Independent Submission
Request for Comments: 7047
Category: Informational
ISSN: 2070-1721

B. Pfaff
B. Davie, Ed.
VMware, Inc.

December 2013

Протокол управления OVSDB

The Open vSwitch Database Management Protocol

Тезисы

Open vSwitch представляет собой программный коммутатор с открытым исходным кодом, разработанный для использования в качестве виртуального коммутатора (vswitch) в виртуализованных средах. Коммутатор vswitch пересылает трафик между различными типами виртуальных машин (VM[1]) на одном физическом хосте, а также между VM и физической сетью. Open vSwitch открыт для программного расширения и управления с использованием протоколов управления OpenFlow и OVSDB (Open vSwitch Database). Этот документ определяет протокол управления OVSDB. Проект Open vSwitch включает реализации клиента и сервера OVSDB с открытым исходным кодом.

Статус документа

Документ не относится к категории Internet Standards Track и публикуется с информационными целями.

Документ входит в серию RFC, но не связан с каким-либо потоком RFC. Редактор RFC самостоятельно принял решение о публикации документа и не делает каких-либо заявлений о его пригодности для реализации или внедрения. Документы, одобренные для публикации лишь редактором RFC, не претендуют на какой-либо статус стандартизации Internet (см. раздел 2 в RFC 5741).

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке http://www.rfc-editor.org/info/rfc7047.

Оглавление

[1]Virtual machine.

1. Введение

В виртуализованных серверных средах обычно нужны виртуальные коммутаторы (vswitch) для пересылки трафика между разными VM на одном физическом хосте, а также между VM и физической сетью. Open vSwitch [OVS] является программным коммутатором с открытым исходным кодом, разработанным для использования в качестве vswitch в таких средах. Open vSwitch (OVS) открыт для программного расширения и управления с использованием протоколов управления OpenFlow [OF-SPEC] и OVSDB (Open vSwitch Database). Этот документ определяет протокол управления OVSDB. Проект Open vSwitch включает реализации клиента и сервера OVSDB с открытым исходным кодом.

Протокол управления OVSDB использует JSON [RFC4627] в качестве формата передачи и основан на JSON-RPC версии 1.0 [JSON-RPC].

Схема базы данных Open vSwitch описана в [DB-SCHEMA]. Этот документ задает протокол для взаимодействия с базой данных при управлении и настройке конфигурации экземпляров Open vSwitch. Описанный в документе протокол также включает способы определения используемой схемы, описанные в параграфе 4.1.2.

Протокол управления OVSDB предназначен для обеспечения программного доступа к базе данных Open vSwitch, в соответствии с описанием [DB-SCHEMA]. Эта база содержит конфигурацию для одного демона Open vSwitch. В соответствии с текущим определением эта информация включает поведение коммутации vswitch и не описывает поведение или конфигурацию системы маршрутизации. Если система будет расширена для поддержки системы маршрутизации, разработчикам и операторам следует ознакомиться с работами группы IETF I2RS, которая задает протоколы и модели данных для взаимодействия с системой маршрутизации по событиям или в реальном масштабе времени.

1.1. Уровни требований

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не нужно** (SHALL NOT), **следует** (SHOULD), **не следует** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **не рекомендуется** (NOT RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе интерпретируются в соответствии с [RFC2119].

1.2. Терминология

UUID
Уникальный универсальный идентификатор — 128-битовое значение, уникальное в пространстве и времени [DCE].

OVS
Open vSwitch — виртуальный коммутатор с открытым исходным кодом.

OVSDB
База данных (БД), используемая для настройки конфигурации экземпляров OVS.

JSON
Javascript Object Notation (обозначение объектов) [RFC4627].

JSON-RPC
JSON Remote Procedure Call (вызов удаленных процедур) [JSON-RPC].

Durable - долговечный
Надежно записанный в энергонезависимое хранилище (например, диск). OVSDB поддерживает опцию для управления долговечностью (хранения) транзакций.

Отметим, что спецификация JSON [RFC4627] обеспечивает точные определения множества важных терминов, включая значения, объекты, массивы, числа и строки JSON. Во всех случаях данный документ использует определения из [RFC4627].

2. Обзор системы

На рисунке 1 показаны основные компоненты Open vSwitch и интерфейсы с кластером управления и администрирования. Экземпляр OVS содержи сервер баз данных (ovsdb-server), демон vswitch (ovs-vswitchd) и может включать модуль пересылки по быстрому пути. Кластер управления и пересылки включает некоторое число контроллеров и менеджеров. Последние используют протокол OVSDB для управления экземплярами OVS. Каждый экземпляр OVS управляется хотя бы одним менеджером. Контроллеры используют OpenFlow для задания состояний потоков в коммутаторах OpenFlow. Экземпляр OVS может поддерживать множество логических путей данных, называемых мостами (bridge). Для каждого моста OpenFlow имеется хотя бы один контроллер.

Интерфейс управления OVSDB служит для выполнения операций управления и настройки экземпляров OVS. По сравнению с OpenFlow операции управления OVSDB выполняются в достаточно продолжительном интервале времени. Примеры операций, поддерживаемых OVSDB, включают:

- создание, изменение и удаление мостов (путей данных0 OpenFlow, которых может быть много в одном экземпляре OVS;
- настройку набора контроллеров, с которым следует соединить путь данных OpenFlow;
- настройку набора менеджеров, с которым следует соединить сервер OVSDB;

- создание, изменение и удаление портов пути данных OpenFlow;
- создание, изменение и удаление туннельных интерфейсов путей данных OpenFlow;
- создание, изменение и удаление очередей;
- настройку правил QoS[1] и связывание этих правил с очередями;
- сбор статистики.

OVSDB не выполняет задач на уровне отдельных потоков, оставляя это протоколу OpenFlow.

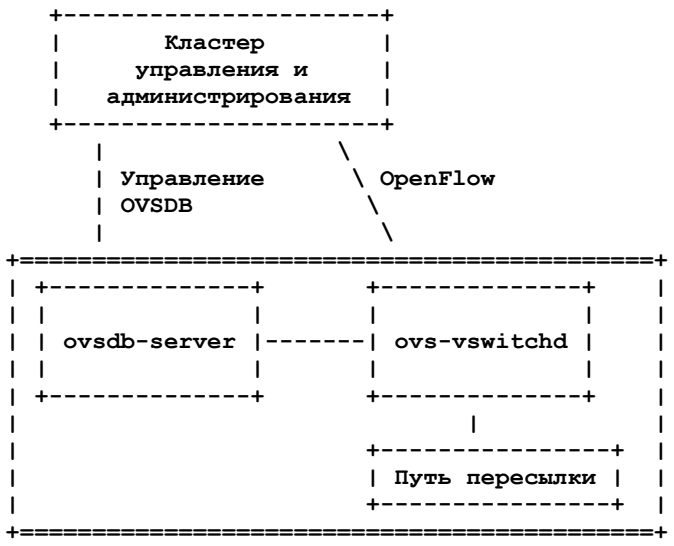

Рисунок 1. Интерфейсы Open vSwitch.

Дополнительная информация об использовании протокола управления OVSDB приведена в [DB-SCHEMA].

3. Структура OVSDB

В этом разделе описана структура баз данных OVSDB. Эти базы являются достаточно обобщенными. Полное описание текущей схемы баз данных, используемой в OVS, приведено в [DB-SCHEMA]. В параграфе 4.1.2 рассказано о том, как протокол управления OVSDB может применяться для обнаружения используемой базы данных.

3.1. Использование JSON

OVSDB использует нотацию JSON [RFC4627] для формата схемы и протокола передачи. Реализация JSON в Open vSwitch имеет два ограничения:

- Null-байты (\u0000) **не следует** использовать в строках;
- поддерживается только кодировка UTF-8.

В последующих описаниях применяются сокращенные обозначения значений JSON, соответствующие [RFC4627].

<string>
Строка JSON, в которой разрешены любые символы Unicode. Реализациям **следует** запрещать null-байты.

<id>
Строка JSON, соответствующая [a-zA-Z_][a-zA-Z0-9_]*. Идентификаторы, начинающиеся с символа подчеркивания (_) зарезервированы для реализации, их применение пользователями **недопустимо**.

<version>
Строка JSON с номером версии, соответствующая формату [0-9]+\.[0-9]+\.[0-9]+

<boolean>
Значение JSON true или false.

<number>
Число JSON.

<integer>
Целое число JSON из диапазона $-(2^{63})$ … $+(2^{63})-1$.

<json-value>
Любое значение JSON.

<nonnull-json-value>
Любое значение JSON кроме null.

<error>
Объект JSON с указанными ниже членами:

```
"error": <string>          обязательно
"details": <string>        необязательно
```

Значением элемента "error" является короткая строка, заданная в этом документе, которая показывает класс ошибки. Большинство строк "error", связанных с конкретным контекстом, описаны в этом документе, но приведенные здесь строки "error" могут встречаться в любом контексте, где разрешены объекты <error>.

```
"error": "resources exhausted"
```
Операция требует больше ресурсов (память, диса, CPU и т. п.), чем в данный момент доступно серверу БД.

```
"error": "I/O error"
```

[1]Quality of service — качество обслуживания.

Проблемы доступа к диску, сети или другим нужным ресурсам, не позволяющие выполнить операцию целиком. Реализации БД **могут** использовать строки "error", заданные в этом документе, для индикации ошибок, которые не попадают ни в одну из указанных категорий. Объект <error> **может** включать элемент "details", значение которого является строкой, описывающей ошибку более подробно для пользователей или администратора. Данный документ не задает формат или содержимое строк "details". Объект <error> **может** также включать другие элменты для более подробного описания ошибок. Документ не задает имен и значений для таких элементов.

3.2. Формат схемы

Конфигурационная БД Open vSwitch состоит из набора таблиц, каждая из которых содержит множество полей (столбцов) и может включать множество (возможно пустое) строк. Схема БД представлена объектом <database-schema>, описанным ниже.

<database-schema>

Объект JSON, включающий следующие элементы:

"name": <id>	**обязательно**
"version": <version>	**обязательно**
"cksum": <string>	**необязательно**
"tables": {<id>: <table-schema>, ...}	**обязательно**

Поле "name" идентифицирует БД в целом. Это имя требуется для большинства запросов JSON-RPC и указывает БД, с которой осуществляется работа.

Поле "version" указывает версию схемы БД. Это поле является **обязательным**. Семантика Open vSwitch для поля "version" описана в [DB-SCHEMA]. Другие схемы могут использовать это поле иначе.

Необязательное поле "cksum" содержит заданную реализацией контрольную сумму для схемы БД. Она служит прежде всего инструментом для разработчиков и клиентам **следует** игнорировать контрольные суммы.

Поле "tables" указывает объекты JSON, чьи имена являются именами таблиц, а значения - объектами <table-schema>.

<table-schema>

Объект JSON, включающий следующие элементы:

"columns": {<id>: <column-schema>, ...}	**обязательно**
"maxRows": <integer>	**необязательно**
"isRoot": <boolean>	**необязательно**
"indexes": [<column-set>*]	**необязательно**

Поле "columns" является объектом JSON, имя, имена и значения поле которого являются объектами <column-schema>.

Каждая таблица имеет следующие поля, определения которых не включены в схему:

"_uuid" — это поле содержит в точности одно значение UUID и инициализируется случайным образом при создании строки БД. Поле предназначено только для чтения и никогда не меняется в течение срока существования строки.

"_version" — подобно "_uuid", содержит единственное значение UUID, инициализируемое случайным числом при создании строки и доступное только для чтения. Однако значение поля меняется на новое случайное число при изменении других полей строки. Кроме того, это поле является короткоживущим (эфемерным) — при закрытии БД и последующем ее открытии или при остановке и повторном запуске процесса БД поле "_version" получает новое случайное значение.

Если поле "maxRows" содержит положительное целое число, оно ограничивает максимальное число строк в таблице. Это «отложенное» ограничение применяется только в момент представления транзакции (см. описание запроса "transact" в параграфе 4.1.3). Если поле "maxRows" не задано, размер таблицы ограничивается лишь доступными на сервере БД ресурсами. Ограничения "maxRows" применяются после удаления из таблицы строк без ссылок на них при "isRoot" = false.

Логическое значение "isRoot" указывает, нужны ли в таблице строгие ссылки из других таблиц и позволяет избавляться от «мусора» (смотри обсуждение ссылок "strong" и "weak" в описании <base-type> ниже). Если в поле "isRoot" задано значение true, строка в таблице существует независимо от других ссылок (ее можно считать частью «корневого набора» при сборке мусора). Если поле "isRoot" не задано или содержит значение false, любая строка в таблице может существовать лишь при наличии хотя бы одной ссылки на нее с refType = "strong" (из той же или другой таблицы). Это «отложенное» действие и строки без ссылок на них удаляются перед представлением транзакии.

Для совместимости со схемами, созданными до определения поля "isRoot" при опущенном или имеющем значение false поле "isRoot" в каждом объекте <table-schema> данного объекта <database-schema> каждая таблица считается частью корневого набора.

Если поле "indexes" задано, оно должно быть массивом (возможно пустым) объектов <column-set>. Объект <column-set> является массивом (возможно пустым) строк с именами полей. Каждый массив <column-set> является набором полей, чьи значения, собранные вместе из данной строки, должны быть уникальными в рамках таблицы. Это «отложенное» ограничение, которое применяется в момент представления транзакции, после удаления строк без ссылок на них и строк с «подвешенными» мягкими ссылками на них. Эфемерные поля не могут быть частью индексов.

<column-schema>

Объект JSON, включающий следующие элементы:

"type": <type>	**обязательно**
"ephemeral": <boolean>	**необязательно**
"mutable": <boolean>	**необязательно**

Поле "type" задает тип данных, хранящихся в этой колонке (поле).

Если "ephemeral" имеет значение true, для значений в колонке не гарантируется долговечность и они могут быть потеряны при перезапуске БД. Колонки, чей тип (ключ или значение) относится к строгим ссылкам на таблицу, не являющуюся частью корневого набора, являются долговечными независимо от значения этого поля (в противном случае при перезапуске БД может теряться строка целиком).

Если указано "mutable" = false, значения колонки (поля) не могут быть изменены после их установки операцией "insert".

<type>

Тип колонки (поля) БД. Это <atomic-type> или объект JSON, описывающий тип колонки БД со следующими элементами:

"key": <base-type>	**обязательно**
"value": <base-type>	**необязательно**
"min": <integer>	**необязательно**
"max": <integer> или "unlimited"	**необязательно**

Для "min" и "max" по умолчанию принято значение 1. Если для "max" указано значение "unlimited", максимальное число элементов не ограничено, но реализация может задать свои ограничения. После рассмотрения принятых по умолчанию значения "min" должно иметь значение 0 или 1, а "max" — не меньше 1 и не меньше значения "min".

Если "min" и "max" имеют значение 1, а "value" не задано, типом будет скаляр, указанный элементом "key".

Если "min" или "max" отлично от 1, а "value" не задано, типом будет набор скаляров, указанных элементом "key".

Если элемент "value" задан, тип отображается из "key" в "value".

<base-type>

Тип ключа или значения в колонке (поле) БД. Это <atomic-type> или объект JSON со следующими элементами:

"type": <atomic-type>	**обязательно**
"enum": <value>	**необязательно**
"minInteger": <integer>	**необязательно, только целые числа**
"maxInteger": <integer>	**необязательно, только целые числа**
"minReal": <real>	**необязательно, только действительные числа**
"maxReal": <real>	**необязательно, только действительные числа**
"minLength": <integer>	**необязательно, только строки**
"maxLength": <integer>	**необязательно, только строки**
"refTable": <id>	**необязательно, только UUID**
"refType": "strong" или "weak"	**необязательно, только с "refTable"**

Тип <atomic-type> сам по себе является эквивалентом объекта JSON с одним элементом "type", имеющим значение <atomic-type>.

Элемент "enum" может быть задан как <value>, чьим типом является одно или множество значения, заданных для элемента "type". Если элемент "enum" указан, пригодные значения <base-type> ограничены включенными в <value>.

Элемент "enum" не совместим с перечисленными ниже случаями.

Если "type" = "integer", может быть задано "minInteger", "maxInteger" или оба для ограничения диапазона пригодных значений. При задании обоих значение "maxInteger" должно быть не меньше "minInteger".

Если "type" = "real", может быть задано "minReal", "maxReal" или оба для ограничения диапазона пригодных значений. При задании обоих значение "maxReal" должно быть не меньше to "minReal".

Если "type" = "string", может быть задано "minLength" and "maxLength" или оба для ограничения диапазона пригодных значений. При задании обоих значение "maxLength" должно быть не меньше "minLength". Размер строки указывается числом символов.

Если "type" = "uuid", поле "refTable" (при его наличии) должно быть именем таблицы в данной БД. При заданном "refTable" должно быть указано и "refType", и эффект "refTable" зависит от "refType":

- если "refType" = "strong" или отсутствует, разрешенные значения UUID ограничены идентификаторами строк в указанной таблице;
- если "refType" = "weak", разрешены любые UUID, но идентификаторы, не соответствующие строкам указанной таблицы, будут автоматически удаляться. При возникновении такой ситуации для отображения будут удаляться и ключ, и значение.

Элемент "refTable" задает «отложенные» ограничения, которые применяются в момент представления транзакции (см. описание запроса "transact" в параграфе 4.1.3). Остальные ограничения <base-type> применяются сразу же при каждой операции ("immediate").

<atomic-type>

Одно из значений "integer", "real", "boolean", "string" или "uuid", представляющее скалярный тип.

4. Протокол передачи

Протокол передачи БД реализован в JSON-RPC 1.0 [JSON-RPC]. Хотя спецификация JSON-RPC допускает различный транспорт, реализациям данной спецификации **следует** работать непосредственно на базе TCP, через выделенный для этого порт (см. раздел 6).

4.1. Методы RPC

В последующих параграфах описаны поддерживаемые протоколом методы RPC. Как указано в спецификации JSON-RPC 1.0, каждый запрос представляет собой строку, содержащую имя метода, (возможно пустой) массив передаваемых методу параметров, а также идентификатор запроса, который может служить для сопоставления откликов с запросами. Каждый отклик содержит объект результата (непустой при успешном вызове, объект ошибки (непустой при наличии ошибки) и идентификатор для сопоставления с запросом. Более подробное рассмотрение каждого метода, его параметров и результатов приведено ниже.

Сервер OVSDB **должен** реализовать все описанные здесь методы. Клиент OVSDB **должен** реализовать метод "Echo" и может реализовать другие методы в соответствии со своими потребностями.

Операции, которые могут выполняться для БД OVS и применением этих методов (например, "transact"), описаны в разделе 5.

4.1.1. Список баз данных

Эта операция отыскивает массив, элементами которого являются имена БД, которые доступны через соединение данного протокола управления.

Объект запроса содержит следующие элементы:

- **"method": "list_dbs"**

- `"params": []`
- `"id": <nonnull-json-value>`

Объект отклика содержит перечисленные элементы:

- `"result": [<db-name>,…]`
- `"error": null`
- `"id": совпадает с идентификатором в запросе`

4.1.2. Получение схема

Эта операция отыскивает схему <database-schema>, которая описывает БД <db-name>.

Объект запроса содержит следующие элементы:

- `"method": "get_schema"`
- `"params": [<db-name>]`
- `"id": <nonnull-json-value>`

Объект отклика содержит перечисленные элементы:

- `"result": <database-schema>`
- `"error": null`
- `"id": совпадает с идентификатором в запросе`

Если указанной БД не существует, сервер возвращает ошибку JSON-RPC вида:

- `"result": null`
- `"error": "unknown database"`
- `"id": совпадает с идентификатором в запросе`

4.1.3. Транзакция

Этот метод RPC инициирует выполнение на сервере БД последовательности операций в заданном порядке для указанной БД.

Объект запроса содержит следующие элементы:

- `"method": "transact"`
- `"params": [<db-name>, <operation>*]`
- `"id": <nonnull-json-value>`

Значение "id" **должно** быть уникальным для всех выполняющихся в данный момент транзакций в рамках сессии JSON-RPC. В противном случае сервер может возвратить ошибку JSON-RPC.

Массив "params" для этого метода состоит из <db-name>, идентифицирующих БД, к которой применяется транзакция, далее могут следовать объекты JSON, каждый из которых представляет одну операцию БД. Допустимые операции описаны в разделе 5. Сервер БД выполняет каждую из операций в заданном порядке, прерывая выполнение при возникновении той или иной ошибки. Набор операций выполняется как одна неделимая, согласованная и изолированная транзакция. Транзакция выполняется тогда и только тогда, когда каждая операция в ней успешна. Стойкость представления не гарантируется, пока в набор операций не включена операция "commit" с "durable" = true. Более подробное описание операций приведено в разделе 5.

Объект отклика содержит перечисленные элементы:

- `"result": [<object>*]`
- `"error": null`
- `"id": совпадает с идентификатором в запросе`

Независимо от наличия ошибок при операциях с БД ответом всегда будет отклик JSON-RPC с (возможно) пустым (null) элементом "error" и элементом "result", который является массивом с таким же числом элементов, которое было в "params". Каждый элемент массива "result" соответствует такому же элементу массива "params". Интерпретация элементов массива "result" описана ниже.

- Объект JSON, который не содержит элемента "error", говорит об успешном выполнении операции. Отдельные элементы объекта описаны ниже вместе с соответствующими операциями. Некоторые операции не выдают каких-либо результатов и в этом случае объект не будет включать элементов.
- Элемент <error> указывает возникновение ошибки при выполнении соответствующей операции.
- Значение JSON null указывает, что операция не выполнялась по причине отказа предыдущей операции.

В общем случае "result" содержит некоторое число успешных результатов, за которыми может следовать ошибка, а за ней - значения JSON null для совпадения с числом элементов в массиве "params". Здесь имеется одно исключение — если все операции завершились успешно, но результат не может быть представлен, "result" будет включать на один элемент больше, чем "params" и этим дополнительным элементом будет <error>. В таких случаях возможны перечисленные ниже строки "error".

"error": "referential integrity violation"

При попытке выполнения значения поля, указанного UUID для строки таблицы не было обнаружено в таблице, указанной ключом <base-type> или значением "refTable" с "refType" = "strong" (это может быть обусловлено вставкой строки, ссылающейся на отсутствующую строку, удалением строки, на которую продолжает ссылаться другая строка, указанием UUID для строки неверной таблицы или другими причинами).

"error": "constraint violation"

Может возникать множество ситуаций, при которых попытка представления транзакции будет приводить к нарушению ограничений БД (см. параграф 3.2, где обсуждаются ограничения). Некоторые из таких ситуаций приведены ниже.

- Число строк в таблице превосходит максимум, заданный значением "maxRows" для таблицы.
- Две или более строк таблицы имеют совпадающие значения в полях, образующих индекс.

- Колонка (поле) с ключом <base-type> или значение "refTable" с "refType" = "weak" становится пустым в результате удаления, а это поле не может быть пустым, поскольку для его типа <type> задано "min" = 1. Такое удаление может быть результатом удаления строки на которую указывает ссылка (или отсутствие такой строки, если строка колонки (поля) была вставлена в рамках транзакции).

"error": "resources exhausted"

Операция требует ресурсов (память, диск, CPU и т. п.) больше, чем доступно на сервере БД.

"error": "I/O error"

Проблема доступа к диску, сети или другому ресурсу, препятствующая выполнению операции целиком.

Если "params" содержит одну или несколько операций "wait", транзакция может занимать неопределенное время. Реализация БД **должна** быть способна воспринимать, выполнять и отвечать на другие транзакции и иные запросы JSON-RPC, пока одна или несколько транзакций с операциями "wait" находятся в состоянии исполнения для той же или другой сессии JSON-RPC.

4.1.4. Отмена

Метод "cancel" является уведомлением JSON-RPC, т. е. соответствующего отклика не предоставляется. Метод говорит серверу БД о необходимости незамедлительно завершить или прервать запрос "transact", для которого "id" совпадает со значением "params" в уведомлении. Элементы объекта уведомления перечислены ниже.

- **"method": "cancel"**
- **"params": ["id" незавершенного запроса]**
- **"id": null**

Запрос "transact" может быть выполнен полностью и тогда сервер возвращает отклик в форме, описанной для "transact" (параграф 4.1.3). В остальных случаях сервер передает отклик JSON-RPC с ошибкой, как показано ниже.

- **"result": null**
- **"error": "canceled"**
- **"id": "id" отмененного запроса.**

Для самого уведомления "cancel" не передается отклика.

4.1.5. Мониторинг

Запрос "monitor" позволяет клиенту реплицировать таблицы или их части в БД OVSDB путем запроса изменений этих таблиц и получения полного начального состояния таблицы или ее части. Элементы запроса показаны ниже.

- **"method": "monitor"**
- **"params": [<db-name>, <json-value>, <monitor-requests>]**
- **"id": <nonnull-json-value>**

Параметр <json-value> служит для сопоставления последующих уведомлений об изменениях (см. ниже) с данным запросом. Объект <monitor-requests> отображает имя таблицы для мониторинга на массив объектов <monitor-request>.

Каждый <monitor-request> является объектом с перечисленными ниже элементами.

"columns": [<column>*]	**необязательно**
"select": <monitor-select>	**необязательно**

Поле columns (при его наличии) задает колонки (поля) таблицы для мониторинга, а объект <monitor-select> включает показанные ниже элементы.

"initial": <boolean>	**необязательно**
"insert": <boolean>	**необязательно**
"delete": <boolean>	**необязательно**
"modify": <boolean>	**необязательно**

Содержимое этого объекта задает режим мониторинга полей или таблицы (см. ниже).

Объект отклика содержит перечисленные ниже элементы.

- **"result": <table-updates>**
- **"error": null**
- **"id": совпадает с идентификатором в запросе**

Объект <table-updates>, подробно описанный в параграфе 4.1.6, включает содержимое таблиц, для которых выбраны строки "initial". Если начальное содержимое таблиц не было запрошено, возвращается пустой объект "result".

Позднее, когда в указанные таблицы будут внесены изменения, эти обновления будут автоматически отправляться клиенту с использованием уведомлений мониторинга "update" (параграф 4.1.6). Мониторинг будет выполняться до завершения сессии JSON-RPC или получения от клиента запроса "monitor_cancel".

Каждый запрос <monitor-request> задает одно или множество полей и способ мониторинга этих полей (или всей таблицы). Элемент "columns" указывает поля для мониторинга и в нем **недопустимы** дубликаты. Если элемент "columns" опущен, мониторинг выполняется для всех полей таблицы за исключением "_uuid". Обстоятельства, при которых передается уведомление "update" для строки в таблице, определяются элементом <monitor-select> как показано ниже.

- Если "initial" отсутствует или имеет значение true, каждая строка таблицы передается как часть отклика на запрос "monitor".
- Если "insert" отсутствует или имеет значение true, уведомления "update" передаются для вставленных в таблицу строк.
- Если "delete" отсутствует или имеет значение true, уведомления "update" передаются при удалении строк.
- Если "modify" отсутствует или имеет значение true, уведомления "update" передаются при изменении строк.

Если в массиве более одного <monitor-request>, каждому запросу в массиве следует указывать "columns" и "select", при этом набор "columns" **должен** исключать совпадения.

4.1.6. Уведомление Update

Уведомления "update" передаются сервером клиенту для информирования того об изменениях в таблицах, для которых клиентом был передан запрос "monitor", описанный выше. Элементы уведомления показаны ниже.

- **"method": "update"**
- **"params": [<json-value>, <table-updates>]**
- **"id": null**

Поле <json-value> в элементе "params" совпадает со значением <json-value> в "params" соответствующего запроса "monitor". Поле <table-updates> является объектом, который отображает имя таблицы в <table-update>. Объект <table-update> отображает UUID строки на объект <row-update>. Элементы <row-update> показаны ниже.

"old": <row> присутствует для обновлений "delete" и "modify";
"new": <row> присутствует для обновлений "initial", "insert" и "modify".

Формат <row> описан в параграфе 5.1.

Каждая таблица, в которой изменилась хотя бы одна строка (или начальный вид которой был представлен), включается в <table-updates>. Каждая такая строка представляется в <table-update> как элемент с именем, взятым из элемента "_uuid" этой строки. Соответствующее значение является <row-update>.

- Элемент "old" присутствует для обновлений "delete" и "modify". Для обновлений "delete" включается каждый столбец, для которого задан мониторинг. Для обновлений "modify" включается предшествующее значение для каждого столбца, который отслеживается, если значение было изменено (столбцы, которые не изменились, указываются в "new").
- Элемент "new" присутствует для обновлений "initial", "insert" и "modify". Для "initial" и "insert" включается каждое поле, для которого задан мониторинг. Для обновлений "modify" включается новое значение в каждом отслеживаемом поле.

Отметим, что начальное представление строк не включается в уведомления об изменении, но включается в объект отклика на запрос мониторинга. Форматирование объекта <table-updates> во всех случаях одинаково.

4.1.7. Отмена мониторинга

Запрос "monitor_cancel" отменяет введенный ранее запрос мониторинга. Элементы запросы отмены показаны ниже.

- **"method": "monitor_cancel"**
- **"params": [<json-value>]**
- **"id": <nonnull-json-value>**

Поле <json-value> в элементе "params" совпадает со значением <json-value> в "params" отменяемого запроса "monitor". После этого отправка сообщений "update" для этого монитора таблиц будет прекращена. Элементы отклика показаны ниже.

- **"result": {}**
- **"error": null**
- **"id": совпадает с идентификатором в отменяемом запросе**

Если при отмене мониторинга отменяемый запрос указан некорректно, возвращается показанный ниже отклик с ошибкой.

- **"result": null**
- **"error": "unknown monitor"**
- **"id": совпадает с идентификатором в отменяемом запросе**

4.1.8. Блокировка операций

Три метода RPC - "lock", "steal" и "unlock" обеспечивают клиентам поддержку блокировки операций с БД. Сервер БД поддерживает произвольное число блокировок, каждая из которых указывается заданным клиентом идентификатором. В любой момент каждая из блокировок может иметь не более одного владельца. Точное использование блокировки определяется клиентом. Например, множество клиентов может согласовать для той или иной таблицы возможность записи только для владельца некой блокировки. Протокол OVSDB сам по себе не вносит ограничений на использование блокировок, он просто обеспечивает для каждой блокировки не более одного владельца.

Элементы запроса показаны ниже.

- **"method": "lock", "steal" или "unlock"**
- **"params": [<id>]**
- **"id": <nonnull-json-value>**

Отклик зависит от запроса и включает перечисленные ниже элементы.

- **"result": {"locked": boolean} для "lock"**
- **"result": {"locked": true} для "steal"**
- **"result": {} для "unlock"**
- **"error": null**
- **"id": совпадает с идентификатором в запросе**

Описания каждого из трех методов приведены ниже.

- "lock" — БД будет назначать этого клиента владельцем блокировки, как только она станет доступной. Если множество клиентов запрашивает одну и ту же блокировку, они будут получать ее в порядке очереди.
- "steal" — БД незамедлительно назначает клиента владельцем блокировки. Предшествующий владелем (если он был) теряет блокировку.
- "unlock" — если клиент владеет блокировкой, она освобождается. Если клиент клиент запросил владение блокировкой, запрос отменяется.

  Закрытие или иной разрыв соединения клиента с БД отменяет все его блокировки.

Для любой данной блокировки клиент **должен** чередовать операции "lock" или "steal" с операцией "unlock". Т. е. если предыдущей операцией была "lock" или "steal", за ней **должна** следовать операция "unlock" и наоборот.

Для операции "lock" элемент "locked" в объекте отклика имеет значение true, если блокировка была получена, и false, если блокировкой владеет другой клиент и запрос помещен в очередь. В последнем случае клиент будет позднее уведомлен сообщением "locked" (параграф 4.1.9) о получении блокировки.

Эти запросы выполняются быстро и на них передаются соответствующие отклики без ожидания. Уведомления "locked" и "stolen" (см. ниже) передаются асинхронно при смене владельца блокировки.

Отметим, что областью действия блокировки является сервер БД, а не база данных, размещенная на этом сервере. Клиент может реализовать соглашения по именованию типа "<db-name>__<lock-name>", которые позволяют ограничить область действия блокировки конкретной базой данных.

4.1.9. Уведомления о блокировке

Уведомление "locked" предназначено для информирования клиента о предоставлении ему блокировки, запрошенной ранее с помощью метода "lock". Элементы уведомления показаны ниже.

- `"method": "locked"`
- `"params": [<id>]`
- `"id": null`

Элемент "params" указывает блокировку, заданную в запросе "lock". Уведомляемый клиент в данный момент владеет блокировкой, указанной в "params".

Сервер БД передает такое уведомление после отклика на соответствующий запрос "lock" (но только в том случае, когда в отклике элемент "locked" имел значение false), но до отклика на последующий запрос "unlock" от этого клиента.

4.1.10. Уведомления о «краже» блокировки

Уведомление "stolen" служит для информирования клиента, владевшего блокировкой о том, что она была «украдена» другим клиентом. Элементы уведомления показаны ниже.

- `"method": "stolen"`
- `"params": [<id>]`
- `"id": null`

Уведомляемый клиент больше не владеет блокировкой, указанной в "params". Клиент по-прежнему **должен** использовать запрос "unlock" перед отправкой последующего запроса "lock" или "steal" для этой блокировки.

Если клиент изначально получил «украденную» блокировку по запросу "lock", она будет автоматически возвращена ему после освобождения укравшим блокировку клиентом (сервер БД в этом случае передаст клиенту уведомление "locked").

Если клиент изначально получил «украденную» блокировку по запросу "steal", сервер БД не будет автоматически возвращать блокировку при ее освобождении. Для повторного ее получения клиент должен передать запрос "unlock", а затем "lock" или "steal".

4.1.11. Эхо

Метод "echo" может использоваться клиентами или сервером для проверки соединения с БД и **должен** быть реализован как на серверах, так и на клиентах. Элементы запроса показаны ниже.

- `"method": "echo"`
- `"params": массив JSON с произвольным содержимым`
- `"id": <json-value>`

Объект отклика имеет показанные ниже элементы.

- `"result": совпадает с "params" в запросе`
- `"error": null`
- `"id": совпадает с идентификатором в запросе`

5. Операции с базой данных

В этом разделе описаны операции, которые могут использоваться в методе "transact", описанном в параграфе 4.1.3.

5.1. Обозначения

Ниже приведены обозначения, используемые при описании операций.

<db-name>
: Идентификатор <id>, указывающий БД. Пригодные значения <db-name> можно получить с помощью запроса "list_dbs". Значение <db-name> берется из элемента "name" в объекте <database-schema>.

<table>
: Идентификатор <id>, указывающий таблицу.

<column>
: Идентификатор <id>, указывающий поле таблицы.

<row>
: Объект JSON, который описывает строку таблицы или ее часть. Каждый элемент является именем столбца (поля) таблицы в паре со значением <value> этого столбца.

<value>
: Значение JSON, которое представляет значение поля в строке таблицы (<atom>, <set> или <map>).

<atom>
: Значение JSON, которое представляет скалярное значение для поля. Может быть <string>, <number>, <boolean>, <uuid> или <named-uuid>.

<set>

Элемент <atom>, представляющий набор с единственным элементом, или 2-элементный массив JSON, представляющий значение набора БД. Первым элементом массива должна быть строка "set", а второй должен быть массивом (возможно пустым) <atom> со значениями набора. Все элементы <atom> должны быть одного типа.

<map>

2-элементный массив JSON, представляющий значение отображения БД. Первым элементом массива должна быть строка "map", а второй должен быть массивом (возможно пустым) <pair>, задающим значения в отображении. Все <pair> должны иметь один тип ключей и значений.

Объекты JSON не применяются для представления <map>, поскольку JSON разрешает в объектах лишь имена строк.

<pair>

2-элементный массив JSON, представляющий пару в отображении БД. Первым элементом является <atom>, представляющий ключ, вторым - <atom> представляющий значение.

<uuid>

2-элементный массив JSON, представляющий UUID. Первым элементом массива должна быть строка "uuid", а второй должен быть 36-символьной строкой UUID в формате, описанном в RFC 4122 [RFC4122]. Ниже показан пример массива <uuid>, представляющего UUID 550e8400-e29b-41d4-a716-446655440000

```
["uuid", "550e8400-e29b-41d4-a716-446655440000"]
```

<named-uuid>

2-элементный массив JSON, представляющий UUID строки, вставленной операцией "insert" в той же транзакции. Первым элементом массива должна быть строка "named-uuid", а в качестве второго следует использовать <id>, заданный в качестве "uuid-name" для операции "insert" в той же транзакции. Например, если операция "insert" в транзакции задает "uuid-name" = "myrow", приведенный ниже массив <named-uuid> представляет UUID, созданный этой операцией

```
["named-uuid", "myrow"]
```

Массив <named-uuid> может использоваться в любом месте, где действует <uuid>. Это позволяет в одной транзакции создать строку, а затем ссылаться на нее, используя элемент "uuid-name", который был связан с этой строкой при ее вставке. Отметим, что "uuid-name" имеет смысл только в рамках одной транзакции.

<condition>

3-элементный массив JSON вида [<column>, <function>, <value>], который представляет проверка значения поля. Если ниже не указано иное, <value> **должно** иметь такой же тип, как <column>. Трактовка зависит от типа <column>, как показано ниже.

integer или real

В качестве <function> должно использоваться "<", "<=", "==", "!=", ">=", ">", "includes" или "excludes".

Проверка дает результат true, если значение в столбце соответствует выражению <function> <value> (например, если поле имеет значение 1, а <value> = 2, проверка даст результат true для функций "<", "<=" или "!=").

Функция "includes" эквивалентна "==", "excludes" - "!=".

boolean, string или uuid

В качестве <function> должно использоваться "!=", "==", "includes" или "excludes".

Если <function> имеет значение "==" или "includes", проверка даст результат true при совпадении значения поля с <value>. Если <function> имеет значение "!=" или "excludes", результат будет обратным.

set или map

В качестве <function> должно использоваться "!=", "==", "includes" или "excludes".

Для <function> = "==" результат будет true, если поле содержит в точности те же значения (для set) или пары (для map), что и <value>. Если в качестве функции задано "!=", результат будет обратным.

При <function> = "includes" результат будет true, если поле содержит все значения (для set) или пары (для map) из <value>. Поле может также включать другие значения или пары.

Для <function> = "excludes" результат будет true, если поле не содержит ни одного значения (для set) или пары (для map) из <value>. Поле может также включать другие значения или пары, которых нет в <value>.

Если в качестве <function> задано "includes" или "excludes", требования к типу <value> несколько смягчаются и он может иметь число элементов меньше минимального для типа поля. Для <function> = "excludes", требования к типу дополнительно смягчаются, позволяя <value> иметь число элементов, превышающее максимум для типа поля.

<function>

Один из элементов "<", "<=", "==", "!=", ">=", ">", "includes" или "excludes".

<mutation>

3-элементный массив JSON вида [<column>, <mutator>, <value>], который представляет смену значения поля. Если ниже не указано иное, <value> **должно** иметь такой же тип, как <column>. Трактовка зависит от типа <column>, как показано ниже.

integer или real

Элемент <mutator> должен быть "+=", "-=", "*=", "/=", или (только для целых чисел) "%=". Значение <column> меняется на сумму, разность, произведение, частное от деления или остаток для элементов <column> и <value>.

Ограничения для <column> игнорируются при анализе <value>.

boolean, string или uuid

Для этого типа в настоящее время нет действующего значения <mutator>.

set

Любой оператор <mutator>, пригодный для типа элементов в наборе, может быть применен к набору и в этом случае изменение выполняется индивидуально для каждого элемента в наборе. В качестве <value> должно применяться скалярное значение того же типа, какой имеют элементы набора за исключением того, что ограничения игнорируются при анализе <value>.

Если <mutator> = "insert", каждое из значения в наборе из <value> добавляется в <column>, если оно там еще не присутствует. Требования к типу <value> несколько смягчены, он может иметь меньшее число элементов, чем заданный для типа столбца минимум.

Если <mutator> = "delete", каждое из значения в наборе из <value> удаляется из <column>, если оно там есть. Требования к типу <value> несколько смягчены, он может иметь меньшее или большее число элементов, чем заданный для типа столбца максимум.

map

В качестве <mutator> должно задаваться "insert" или "delete".

Если <mutator> = "insert", каждая пара key-value в отображении из <value> добавляется в <column>, если оно там еще не присутствует. Требования к типу <value> несколько смягчены, он может иметь меньшее число элементов, чем заданный для типа столбца минимум.

Если <mutator> = "delete", <value> может иметь то же тип, что и <column> (тип отображения) или может быть набором, элементы которого имеют такой же тип, ак ключ <column>:

- если <value> - это отображение, изменение состоит в удалении из <column> каждой пары key-value, для которой ключ и значение совпадают с одной из пар key-value в <value>;
- если <value> - это набор, , изменение состоит в удалении из <column> каждой пары key-value, для которой ключ совпадает с обним из значений в <value>.

Для "delete" поле <value> может включать любое число элементов, независимо от ограничений на число элементов в <column>.

<mutator>

Один из элементов "+=", "-=", "*=", "/=", "%=", "insert" или "delete".

5.2. Операции

В последующих параграфах описаны операции, которые могут быть выполнены как часть запроса RPC "transact" (параграф 4.1.3). Каждая из этих операция является объектом JSON, который может быть включен в качестве одного из элементов в массив "params" запроса "transact". Детали каждого объекта, семантика, результаты и возможные ошибки описаны ниже.

5.2.1. Вставка

Объект "insert" содержит перечисленные ниже элементы.

```
"op": "insert"          обязательно
"table": <table>        обязательно
"row": <row>            обязательно
"uuid-name": <id>       необязательно
```

Сооьветствующий объект результата имеет единственный элемент

```
"uuid": <uuid>
```

Операция вставляет строку "row" в таблицу "table". Если "row" на задает значения для всех столбцов (полей) в "table", пропущенные поля получают принятые по умолчанию значения, которые зависят от типа столбца. Поля, в которых <type> задает "min" = 0, по умолчанию остаются пустыми. В остальных случаях по умолчанию используется одно значение или пара key-value, определяемые <atomic-type>:

- `"integer" или "real": 0`
- `"boolean": false`
- `"string": "" (пустая строка)`
- `"uuid": 00000000-0000-0000-0000-000000000000`

Новая строка получает случайное значение UUID. При заданном "uuid-name" возникнет ошибка, если значение <id> не будет уникальным среди "uuid-name", представленных во всех операциях "insert" данной транзакции. UUID для новой строки возвращается в качестве элемента "uuid" в объекте результата.

Возможные ошибки при этой операции приведены ниже.

"error": "duplicate uuid-name"

Одно значение "uuid-name" указано в нескольких операциях "insert" данной транзакции.

"error": "constraint violation"

Одно из значений в "row" не соответствует ограничениям для <base-type> в полях. Эта ошибка может возникать для столбцов, которые не заданы явно в "row", если принятое по умолчанию значение не соответствует ограничениям для столбца.

5.2.2. Выбор

Ниже показаны элементы объекта "select".

```
"op": "select"              обязательно
"table": <table>            обязательно
"where": [<condition>*]     обязательно
"columns": [<column>*]      необязательно
```

Соответствующий отклик имеет единственный элемент

```
"rows": [<row>*]
```

Операция выполняет в таблице "table" поиск всех строк, которые соответствуют условиям, заданным в "where". Если "where" является пустым массивом, будет выбрана каждая строка "table".

Элемент "rows" в результате является массивом объектов, каждый из которых представляет соответствующую условиям поиска строку с полями, заданными в элементом "columns", при этом имя столбца служит именем элемента, а значение — значением элемента. Если элемент "columns", включаются все столбцы из таблицы (в том числе "_uuid" и "_version"). Если две строки результата имеют совпадающие значения всех включенных полей, в "rows" помещается только один экземпляр. Задание "_uuid" в элементе "columns" предотвратит появление таких дубликатов, поскольку каждая строка имеет уникальное значение UUID.

Порядок строк в "rows" не задается.

5.2.3. Обновление

Ниже приведены элементы объекта "update".

```
"op": "update"                  обязательно
"table": <table>                обязательно
"where": [<condition>*]         обязательно
"row": <row>                    обязательно
```

Соответствующий объект отклика содержит единственный элемент

```
"count": <integer>
```

Операция обновляет строки таблицы "table", соответствующие заданным элементом "where" условиям. Для каждой найденной строки изменяются значения во всех полях, указанных в "row", на значения, заданные этим же элементом "row". Значения "_uuid" и "_version" данная операция не может обновлять напрямую. Столбцы, открытые лишь для чтения, также не могут обновляться.

Элемент "count" в объекте результата указывает число соответствующих заданным условиям строк таблицы.

При выполнении операции может возникать ошибка, показанная ниже.

"error": "constraint violation"
Одно из значений в элементе "row" не соответствует применяемым сразу же ограничениям для <base-type> столбца.

5.2.4. Изменение

Элементы объекта "mutate" перечислены ниже

```
"op":  "mutate"                 обязательно
"table": <table>                обязательно
"where": [<condition>*]         обязательно
"mutations": [<mutation>*]      обязательно
```

Соответствующий объект отклик включает один элемент

```
"count": <integer>
```

Операция изменяет строки таблицы, проводя в "table" поиск строк, соответствующих условиям, заданным в "where". Для каждой соответствующей строки операция меняет поля в соответствии с каждым <mutation> элемента "mutations" в указанном порядке.

Столбцы "_uuid" и "_version" в таблице не могут напрямую изменяться этой операцией. Столбцы, открытые лишь для чтения, также не могут быть изменены.

Элемент "count" в объекте результата указывает число соответствующих заданным условиям строк таблицы.

При выполнении операции могут возникать перечисленные ниже ошибки.

"error": "domain error"
Результат изменения не определен математически (например, деление на 0).
"error": "range error"
Результат изменения не может быть представлен в формате БД (например, целое число выходит за пределы диапазона INT64_MIN...INT64_MAX или действительное часло выходит за пределы -DBL_MAX...DBL_MAX).
"error": "constraint violation"
Изменение ведет к нарушению заданных для столбца ограничений (например, столбец может иметь число значений больше или меньше дозволенного предела, арифметическая информация приводит к дубликатам в set или map, нарушаются ограничения, заданные для столбца <base-type>).

5.2.5. Удаление

Элементы объекта "delete" перечислены ниже.

```
"op":  "delete"                 обязательно
"table": <table>                обязательно
"where": [<condition>*]         обязательно
```

Соответствующий объект результата имеет единственный элемент.

```
"count": <integer>
```

Операция удаляет из таблицы "table" строки, соответствующие услоиям, заданным в "where". Элемент "count" показывает число удаленных из таблицы строк.

5.2.6. Ожидание

Ниже перечислены элементы объекта "wait".

```
"op": "wait"                    обязательно
"timeout": <integer>            необязательно
"table": <table>                обязательно
"where": [<condition>*]         обязательно
"columns": [<column>*]          обязательно
"until": "==" or "!="           обязательно
"rows": [<row>*]                обязательно
```

Операция не имеет объекта для результата.

Операция инициирует ожидание соблюдения заданных условий.

Если элемент "until" имеет значение "==", ожидание длится до того момента, когда запрос к таблице "table", заданный элементами "where" и "columns" (см. описание операции "select"), вернет в результате набор, заданный элементом "rows". В этом случае ожидание завершается успешно, в остальных случаях транзакция отменяется полностью (откат к

прежнему состоянию). Она автоматически запускается позднее, когда изменения в БД сделают возможным выполнение операции. Клиент не получит отклика, пока операция не завершится успехом или отказом.

Если элемент "until" имеет значение "!=", смысл проверки обращается. Т. Е пока запрос к "table", заданный элементами "where" и "columns", возвращает "rows", транзакция будет «откатываться» назад и автоматически повторяться позднее.

Если задан элемент "timeout", транзакция прерывается по истечении указанного числа миллисекунд. Гарантируется хотя бы обна попутка выполнения транзакции до ее прерывания, "timeout" = 0 будет прерывать транзакцию при первом несоответствии.

Операция может приводить к возврату показанной ниже ошибки.

"error": "timed out"
Прошло время, заданное элементом "timeout", а транзакцию выполнить не удалось.

5.2.7. Фиксация

Ниже показаны элементы объекта "commit".

```
"op": "commit"                    обязательно
"durable": <boolean>              обязательно
```

Операция не имеет объекта для результата.

Если задано "durable" = true, транзакция при ее фиксации будет сохраняться долговременно (на диске) до того, как клиенту будет отправлен отклик. Эта операция с "durable" = false эффективно соответствует отсутствию операции.

Операция может приводить к возврату показанной ниже ошибки.

"error": "not supported"
Задано "durable" = true, а данная реализация БД не поддерживает долговременной фиксации.

5.2.8. Прерывание

Элементы объекта "abort" перечислены ниже.

```
"op": "abort"                     обязательно
```

Операция не имеет объекта для результата (операция не может приводить к успеху).

Эта операция прерывает выполнение транзакции с возвратом ошибки. Это может быть полезно для тестов.

Операция может приводить к возврату показанной ниже ошибки.

"error": "aborted"
Данная операция всегда возвращает такую ошибку.

5.2.9. Комментарий

Элементы объекта "comment" приведены ниже.

```
"op": "comment"                   обязательно
"comment": <string>               обязательно
```

Операция не имеет объекта для результата.

Операция предоставляет администратору БД информацию о цели транзакции. реализация ovsdb-server, например, добавляет комментарии в транзакции, которые меняют БД в журнал базы данных. Это может помочь при отладке (например, когда множество клиентов записывает данные в базу). Примером может служить консольная программа ovs-vsctl, которая взаимодействует с сервером ovsdb-server. При выполнении операций с БД она включает вызванную команду (например, "ovs-vsctl add-br br0") в комментарий транзакции, который можно увидеть в журнальном файле вместе с изменениями, внесенными в таблицы БД.

5.2.10. Защита прав владения

Ниже перечислены элементы объекта "assert".

```
"op": "assert"                    обязательно
"lock": <id>                      обязательно
```

Объект результата не имеет элементов.

Операция assert вызывает прерывание транзакции, если клиент не владеет блокировкой, указанной элементом <id>.

Операция может приводить к возврату показанной ниже ошибки.

"error": "not owner"
Клиент не владеет указанной блокировкой.

6. Взаимодействие с IANA

Агентство IANA выделило порт TCP 6640 для этого протокола. Ранние реализации OVSDB используют другой номер порта, но совместимым со спецификацией реализациям следует применять выделенный IANA номер порта.

Агентство IANA обновило ссылку на порт 6640 с указанием данного документа.

7. Вопросы безопасности

Основным вопросом безопасности, который требуется решить для протокола OVSDB, является проверка подлинности, защита целостности и конфиденциальности при взаимодействии между клиентами и серверами, реализующими протокол. Для обеспечения такой защиты в соединениях OVSDB **следует** применять протокол TLS[1] [RFC5246]. Точные детали взаимной проверки подлинности клиентом и сервером зависят от рабочей среды. Зачастую клиенты и серверы

[1]Transport Layer Security — защита транспортного уровня.

OVSDB размещаются в одной контролируемой среде, например, на машинах одного ЦОД, где взаимодействие между ними может быть организовано через изолированную сеть управления.

8. Благодарности

Спасибо Jeremy Stribling и Justin Pettit за их полезные предложения.

9. Литература

9.1. Нормативные документы

[DCE] "DCE: Remote Procedure Call", Open Group CAE Specification C309[1], ISBN 1-85912-041-5, August 1994.

[JSON-RPC] "JSON-RPC Specification, Version 1.0"[2], <http://json-rpc.org/wiki/specification>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

9.2. Дополнительная литература

[DB-SCHEMA] "Open vSwitch Database Schema", <http://openvswitch.org/ovs-vswitchd.conf.db.5.pdf>.

[OF-SPEC] Open Networking Foundation, "OpenFlow Switch Specification, version 1.3.3", October 2013, <https://www.opennetworking.org>.

[OVS] "Open vSwitch", <http://openvswitch.org/>.

Адреса авторов

Ben Pfaff

VMware, Inc.

3401 Hillview Ave.

Palo Alto, CA 94304

USA

EMail: blp@nicira.com

Bruce Davie (редактор)

VMware, Inc.

3401 Hillview Ave.

Palo Alto, CA 94304

USA

EMail: bsd@nicira.com

Перевод на русский язык

Николай Малых

nmalykh@gmail.com

[1]Версия 1.1 (C706) доступна по ссылке. *Прим. перев.*
[2]В настоящее время по приведенной ссылке доступна также спецификация версии 2.0. *Прим. перев.*