NAME

ovsdb-server – Open vSwitch Database Server Protocol

DESCRIPTION

ovsdb−server implements the Open vSwitch Database (OVSDB) protocol specified in RFC 7047. This document provides clarifications for how **ovsdb−server** implements the protocol and describes the extensions that it provides beyond RFC 7047. Numbers in section headings refer to corresponding sections in RFC 7047.

3.1 JSON Usage

RFC 4627 says that names within a JSON object should be unique. The Open vSwitch JSON parser discards all but the last value for a name that is specified more than once.

The definition of <error> allows for implementation extensions. Currently **ovsdb−server** uses the following additional **error** strings (which might change in later releases):

syntax error or unknown column
: The request could not be parsed as an OVSDB request. An additional **syntax** member, whose value is a string that contains JSON, may narrow down the particular syntax that could not be parsed.

internal error
: The request triggered a bug in **ovsdb−server**.

ovsdb error
: A map or set contains a duplicate key.

permission error
: The request was denied by the role−based access control extension, introduced in version 2.8.

3.2 Schema Format

RFC 7047 requires the **version** field in <database−schema>. Current versions of **ovsdb−server** allow it to be omitted (future versions are likely to require it).

RFC 7047 allows columns that contain weak references to be immutable. This raises the issue of the behavior of the weak reference when the rows that it references are deleted. Since version 2.6, **ovsdb−server** forces columns that contain weak references to be mutable.

Since version 2.8, the table name **RBAC_Role** is used internally by the role−based access control extension to **ovsdb−server** and should not be used for purposes other than defining mappings of role names to table access permissions. This table has one row per role name and the following columns:

name The role name.

permissions
: A map of table name to a reference to a row in a separate permission table.

The separate RBAC permission table has one row per access control configuration and the following columns:

name The name of the table to which the row applies.

authorization
: The set of column names and column:key pairs to be compared with the client ID in order to determine the authorization status of the requested operation.

insert_delete
: A boolean value, true if authorized insertions and deletions are allowed, false if no insertions or deletions are allowed.

update The set of columns and column:key pairs for which authorized update and mutate operations should be permitted.

4 Wire Protocol

The original OVSDB specifications included the following reasons, omitted from RFC 7047, to operate JSON−RPC directly over a stream instead of over HTTP:

- JSON−RPC is a peer−to−peer protocol, but HTTP is a client−server protocol, which is a poor match. Thus, JSON−RPC over HTTP requires the client to periodically poll the server to receive server requests.
- HTTP is more complicated than stream connections and doesn't provide any corresponding advantage.
- The JSON−RPC specification for HTTP transport is incomplete.

4.1.3 Transact

Since version 2.8, role−based access controls can be applied to operations within a transaction that would modify the contents of the database (these operations include row insert, row delete, column update, and column mutate). Role−based access controls are applied when the database schema contains a table with the name **RBAC_Role** and the connection on which the transaction request was received has an associated role name (from the **role** column in the remote connection table). When role−based access controls are enabled, transactions that are otherwise well−formed may be rejected depending on the client's role, ID, and the contents of the **RBAC_Role** table and associated permissions table.

4.1.5 Monitor

For backward compatibility, **ovsdb−server** currently permits a single <monitor−request> to be used instead of an array; it is treated as a single−element array. Future versions of **ovsdb−server** might remove this compatibility feature.

Because the <json−value> parameter is used to match subsequent update notifications (see below) to the request, it must be unique among all active monitors. **ovsdb−server** rejects attempt to create two monitors with the same identifier.

When a given client sends a **transact** request that changes a table that the same client is monitoring, **ovsdb−server** always sends the **update** (or **update2** or **update3**) for these changes before it sends the reply to the **transact** request. Thus, when a client receives a **transact** reply, it can know immediately what changes (if any) the transaction made. (If ovsdb−server might use the other order, then a client that wishes to act on based on the results of its own transactions would not know when this was guaranteed to have taken place.)

4.1.7 Monitor Cancellation

When a database monitored by a session is removed, and database change awareness is enabled for the session (see Section 4.1.16), the database server spontaneously cancels all monitors (including conditional monitors described in Section 4.1.12) for the removed database. For each canceled monitor, it issues a notification in the following form:

```
"method": "monitor_canceled"
"params": [<json-value>]
"id": null
```

4.1.12 Monitor_cond

A new monitor method added in Open vSwitch version 2.6. The **monitor_cond** request enables a client to replicate subsets of tables within an OVSDB database by requesting notifications of changes to rows matching one of the conditions specified in **where** by receiving the specified contents of these rows when table updates occur. **monitor_cond** also allows a more efficient update notifications by receiving <table−updates2> notifications (described below).

The **monitor** method described in Section 4.1.5 also applies to **monitor_cond**, with the following exceptions:

- RPC request method becomes **monitor_cond**.
- Reply result follows <table−updates2>, described in Section 4.1.14.
- Subsequent changes are sent to the client using the **update2** monitor notification, described in Section 4.1.14
- Update notifications are being sent only for rows matching [<condition>*].

The request object has the following members:

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

The <json−value> parameter is used to match subsequent update notifications (see below) to this request. The <monitor−cond−requests> object maps the name of the table to an array of <monitor−cond−request>.

Each <monitor−cond−request> is an object with the following members:

```
"columns": [<column>*]            optional
"where": [<condition>*]           optional
"select": <monitor-select>        optional
```

The **columns**, if present, define the columns within the table to be monitored that match conditions. If not present, all columns are monitored.

The **where**, if present, is a JSON array of <condition> and boolean values. If not present or condition is an empty array, implicit True will be considered and updates on all rows will be sent.

<monitor−select> is an object with the following members:

```
"initial": <boolean>              optional
"insert": <boolean>               optional
"delete": <boolean>               optional
"modify": <boolean>               optional
```

The contents of this object specify how the columns or table are to be monitored as explained in more detail below.

The response object has the following members:

```
"result": <table-updates2>
"error": null
"id": same "id" as request
```

The <table−updates2> object is described in detail in Section 4.1.14. It contains the contents of the tables for which initial rows are selected. If no tables initial contents are requested, then **result** is an empty object.

Subsequently, when changes to a specified table that match one of the conditions in <monitor−cond−request> are committed, the changes are automatically sent to the client using the **update2** monitor notification (see Section 4.1.14). This monitoring persists until the JSON−RPC session terminates or until the client sends a **monitor_cancel** JSON−RPC request.

Each <monitor−cond−request> specifies one or more conditions and the manner in which the rows that

match the conditions are to be monitored. The circumstances in which an **update** notification is sent for a row within the table are determined by <monitor−select>:

- If **initial** is omitted or true, every row in the original table that matches one of the conditions is sent as part of the response to the **monitor_cond** request.
- If **insert** is omitted or true, update notifications are sent for rows newly inserted into the table that match conditions or for rows modified in the table so that their old version does not match the condition and new version does.
- If **delete** is omitted or true, update notifications are sent for rows deleted from the table that match conditions or for rows modified in the table so that their old version does match the conditions and new version does not.
- If **modify** is omitted or true, update notifications are sent whenever a row in the table that matches conditions in both old and new version is modified.

Both **monitor** and **monitor_cond** sessions can exist concurrently. However, **monitor** and **monitor_cond** shares the same <json−value> parameter space; it must be unique among all **monitor** and **monitor_cond** sessions.

4.1.13 Monitor_cond_change

The **monitor_cond_change** request enables a client to change an existing **monitor_cond** replication of the database by specifying a new condition and columns for each replicated table. Currently changing the columns set is not supported.

The request object has the following members:

```
"method": "monitor_cond_change"
"params": [<json-value>, <json-value>, <monitor-cond-update-requests>]
"id": <nonnull-json-value>
```

The <json−value> parameter should have a value of an existing conditional monitoring session from this client. The second <json−value> in params array is the requested value for this session. This value is valid only after **monitor_cond_change** is committed. A user can use these values to distinguish between update messages before conditions update and after. The <monitor−cond−update−requests> object maps the name of the table to an array of <monitor−cond−update−request>. Monitored tables not included in <monitor−cond−update−requests> retain their current conditions.

Each <monitor−cond−update−request> is an object with the following members:

```
"columns": [<column>*]          optional
"where": [<condition>*]         optional
```

The **columns** specify a new array of columns to be monitored, although this feature is not yet supported.

The **where** specify a new array of conditions to be applied to this monitoring session.

The response object has the following members:

```
"result": {}
"error": null
"id": same "id" as request
```

Subsequent <table−updates2> notifications are described in detail in Section 4.1.14 in the RFC. If insert contents are requested by original monitor_cond request, <table−updates2> will contain rows that match the new condition and do not match the old condition. If deleted contents are requested by origin monitor

request, <table−updates2> will contain any matched rows by old condition and not matched by the new condition.

Changes according to the new conditions are automatically sent to the client using the **update2** or **update3** monitor notification depending on the monitor method. An update, if any, as a result of a condition change, will be sent to the client before the reply to the **monitor_cond_change** request.

4.1.14 Update2 notification

The **update2** notification is sent by the server to the client to report changes in tables that are being monitored following a **monitor_cond** request as described above. The notification has the following members:

```
"method": "update2"
"params": [<json-value>, <table-updates2>]
"id": null
```

The <json−value> in **params** is the same as the value passed as the <json−value> in **params** for the corresponding **monitor** request. <table−updates2> is an object that maps from a table name to a <table−update2>. A <table−update2> is an object that maps from row's UUID to a <row−update2> object. A <row−update2> is an object with one of the following members:

"initial": <row>
: present for **initial** updates

"insert": <row>
: present for **insert** updates

"delete": <row>
: present for **delete** updates

"modify": <row>"
: present for **modify** updates

The format of <row> is described in Section 5.1.

<row> is always a null object for a **delete** update. In **initial** and **insert** updates, <row> omits columns whose values equal the default value of the column type.

For a **modify** update, <row> contains only the columns that are modified. <row> stores the difference between the old and new value for those columns, as described below.

For columns with single value, the difference is the value of the new column.

The difference between two sets are all elements that only belong to one of the sets.

The difference between two maps are all key−value pairs whose keys appears in only one of the maps, plus the key−value pairs whose keys appear in both maps but with different values. For the latter elements, <row> includes the value from the new column.

Initial views of rows are not presented in update2 notifications, but in the response object to the **monitor_cond** request. The formatting of the <table−updates2> object, however, is the same in either case.

4.1.15 Monitor_cond_since

A new monitor method added in Open vSwitch version 2.12. The **monitor_cond_since** request enables a client to request changes that happened after a specific transaction id. A client can use this feature to request only latest changes after a server connection reset instead of re−transfer all data from the server again.

The **monitor_cond** method described in Section 4.1.12 also applies to **monitor_cond_since**, with the following exceptions:

- RPC request method becomes **monitor_cond_since**.
- Reply result includes extra parameters.
- Subsequent changes are sent to the client using the **update3** monitor notification, described in Section 4.1.16

The request object has the following members:

```
"method": "monitor_cond_since"
"params": [<db-name>, <json-value>, <monitor-cond-requests>, <last-txn-id>]
"id": <nonnull-json-value>
```

The <last−txn−id> parameter is the transaction id that identifies the latest data the client already has, and it requests server to send changes AFTER this transaction (exclusive).

All other parameters are the same as **monitor_cond** method.

The response object has the following members:

```
"result": [<found>, <last-txn-id>, <table-updates2>]
"error": null
"id": same "id" as request
```

The <found> is a boolean value that tells if the <last−txn−id> requested by client is found in server's history or not. If true, the changes after that version up to current is sent. Otherwise, all data is sent.

The <last−txn−id> is the transaction id that identifies the latest transaction included in the changes in <table−updates2> of this response, so that client can keep tracking. If there is no change involved in this response, it is the same as the <last−txn−id> in the request if <found> is true, or zero uuid if <found> is false. If the server does not support transaction uuid, it will be zero uuid as well.

All other parameters are the same as in response object of **monitor_cond** method.

Like in **monitor_cond**, subsequent changes that match conditions in <monitor−cond−request> are automatically sent to the client, but using **update3** monitor notification (see Section 4.1.16), instead of **update2**.

4.1.16 Update3 notification

The **update3** notification is sent by the server to the client to report changes in tables that are being monitored following a **monitor_cond_since** request as described above. The notification has the following members:

```
"method": "update3"
"params": [<json-value>, <last-txn-id>, <table-updates2>]
"id": null
```

The <last−txn−id> is the same as described in the response object of **monitor_cond_since**.

All other parameters are the same as in **update2** monitor notification (see Section 4.1.14).

4.1.17 Get Server ID

A new RPC method added in Open vSwitch version 2.7. The request contains the following members:

```
"method": "get_server_id"
"params": null
"id": <nonnull-json-value>
```

The response object contains the following members:

```
"result": "<server_id>"
"error": null
"id": same "id" as request
```

<server_id> is JSON string that contains a UUID that uniquely identifies the running OVSDB server process. A fresh UUID is generated when the process restarts.

4.1.18 Database Change Awareness

RFC 7047 does not provide a way for a client to find out about some kinds of configuration changes, such as about databases added or removed while a client is connected to the server, or databases changing between read/write and read−only due to a transition between active and backup roles. Traditionally, **ovsdb−server** disconnects all of its clients when this happens, because this prompts a well−written client to reassess what is available from the server when it reconnects.

OVS 2.9 provides a way for clients to keep track of these kinds of changes, by monitoring the **Database** table in the **_Server** database introduced in this release (see **ovsdb−server(5)** for details). By itself, this does not suppress **ovsdb−server** disconnection behavior, because a client might monitor this database without understanding its special semantics. Instead, **ovsdb−server** provides a special request:

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

If the boolean in the request is true, it suppresses the connection−closing behavior for the current connection, and false restores the default behavior. The reply is always the same:

```
"result": {}
"error": null
"id": same "id" as request
```

4.1.19 Schema Conversion

Open vSwitch 2.9 adds a new JSON−RPC request to convert an online database from one schema to another. The request contains the following members:

```
"method": "convert"
"params": [<db-name>, <database-schema>]
"id": <nonnull-json-value>
```

Upon receipt, the server converts database <db−name> to schema <database−schema>. The schema's name must be <db−name>. The conversion is atomic, consistent, isolated, and durable. The data in the database must be valid when interpreted under <database−schema>, with only one exception: data for tables and columns that do not exist in the new schema are ignored. Columns that exist in <database−schema> but not in the database are set to their default values. All of the new schema's constraints apply in full.

If the conversion is successful, the server notifies clients that use the **set_db_change_aware** RPC introduced in Open vSwitch 2.9 and cancels their outstanding transactions and monitors. The server disconnects other clients, enabling them to notice the change when they reconnect. The server sends the following reply:

```
"result": {}
"error": null
"id": same "id" as request
```

If the conversion fails, then the server sends an error reply in the following form:

```
"result": null
"error": [<error>]
"id": same "id" as request
```

5.1 Notation

For <condition>, RFC 7047 only allows the use of **!=**, **==**, **includes**, and **excludes** operators with set types. Open vSwitch 2.4 and later extend <condition> to allow the use of **<**, **<=**, **>=**, and **>** operators with a column with type "set of 0 or 1 integer" and an integer argument, and with "set of 0 or 1 real" and a real argument. These conditions evaluate to false when the column is empty, and otherwise as described in RFC 7047 for integer and real types.

<condition> is specified in Section 5.1 in the RFC with the following change: A condition can be either a 3−element JSON array as described in the RFC or a boolean value. In case of an empty array an implicit true boolean value will be considered.

5.2.1 Insert

As an extension, Open vSwitch 2.13 and later allow an optional **uuid** member to specify the UUID for the new row. The specified UUID must be unique within the table when it is inserted and not the UUID of a row previously deleted within the transaction. If the UUID violates these rules, then the operation fails with a **duplicate uuid** error.

5.2.6 Wait, 5.2.7 Commit, 5.2.9 Comment

RFC 7047 says that the **wait**, **commit**, and **comment** operations have no corresponding result object. This is not true. Instead, when such an operation is successful, it yields a result object with no members.

AUTHOR

The Open vSwitch Development Community

COPYRIGHT