

Introduction

wal2json is an output plugin for logical decoding. It means that the plugin have access to tuples produced by INSERT and UPDATE. Also, UPDATE/DELETE old row versions can be accessed depending on the configured replica identity. Changes can be consumed using the streaming protocol (logical replication slots) or by a special SQL API.

format version 1 produces a JSON object per transaction. All of the new/old tuples are available in the JSON object. Also, there are options to include properties such as transaction timestamp, schema-qualified, data types, and transaction ids.

format version 2 produces a JSON object per tuple. Optional JSON object for beginning and end of transaction. Also, there are a variety of options to include properties.

Build and Install

This extension is supported on [those platforms](#) that PostgreSQL is. The installation steps depend on your operating system. [PostgreSQL yum repository](#) and [PostgreSQL apt repository](#) provide wal2json packages.

In Red Hat/CentOS:

```
$ sudo yum install wal2json12
```

In Debian/Ubuntu:

```
$ sudo apt-get install postgresql12-wal2json
```

You can also keep up with the latest fixes and features cloning the Git repository.

```
$ git clone https://github.com/eulerto/wal2json.git
```

Unix based Operating Systems

Before installing **wal2json**, you should have PostgreSQL 9.4+ installed (including the header files). If PostgreSQL is not in your search path, add it. If you are using [PostgreSQL yum repository](#), install `postgresql12-devel` and add `/usr/pgsql-12/bin` to your search path (yum uses 12, 11, 10, 96, 95 or 94). If you are using [PostgreSQL apt repository](#), install `postgresql-server-dev-12` and add `/usr/lib/postgresql/12/bin` to your search path. (apt uses 12, 11, 10, 9.6, 9.5 or 9.4).

If you compile PostgreSQL by yourself and install it in `/home/euler/pg12`:

```
$ tar -zxf wal2json-wal2json_2_2.tar.gz
$ cd wal2json-wal2json_2_2
$ export PATH=/home/euler/pg12/bin:$PATH
$ make
$ make install
```

If you are using [PostgreSQL yum repository](#):

```
$ sudo yum install postgresql12-devel
$ tar -zxf wal2json-wal2json_2_2.tar.gz
$ cd wal2json-wal2json_2_2
$ export PATH=/usr/pgsql-12/bin:$PATH
$ make
$ make install
```

If you are using [PostgreSQL apt repository](#):

```
$ sudo apt-get install postgresql-server-dev-12
$ tar -zxf wal2json-wal2json_2_2.tar.gz
$ cd wal2json-wal2json_2_2
$ export PATH=/usr/lib/postgresql/12/bin:$PATH
$ make
$ make install
```

Windows

There are several ways to build **wal2json** on Windows. If you are build PostgreSQL too, you can put **wal2json** directory inside contrib, change the contrib Makefile (variable SUBDIRS) and build it following the [Installation from Source Code on Windows](#) instructions. However, if you already have PostgreSQL installed, it is also possible to compile **wal2json** out of the tree. Edit `wal2json.vcxproj` file and change `c:\postgres\pg103` to the PostgreSQL prefix directory. The next step is to open this project file in MS Visual Studio and compile it. Final step is to copy `wal2json.dll` to the `pg_config --pkglibdir` directory.

Configuration

postgresql.conf

You need to set up at least two parameters at `postgresql.conf`:

```
wal_level = logical
#
# these parameters only need to set in versions 9.4, 9.5 and 9.6
# default values are ok in version 10 or later
#
max_replication_slots = 10
max_wal_senders = 10
```

After changing these parameters, a restart is needed.

Parameters

- `include-xids`: add *xid* to each changeset. Default is *false*.
- `include-timestamp`: add *timestamp* to each changeset. Default is *false*.
- `include-schemas`: add *schema* to each change. Default is *true*.
- `include-types`: add *type* to each change. Default is *true*.
- `include-typmod`: add modifier to types that have it (eg. `varchar(20)` instead of `varchar`). Default is *true*.
- `include-type-oids`: add type oids. Default is *false*.
- `include-domain-data-type`: replace domain name with the underlying data type. Default is *false*.
- `include-column-positions`: add column position (`pgattribute.attnum_`). Default is *false*.
- `include-not-null`: add *not null* information as *columnoptionals*. Default is *false*.
- `include-default`: add default expression. Default is *false*.
- `include-pk`: add *primary key* information as *pk*. Column name and data type is included. Default is *false*.
- `pretty-print`: add spaces and indentation to JSON structures. Default is *false*.
- `write-in-chunks`: write after every change instead of every changeset. Default is *false*.
- `include-lsn`: add *nextlsn* to each changeset. Default is *false*.
- `include-unchanged-toast` (deprecated): Don't use it. It is deprecated.
- `filter-origins`: exclude changes from the specified origins. Default is empty which means that no origin will be filtered. It is a comma separated value.
- `filter-tables`: exclude rows from the specified tables. Default is empty which means that no table will be filtered. It is a comma separated value. The tables should be schema-qualified. `*.foo` means table `foo` in all schemas and `bar.*` means all tables in schema `bar`. Special characters (space, single quote, comma, period, asterisk) must be escaped with backslash. Schema and table are case-sensitive. Table `"public"."Foo bar"` should be specified as `public.Foo\ bar`.

- **add-tables**: include only rows from the specified tables. Default is all tables from all schemas. It has the same rules from **filter-tables**.
- **filter-msg-prefixes**: exclude messages if prefix is in the list. Default is empty which means that no message will be filtered. It is a comma separated value.
- **add-msg-prefixes**: include only messages if prefix is in the list. Default is all prefixes. It is a comma separated value. **wal2json** applies **filter-msg-prefixes** before this parameter.
- **format-version**: defines which format to use. Default is *1*.
- **actions**: define which operations will be sent. Default is all actions (insert, update, delete, and truncate). However, if you are using **format-version** 1, truncate is not enabled (backward compatibility).

Examples

There are two ways to obtain the changes (JSON objects) from **wal2json** plugin: calling functions via SQL or `pg_recvlogical`.

`pg_recvlogical`

Besides the configuration above, it is necessary to configure a replication connection to use `pg_recvlogical`. A logical replication connection in version 9.4, 9.5, and 9.6 requires **replication** keyword in the database column. Since version 10, logical replication matches a normal entry with a database name or keywords such as **all**.

First, add a replication connection rule at `pg_hba.conf` (9.4, 9.5, and 9.6):

```
local    replication    myuser                trust
```

If you are using version 10 or later:

```
local    mydatabase     myuser                trust
```

Also, set `max_wal_senders` at `postgresql.conf`:

```
max_wal_senders = 1
```

A restart is necessary if you changed `max_wal_senders`.

You are ready to try **wal2json**. In one terminal:

```
$ pg_recvlogical -d postgres --slot test_slot --create-slot -P wal2json
$ pg_recvlogical -d postgres --slot test_slot --start -o pretty-print=1 -f -
```

In another terminal:

```
$ cat /tmp/example1.sql
CREATE TABLE table_with_pk (a SERIAL, b VARCHAR(30), c TIMESTAMP NOT NULL, PRIMARY KEY(a, c));
CREATE TABLE table_without_pk (a SERIAL, b NUMERIC(5,2), c TEXT);

BEGIN;
INSERT INTO table_with_pk (b, c) VALUES('Backup and Restore', now());
INSERT INTO table_with_pk (b, c) VALUES('Tuning', now());
INSERT INTO table_with_pk (b, c) VALUES('Replication', now());
DELETE FROM table_with_pk WHERE a < 3;

INSERT INTO table_without_pk (b, c) VALUES(2.34, 'Tapir');
-- it is not added to stream because there isn't a pk or a replica identity
UPDATE table_without_pk SET c = 'Anta' WHERE c = 'Tapir';
COMMIT;
```

```
$ psql -At -f /tmp/example1.sql postgres
CREATE TABLE
CREATE TABLE
BEGIN
INSERT 0 1
INSERT 0 1
INSERT 0 1
DELETE 2
INSERT 0 1
UPDATE 1
COMMIT
```

The output in the first terminal is:

```
{
  "change": [
  ]
}
{
  "change": [
  ]
}
WARNING: table "table_without_pk" without primary key or replica identity is nothing
{
  "change": [
```

```

{
  "kind": "insert",
  "schema": "public",
  "table": "table_with_pk",
  "columnnames": ["a", "b", "c"],
  "columnvalues": [1, "Backup and Restore", "2018-03-27 11:58:28.988414"]
}
,{
  "kind": "insert",
  "schema": "public",
  "table": "table_with_pk",
  "columnnames": ["a", "b", "c"],
  "columnvalues": [2, "Tuning", "2018-03-27 11:58:28.988414"]
}
,{
  "kind": "insert",
  "schema": "public",
  "table": "table_with_pk",
  "columnnames": ["a", "b", "c"],
  "columnvalues": [3, "Replication", "2018-03-27 11:58:28.988414"]
}
,{
  "kind": "delete",
  "schema": "public",
  "table": "table_with_pk",
  "oldkeys": {
    "keynames": ["a", "c"],
    "keytypes": ["integer", "timestamp without time zone"],
    "keyvalues": [1, "2018-03-27 11:58:28.988414"]
  }
}
,{
  "kind": "delete",
  "schema": "public",
  "table": "table_with_pk",
  "oldkeys": {
    "keynames": ["a", "c"],
    "keytypes": ["integer", "timestamp without time zone"],
    "keyvalues": [2, "2018-03-27 11:58:28.988414"]
  }
}
,{
  "kind": "insert",

```

```

        "schema": "public",
        "table": "table_without_pk",
        "columnnames": ["a", "b", "c"],
        "columnntypes": ["integer", "numeric(5,2)", "text"],
        "columnvalues": [1, 2.34, "Tapir"]
    }
]
}

```

Dropping the slot in the first terminal:

Ctrl+C

```
$ pg_recvlogical -d postgres --slot test_slot --drop-slot
```

SQL functions

```
$ cat /tmp/example2.sql
```

```
CREATE TABLE table2_with_pk (a SERIAL, b VARCHAR(30), c TIMESTAMP NOT NULL, PRIMARY KEY(a, c));
CREATE TABLE table2_without_pk (a SERIAL, b NUMERIC(5,2), c TEXT);
```

```
SELECT 'init' FROM pg_create_logical_replication_slot('test_slot', 'wal2json');
```

```
BEGIN;
```

```
INSERT INTO table2_with_pk (b, c) VALUES('Backup and Restore', now());
```

```
INSERT INTO table2_with_pk (b, c) VALUES('Tuning', now());
```

```
INSERT INTO table2_with_pk (b, c) VALUES('Replication', now());
```

```
DELETE FROM table2_with_pk WHERE a < 3;
```

```
INSERT INTO table2_without_pk (b, c) VALUES(2.34, 'Tapir');
```

```
-- it is not added to stream because there isn't a pk or a replica identity
```

```
UPDATE table2_without_pk SET c = 'Anta' WHERE c = 'Tapir';
```

```
COMMIT;
```

```
SELECT data FROM pg_logical_slot_get_changes('test_slot', NULL, NULL, 'pretty-print', '1');
```

```
SELECT 'stop' FROM pg_drop_replication_slot('test_slot');
```

The script above produces the output below:

```
$ psql -At -f /tmp/example2.sql postgres
```

```
CREATE TABLE
```

```
CREATE TABLE
```

```
init
```

```
BEGIN
```

```
INSERT 0 1
```

```

INSERT 0 1
INSERT 0 1
DELETE 2
INSERT 0 1
UPDATE 1
COMMIT
psql:/tmp/example2.sql:17: WARNING:  table "table2_without_pk" without primary key or replication
{
  "change": [
    {
      "kind": "insert",
      "schema": "public",
      "table": "table2_with_pk",
      "columnnames": ["a", "b", "c"],
      "columnvalues": [1, "Backup and Restore", "2018-03-27 12:05:29.914496"]
    },
    {
      "kind": "insert",
      "schema": "public",
      "table": "table2_with_pk",
      "columnnames": ["a", "b", "c"],
      "columnvalues": [2, "Tuning", "2018-03-27 12:05:29.914496"]
    },
    {
      "kind": "insert",
      "schema": "public",
      "table": "table2_with_pk",
      "columnnames": ["a", "b", "c"],
      "columnvalues": [3, "Replication", "2018-03-27 12:05:29.914496"]
    },
    {
      "kind": "delete",
      "schema": "public",
      "table": "table2_with_pk",
      "oldkeys": {
        "keynames": ["a", "c"],
        "keytypes": ["integer", "timestamp without time zone"],
        "keyvalues": [1, "2018-03-27 12:05:29.914496"]
      }
    },
    {
      "kind": "delete",
      "schema": "public",

```

```

        "table": "table2_with_pk",
        "oldkeys": {
            "keynames": ["a", "c"],
            "keytypes": ["integer", "timestamp without time zone"],
            "keyvalues": [2, "2018-03-27 12:05:29.914496"]
        }
    },{
        "kind": "insert",
        "schema": "public",
        "table": "table2_without_pk",
        "columnnames": ["a", "b", "c"],
        "columntypes": ["integer", "numeric(5,2)", "text"],
        "columnvalues": [1, 2.34, "Tapir"]
    }
]
}
stop

```

Let's repeat the same example with format-version 2:

```

$ cat /tmp/example3.sql
CREATE TABLE table2_with_pk (a SERIAL, b VARCHAR(30), c TIMESTAMP NOT NULL, PRIMARY KEY(a, c));
CREATE TABLE table2_without_pk (a SERIAL, b NUMERIC(5,2), c TEXT);

SELECT 'init' FROM pg_create_logical_replication_slot('test_slot', 'wal2json');

BEGIN;
INSERT INTO table2_with_pk (b, c) VALUES('Backup and Restore', now());
INSERT INTO table2_with_pk (b, c) VALUES('Tuning', now());
INSERT INTO table2_with_pk (b, c) VALUES('Replication', now());
DELETE FROM table2_with_pk WHERE a < 3;

INSERT INTO table2_without_pk (b, c) VALUES(2.34, 'Tapir');
-- it is not added to stream because there isn't a pk or a replica identity
UPDATE table2_without_pk SET c = 'Anta' WHERE c = 'Tapir';
COMMIT;

SELECT data FROM pg_logical_slot_get_changes('test_slot', NULL, NULL, 'format-version', '2');
SELECT 'stop' FROM pg_drop_replication_slot('test_slot');

```

The script above produces the output below:

```

$ psql -At -f /tmp/example3.sql postgres
CREATE TABLE

```

```

CREATE TABLE
init
BEGIN
INSERT 0 1
INSERT 0 1
INSERT 0 1
DELETE 2
INSERT 0 1
UPDATE 1
COMMIT
psql:/tmp/example3.sql:17: WARNING:  no tuple identifier for UPDATE in table "public"."table2_with_pk"
{"action": "B"}
{"action": "I", "schema": "public", "table": "table2_with_pk", "columns": [{"name": "a", "type": "integer"}]}
{"action": "I", "schema": "public", "table": "table2_with_pk", "columns": [{"name": "a", "type": "integer"}]}
{"action": "I", "schema": "public", "table": "table2_with_pk", "columns": [{"name": "a", "type": "integer"}]}
{"action": "D", "schema": "public", "table": "table2_with_pk", "identity": [{"name": "a", "type": "integer"}]}
{"action": "D", "schema": "public", "table": "table2_with_pk", "identity": [{"name": "a", "type": "integer"}]}
{"action": "I", "schema": "public", "table": "table2_without_pk", "columns": [{"name": "a", "type": "integer"}]}
{"action": "C"}
stop

```

License

Copyright (c) 2013-2020, Euler Taveira de Oliveira All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the Euler Taveira de Oliveira nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,

THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.