Arenadata™ Grid

Версия - v2.3.1-nsud

Роли в Arenadata Grid

2020 Arenadata™

Оглавление

1 Встроенные роли **3**

2 Роли Arenadata Grid **4**
2.1 Процессор обработки входных данных 4
2.2 Коннектор к Kafka 6
2.3 Процессор обработки выходных данных 8
2.4 Планировщик заданий 10
2.5 Сервис внешнего API 10
2.6 Сервис хранения 10

3 Конфигурация yaml-файлов **13**
3.1 Конфигурация 13

Кластер **Arenadata Grid** распределяет функции экземпляров на основе ролей.

Кластерные роли Это Lua-модули, реализующие некоторые заданные для экземпляра функции и/или логику.

Поскольку все экземпляры, на которых запущены кластерные приложения, используют один и тот же исходный код и знают обо всех определенных ролях (и подключенных модулях), можно динамически включать и отключать несколько разных ролей на любом количестве экземпляров без перезапусков даже во время работы кластера.

Глава 1

Встроенные роли

В модуль *cartridge* входят две встроенные роли, которые реализуют автоматический шардинг:

- *vshard-router* – обрабатывает ресурсоемкие вычисления в vshard: направляет запросы к узлам хранения данных;
- *vshard-storage* – работает с большим количеством транзакций в vshard: хранит подмножество набора данных и управляет им.

Глава 2

Роли Arenadata Grid

2.1 Процессор обработки входных данных

Используется для преобразования и обработки данных, приходящих из внешних систем.

2.1.1 Публичные методы

insert_messages_from_kafka

Метод, предназначенный для вставки данных из **Kafka** в спейсы роли *storage*. Требует для работы наличие `schema_registry`, которое предоставляет схемы *avro*.

Параметры:

- `messages` – массив сообщений Kafka, которые представляют собой lua-таблицу структуры:

```
{
   "topic":"имя топика",
   "partition":"номер партиции",
   "offset":"оффсет",
   "key":"ключ",
   "value":"значение"
}
```

- `parse_key_function_str` – имя метода, использующегося для парсинга ключа;
- `parse_value_function_str` – имя метода, использующегося для парсинга значений.

Возвращаемое значение: Возвращает результат обработки данных из Kafka. Выходной формат:

```
{
    "topic":"имя топика",
    "partition":"номер партиции",
    "offset":"оффсет",
    "key":"ключ",
    "result": "результат операции true/false",
    "error": "сообщение об ошибке или nil"
 }
```

Конфигурация:

Использует параметры из настроек топиков кафки и настроек schema_registry.

load_csv_lines

Метод, предназначенный для вставки строк *csv* в спейсы роли *storage*.

Параметры:

- `space_name` – имя спейса, куда производится вставка;
- `lines` – массив строк csv для вставки.

Возвращаемое значение:

- *true* – при успешной вставке;
- *false,error* – при ошибке.

get_metric

Метод, предназначенный для передачи метрик данной роли в систему сбора метрик.

Возвращаемое значение: Набор метрик в формате *JSON*.

2.1.2 Приватные методы

parse_avro

Метод, предназначенный для разбора *avro* и преобразования его в формат, необходимый для вставки в *storage*.

Параметры:

- `schema` – имя схемы для запроса в schema_registry;
- `value` – значение для вставки в формате avro.

Возвращаемое значение:

- *true* – при успешной загрузке;
- *false,error* – при ошибке.

load_avro

Метод, предназначенный для загрузки данных из `parse_avro` в *storage*.

Параметры:

- `space_name` – имя спейса, куда производится вставка строк;
- `lines` – строки, которые необходимо вставить.

Возвращаемое значение:

- *true,разобранные данные* – при успешном разборе;
- *false,error* – при ошибке.

prepare_kafka_message_for_insert

Метод, предназначенный для предварительного разбора данных из `insert_messages_from_kafka`.

Параметры:

- `topic` – имя топика Kafka, откуда пришли данные;
- `data` – данные для вставки в *storage*;

- `parse_function` – функция для разбора данных.

  Возвращаемое значение:

- *true,разобранные данные* – при успешном разборе;
- *false,error* – при ошибке.

get_function_by_name

Метод, предназначенный для возврата функции по ее имени.

Параметры:

- `function_name` – имя функции для поиска в скоупе;

  Возвращаемое значение: Функция или nil.

2.2 Коннектор к Kafka

Включает в себя *Kafka producer* и *Kafka consumer*.

2.2.1 Публичные методы

get_messages_from_kafka

Метод, предназначенный для получения сообщений из канала потребителя **Kafka**, отправки их в `adg_input_processor` для записи в *storage* и последующего коммита в **Kafka**.

Запускается в отдельном файбере *insert_fiber*. По умолчанию вызывается каждые *0.2* секунды.

send_messages_to_kafka

Метод, предназначенный для отправки сообщений в **Kafka**.

Параметры:

- `topic_name` – имя топика, куда производится отправка;
- `messages` – массив сообщений для отправки в Kafka;
- `opts` – дополнительные опции отправки.

  Возвращаемое значение:

- *true* – при успешной вставке;
- *false,error* – при ошибке.

get_metric

Метод, предназначенный для передачи метрик данной роли в систему сбора метрик.

Возвращаемое значение: Набор метрик в формате *JSON*.

2.2.2 Kafka producer

Предназначен для отправки сообщений в **Kafka**.

Публичные методы

create

Метод, предназначенный для создания экземпляра продьюсера кафки *producer*.

Параметры:

- `brokers` – список пар хост/порт для подключения к кластеру Kafka;
- `options` – список пар ключ/значение из https://github.com/edenhill/librdkafka/blob/master/CONFIGURATION.md
- `additional_opts` – список дополнительных настроек подключения.

Конфигурация:

- Настройка брокеров
- Настройка продьюсера

produce

Метод, предназначенный для отправки сообщений в **Kafka** с возможностью выбора синхронно/асинхронно.

Параметры:

- `topic_name` – имя топика, куда необходимо отправить сообщения;
- `messages` – список сообщений, которые необходимо загрузить в Kafka;
- `opts` – дополнительные опции.

close

Метод, предназначенный для удаления экземпляра продьюсера кафки *producer*.

get_producer

Метод, предназначенный для возврата текущего экземпляра *producer*.

Возвращаемое значение: Текущий экземпляр *producer*.

Приватные методы

produce_messages

Метод, предназначенный для отправки сообщений в **Kafka**.

Параметры:

- `topic_name` – имя топика, куда необходимо отправить сообщения;
- `messages` – список сообщений, которые необходимо загрузить в Kafka;
- `is_async` – асинхронно или нет отправлять сообщения в Kafka.

2.2.3 Kafka consumer

Предназначен для отправки сообщений в **Kafka**.

Публичные методы

create

Метод, предназначенный для создания экземпляра продьюсера кафки *producer*.

Параметры:

- `brokers` – список пар хост/порт для подключения к кластеру Kafka;
- `options` – список пар ключ/значение из https://github.com/edenhill/librdkafka/blob/master/CONFIGURATION.md
- `additional_opts` – список дополнительных настроек подключения.

Конфигурация:

- Настройка брокеров
- Настройка продьюсера

produce

Метод, предназначенный для отправки сообщений в **Kafka** с возможностью выбора синхронно/асинхронно.

Параметры:

- `topic_name` – имя топика, куда необходимо отправить сообщения;
- `messages` – список сообщений, которые необходимо загрузить в кафку;
- `opts` – дополнительные опции.

close

Метод, предназначенный для удаления экземпляра продьюсера кафки *producer*.

get_producer

Метод, предназначенный для возврата текущего эксземпляра *producer*.

Возвращаемое значение: Текущий экземпляр *producer*.

Приватные методы

produce_messages

Метод, предназначенный для отправки сообщений в **Kafka**.

Параметры:

- `topic_name` – имя топика, куда необходимо отправить сообщения;
- `messages` – список сообщений, которые необходимо загрузить в Kafka;
- `is_async` – асинхронно или нет отправлять сообщения в Kafka.

2.3 Процессор обработки выходных данных

Используется для преобразования и обработки данных, которые необходимо отправить во внешние системы.

2.3.1 Публичные методы

send_simple_msg_to_kafka

Метод, предназначенный для отправки сообщения в **Kafka**, через `adg_kafka_connector`.

Параметры:

- `topic_name` – имя топика, куда производится отправка;
- `key` – ключ сообщения;
- `value` – значение сообщения.

send_messages_to_kafka

Метод, предназначенный для отправки набора сообщений в **Kafka** через `adg_kafka_connector`.

Параметры:

- `topic_name` – имя топика, куда производится отправка;
- `messages` – массив сообщений для отправки в Kafka;
- `opts` – дополнительные опции отправки.

Возвращаемое значение:

- *true* – при успешной вставке;
- *false,error* – при ошибке.

send_query_to_kafka

Метод, предназначенный для отправки результата произвольного sql-запроса в **Kafka** через `adg_kafka_connector`.

Параметры:

- `topic_name` – имя топика, куда производится отправка;
- `query` – произвольный sql-запрос для выполнения на *storage*;
- `opts` – дополнительные параметры.

send_table_to_kafka

Метод, предназначенный для отправки строк из конкретной таблицы в **Kafka** через `adg_kafka_connector`.

Параметры:

- `topic_name` – имя топика, куда производится отправка;
- `table` – спейс на *storage* для забора данных;
- `filter` – фильтр на запрос к таблице;
- `opts` – дополнительные параметры.

Возвращаемое значение:

- *true* – при успешной вставке;
- *false,error* – при ошибке.

get_metric

Метод, предназначенный для передачи метрик данной роли в систему сбора метрик.

Возвращаемое значение: Набор метрик в формате *JSON*.

2.4 Планировщик заданий

Используется для вызова заданий по расписанию.

2.4.1 Публичные методы

get_metric

Метод, предназначенный для передачи метрик данной роли в систему сбора метрик.

Возвращаемое значение: Набор метрик в формате *JSON*.

2.4.2 Приватные методы

event_loop_run

Метод, предназначенный для получения списка задач из конфига планировщика и их запуск. По умолчанию вызывается *1* раз в секунду.

2.5 Сервис внешнего API

2.5.1 Назначение

2.5.2 Публичные методы

2.5.3 Приватные методы

2.6 Сервис хранения

Используется для хранения и обработки данных.

2.6.1 Публичные методы

get_metric

Метод, предназначенный для передачи метрик данной роли в систему сбора метрик.

Возвращаемое значение: Набор метрик в формате *JSON*.

prep_sql

Метод, предназначенный для генерации и кеширования *prepared statement* запроса.

Параметры:

- `query` – SQL-код запроса.

execute_sql

Метод, предназначенный для запуска SQL-запроса на инстансе.

Параметры:

- `query` – SQL-код запроса;
- `params` – биндинг-параметры запроса.

  Возвращаемое значение:

- Для select-запроса:
  - *набор данных, nil* – при успешном выполнении;
  - *false,error* – при ошибке выполнения;
- Для остальных запросов:
  - *true, nil* – при успешном выполнении;
  - *false,error* – при ошибке выполнения.

set_schema_ddl

Метод, предназначенный для обновления ddl-схемы на данном *storage*. Данные берутся из конфига кластера.

Возвращаемое значение:

- *true, nil* – при успешном выполнении;
- *false,error* – при ошибке выполнения.

2.6.2 Приватные методы

insert_tuples

Метод, предназначенный для вставки туплов.

Параметры:

- `tuples` – список пар `space_name`/`by_space` для вставки в спейсы.

  Возвращаемое значение:

- *true, nil* – при успешном выполнении;
- *false,error* – при ошибке выполнения.

storage_drop_all

Метод, предназначенный для дропа всех спейсов.

Возвращаемое значение:

- *true* – при успешном выполнении;
- *false,error* – при ошибке выполнения.

storage_space_len

Метод, предназначенный для расчета количества строк в спейсе.

Параметры:

- `space_name` – имя спейса, для которого требуется вычислить количество строк.

  Возвращаемое значение: Количество строк в спейсе.

get_storage_ddl

Метод, предназначенный для получения схемы данных данного *storage*.

Возвращаемое значение: ddl-схема текущего *storage*.

Глава 3

Конфигурация yaml-файлов

3.1 Конфигурация

Доступны следующие операции:

- Хранить настройки пользовательских ролей в виде разделов в конфигурации на уровне кластера, например:

```
# YAML configuration file
my_role:
  notify_url: "https://localhost:8080"
```

- Загружать и выгружать конфигурацию всего кластера через веб-интерфейс кластера или с помощью API (запросы GET/PUT к конечной точке admin/config: `curl localhost:8081/admin/config` и `curl -X PUT -d "{'my_parameter': 'value'}"localhost:8081/admin/config`).
- Использовать конфигурацию в своей функции *apply_config()*.

Каждый экземпляр в кластере хранит копию конфигурационного файла в своей рабочей директории (которую можно задать с помощью `cartridge.cfg({workdir = ...})`):

- */var/lib/tarantool/<instance_name>/config.yml* – для экземпляров, развернутых из RPM-пакетов, под управлением *systemd*;
- */home/<username>/tarantool_state/var/lib/tarantool/config.yml* – для экземпляров, развернутых из архивов, под управлением *tarantoolctl*.

Конфигурация кластера представляет собой Lua-таблицу. Если некоторые данные конфигурации для конкретного приложения (например, схему базы данных, описанную с помощью языка определения данных DDL) необходимо хранить в каждом экземпляре кластера, можно использовать свой собственный API, добавив в таблицу специальный раздел. Кластер поможет безопасно передать его всем экземплярам.

Такой раздел создается параллельно (в одном файле) с разделами о топологии и о *vshard*, которые кластер генерирует автоматически. В отличие от сгенерированных разделов – логику изменения, проверки и применения конфигурации в специальном разделе необходимо определять вручную.

Конфигурация ролей ADG:

- *Настройки подключения к Kafka*
- *Настройки топиков Kafka*
- *Настройки потребителей Kafka*
- *Настройки производителей Kafka*

- *Настройки подключений к schema_registry*
- *Настройки планировщика заданий*

3.1.1 Настройки подключения к Kafka

Доступные опции:

- `bootstrap_connection_string` – список пар хост/порт для подключения к кластеру Kafka.

  Пример значения: `'bootstrap_connection_string': '10.18.84.6:9092,10.18.84.7:9092'`

3.1.2 Настройки топиков Kafka

Список пар топик/настройки топиков. Топик – значение из опции *topics* раздела конфигурации *Настройки потребителей Kafka*.

Доступные опции:

- `error_topic` – топик, куда уходят сообщения об ошибках загрузки;
- `success_topic` – топик, куда уходят сообщения об успешной загрузке;
- `target_table` – целевая таблица на storage, куда записываются строки из сообщений Kafka;
- `schema_data` – наименование схемы из `schema_registry` для значения сообщения;
- `schema_key` – наименование схемы из `schema_registry` для ключа сообщения.

Пример:

```
{   "EMPLOYEES":{
    "error_topic":"input_test2",
    "success_topic":"input_test",
    "target_table":"EMPLOYEES",
    "schema_data":"employees",
    "schema_key":"adb_upload_request"

},
 "DOCS":{
    "error_topic":"input_test3",
    "success_topic":"input_test4",
    "target_table":"DOCS",
    "schema_data":"docs",
    "schema_key":"adb_upload_request"

}
}
```

3.1.3 Настройки потребителей Kafka

Доступные опции:

- `custom_properties` – список пар ключ/значение для кастомных настроек потребителя.

Пример значения:

```
{
    "custom_properties":{
        "log_level":7
  }
}
```

- `properties` – список пар ключ/значение из https://github.com/edenhill/librdkafka/blob/master/CONFIGURATION.md

  Пример значения:

```
{
    "properties":{
        "enable.auto.offset.store":"false",
        "auto.offset.reset":"latest",
        "enable.auto.commit":"false",
        "group.id":"tarantool-group-csv",
        "partition.assignment.strategy":"roundrobin",
        "enable.partition.eof":"false"

  }
}
```

- `topics` – список топиков, на которые подписывается потребитель.

  Пример значения:

```
{  "topics":[
    "EMPLOYEES",
    "DOCS"
]}
```

3.1.4 Настройки производителей Kafka

Доступные опции:

- `custom_properties` – список пар ключ/значение для кастомных настроек потребителя.

  Пример значения:

```
{
    "custom_properties":{
        "log_level":7
  }
}
```

- `properties` – список пар ключ/значение из https://github.com/edenhill/librdkafka/blob/master/CONFIGURATION.md

  Пример значения:

```
{
    "properties":{
      "message.send.max.retries" : 500,
      "compression.codec": "gzip"
  }
}
```

3.1.5 Настройки подключений к schema_regisrty

Доступные опции:

- `host` – хост `schema_regisrty` для подключения.

  Пример значения: `"host":"10.18.84.6"`.

- `port` – порт `schema_regisrty` для подключения.

  Пример значения: `"port":"8081"`

3.1.6 Настройки планировщика заданий

Список пар задача/настройки задачи.

Доступные опции:

- `funcname` – название метода, который вызывается планировщиком;
- `kind` - тип задачи на текущий момент, поддерживается только *periodical*;
- `rolename` - название роли, на которой вызывается *funcname*;
- `schedule` - крон-строка расписания.

  Пример:

```
{
    "scheduler.tasks":{
       "kafka_send":{
          "funcname":"test_msg_to_kafka",
          "kind":"periodical",
          "rolename":"app.roles.adg_output_processor",
          "schedule":"5,10,15,20,25,30,35,40,45,50,55 * * * * *"

 },
       "kafka_sink":{
          "funcname":"get_messages_from_kafka",
          "kind":"periodical",
          "rolename":"app.roles.adg_kafka_connector",
          "schedule":"1-59 * * * * *"

 }

 }
 }
```

Important: Контактная информация службы поддержки – e-mail: info@arenadata.io