoft/mssql-release https://github.com/Microsoft/homebrew-mssql-release
$ brew install --no-sandbox msodbcsql17 mssql-tools
```

For El Capitan and Sierra use brew to install:

```
$ brew tap microsoft/mssql-release https://github.com/Microsoft/homebrew-mssql-release
$ brew install --no-sandbox msodbcsql@13.1.9.2 mssql-tools@14.0.6.0
```

Examples below are fair to msodbcsql 13

/usr/local/etc/odbcinst.ini contents:

```
[ODBC Driver 13 for SQL Server]
Description=Microsoft ODBC Driver 13 for SQL Server
Driver=/usr/local/lib/libmsodbcsql.13.dylib
UsageCount=1
```

/usr/local/etc/odbc.ini contents:

```
[<dsn_name>]
Description=SQL Server
Driver=ODBC Driver 13 for SQL Server
Server=<host>,<port>
```

FYI:

Uid, Pwd etc are placed into connstring

Example

MySQL

Linux

Please follow to the official installation guide.

MacOS

Download and install:

1. http://www.iodbc.org/dataspace/doc/iodbc/wiki/iodbcWiki/Downloads
2. https://dev.mysql.com/downloads/connector/odbc/

Add to file /usr/local/etc/odbcinst.ini:

```
[MySQL ODBC 8.0 Driver]
Driver=/usr/local/mysql-connector-odbc-8.0.12-macos10.13-x86-64bit/lib/libmyodbc8a.so
UsageCount=1
```

Add to file /usr/local/etc/odbc.ini:

```
[<dsn>]
Driver = MySQL ODBC 8.0 Driver
Server = <host>
PORT = <port>
```

FYI:

USER, DATABASE etc are placed into connstring

Example

Sybase ASE

MacOS

Run brew install freetds

Add to file /usr/local/etc/freetds.conf

```
[sybase]
host = localhost
port = 8000
tds version = auto
```

Add to file /usr/local/etc/odbcinst.ini:

```
[Sybase Driver]
Driver=/usr/local/lib/libtdsodbc.so
UsageCount=1
```

Add to file /usr/local/etc/odbc.ini:

```
[default]
Driver=/usr/local/lib/libtdsodbc.so
Port=8000

[sybase]
Driver=Sybase Driver
Description=Sybase ASE
DataSource=<datasource>
ServerName=sybase
Database=<database>
```

Example

References

1. Tarantool - in-memory database and application server.
2. PostgreSQL ODBC
3. MS SQL Server ODBC
4. MySQL ODBC

6.5 oracle

6.5.1 Oracle connector

The oracle package exposes some functionality of OCI. With this package, Tarantool Lua applications can send and receive data over Oracle protocol.

The advantage of integrating oracle with Tarantool, which is an application server plus a DBMS, is that anyone can handle all of the tasks associated with Oracle (control, manipulation, storage, access) with the same high-level language (Lua) and with minimal delay.

Table of contents

- API reference

Prerequisites

- An operating system with developer tools including cmake, C compiler with gnu99 support, git and Lua.
- Tarantool 1.6.5+ with header files (tarantool and tarantool-dev packages).
- Oracle OCI 10.0+ header files and dynamic libs.

Automatic build

Important: Builder requires Oracle Instant Client zip archives. You need to download them from Oracle into the source tree:

```
curl -O https://raw.githubusercontent.com/bumpx/oracle-instantclient/master/instantclient-basic-linux.x64-12.2.0.
↪1.0.zip
curl -O https://raw.githubusercontent.com/bumpx/oracle-instantclient/master/instantclient-sdk-linux.x64-12.2.0.1.
↪0.zip
sha256sum -c instantclient.sha256sum
```

To build a complete oracle package, you need to run package.sh script first (depends on docker). Packages will be available in build/ directory. Example:

```
wget <oracle-client.rpm>
wget <oracle-devel.rpm>
$./package.sh
...
done
$ls -1 build
oracle-instantclient12.2-basic-12.2.0.1.0-1.x86_64.rpm
oracle-instantclient12.2-devel-12.2.0.1.0-1.x86_64.rpm
tarantool-oracle-1.0.0.0-1.el7.centos.src.rpm
tarantool-oracle-1.0.0.0-1.el7.centos.x86_64.rpm
tarantool-oracle-debuginfo-1.0.0.0-1.el7.centos.x86_64.rpm
```

After that you can install oracle package on the target machine:

```
rpm -Uvh tarantool-oracle-1.0.0.0-1.el7.centos.x86_64.rpm
```

Getting started

Start Tarantool in the interactive mode. Execute these requests:

```
tarantool> oracle = require('oracle')
tarantool> env, errmsg = oracle.new()
tarantool> if not env then error("Failed to create environment: "..errmsg) end
tarantool> c, errmsg = env:connect({username='system', password='oracle', db='localhost:1511/myspace'})
tarantool> if not c then error("Failed to connect: "..errmsg) end
tarantool> c:exec('CREATE TABLE test(i int, s varchar(20))')
tarantool> c:exec('INSERT INTO test(i, s) VALUES(:I, :S)', {I=1, S='Hello!'})
tarantool> rc, result_set = c:exec('SELECT * FROM test')
```

If all goes well, you should see:

```
tarantool> result_set[1][2] -- 'Hello!'
```

This means that you have successfully installed tarantool/oracle and successfully executed an instruction that brought data from an Oracle database.

API reference

function new([opts])

Create Oracle connection environment.

Accepts parameters:

- [optional] table of options:
  - charset - client-side character and national character set. If not set or set improperly, NLS_LANG setting is used.

Returns: * env - environment object in case of success, nil otherwise, * err [OPTIONAL] - error string in case of error.

function env:connect(credentials [, additional options])

Connect to the Oracle database.

Accepts parameters: * credentials (table):

- username (str) - user login,
- password (str) - user password,
- db (str) - database URL.
- additional options (as table):
  - prefetch_count (int) - prefetch row count amount from Oracle,
  - prefetch_size (int) - memory limit for prefetching (in MB),
  - batch_size (int) - the size of each SELECT loop batch on exec() and cursor:fetchall().

Returns: * conn - connection object in case of success, nil otherwise, * err [OPTIONAL] - error string or table structure in case of error.

function env:version()

Get version string.

function conn:exec(sql [, args])

Execute an operation.

Accepts parameters:

- sql - SQL statement,
- [optional] statement arguments.

Returns:

- rc - result code (0 - Success, 1 - Error)
- result_set - result table, err - table with error (see below) in case of error
- row_count - number of rows in result_set
- err [OPTIONAL] - table with error in case of warning from Oracle

Examples:

```
-- Schema - create table(a int, b varchar(25), c number)
conn:exec("insert into table(a, b, c) values(:A, :B, :C)", {A=1, B='string', C=0.1})
```

```
-- Schema - create table(a int, b varchar(25), c number)
rc, res = conn:exec("SELECT a, b, c FROM table")
res[1][1] -- a
res[1][2] -- b
res[1][3] -- c
```

function conn:cursor(sql [, args, opts])

Create cursor to fetch SELECT results.

Accepts parameters:

- sql - SELECT SQL statement,
- [optional] statement arguments,
- [optional] table of options:
  - scrollable - enable cursor scrollable mode (false by default).

Returns:

- cursor - cursor object in case of success, nil otherwise
- err [OPTIONAL] - table with error, same format as exec function one

function conn:close()

Close connection and all associated cursors.

function cursor:fetch_row()

Fetch one row of resulting set.

Returns:

- rc - result code (0 - Success, 1 - Error),
- result_set - result table, err - table with error in case of error,
- row_count - number of rows in result_set.

function cursor:fetch(fetch_size)

Fetch fetch_size rows of resulting set.

Accepts parameters:

- fetch_size - number of rows (positive integer).

Returns:

- rc - result code (0 - Success, 1 - Error),
- result_set - result table, err - table with error in case of error,
- row_count - number of rows in result_set.

function cursor:fetch_all()

Fetch all remaining rows of resulting set.

Returns:

- rc - result code (0 - Success, 1 - Error),
- result_set - result table, err - table with error in case of error,
- row_count - number of rows in result_set.

function cursor:fetch_first(fetch_size)

Scrollable only.

Fetch first fetch_size rows of resulting set.

Accepts parameters:

- fetch_size - number of rows (positive integer).

Returns:

- rc - result code (0 - Success, 1 - Error),
- result_set - result table, err - table with error in case of error,
- row_count - number of rows in result_set.

function cursor:fetch_last()

Scrollable only.

Fetch last row of resulting set.

Returns:

- rc - result code (0 - Success, 1 - Error),
- result_set - result table, err - table with error in case of error,
- row_count - number of rows in result_set.

function cursor:fetch_absolute(fetch_size, offset)

Scrollable only.

Fetch fetch_size rows of resulting set, starting from offset absolute position (including offset row).

Accepts parameters:

- fetch_size - number of rows (positive integer),
- offset - absolute cursor offset (positive integer).

Returns:

- rc - result code (0 - Success, 1 - Error),
- result_set - result table, err - table with error in case of error,
- row_count - number of rows in result_set.

function cursor:fetch_relative(fetch_size, offset)

Scrollable only.

Fetch fetch_size rows of resulting set, starting from current + offset absolute position(including current + offset row).

Accepts parameters:

- fetch_size - number of rows (positive integer),
- offset - relative cursor offset (signed integer).

Returns:

- rc - result code (0 - Success, 1 - Error),
- result_set - result table, err - table with error in case of error,
- row_count - number of rows in result_set.

function cursor:fetch_current(fetch_size)

Scrollable only.

Fetch fetch_size rows of resulting set, starting from current position (including current row).

Accepts parameters:

- fetch_size - number of rows (positive integer).

Returns:

- rc - result code (0 - Success, 1 - Error),
- result_set - result table, err - table with error in case of error,
- row_count - number of rows in result_set.

function cursor:fetch_prior(fetch_size)

Scrollable only.

Fetch fetch_size rows of resulting set, starting from previous row from the current position (including previous row).

Accepts parameters:

- fetch_size - number of rows (positive integer).

Returns:

- rc - result code (0 - Success, 1 - Error),
- result_set - result table, err - table with error in case of error,
- row_count - number of rows in result_set.

function cursor:get_position()

Scrollable only.

Get current cursor position.

Returns:

- rc - result code (0 - Success, 1 - Error)
- position - current cursor position, err - table with error in case of error

function cursor:close()

Closes cursor. After this was executed, cursor is no longer available for fetching results.

function cursor:is_closed()

Returns:

- is_closed - true if closed, false otherwise.

function cursor:ipairs()

Lua 5.1 version of ipairs(cursor) operator.

Example:

```
-- Foo(row) is some function for row processing
for k, row in cursor:ipairs() do
    Foo(row)
end
```

function conn:close()

Close connection and all associated cursors.

Error handling

In case of error returns nil, err where err is table with the next fields:

- type - type of error (1 - Oracle error is occured, 0 - Connector error is occured)
- msg - message with text of error
- code - error code (now defined ONLY for Oracle error codes)

6.5.2 Deploy Oracle in Docker

Description (For Oracle EE v12.2):

1. You should have working Docker and Oracle accounts.
2. instantclient-sqlplus.
3. Go to this page and follow instructions. You need to follow steps 1.a-1.d steps. For short here they are:

- 1.a Log in to Docker Store with your Docker credentials.
- 1.b Search for „oracle database“ and select the Oracle Database Enterprise Edition image.
- 1.c Click through and accept terms if needed.
- 1.d View Setup Instructions.
- 1.d.1 Download image with

```
docker pull store/oracle/database-enterprise:12.2.0.1
```

- 1.d.2 Run container with -P option, it will allocate port to access database outside docker container.

```
docker run -d -it --name OraDBEE -P store/oracle/database-enterprise:12.2.0.1
```

Note: type docker ps to get allocated port, you need PORTS section (or docker port CONTAINER_NAME), also check that status is healthy (if not, repeat 1.d.2 - in my case probability to create working container was 50/50 :P). Assume port is 32771 for farther instructions.

4. Go inside container, create a user, grant all necessary permissions:

```
docker exec -it OraDBEE bash -c "source /home/oracle/.bashrc; sqlplus sys/Oradoc_db1@ORCLCDB as sysdba"
...

SQL> alter session set "_ORACLE_SCRIPT"=true;
SQL> CREATE USER user1 IDENTIFIED BY qwerty123;
SQL> GRANT CONNECT, RESOURCE, DBA TO user1;
```

5. Check connection and access rights outside container for a new user:

```
> sqlplus user1/qwerty123@localhost:32771/ORCLCDB.localdomain
```

Try to create a table, insert some rows, select them, then drop table.

6. Follow Getting started part. Use this user credentials in connect method:

```
tarantool> c, err = ora.connect({username='user1', password='qwerty123', db='localhost:32771/ORCLCDB.
↪localdomain'})
```

Troubleshooting

If Docker can't get Oracle image with

```
docker pull store/oracle/database-enterprise:12.2.0.1
```

try to login with „docker login". If still nothing is happening, go to docker store and check license (Terms of Service).

6.5.3 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

[Unreleased]

Changed

- Removed get_position for non-scrollable cursors

[1.3.0] - 2019-11-28

Added

- Added support for connection environment

Changed

- Direct call of „oracle.connect" is deprecated and will print warnings on call
- Added explicit error return for fiber cancelled errors
- Fixed segmentation fault on fiber cancel in the middle of fetch, cursor and connection create
- Fixed freeze on wrong credentials

[1.2.2] - 2019-09-30

Added

- Added support for scrollable cursors

Changed

- Fixed bug when cursor had no explicit Lua link to connection object and collecting connection object by gc made cursor unusable

[1.2.1] - 2019-09-16

Changed

- Fixed bug when connection close affected other connections cursors
- Fixed bug when cursors remained open on the Oracle side when closed manually

[1.2.0] - 2019-08-22

Added

- Added support for non-scrollable cursors
- Added support for connection caching parameters

[1.1.6] - 2019-07-08

Changed

- Removed memory leak when fiber with connection has been killed
- Fixed unresponsive event loop when fiber with long-running request is killed

[1.1.5] - 2019-05-27

Changed

- Bugfixes and other improvements

[1.1.4] - 2019-04-04

Changed

- Improved error handling
- Fixed pushing double value into Lua stack
- Fixed Lua VM freeze while connecting to Oracle DB
- Various bugfixes

Added

- Added OCI libs as a separate rock with dependency

[1.1.0] - 2019-02-01

Changed

- Bugfixes
- Several stability improvements

Added

- Oracle OCCI switched to 11.2
- Detect and bind params from statement
- Removed autocommit
- Added more data conversions for returning dataset
- Use one method for read and write requests
- exec_once() removed
- Tests added
- CentOS 7 Dockerfile added
- Add support for blob parameters

[1.0.0] - 2017-04-12

Added

- Basic functionality

6.6 space-explorer

6.6.1 Space explorer

Rock for exploring tarantool spaces in cartridge

Installation

- Add „space-explorer == ...“ to rockspec dependencies
- Run tarantoolctl rocks make
- Add require(„space-explorer“).init() after cartridge.cfg call

Example

example-app-scm-1.rockspec

```
package = 'example-app'
version = 'scm-1'
source  = {
    url = '/dev/null',
}
dependencies = {
    'tarantool',
    'lua >= 5.1',
    'luatest == 0.2.0-1',
    'ldecnumber == 1.1.3-1',
    'cartridge == scm-1',
    'space-explorer == scm-1'
}
build = {
    type = 'none';
}
```

init.lua

```
#!/usr/bin/env tarantool

require('strict').on()

local cartridge = require('cartridge')

local ok, err = cartridge.cfg({
    roles = {
        'cartridge.roles.vshard-storage',
        'cartridge.roles.vshard-router',
        'app.roles.api',
        'app.roles.storage',
    },
    cluster_cookie = 'example-app-cluster-cookie',
})

require('space-explorer').init()

assert(ok, tostring(err))
```

Guide

Space explore guide can be seen here.

В данной главе рассматриваются Lua-модули с открытым и закрытым исходным кодом для Tarantool Enterprise, которые включены в дистрибутив в качестве автономного репозитория сторонних библиотек.

6.7 Модули с открытым исходным кодом:

- avro-schema – набор инструментов для схемы Apache Avro;
- cartridge – это высокоуровневый интерфейс управления кластером, который содержит несколько модулей:

  - rpc обеспечивает удаленные вызовы процедур между экземплярами кластера и позволяет ролям, запущенным на некоторых экземплярах, взаимодействовать с другими ролями в других экземплярах.
  - service-registry обеспечивает взаимодействие между ролями и позволяет различным ролям взаимодействовать друг с другом в рамках одного экземпляра.
  - confapplier обеспечивает валидацию и применение конфигурации в масштабе всего кластера с помощью двухфазной фиксации.
  - auth управляет аутентификацией.
  - pool повторно использует соединения net.box в Tarantool.
  - admin обеспечивает функции администрирования.
- cartridge-cli – это интерфейс командной строки для модуля cartridge.
- checks – это модуль контроля типов функциональных аргументов. Эта библиотека объявляет функцию checks() и таблицу checkers, которые позволяют быстро и незаметно проверять параметры, передаваемые в Lua-функцию.
- http – это встроенный HTTP-сервер, который дополняет стандартный HTTP-клиент и требует установки, как описано в разделе по установке.
- icu-date – библиотека форматирования даты и времени для Tarantool, которая основана на библиотеке для работы с Unicode (International Components for Unicode);
- kafka – это полноценная высокопроизводительная библиотека kafka для Tarantool на основе librdkafka;
- ldecnumber – библиотека для десятичной арифметики;
- luacheck is a static analyzer and linter for Lua, preconfigured for Tarantool.
- luarapidxml – быстрый анализатор XML.
- luatest – среда тестирования Tarantool, которая написана на Lua.
- membership создает сеть из нескольких экземпляров Tarantool на основе протокола gossip. Сеть сама контролирует себя, помогает участникам обнаружить всех остальных в группе и получать уведомления об изменениях своего статуса с низкой задержкой. Модуль основан на концепциях из Consul или, точнее, алгоритма SWIM.
- stat – набор полезных метрик для мониторинга.
- vshard – автоматическая система шардинга для горизонтального масштабирования экземпляров СУБД Tarantool.

6.8 Модули с закрытым исходным кодом

- ldap позволяет осуществлять аутентификацию на сервере LDAP и выполнять поиск.
- odbc – коннектор ODBC для Tarantool на основе unixODBC.
- oracle – коннектор Oracle для Lua-приложений, с помощью которого они могут обращаться к базам данных Oracle. Преимущество интеграции Tarantool-Oracle состоит в том, что любой может выполнять задачи по работе с СУБД Oracle (управление, обработка, хранение, доступ) на одном языке высокого уровня (Lua) с минимальной задержкой.
- task – модуль для управления фоновыми задачами в Tarantool-кластере.
- tracing – модуль для отладки проблем с производительностью.

- space-explorer – модуль для просмотра спейсов Tarantool в cartridge.

6.9 Установка и использование модулей

Чтобы использовать модуль, установите следующие элементы:

1. Все необходимые сторонние пакеты программного обеспечения (при необходимости). Список программ см. в требованиях модуля.
2. Сам модуль на каждый экземпляр Tarantool:

```
$ tarantoolctl rocks install <module_name> [<module_version>]
```

Для получения информации об управлении модулями Tarantool см. также другие важные команды tarantoolctl.

ГЛАВА 7

Приложения

7.1 Приложение А. Журнал аудита

Журнал аудита содержит записи о событиях СУБД Tarantool в формате JSON. Доступны следующие журналы событий:

- пройденная/непройденная аутентификация и авторизация пользователя,
- закрытое соединение,
- изменение пароля,
- создание/удаление пользователя/роли,
- включение/отключение пользователя,
- изменение прав пользователя/роли.

7.1.1 Структура журнала

Ключ	Тип	Описание	Пример
type	строка	тип события	<“access_denied”>
type_id	число	идентификатор события	<8>
description	строка	описание события	<“Authentication failed”>
time	строка	время события	<“YYYY-MM-DDTHH:MM:SS.03f[+|-]GMT”>
peer	строка	удаленный клиент	<“ip:port”>
user	строка	пользователь	<“user”>
param	строка	параметры события	см. ниже

7.1.2 Описание события

Событие	Ключ	Параметры
авторизация пользователя пройдена	auth_ok	{“name”: “user”}
авторизация пользователя не пройдена	auth_fail	{“name”: “user”}
пользователь вышел из системы или завершил сеанс	disconnect	
неудачные попытки доступа к конфиденциальным данным (личные записи, данные, геолокация и т. д.)	access_denied	{“name”: “obj_name”, “obj_type“: “space”, “access_type”: “read”}
создание пользователя	user_create	{“name”: “user”}
удаление пользователя	user_drop	{“name”: “user”}
отключение пользователя	user_disable	{“name”: “user”}
включение пользователя	user_enable	{“name”: “user”}
выдача (изменение) прав (роли, профили и т.д.) пользователю	user_priv	{“name”: “user”} “obj_name”: “obj_name”, “obj_type”: “space”, “old_priv”: “”, “new_priv”: “read,write”}
сброс пароля пользователя (должен быть указан пользователь, который вносит изменения)	password_change	{“name”: “user”}
создание роли	role_create	{“name”: “role”}
выдача (изменение) прав для роли	role_priv	{“name”: “role”} “obj_name”: “obj_name”, “obj_type”: “space”, “old_priv”: “”, “new_priv”: “read,write”}

7.2 Приложение B. Важные параметры Tarantool

- box.info
- box.info.replication
- box.info.memory
- box.stat
- box.stat.net
- box.slab.info
- box.slab.stats

Для получения подробной информации см. справочник по модулю ‘box‘ в основной документации по Tarantool.

7.3 Приложение C. Метрики системы мониторинга

Параметр	Описание	Тип по SNMP	Единицы измерения	Предел
Version	Версия Tarantool	DisplayString		
IsAlive	Показатель доступности экземпляра	Integer (список)		0 - недоступен 1 - доступен
MemoryLua	объем памяти, занятый Lua	Gauge32	МБайт	900
MemoryData	объем, используемый для хранения данных	Gauge32	МБайт	значение задается вручную
MemoryNet	объем, используемый для сетевого ввода-выводы	Gauge32	МБайт	1024
MemoryIndex	объем, используемый для хранения индексов	Gauge32	МБайт	значение задается вручную
MemoryCache	объем, используемый для хранения кэша (только для vinyl)	Gauge32	МБайт	
ReplicationLag	время задержки с момента последней синхронизации (максимальное значение, если есть несколько файберов)	Integer32	сек.	5
FiberCount	количество файберов	Gauge32	шт.	1000

Продолжается на следующей странице

Таблица 1 – продолжение с предыдущей страницы

Параметр	Описание	Тип по SNMP	Единицы измерения	Предел
CurrentTime	текущее время в секундах, с 1 января 1970г.	Unsigned32	временная отметка Unix в сек.	
StorageStatus	статус набора реплик	Integer	список	> 1
StorageAlerts	количество	Gauge32	шт.	$>= 1$
StorageTotalBkts	общее количество сегментов в хранилище	Gauge32	шт.	< 0
StorageActiveBkts	количество сегментов в статусе ACTIVE	Gauge32	шт.	< 0
StorageGarbageBkts	количество сегментов в статусе GARBAGE	Gauge32	шт.	< 0
StorageReceivingBkts	количество сегментов в статусе RECEIVING	Gauge32	шт.	< 0
StorageSendingBkts	количество сегментов в статусе SENDING	Gauge32	шт.	< 0
RouterStatus	статус роутера	Integer	список	> 1
RouterAlerts	количество предупреждений для роутера	Gauge32	шт.	$>= 1$
RouterKnownBkts	количество сегментов в известных наборах реплик	Gauge32	шт.	< 0
RouterUnknownBkts	количество реплик, неизвестных роутеру	Gauge32	шт.	< 0
RequestCount	общее количество запросов	Counter64	шт.	
InsertCount	общее количество запросов вставки	Counter64	шт.	
DeleteCount	общее количество запросов на удаление	Counter64	шт.	
ReplaceCount	общее количество запросов замены	Counter64	шт.	
UpdateCount	общее количество запросов на обновление	Counter64	шт.	
SelectCount	общее количество запросов выборки	Counter64	шт.	
EvalCount	количество вызовов через Eval	Counter64	шт.	

Продолжается на следующей странице

Таблица 1 – продолжение с предыдущей страницы

Параметр	Описание	Тип по SNMP	Единицы измерения	Предел
CallCount	количество вызовов через call	Counter64	шт.	
ErrorCount	количество ошибок в Tarantool	Counter64	шт.	
AuthCount	количество завершенных операций по аутентификации	Counter64	шт.	

7.4 Приложение D. Устаревшие функции

Прекращена поддержка ZooKeeper и orchestrator. Тем не менее, при необходимости их можно использовать.

В следующих разделах описаны соответствующие функциональные возможности.

7.4.1 Управление кластером по API

Для управления кластером используйте оркестратор orchestrator, включенный в комплект поставки. orchestrator использует ZooKeeper для хранения и передачи конфигурации. Для управления кластером в orchestrator есть REST API. Изменение конфигураций в ZooKeeper осуществляется в результате вызова API-функций из orchestrator, что, в свою очередь, приводит к изменениям конфигурации узлов Tarantool.

Мы рекомендуем использовать интерфейс командной строки curl для вызова API-функций из orchestrator.

В следующем примере показано, как зарегистрировать новую зону доступности (DC):

```
$ curl -X POST http://HOST:PORT/api/v1/zone \
   -d '{
  "name": "Caucasian Boulevard"
  }'
```

Чтобы проверить, была ли выполнена регистрация DC, попробуйте использовать следующую команду. Результатом будет список всех зарегистрированных узлов в формате JSON:

```
$ curl http://HOST:PORT/api/v1/zone| python -m json.tool
```

Чтобы применить новую конфигурацию непосредственно к узлам Tarantool, увеличьте номер версии конфигурации после вызова API-функции. Для этого используйте запрос POST к /api/v1/version:

```
$ curl -X POST http://HOST:PORT/api/v1/version
```

В целом, чтобы обновить конфигурацию кластера:

1. Вызовите метод POST/PUT в orchestrator. В результате будут обновлены узлы в ZooKeeper, а также будет инициировано последующее обновление узлов Tarantool.
2. Обновите версию конфигурации, используя запрос POST к /api/v1/version. В результате конфигурация применится к узлам Tarantool.

Для получения дополнительной информации об API оркестратора см. Приложение E.

7.4.2 Настройка геодублирования

По логике узлы кластера могут относиться к некоторой зоне доступности. Физически же зона доступности – это отдельный DC или стойка внутри DC. Можно задать матрицу весов (расстояний) для зон доступности.

Новые зоны добавляются путем вызова соответствующего метода API оркестратора.

По умолчанию матрица весов (расстояний) для зон не настроена, а геодублирование для таких конфигураций работает следующим образом:

- Данные всегда записываются на мастер.
- Если мастер доступен, то он используется для чтения.
- Если мастер недоступен, то любая доступная реплика используется для чтения.

Когда вы задаете матрицу весов (расстояний), вызывая /api/v1/zones/weights, система автоматического масштабирования СУБД Tarantool находит реплику, которая ближе всех к указанному роутеру по весам, и начинает использовать эту реплику для чтения. Если эта реплика недоступна, то выбирается следующая ближайшая реплика с учетом расстояний, указанных в конфигурации.

7.5 Приложение E. Справочник по Orchestrator API

7.5.1 Configuring the zones

- POST /api/v1/zone
- GET /api/v1/zone/
- PUT /api/v1/zone/
- DELETE /api/v1/zone/

POST /api/v1/zone

Создание новой зоны.

Запрос

```
{
"name": "zone 1"
}
```

Ответ

```
{
"error": {
    "code": 0,
    "message": "ok"
},
"data": {
    "id": 2,
    "name": "zone 2"
},
"status": true
}
```

Возможные ошибки

- zone_exists – указанная зона уже существует

GET /api/v1/zone/{zone_id: optional}
Возврат информации об указанной зоне или обо всех зонах.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": [
      {
         "id": 1,
         "name": "zone 11"
      },
      {
         "id": 2,
         "name": "zone 2"
      }
   ],
   "status": true
}
```


Возможные ошибки

- zone_not_found – указанная зона не обнаружена

PUT /api/v1/zone/{zone_id}
Обновление информации о зоне.

Тело


```
{
   "name": "zone 22"
}
```


Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": {},
   "status": true
}
```


Возможные ошибки

- zone_not_found – указанная зона не обнаружена

DELETE /api/v1/zone/{zone_id}
Удаление зоны, если в ней нет узлов.

Ответ


```
{
   "error": {
```


(continues on next page)

(продолжение с предыдущей страницы)


```
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```


Возможные ошибки

- zone_not_found – указанная зона не обнаружена
- zone_in_use – в указанной зоне есть хотя бы один узел

7.5.2 Настройка веса зоны

- GET /api/v1/zones/weights
- POST /api/v1/zones/weights

POST /api/v1/zones/weights

Конфигурация веса зоны.

Тело


```
{
    "weights": {
        "1": {
            "2": 10,
            "3": 11
        },
        "2": {
            "1": 10,
            "3": 12
        },
        "3": {
            "1": 11,
            "2": 12
        }
    }
}
```


Ответ


```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```


Возможные ошибки

- zones_weights_error – ошибка конфигурации

GET /api/v1/zones/weights

Возврат конфигурации веса зоны.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": {
      "1": {
         "2": 10,
         "3": 11
      },
      "2": {
         "1": 10,
         "3": 12
      },
      "3": {
         "1": 11,
         "2": 12
      }
   },
   "status": true
}
```


Возможные ошибки

- zone_not_found – указанная зона не обнаружена

7.5.3 Настройка реестра

- GET /api/v1/registry/nodes/new
- POST /api/v1/registry/node
- PUT /api/v1/registry/node/
- GET /api/v1/registry/node/
- DELETE /api/v1/registry/node/

GET /api/v1/registry/nodes/new
Возврат всех обнаруженных узлов.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": [
      {
         "uuid": "uuid-2",
         "hostname": "tnt2.public.i",
         "name": "tnt2"
      }
   ],
   "status": true
}
```

POST /api/v1/registry/node
: Регистрация обнаруженного узла.

  Тело

```
{
   "zone_id": 1,
   "uuid": "uuid-2",
   "uri": "tnt2.public.i:3301",
   "user": "user1:pass1",
   "repl_user": "repl_user1:repl_pass1",
   "cfg": {
      "listen": "0.0.0.0:3301"
   }
}
```

  Ответ

```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": {},
   "status": true
}
```

  Возможные ошибки

  - node_already_registered – указанный узел уже зарегистрирован
  - zone_not_found – указанная зона не обнаружена
  - node_not_discovered – указанный узел не обнаружен

PUT /api/v1/registry/node/{node_uuid}
: Обновление параметров зарегистрированного узла.

  Тело

  Передача только обновляемых параметров.

```
{
   "zone_id": 1,
   "repl_user": "repl_user2:repl_pass2",
   "cfg": {
      "listen": "0.0.0.0:3301",
      "memtx_memory": 100000
   }
}
```

  Ответ

```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": {},
   "status": true
}
```

Возможные ошибки

- node_not_registered – указанный узел не зарегистрирован

GET /api/v1/registry/node/{node_uuid: optional}
Возврат информации об узлах в кластере. Если передать node_uuid, вернется информация только по данному узлу.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": {
      "uuid-1": {
         "user": "user1:pass1",
         "hostname": "tnt1.public.i",
         "repl_user": "repl_user2:repl_pass2",
         "uri": "tnt1.public.i:3301",
         "zone_id": 1,
         "name": "tnt1",
         "cfg": {
            "listen": "0.0.0.0:3301",
            "memtx_memory": 100000
         },
         "zone": 1
      },
      "uuid-2": {
         "user": "user1:pass1",
         "hostname": "tnt2.public.i",
         "name": "tnt2",
         "uri": "tnt2.public.i:3301",
         "repl_user": "repl_user1:repl_pass1",
         "cfg": {
            "listen": "0.0.0.0:3301"
         },
         "zone": 1
      }
   },
   "status": true
}
```


Возможные ошибки

- node_not_registered – указанный узел не зарегистрирован

DELETE /api/v1/registry/node/{node_uuid}
Удаление узла, если он не относится ни к одному набору реплик.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": {},
```


(continues on next page)

(продолжение с предыдущей страницы)

```
    "status": true
}
```

Возможные ошибки

- node_not_registered – указанный узел не зарегистрирован
- node_in_use – указанный узел используется в наборе реплик

7.5.4 API по роутерам

- GET /api/v1/routers
- POST /api/v1/routers
- DELETE /api/v1/routers/{uuid}

GET /api/v1/routers

Возврат списка всех узлов, которые представляют собой роутер.

Ответ

```
{
    "data": [
        "uuid-1"
    ],
    "status": true,
    "error": {
        "code": 0,
        "message": "ok"
    }
}
```

POST /api/v1/routers

Присвоение узлу роли роутера.

Тело

```
{
    "uuid": "uuid-1"
}
```

Ответ

```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```

Возможные ошибки

- node_not_registered – указанный узел не зарегистрирован

DELETE /api/v1/routers/{uuid}

Снятие роли роутера с узла.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": {},
   "status": true
}
```


7.5.5 Настройка наборов реплик

- POST /api/v1/replicaset
- PUT /api/v1/replicaset/
- GET /api/v1/replicaset/
- DELETE /api/v1/replicaset/
- POST /api/v1/replicaset/{replicaset_uuid}/master
- POST /api/v1/replicaset/{replicaset_uuid}/node
- DELETE /api/v1/zone/

POST /api/v1/replicaset

Создание набора реплик со всеми зарегистрированными узлами.

Тело


```
{
   "uuid": "optional-uuid",
   "replicaset": [
      {
         "uuid": "uuid-1",
         "master": true
      }
   ]
}
```


Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": {
      "replicaset_uuid": "cc6568a2-63ca-413d-8e39-704b20adb7ae"
   },
   "status": true
}
```


Возможные ошибки

- replicaset_exists – указанный набор реплик уже существует
- replicaset_empty – указанный набор реплик не содержит ни одного узла

- node_not_registered – указанный узел не зарегистрирован
- node_in_use – указанный узел используется в другом наборе реплик

PUT /api/v1/replicaset/{replicaset_uuid}
Обновление параметров набора реплик.

Тело

```
{
   "replicaset": [
      {
         "uuid": "uuid-1",
         "master": true
      },
      {
         "uuid": "uuid-2",
         "master": false,
         "off": true
      }
   ]
}
```

Ответ

```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": {},
   "status": true
}
```

Возможные ошибки

- replicaset_empty – указанный набор реплик не содержит ни одного узла
- replicaset_not_found – указанный набор реплик не обнаружен
- node_not_registered – указанный узел не зарегистрирован
- node_in_use – указанный узел используется в другом наборе реплик

GET /api/v1/replicaset/{replicaset_uuid: optional}
Возврат информации обо всех компонентах кластера. Если передать replicaset_uuid, вернется информация только по данному набору реплик.

Тело

```
{
   "name": "zone 22"
}
```

Ответ

```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
```

(continues on next page)

(продолжение с предыдущей страницы)


```
    "data": {
        "cc6568a2-63ca-413d-8e39-704b20adb7ae": {
            "uuid-1": {
                "hostname": "tnt1.public.i",
                "off": false,
                "repl_user": "repl_user2:repl_pass2",
                "uri": "tnt1.public.i:3301",
                "master": true,
                "name": "tnt1",
                "user": "user1:pass1",
                "zone_id": 1,
                "zone": 1
            },
            "uuid-2": {
                "hostname": "tnt2.public.i",
                "off": true,
                "repl_user": "repl_user1:repl_pass1",
                "uri": "tnt2.public.i:3301",
                "master": false,
                "name": "tnt2",
                "user": "user1:pass1",
                "zone": 1
            }
        }
    },
    "status": true
}
```


Возможные ошибки

- replicaset_not_found – указанный набор реплик не обнаружен

DELETE /api/v1/replicaset/{replicaset_uuid}

Удаление набора реплик.

Ответ


```
{
    "error": {
        "code": 0,
        "message": "ok"
    },
    "data": {},
    "status": true
}
```


Возможные ошибки

- replicaset_not_found – указанный набор реплик не обнаружен

POST /api/v1/replicaset/{replicaset_uuid}/master

Смена мастера в наборе реплик.

Тело


```
{
    "instance_uuid": "uuid-1",
    "hostname_name": "hostname:instance_name"
}
```

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": {},
   "status": true
}
```


Возможные ошибки

- replicaset_not_found – указанный набор реплик не обнаружен
- node_not_registered – указанный узел не зарегистрирован
- node_not_in_replicaset – указанный узел находится не в указанном наборе реплик

POST /api/v1/replicaset/{replicaset_uuid}/node

Добавление узла в набор реплик.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": {},
   "status": true
}
```


Тело


```
{
   "instance_uuid": "uuid-1",
   "hostname_name": "hostname:instance_name",
   "master": false,
   "off": false
}
```


Возможные ошибки

- replicaset_not_found – указанный набор реплик не обнаружен
- node_not_registered – указанный узел не зарегистрирован
- node_in_use – указанный узел используется в другом наборе реплик

GET /api/v1/replicaset/status

Возврат статистики по кластеру.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "data": {
```


(continues on next page)

(продолжение с предыдущей страницы)


```
    "cluster": {
      "routers": [
        {
          "zone": 1,
          "name": "tnt1",
          "repl_user": "repl_user1:repl_pass1",
          "hostname": "tnt1.public.i",
          "status": null,
          "uri": "tnt1.public.i:3301",
          "user": "user1:pass1",
          "uuid": "uuid-1",
          "total_rps": null
        }
      ],
      "storages": [
        {
          "hostname": "tnt1.public.i",
          "repl_user": "repl_user2:repl_pass2",
          "uri": "tnt1.public.i:3301",
          "name": "tnt1",
          "total_rps": null,
          "status": 'online',
          "replicas": [
            {
              "user": "user1:pass1",
              "hostname": "tnt2.public.i",
              "replication_info": null,
              "repl_user": "repl_user1:repl_pass1",
              "uri": "tnt2.public.i:3301",
              "uuid": "uuid-2",
              "status": 'online',
              "name": "tnt2",
              "total_rps": null,
              "zone": 1
            }
          ],
          "user": "user1:pass1",
          "zone_id": 1,
          "uuid": "uuid-1",
          "replicaset_uuid": "cc6568a2-63ca-413d-8e39-704b20adb7ae",
          "zone": 1
        }
      ]
    }
  },
  "status": true
}
```


Возможные ошибки

- zone_not_found – указанная зона не обнаружена
- zone_in_use – в указанной зоне есть хотя бы один узел

7.5.6 Настройка версии конфигурации

- POST /api/v1/version

- GET /api/v1/version

POST /api/v1/version
Настройка версии конфигурации.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "status": true,
   "data": {
      "version": 2
   }
}
```


Возможные ошибки

- cfg_error – ошибка конфигурации

GET /api/v1/version
Возврат версии конфигурации.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "status": true,
   "data": {
      "version": 2
   }
}
```


7.5.7 Конфигурация шардинга

- POST /api/v1/sharding/cfg
- GET /api/v1/sharding/cfg

POST /api/v1/sharding/cfg
Добавление новой конфигурации шардинга.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "status": true,
   "data": {}
}
```


GET /api/v1/sharding/cfg
Возврат текущей конфигурации шардинга.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "status": true,
   "data": {}
}
```


7.5.8 Сброс конфигурации кластера

- POST /api/v1/clean/cfg
- POST /api/v1/clean/all

POST /api/v1/clean/cfg

Сброс конфигурации кластера.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "status": true,
   "data": {}
}
```


POST /api/v1/clean/all

Сброс конфигурации кластера и удаление информации об узлах кластера из каталогов ZooKeeper.

Ответ


```
{
   "error": {
      "code": 0,
      "message": "ok"
   },
   "status": true,
   "data": {}
}
```