Документация

Выпуск 1.26

Руководство программиста XSQUARE-xDAC

апр. 27, 2023

Содержание

ГЛАВА 1

Общие сведения

При подключении к базе данных приложение сканирует набор процедур и функций, которые расположены в схеме, указанной в настроечном параметре “parsingSchema”. Результат сканирования записывается в кэш приложения, который далее используется для сопоставления запрашиваемых процедур и параметров переданных в запросе.

Каждая процедура и функция, которая была получена в результате сканирования, публикуется как endpoint приложения xdac. Доступ осуществляется по наименованию процедуры или функции, который указывается в пути вызова сервиса. Параметры передаваемые в процедуру передаются в формате json, и сопоставляется по наименованию. Результат работы функции преобразуется в скалярное значение json и отдаётся конечному потребителю.

ГЛАВА 2

XDAC - Data Access Connector

2.1 Проблематика

Разработка API для приложения часто требует от разработчика дополнительных навыков, таких как понимание работы протокола http, знание дополнительного языка программирования и инструментария, который этот язык предоставляет, а также умения администрирования (установка и мониторинг) готового API.

Зачастую вся логика и данные, которые необходимо опубликовать через API уже существуют в том или ином виде на стороне базы данных. В этом случае разработчик вынужден разрабатывать точки доступа для этих данных, при этом процесс разработки сводится к рутинному и трудозатратному процессу написанию однотипного кода.

2.2 Решение

XDAC позволяет автоматизировать процесс публикации API, предоставляя доступ к процедурам и функциям хранимых на стороне Postgresql. Используя данное решение разработчику базы данных больше не потребуется писать отдельное приложение.

ГЛАВА 3

Вызов процедур и функций

Каждая процедура и функция, полученная в результате сканирования схемы, публикуется сервисом с префиксом `/rpc/`

```
curl "http://localhost:3000/xdac/rpc/function_name" -X POST
```

Для передачи параметров необходимо включить в тело запроса json, наименование полей которого будет соответствовать наименованию аргументов функции. Для примера создадим функцию следующего вида:

```
CREATE OR REPLACE FUNCTION api.add_numbers (
    first integer,
    second integer
)
RETURNS integer AS
$body$
BEGIN
    return first + second;
END;
$body$
LANGUAGE 'plpgsql'
```

Для вызова функции необходимо передать запрос следующего вида:

```
curl -d '{"first":10,"second":30}' 'http://localhost:3000/xdac/rpc/add_numbers'
```

```
40
```

Каждый вызов процедуры или функции осуществляется в своей транзакции. Наличие оператора `COMMIT` в теле функции приведет к ошибке.

3.1 Правила наименования

Все наименования (имена процедур и функций, наименования аргументов) - регистрозависимые.

Для вызова функции с именем `StRaNgeNamE` необходимо вызвать метод следующим образом:

```
curl "http://localhost:3000/xdac/rpc/StRaNgeNamE"
```

Вызов в другом формате приведет к ошибке:

```
{
    "code": "XDAC001",
    "hint": "Make sure that routine name is correct or try to reload applicaion",
    "message": "Routine strangename was not found in app cache",
    "details": ""
}
```

3.2 Вызов процедуры с JSON аргументом

Если необходимо передать json внутрь функции или процедуры, необходимо передать заголовки `Content-Type: application-jsonи Prefer:params=single-object`

```
curl -H «Content-Type: application/json» -H «Prefer: params=single-object» -d „
↪{«text»:»Test message»,»input»:[1,2,3]}“ «http://localhost:3000/xdac/rpc/jsonInput»
```

Сама процедура должна иметь только один аргумент с типом данных **json** или **jsonb**:

```
CREATE OR REPLACE FUNCTION api."jsonInput" (
    "myPar" json
)
RETURNS json AS
$body$
begin
return "myPar";
END;
$body$
LANGUAGE 'plpgsql'
```

3.3 Загрузка бинарного файла

Для того чтобы передать бинарный файл в процедуру или функцию необходимо, чтобы процедура имела один аргумент с типом bytea, а запрос содержал заголовок **“Content-Type: application/octet-stream”**

```
curl "http://localhost:3000/xdac/rpc/upload_binary" \
    -X POST -H "Content-Type: application/octet-stream" \
    --data-binary "@file_name.ext"
```

```
CREATE OR REPLACE PROCEDURE api."upload_binary" (
    bytea
)
AS
```

(continues on next page)

(продолжение с предыдущей страницы)

```
$body$
BEGIN
    insert into api.files(blob)
    values ($1);
END;
$body$
LANGUAGE 'plpgsql'
```

ГЛАВА 4

Работа с заголовками

Все заголовки передаваемые в сервис записываются в GUC параметры сессии postgresql. Следующие параметры доступны для доступа `request.headers`, `request.method` and `request.path`

```
-- Получить все заголовки. Они преобразуются в json
SELECT current_setting('request.headers', true)::json;

-- Метод с помощью которого был вызван сервис
SELECT current_setting('request.method', true); --GET/POST/PUT....

-- Путь по которому был вызван сервис
SELECT current_setting('request.path', true); --/xdac/rpc/test
```

Для того чтобы установить заголовки ответа и код ответа используются GUC `response.headers` и `response.status`

```
select set_config('response.headers','[{"Content-Type":"application/json"},{"Set-
↪Cookie":"foo=bar"}]',true);
--необходимо передать массив объектов json
select set_config('response.status','405',true); -- устанавливает код ответа в 405
```

ГЛАВА 5

Обработка ошибок

В случае возникновения ошибки сервис вернёт её в формате:

```
{
  "code": "",
  "hint": "",
  "message": "",
  "details": ""
}
```

Сервис предоставляет возможность обрабатывать пользовательские ошибки вызванные оператором raise

```
{
  "code": "SRV01",
  "hint": "Try GET method",
  "message": "This is postgresql error",
  "details": "This is detail"
}
```

При этом транзакция будет завершена и откачена с помощью оператора `ROLLBACK`.

Если необходимо передать пользовательскую ошибку в сервис, при этом сохранить результаты выполнения функции, можно использовать установку статуса ответа через GUC

```
perform set_config('response.status','405',true);
perform set_config('response.headers','[{"Content-Type":"application/json"}]',true);
return json_build_object('message','Only GET method is allowed');
```

ГЛАВА 6

Управление кэшом приложения

Все процедуры и функции доступные схеме сканируются при запуске приложения и помещаются в кэш. При изменении сигнатуры процедур или создании новых функций они не будут доступны для вызова через API. Для этого необходимо перезагрузить приложение, для того чтобы кэш обновился.

Для автоматизации перезагрузки кэша приложения, XDAC предоставляет возможность перезагрузить кэш с помощью механизма уведомлений `NOTIFY`.

Пользователь может отправить уведомление в канал xdac с уведомлением о перезагрузке кэша:

```
NOTIFY xdac, 'reload';
```

6.1 Автоматизация уведомлений

Следующий способ автоматизировать перезагрузку кэша - создание event trigger на операции ddl. Пользователь с правами SUPERUSER может создать следующий триггер:

```
CREATE OR REPLACE FUNCTION public.xdac_watch() RETURNS event_trigger
LANGUAGE plpgsql
AS $$
 BEGIN
   NOTIFY xdac, 'reload';
 END;
$$;

CREATE EVENT TRIGGER xdac_watch  ON ddl_command_end  EXECUTE PROCEDURE public.xdac_
↪watch();
```

При создании нового объекта базы данных, приложение автоматически получит уведомление о необходимости перезагрузить кэш.