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 module 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 wal2json_18
```

In Debian/Ubuntu:

```
$ sudo apt-get install postgresql-18-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 10+ installed (including the header files). If PostgreSQL is not in your search path, add it. If you are using PostgreSQL yum repository, install `postgresql18-devel` and add `/usr/pgsql-18/bin` to your search path (yum uses `18, 17, 16, 15, 14, 13, 12, 11 or 10`). If you are using PostgreSQL apt repository, install `postgresql-server-dev-18` and add `/usr/lib/postgresql/18/bin` to your search path. (apt uses `18, 17, 16, 15, 14, 13, 12, 11 or 10`).

If you compile PostgreSQL by yourself and install it in `$HOME/pg18`:

```
$ tar -zxf wal2json-wal2json_2_7.tar.gz
$ cd wal2json-wal2json_2_7
$ export PATH=$HOME/pg18/bin:$PATH
```

```
$ make
$ make install
```

If you are using PostgreSQL yum repository:

```
$ sudo yum install postgresql18-devel
$ tar -zxf wal2json-wal2json_2_7.tar.gz
$ cd wal2json-wal2json_2_7
$ export PATH=/usr/pgsql-18/bin:$PATH
$ make
$ make install
```

If you are using PostgreSQL apt repository:

```
$ sudo apt-get install postgresql-server-dev-18
$ tar -zxf wal2json-wal2json_2_7.tar.gz
$ cd wal2json-wal2json_2_7
$ export PATH=/usr/lib/postgresql/18/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:\pg\18` 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 one parameter at postgresql.conf (the default values of `max_replication_slots` and `max_wal_senders` are enough):

```
wal_level = logical
```

After changing this parameter, a restart is needed.

Since PostgreSQL 18.6, 17.11, 16.15, 15.19, and 14.24, an output plugin can only be loaded if its library is listed in the `output_plugin_libraries` parameter (this restriction was added as a fix for CVE-2026-6471). The default value contains only the output plugins that are shipped with PostgreSQL (`pgoutput` and `test_decoding`), hence, **wal2json** has to be added to it:

```
output_plugin_libraries = 'pgoutput, test_decoding, wal2json'
```

The library name must be exactly `wal2json`. Otherwise, creating or reading from a slot fails with:

```
ERROR:  library "wal2json" may not be used as an output plugin
```

This parameter is only read when the output plugin is loaded, so a reload (`pg_ctl reload` or `SELECT pg_reload_conf()`) is sufficient. Older versions (including PostgreSQL 13 and prior minor versions of the branches listed above) do not have this parameter and do not require this setting.

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 (*pg_attribute.attnum*). Default is *false*.
- `include-origin`: add origin of a piece of data. 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*.
- `numeric-data-types-as-string`: use string for numeric data types. JSON specification does not recognize `Infinity` and `NaN` as valid numeric values. There might be potential interoperability problems for double precision numbers. 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. Only used when `format-version` is `1`. Default is *false*.
- `include-lsn`: add *nextlsn* to each changeset. Default is *false*.
- `include-transaction`: emit records denoting the start and end of each transaction. Default is *true*.
- `include-empty-transaction`: emit transactions without changes (e.g. DDL-only transactions or transactions whose changes were all filtered out). If *false*, the start of a transaction is emitted only before its first change. In format version 2, a transaction without changes produces no output if `include-transaction` is *false*. Default is *true*.

- `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.
- `partition-root`: use the root partitioned table as the table name (similar to `publish_via_partition_root` in `CREATE PUBLICATION`). If *false*, use the partition name. `filter-tables` and `add-tables` match the same name that is used as the table name, hence, if this parameter is *true*, they should specify the root partitioned table. Default is *false*.
- `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 matches a normal entry with a database name or keywords such as `all`. Add a connection rule at pg_hba.conf:

```
local    mydatabase      myuser                      trust
```

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 -o add-msg-prefixes=
```

In another terminal:

```
$ cat /tmp/example1.sql
CREATE TABLE table1_with_pk (a SERIAL, b VARCHAR(30), c TIMESTAMP NOT NULL, PRIMARY KEY(a, c
CREATE TABLE table1_without_pk (a SERIAL, b NUMERIC(5,2), c TEXT);

BEGIN;
INSERT INTO table1_with_pk (b, c) VALUES('Backup and Restore', now());
INSERT INTO table1_with_pk (b, c) VALUES('Tuning', now());
INSERT INTO table1_with_pk (b, c) VALUES('Replication', now());
SELECT pg_logical_emit_message(true, 'wal2json', 'this message will be delivered');
SELECT pg_logical_emit_message(true, 'pgoutput', 'this message will be filtered');
DELETE FROM table1_with_pk WHERE a < 3;
SELECT pg_logical_emit_message(false, 'wal2json', 'this non-transactional message will be de

INSERT INTO table1_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 table1_without_pk SET c = 'Anta' WHERE c = 'Tapir';
COMMIT;

DROP TABLE table1_with_pk;
DROP TABLE table1_without_pk;

$ psql -At -f /tmp/example1.sql postgres
CREATE TABLE
CREATE TABLE
BEGIN
INSERT 0 1
INSERT 0 1
INSERT 0 1
3/78BFC828
3/78BFC880
DELETE 2
3/78BFC990
INSERT 0 1
UPDATE 1
COMMIT
DROP TABLE
DROP TABLE
```

The output in the first terminal is:

```
{
    "change": [
    ]
}
{
```

```
    "change": [
    ]
}
{
    "change": [
        {
            "kind": "message",
            "transactional": false,
            "prefix": "wal2json",
            "content": "this non-transactional message will be delivered even if you rollbac
        }
    ]
}
WARNING:  table "table1_without_pk" without primary key or replica identity is nothing
{
    "change": [
        {
            "kind": "insert",
            "schema": "public",
            "table": "table1_with_pk",
            "columnnames": ["a", "b", "c"],
            "columntypes": ["integer", "character varying(30)", "timestamp without time zone
            "columnvalues": [1, "Backup and Restore", "2018-03-27 11:58:28.988414"]
        }
        ,{
            "kind": "insert",
            "schema": "public",
            "table": "table1_with_pk",
            "columnnames": ["a", "b", "c"],
            "columntypes": ["integer", "character varying(30)", "timestamp without time zone
            "columnvalues": [2, "Tuning", "2018-03-27 11:58:28.988414"]
        }
        ,{
            "kind": "insert",
            "schema": "public",
            "table": "table1_with_pk",
            "columnnames": ["a", "b", "c"],
            "columntypes": ["integer", "character varying(30)", "timestamp without time zone
            "columnvalues": [3, "Replication", "2018-03-27 11:58:28.988414"]
        }
        ,{
            "kind": "message",
            "transactional": true,
            "prefix": "wal2json",
            "content": "this message will be delivered"
        }
```

```
        ,{
            "kind": "delete",
            "schema": "public",
            "table": "table1_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": "table1_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": "table1_without_pk",
            "columnnames": ["a", "b", "c"],
            "columntypes": ["integer", "numeric(5,2)", "text"],
            "columnvalues": [1, 2.34, "Tapir"]
        }
    ]
}
{
    "change": [
    ]
}
{
    "change": [
    ]
}
```

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());
SELECT pg_logical_emit_message(true, 'wal2json', 'this message will be delivered');
SELECT pg_logical_emit_message(true, 'pgoutput', 'this message will be filtered');
DELETE FROM table2_with_pk WHERE a < 3;
SELECT pg_logical_emit_message(false, 'wal2json', 'this non-transactional message will be de

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');

DROP TABLE table2_with_pk;
DROP TABLE table2_without_pk;
```

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
3/78C2CA50
3/78C2CAA8
DELETE 2
3/78C2CBD8
INSERT 0 1
UPDATE 1
COMMIT
{
    "change": [
        {
            "kind": "message",
            "transactional": false,
```

```
            "prefix": "wal2json",
            "content": "this non-transactional message will be delivered even if you rollbac
        }
    ]
}
psql:/tmp/example2.sql:17: WARNING:  table "table2_without_pk" without primary key or replic
{
    "change": [
        {
            "kind": "insert",
            "schema": "public",
            "table": "table2_with_pk",
            "columnnames": ["a", "b", "c"],
            "columntypes": ["integer", "character varying(30)", "timestamp without time zone
            "columnvalues": [1, "Backup and Restore", "2018-03-27 12:05:29.914496"]
        }
        ,{
            "kind": "insert",
            "schema": "public",
            "table": "table2_with_pk",
            "columnnames": ["a", "b", "c"],
            "columntypes": ["integer", "character varying(30)", "timestamp without time zone
            "columnvalues": [2, "Tuning", "2018-03-27 12:05:29.914496"]
        }
        ,{
            "kind": "insert",
            "schema": "public",
            "table": "table2_with_pk",
            "columnnames": ["a", "b", "c"],
            "columntypes": ["integer", "character varying(30)", "timestamp without time zone
            "columnvalues": [3, "Replication", "2018-03-27 12:05:29.914496"]
        }
        ,{
            "kind": "message",
            "transactional": true,
            "prefix": "wal2json",
            "content": "this message will be delivered"
        }
        ,{
            "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
DROP TABLE
DROP TABLE
```

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

```
$ cat /tmp/example3.sql
CREATE TABLE table3_with_pk (a SERIAL, b VARCHAR(30), c TIMESTAMP NOT NULL, PRIMARY KEY(a, c
CREATE TABLE table3_without_pk (a SERIAL, b NUMERIC(5,2), c TEXT);

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

BEGIN;
INSERT INTO table3_with_pk (b, c) VALUES('Backup and Restore', now());
INSERT INTO table3_with_pk (b, c) VALUES('Tuning', now());
INSERT INTO table3_with_pk (b, c) VALUES('Replication', now());
SELECT pg_logical_emit_message(true, 'wal2json', 'this message will be delivered');
SELECT pg_logical_emit_message(true, 'pgoutput', 'this message will be filtered');
DELETE FROM table3_with_pk WHERE a < 3;
SELECT pg_logical_emit_message(false, 'wal2json', 'this non-transactional message will be de

INSERT INTO table3_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 table3_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');

DROP TABLE table3_with_pk;
DROP TABLE table3_without_pk;
```

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
3/78CB8F30
3/78CB8F88
DELETE 2
3/78CB90B8
INSERT 0 1
UPDATE 1
COMMIT
psql:/tmp/example3.sql:20: WARNING:  no tuple identifier for UPDATE in table "public"."table
{"action":"M","transactional":false,"prefix":"wal2json","content":"this non-transactional me
{"action":"B"}
{"action":"I","schema":"public","table":"table3_with_pk","columns":[{"name":"a","type":"inte
{"action":"I","schema":"public","table":"table3_with_pk","columns":[{"name":"a","type":"inte
{"action":"I","schema":"public","table":"table3_with_pk","columns":[{"name":"a","type":"inte
{"action":"M","transactional":true,"prefix":"wal2json","content":"this message will be deliv
{"action":"D","schema":"public","table":"table3_with_pk","identity":[{"name":"a","type":"int
{"action":"D","schema":"public","table":"table3_with_pk","identity":[{"name":"a","type":"int
{"action":"I","schema":"public","table":"table3_without_pk","columns":[{"name":"a","type":"i
{"action":"C"}
stop
DROP TABLE
DROP TABLE
```

License

Copyright (c) 2013-2024, 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.