**# Intructions on how to use this repo**

**# Database Reliability Engineer by Ronaldo Akamine**

Docker Composer to setup 2 PostgreSql databases and enable data replication

Database pg\_master acts as Publisher and pg\_replica as Subscriber.

**## 1. Clone this repo to your local computer**

git clone

**## 2. Make sure you have installed Docker**

https://www.docker.com/products/docker-desktop/

**## 3. Execute the docker compose file**

In your computer's terminal navigate to the repo root folder "postgresql\_replication" and execute:

docker-compose up -d

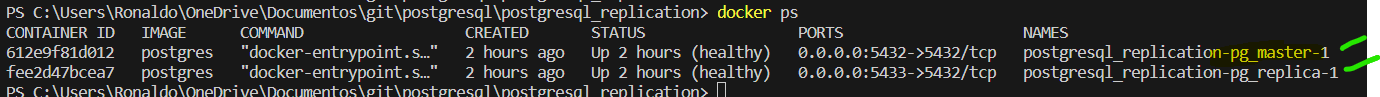
**## 4. Check that you now have 2 docker containers running:**

docker ps

You should see NAMES:

postgresql\_replication-pg\_master-1

postgresql\_replication-pg\_replica-1



**## 4. Configure your DBeaver/other tool to properly query testdb and testdbreplica**

For pg\_master:

host: localhost port: 5432

database: testdb

user: root / password: password

For pg\_replica:

host: localhost port: 5433

database: testdbreplica

user: root / password: password

**## 5. Verify what is the IP address set for pg\_master-1:**

docker network inspect postgresql\_replication\_default

look for "Name": "postgresql\_replication-pg\_master-1" and check it's "IPv4Address".

Take note of this ip address so later we can configure it for the subscription in replica DB.

A computer screen shot of a computer program

Description automatically generated

**## 6. Configure the Replication for the subscription**

The Publisher was already configured in the "01-init.sh" file of pg\_master so we just need to setup the subscriber.

Check how many hows we have in the replica's table:

select count(\*) from cloudwalk.orders;

Log in to pg\_replica with root/password and execute bellow command in postgresql:

CREATE SUBSCRIPTION get\_prod\_orders

CONNECTION 'dbname = testdb

host = <PASTE IP ADDRESS HERE>

user = root

password = password

port = 5432'

PUBLICATION prod\_orders;

**## 7. Compile and execute insert procedure:**

Compile the procure (run script procedure\_prc\_inserts.sql) in pg\_master and call it:

CALL public.prc\_inserts();

This procedure is set to perform 100000 record inserts into cloudwalk.orders table.

And the loop waits 0.2 seconds between each insert so we can simulate real world inserts into our database.

Check on pg\_master and pg\_replica how many how we have in cloudwalk.orders table.

Verify that while the procure runs the replication is working good.

--To check total rows

select count(\*) from cloudwalk.orders;

--To quickly check latest rows added

select \* from cloudwalk.orders order by 1 desc;

**## 8. To make the cloudwalk.orders as a partitioned table with no downtime we have to:**

Create a new partitioned table:

CREATE TABLE IF NOT EXISTS cloudwalk.orders2(

id integer GENERATED BY DEFAULT AS IDENTITY,

product\_name text,

quantity integer,

order\_date date,

constraint pk\_orders primary key (id, order\_date)

) PARTITION BY RANGE (order\_date);

--Create some partitions (child tables)

--Notice that the child table names have timestamp, so change yours first one to today’s date. (My todays date was 0916)

CREATE TABLE cloudwalk.orders2\_20230916 PARTITION OF cloudwalk.orders2

FOR VALUES FROM (now()) TO (now() + interval '1 day');

CREATE TABLE cloudwalk.orders2\_20230917 PARTITION OF cloudwalk.orders2

FOR VALUES FROM (now() + interval '1 day') TO (now() + interval '2 day');

CREATE TABLE cloudwalk.orders2\_20230918 PARTITION OF cloudwalk.orders2

FOR VALUES FROM (now() + interval '2 day') TO (now() + interval '3 day');

CREATE TABLE cloudwalk.orders2\_20230919 PARTITION OF cloudwalk.orders2

FOR VALUES FROM (now() + interval '3 day') TO (now() + interval '4 day');

CREATE TABLE cloudwalk.orders2\_20230920 PARTITION OF cloudwalk.orders2

FOR VALUES FROM (now() + interval '4 day') TO (now() + interval '5 day');

--Notice the partitioned table we created the identity column as DEFAULT instead of ALWAYS

--This allow the trigger to insert the id from the source table.

**## 8.1 Create a trigger that will send all data original table receives to the new partitioned table:**

CREATE FUNCTION fnc\_migrate\_orders() RETURNS trigger AS $$

BEGIN

insert into cloudwalk.orders2

select \* from cloudwalk.orders a

where not exists (select 1

from cloudwalk.orders2 b

where a.id = b.id

and a.order\_date = b.order\_date);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_migrate\_orders BEFORE INSERT ON cloudwalk.orders

FOR EACH STATEMENT EXECUTE FUNCTION fnc\_migrate\_orders();

**## 8.2 Right after the trigger creation check the new partitoned table now has the same data as the original table**

select \* from cloudwalk.orders2; --Partitioned

select \* from cloudwalk.orders; --Not Partitioned

**## 9. To make the cloudwalk.orders as a partitioned table with no downtime we have to:**

1-) Create a view pointing to original table:

create or replace view cloudwalk.orders\_vw as

select \* from cloudwalk.orders;

1.1-) All applications that just needs to read data will have to read data from pg\_master using this view instead of querying directly the real table.

1.2-) Now modify the view so it reads data from partitioned table:

create or replace view cloudwalk.orders\_vw as

select \* from cloudwalk.orders2;

Now we have applications reading data from new partitined table using this view with no Down time in the system.

Views compile time is very fast and also should allow DML operations.

--Ultimately if you need to use original table name for all applications you can rename the table names

**## 10. Renaming tables (some quick downtime can happen):**

--First check what is the latest value for ID column and then make the new sequece (for partitioned table start from that value plus 1000)

select max(id) + 1000 from cloudwalk.orders;

--Suppose above query returns 1560

--Use this value in bellows alter statement:

ALTER SEQUENCE cloudwalk.orders2\_id\_seq RESTART 1560;

--Deprecate table that is not partitioned

alter table cloudwalk.orders rename to orders\_old;

--Make new partitioned table the official one:

alter table cloudwalk.orders2 rename to orders;

--Do not forget to also recompile the view to point to new partitioned table.

create or replace view cloudwalk.orders\_vw as

select \* from cloudwalk.orders order by 1 desc;

This table name renaming also breaks the data replication that needs to be reconfigured.