An Introduction to PostgreSQL 13

Shayon Sanyal, Sr DB Specialist Solutions Architect

August 26th 2021
Agenda

• PostgreSQL – A brief introduction
• Running PostgreSQL as a managed service in AWS
• Why upgrade?
• Why upgrade to PostgreSQL 13
• New features in PostgreSQL 13
• Demo
PostgreSQL – A brief introduction

- PostgreSQL is a powerful, open source object-relational database system

- Origins of PostgreSQL date back to 1986 as part of the POSTGRES project at the University of California at Berkeley

- More than 30 years of active development on the core platform

- Rich features and extensions, reliability and standards compliance, open source license

https://db-engines.com/en/blog_post/85
Running PostgreSQL as a managed service in AWS
Self-managing relational databases is time consuming, complex, and expensive

- Hardware & software installation, configuration, patching, and backups
- Performance and high availability issues
- Capacity planning, and scaling clusters for compute and storage
- Security and compliance
Self-managed vs. fully managed

Spend time innovating & building new apps, not managing infrastructure

<table>
<thead>
<tr>
<th>Self-Managed</th>
<th>App optimization</th>
<th>Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>High availability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Database backups</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB software patches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB software installs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS patches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS installation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server maintenance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rack and stack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power, HVAC, net</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Managed by AWS</th>
<th>App optimization</th>
<th>Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>High availability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Database backups</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB software patches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB software installs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS patches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS installation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server maintenance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rack and stack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power, HVAC, net</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Database on-premises

Databases on EC2

Amazon RDS

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
Amazon RDS
Managed relational database service with a choice of six popular database engines

- **Easy to administer**
  - Easily deploy and maintain hardware, OS and DB software; built-in monitoring

- **Secure & compliant**
  - Data encryption at rest and in transit; industry compliance and assurance programs

- **Available & durable**
  - Automatic Multi-AZ data replication; automated backup, snapshots, failover

- **Performant & scalable**
  - Scale compute and storage with a few clicks; minimal downtime for your application
Amazon Aurora

MySQL and PostgreSQL-compatible relational database built for the cloud
Performance and availability of commercial-grade databases at 1/10th the cost

**Performance and scalability**
- 5x throughput of standard MySQL and 3x of standard PostgreSQL; scale-out up to 15 read replicas

**Availability and durability**
- Fault-tolerant, self-healing storage; six copies of data across three Availability Zones; continuous backup to Amazon S3

**Highly secure**
- Network isolation, encryption at rest/transit, compliance and assurance programs

**Fully managed**
- Managed by RDS: No server provisioning, software patching, setup, configuration, or backups

© 2021, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
Why upgrade?

- Performance improvements
- Data integrity protection enhancements
- Improvements in monitoring and diagnostics
- Current version approaching end-of-life
- Once deprecated, no bug fixing backports or maintenance activities, miss out on security patches or data-integrity-related bugs
Why upgrade to PostgreSQL 13

- Improvements that benefit large databases
- Space savings and performance gains for indexes
- Faster response times for queries that use aggregates or partitions
- Parallelized vacuuming, incremental sorting
- Application developer productivity
- Security improvements
New features in PostgreSQL 13

Themes:

• Continued performance gains
• Administration optimizations
• Application development convenience
• Security enhancements
Continued performance gains
Continued performance gains

- Partitioning enhancements
- B-tree deduplication
- Incremental sort
- Hash aggregation enhancements
- Extended statistics
Logical replication for partitioned tables

- Replicate individual partitions
  - Each partition replicated separately
  - Published individually or via root

```
CREATE PUBLICATION my_pub
  WITH (publish_via_partition_root = true);
```

- Receive into partitioned table
More partitioning enhancements

• Allow pruning of partitions to happen in more cases
• Allow partition-wise joins to happen in more cases
• Support row-level BEFORE triggers on partitioned tables
B-tree deduplication
Incremental sort
Disk-based hash aggregation

PG 12:

• Hash aggregation worked only in-memory, restricted to `work_mem`

```sql
pglab=> explain select count(*) from t2 group by b;

QUERY PLAN

GroupAggregate (cost=1580363.83..1755363.83 rows=10000000 width=12)
  Group Key: b
  -> Sort (cost=1580363.83..1605363.83 rows=10000000 width=4)
    Sort Key: b
      -> Seq Scan on t2 (cost=0.00..144248.00 rows=10000000 width=4)

(5 rows)
```
Disk-based hash aggregation

PG 13:

- Hash aggregation can use disk
- Hash table spilled to disk if it exceeds `work_mem * hash_mem_multiplier`

```
pglab=> explain select count(*) from t2 group by b;

QUERY PLAN

HashAggregate (cost=1219245.30..1475494.71 rows=9999977 width=12)
  Group Key: b
  Planned Partitions: 128
  -> Seq Scan on t2 (cost=0.00..144247.77 rows=9999977 width=4)
(4 rows)
```
Extended statistics

- CREATE STATISTICS
- ALTER STATISTICS ... SET STATISTICS
- Apply multiple extended stats in a single query
- Apply extended stats for OR/IN/ANY (1,2,3, ...) clauses

```
ALTER STATISTICS t3
SET STATISTICS 1000;
```
Administration optimizations
Administration optimizations

• Parallelized Vacuum for Indexes
• INSERTs now trigger Autovacuum
• EXPLAIN tracks write ahead log (WAL) usage
• Replication slot size limits
• Progress Report for ANALYZE
• DROP DATABASE with FORCE
Parallelized vacuum for indexes

- Parallel processing of indexes
- Default: One worker per index
  - `max_parallel_maintenance_workers` (default: 2)
  - `min_parallel_index_scan_size` (default: 512 kB)

```
VACUUM (PARALLEL 2, VERBOSE) test;
```

INFO: vacuuming "public.test"
INFO: launched 2 parallel vacuum workers for index vacuuming (planned: 2)
INFO: scanned index "idx_id" to remove 1403418 row versions by parallel vacuum worker
DETAIL: CPU: user: 0.98 s, system: 0.15 s, elapsed: 2.37 s
INFO: scanned index "idx_name" to remove 1403418 row versions by parallel vacuum worker
DETAIL: CPU: user: 0.88 s, system: 0.27 s, elapsed: 2.60 s ...
INSERTs now trigger autovacuum

- Triggers VACUUM for insert only workloads
- More frequent, inexpensive runs
- New settings:
  - `autovacuum_vacuum_insert_threshold`
  - `autovacuum_vacuum_insert_scale_factor`
EXPLAIN tracks WAL usage

pglab=> CREATE TABLE t4 (i int generated always as identity, t text);
CREATE TABLE
pglab=>
pglab=> EXPLAIN (ANALYZE, WAL)
pglab=> INSERT INTO t4 (t)
pglab=> SELECT md5 (v::text)
pglab=> FROM generate_series (1, 300000) v;

QUERY PLAN

Insert on t4 (cost=0.00..6000.00 rows=300000 width=36) (actual time=566.475..566.476 rows=0 loops=1)
WAL: records=309091 bytes=2850009

-- Function Scan on generate_series v (cost=0.00..6000.00 rows=300000 width=36) (actual time=28.469..309.066 rows=300000 loops=1)
WAL: records=9091 bytes=900009
Planning Time: 0.047 ms
Execution Time: 567.874 ms
(6 rows)
Replication slot size limits

- `max_slot_wal_keep_size`
  - Default: -1

- If replication slot exceeds `max_slot_wal_keep_size`, it is marked invalid.

- **Monitor** `pg_replication_slots.wal_status`: reserved, extended, unreserved, lost
Progress Report for ANALYZE

- How far along is the ANALYZE
- One row for each backend (including autovacuum worker processes) running ANALYZE, showing current progress

```
SELECT * FROM pg_stat_progress_analyze;
```
DROP DATABASE test;
ERROR: database "test" is being accessed by other users
DETAIL: There is 1 other session using the database.

DROP DATABASE test WITH (FORCE);
DROP DATABASE
Application development convenience
Application development convenience

• `gen_random_uuid()`
• JSON path datetime support
• `greatest-common-denominator (gcd)` and `least-common-multiple (lcm)` functions
• `pgbench` allows partitioning “accounts” table
gen_random_uuid()

=> \dx

List of installed extensions

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Schema</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(0 rows)

=> SELECT gen_random_uuid();

    gen_random_uuid

--------------------------------------
16b2d7ec-aec7-4e8e-ac60-06609546b96e

(1 row)
JSON path datetime support

=> SELECT jsonb_path_query('"27-09-2020"', '$.datetime("dd-mm-yyyy")')
  jsonb_path_query
--------------
"2020-09-27"
(1 row)
gcd and lcm functions

```sql
=> SELECT gcd (1071, 462), lcm (1071, 46);
gcd |  lcm
-----+-------
 21 | 49266
(1 row)

=> SELECT gcd (0, 0), lcm (0, 0);
gcd |  lcm
-----+-----
  0 |   0
(1 row)
```
pgbench allows partitioning “accounts” table

$ pgbench -i --partition-method=hash --partitions=5

... creating tables... creating 5 partitions...
generating data (client-side)... 100000 of 100000 tuples (100%) done (elapsed 0.04 s, remaining 0.00 s)
vacuuming... creating primary keys... done in 0.79 s (drop tables 0.02 s, create tables 0.17 s, client-side generate 0.13 s, vacuum 0.26 s, primary keys 0.22 s).

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Type</th>
<th>Owner</th>
<th>Persistence</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>pgbench_accounts</td>
<td>partitioned table</td>
<td>postgres</td>
<td>permanent</td>
<td>0 bytes</td>
</tr>
<tr>
<td>public</td>
<td>pgbench_accounts_1</td>
<td>table</td>
<td>postgres</td>
<td>permanent</td>
<td>2656 kB</td>
</tr>
<tr>
<td>public</td>
<td>pgbench_accounts_2</td>
<td>table</td>
<td>postgres</td>
<td>permanent</td>
<td>2656 kB</td>
</tr>
<tr>
<td>public</td>
<td>pgbench_accounts_3</td>
<td>table</td>
<td>postgres</td>
<td>permanent</td>
<td>2656 kB</td>
</tr>
<tr>
<td>public</td>
<td>pgbench_accounts_4</td>
<td>table</td>
<td>postgres</td>
<td>permanent</td>
<td>2656 kB</td>
</tr>
<tr>
<td>public</td>
<td>pgbench_accounts_5</td>
<td>table</td>
<td>postgres</td>
<td>permanent</td>
<td>2656 kB</td>
</tr>
<tr>
<td>public</td>
<td>pgbench_branches</td>
<td>table</td>
<td>postgres</td>
<td>permanent</td>
<td>40 kB</td>
</tr>
<tr>
<td>public</td>
<td>pgbench_history</td>
<td>table</td>
<td>postgres</td>
<td>permanent</td>
<td>0 bytes</td>
</tr>
<tr>
<td>public</td>
<td>pgbench_tellers</td>
<td>table</td>
<td>postgres</td>
<td>permanent</td>
<td>40 kB</td>
</tr>
</tbody>
</table>
pgbench allows partitioning “accounts” table

```
$ pgbench -i --partition-method=range --partitions=5
...
```

Partitioned table "public.pgbench_accounts"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Nullable</th>
<th>Default</th>
<th>Storage</th>
<th>Stats target</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aid</td>
<td>integer</td>
<td></td>
<td>not null</td>
<td></td>
<td>plain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bid</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
<td>plain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>abalance</td>
<td>integer</td>
<td></td>
<td></td>
<td></td>
<td>plain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>filler</td>
<td>character</td>
<td></td>
<td></td>
<td></td>
<td>extended</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Partition key: RANGE (aid)

Indexes:

"pgbench_accounts_pkey" PRIMARY KEY, btree (aid)

Partitions: pgbench_accounts_1 FOR VALUES FROM (MINVALUE) TO (20001),
  pgbench_accounts_2 FOR VALUES FROM (20001) TO (40001),
  pgbench_accounts_3 FOR VALUES FROM (40001) TO (60001),
  pgbench_accounts_4 FOR VALUES FROM (60001) TO (80001),
  pgbench_accounts_5 FOR VALUES FROM (80001) TO (MAXVALUE)
Security enhancements
Security enhancements

• Minimum TLS version now 1.2
• Trusted Extensions
• Client can require SCRAM channel binding
• Certificate authentication with postgres_fdw
Minimum TLS version now 1.2

- New default: TLSv1.2
- SSL version in libpq connection string
  - `ssl_min_protocol_version`
  - `ssl_max_protocol_version`
Trusted extensions

- Can be installed with CREATE privileges
- Examples: hstore, ltree, pgcrypto, plperl

```sql
pglab=> CREATE USER demo PASSWORD '<<PASSWORD>>';
CREATE ROLE
pglab=> GRANT CREATE ON DATABASE pglab TO demo;
GRANT
pglab=> \connect pglab demo
You are now connected to database "pglab" as user "demo".
pglab=> CREATE EXTENSION hstore;
CREATE EXTENSION
```
Client can require SCRAM channel binding

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>require</td>
<td>Connection must employ channel binding</td>
</tr>
<tr>
<td>prefer</td>
<td>Client will choose channel binding if available</td>
</tr>
<tr>
<td>disable</td>
<td>Prevent the use of channel binding</td>
</tr>
</tbody>
</table>
Certificate authentication with postgres_fdw

• Specify `sslkey` and `sslcert` in connection options
• Specify `password_required=false`
  • Superuser only

=> CREATE USER MAPPING FOR demo SERVER rh1 OPTIONS (user 'demo');
CREATE USER MAPPING
=> CREATE FOREIGN TABLE data1(c1 INT, c2 VARCHAR(10)) SERVER rh1;
CREATE FOREIGN TABLE
=> SELECT * FROM data1;
ERROR: password is required
DETAIL: Non-superusers must provide a password in the user mapping.

=> ALTER USER MAPPING FOR demo SERVER rh1 OPTIONS (password_required 'false');
ALTER USER MAPPING
...and there’s more!

- Improved optimizer’s selectivity estimation for containment/match operators
- Improved performance for truncation of very large tables
- When using LOCK TABLE on a partitioned table, do not check permissions on the child tables
- Show table persistence in psql's `\dt+` command

Release Notes: [https://www.postgresql.org/docs/release/13.0/](https://www.postgresql.org/docs/release/13.0/)
Thank you!

shayons@amazon.com