
AWS
re:Invent

DAT303

Deep dive on PostgreSQL databases on Amazon RDS

Jim Mlodgenski

Principal Database Engineer

AWS

AWS re:Invent

Agenda

Overview

Amazon Relational Database Service (Amazon RDS) PostgreSQL deployment

Performance

Upgrades

Moving data

AWS re:Invent

Amazon Relational Database Service (Amazon RDS)

MANAGED RELATIONAL DATABASE SERVICE WITH YOUR CHOICE OF DATABASE ENGINE

Amazon Aurora | MySQL | PostgreSQL | MariaDB | Microsoft SQL Server | Oracle

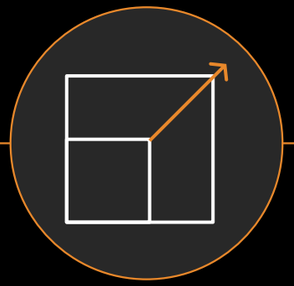
Easy to administer

Easily deploy and maintain hardware, OS and DB software; built-in monitoring

Available and durable

Automatic multi-AZ data replication; automated backup, snapshots, failover

Performant and scalable

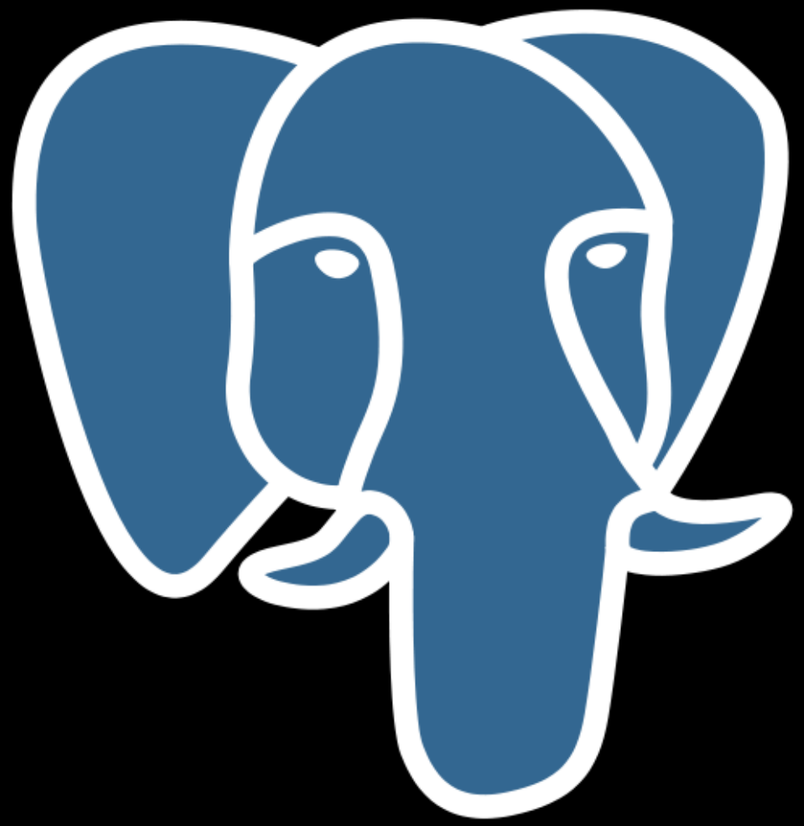


Scale compute and storage with a few clicks; minimal downtime for your application

Secure and compliant

Data encryption at rest and in transit; industry compliance and assurance programs

Why PostgreSQL?



Open source

- In active development for more than 30 years
- Controlled by a community, not a single company

Performance and scale

- Extensive index support for a number of use cases
- Parallel processing for complex queries
- Native partitioning for large tables

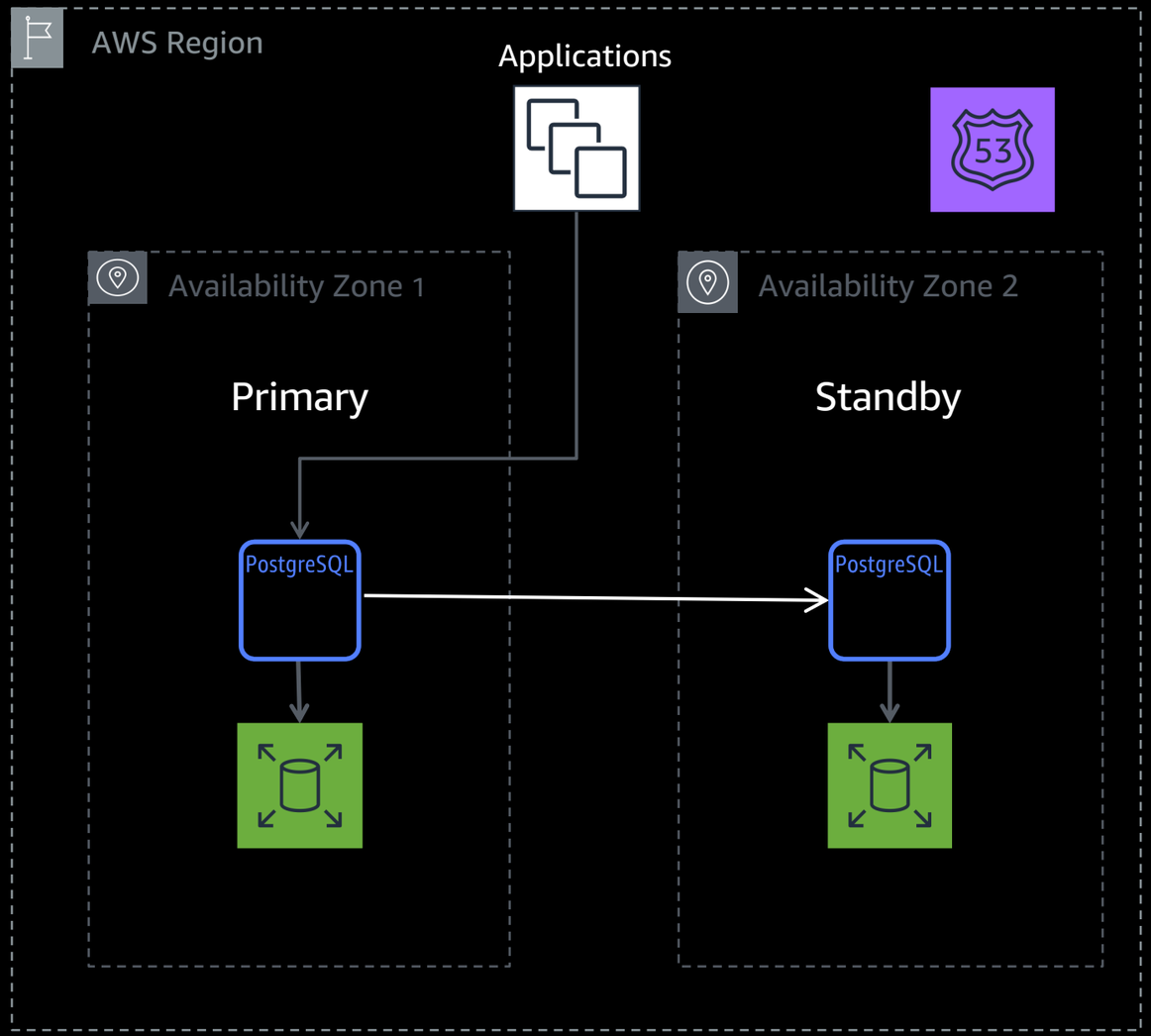
Amazon RDS for PostgreSQL

PostgreSQL community version with easy configuration and management

Supports 9.5, 9.6, 10, 11, and 12

PostgreSQL 13 can be tested on RDS Preview Region (New)

High availability across two availability zones



Amazon Aurora with PostgreSQL compatibility

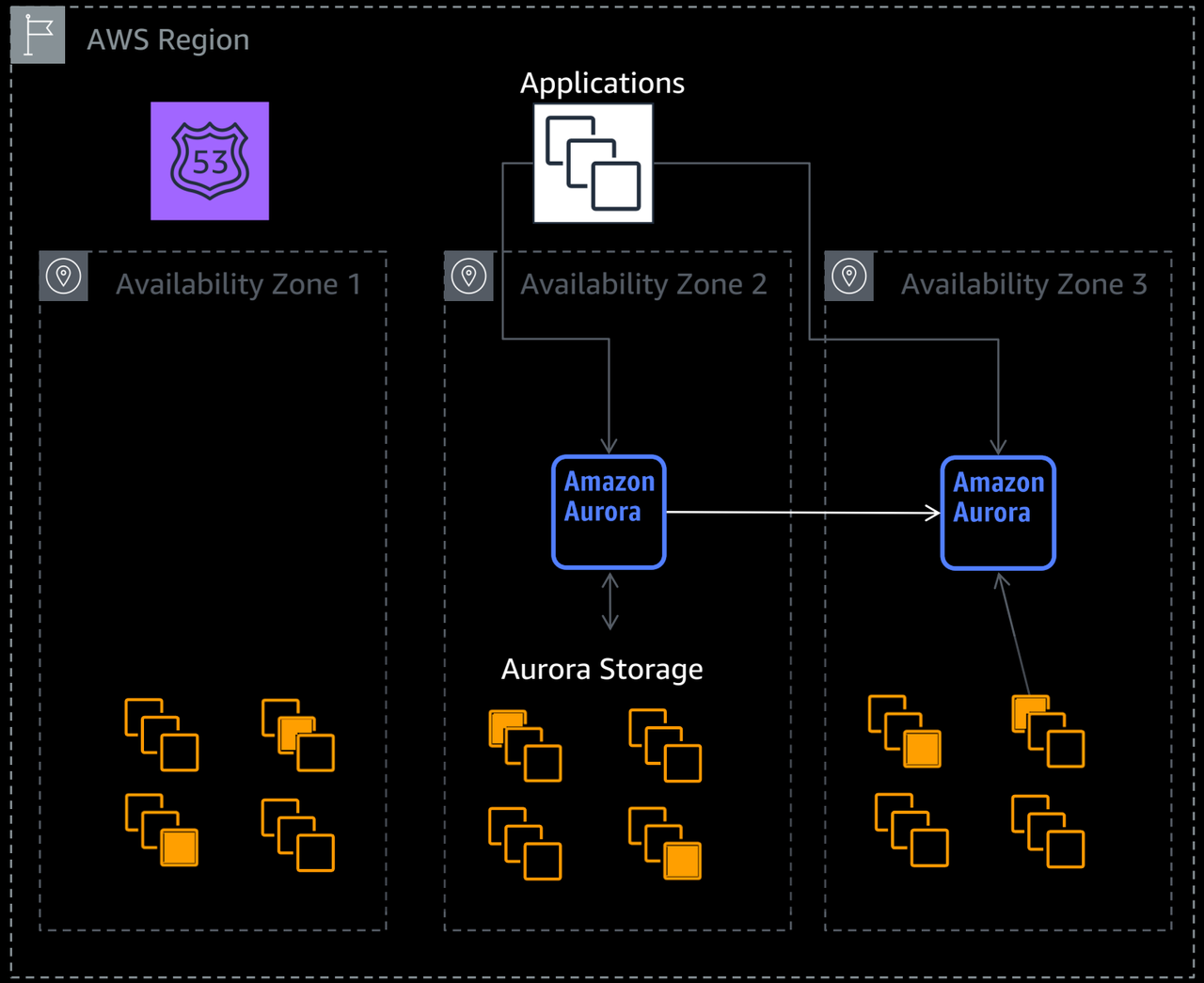
Built from the ground up to leverage AWS services

Supports 9.6, 10, and 11

Up to 2-3x better throughput on the same instance sizes

Highly-available, durable, and fault-tolerant storage layer with 6-way persistence across 3 Availability Zones

Scalable up to 128 TiB



Amazon RDS PostgreSQL deployment

AWS re:Invent

Amazon RDS for PostgreSQL: Instance types

T family

- **Burstable instances**
- 1 vCPU/1 GB RAM > 8 vCPU 32 GB RAM
- Moderate networking performance
- Good for smaller or variable workloads
- T2.micro is eligible for the AWS Free Tier

M family

- **General purpose instances**
- 2 vCPU/8 GiB RAM > 64 vCPU 256 GiB RAM
- High-performance networking
- Good for running CPU-intensive workloads
- M5 offers up to 96 vCPU / 384 GiB RAM

R family

- **Memory-optimized instances**
- 2 vCPU/16 GiB RAM > 64 vCPU 488 GiB RAM
- High-performance networking
- Good for query-intensive workloads or high connection counts
- R5 offers up to 96 vCPU 768 GiB RAM

AWS Graviton2

M6G

- **General purpose instances**
- 1 vCPU/4 GiB RAM > 64 vCPU 256 GiB RAM
- High-performance networking
- Good for running CPU-intensive workloads

R6G

- **Memory-optimized instances**
- 1 vCPU/8 GiB RAM > 64 vCPU 512 GiB RAM
- High-performance networking
- Good for query-intensive workloads or high connection counts

---

64-bit Arm Neoverse cores

4x the number of compute cores

2x larger private caches per core

5x faster memory

2x faster floating-point performance

High-performance database storage

General purpose (GP2)

- SSD storage
- Auto scale up to 64 TiB
- Latency in milliseconds
- IOPS determined by volume size
- Affordable performance

Provisioned IOPS (IO1)

- SSD storage
- Auto scale up to 64 TiB
- Single digit millisecond latencies
- Maximum of 80 K IOPS
- Delivers within 10% of the IOPS performance, 99.9% of the time
- High performance and consistency

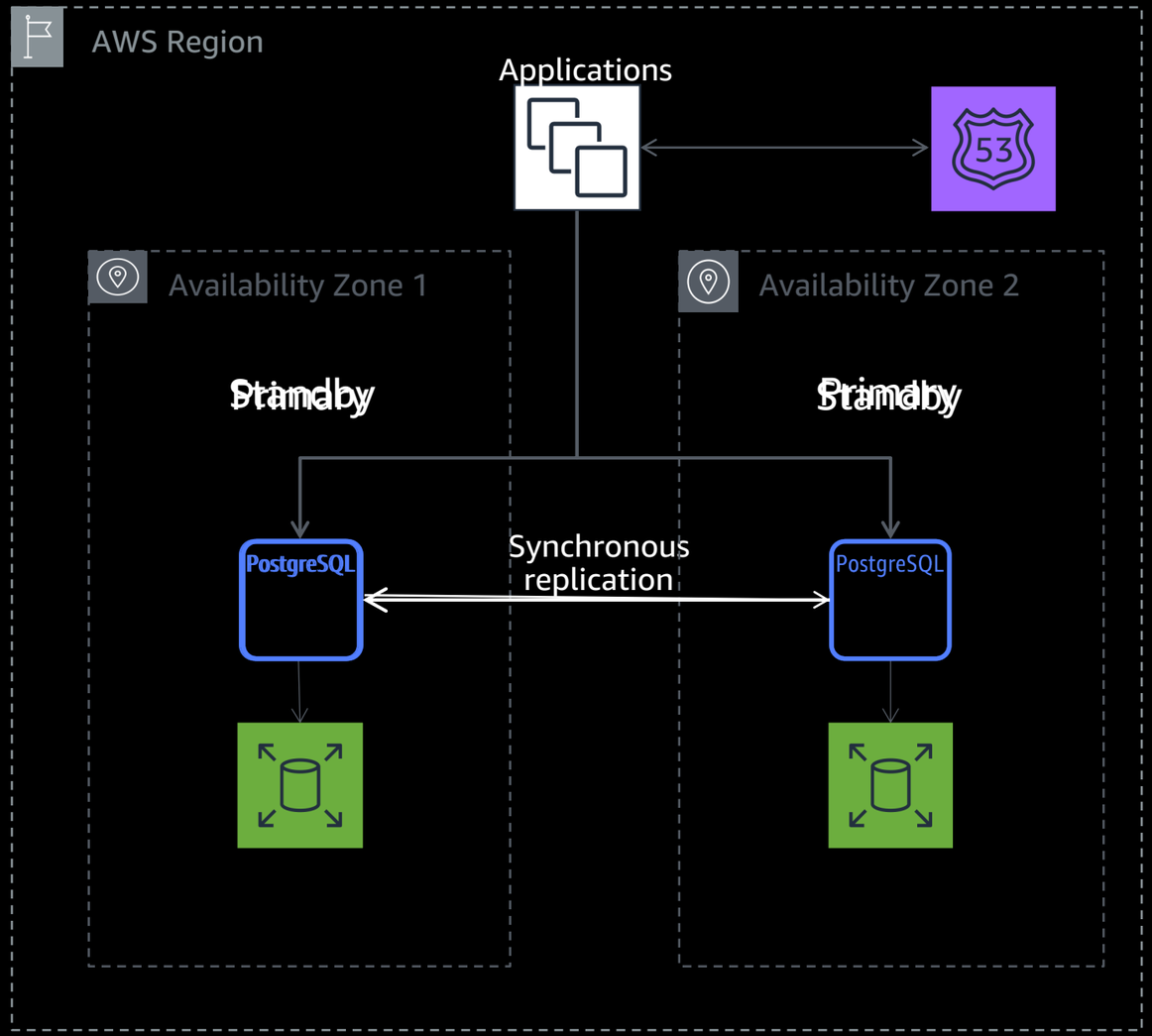
High availability for RDS PostgreSQL

Multi-AZ configurations provide a fault-tolerant solution for PostgreSQL

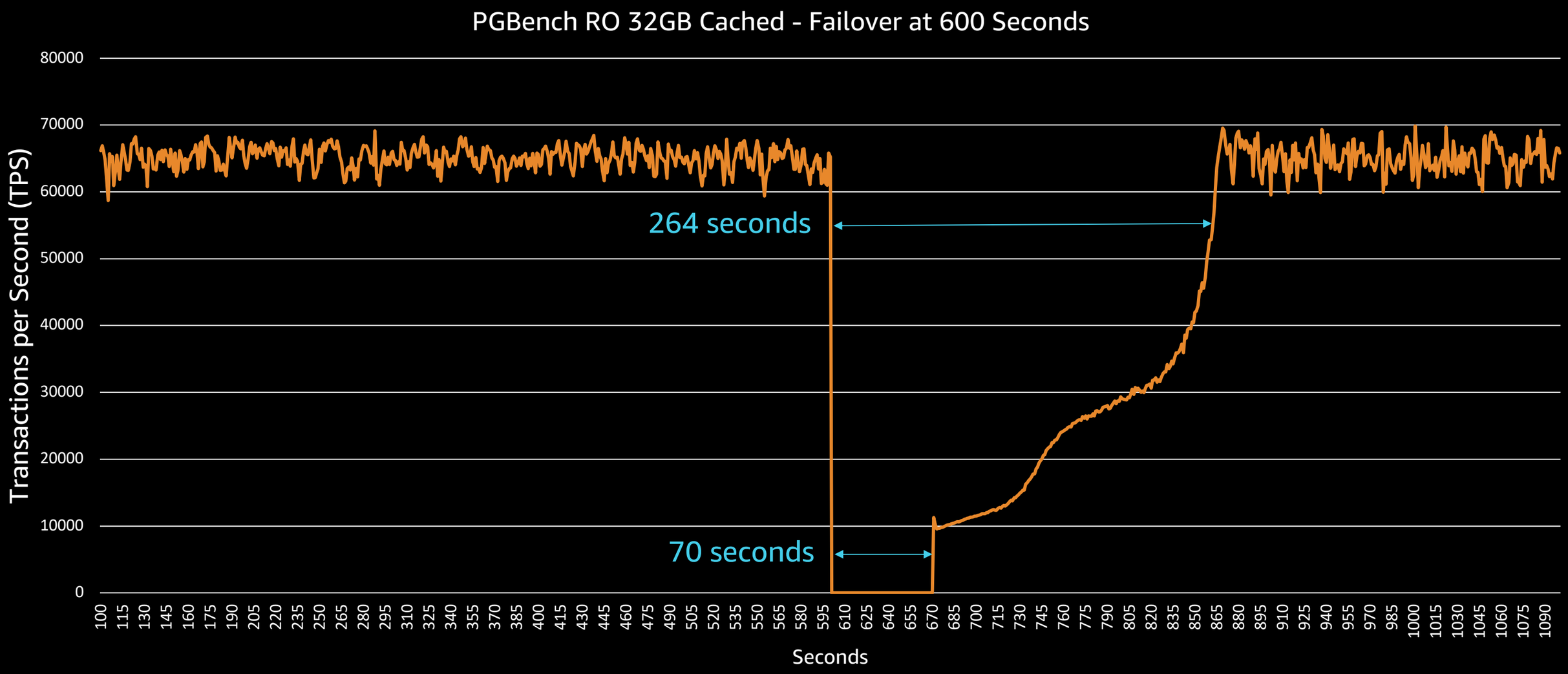
Each host synchronously maintains a set of volumes with a full copy of the data

Redirection to the new primary instance is provided through DNS (Route 53)

Detects infrastructure issues, not database engine problems



Cache warming

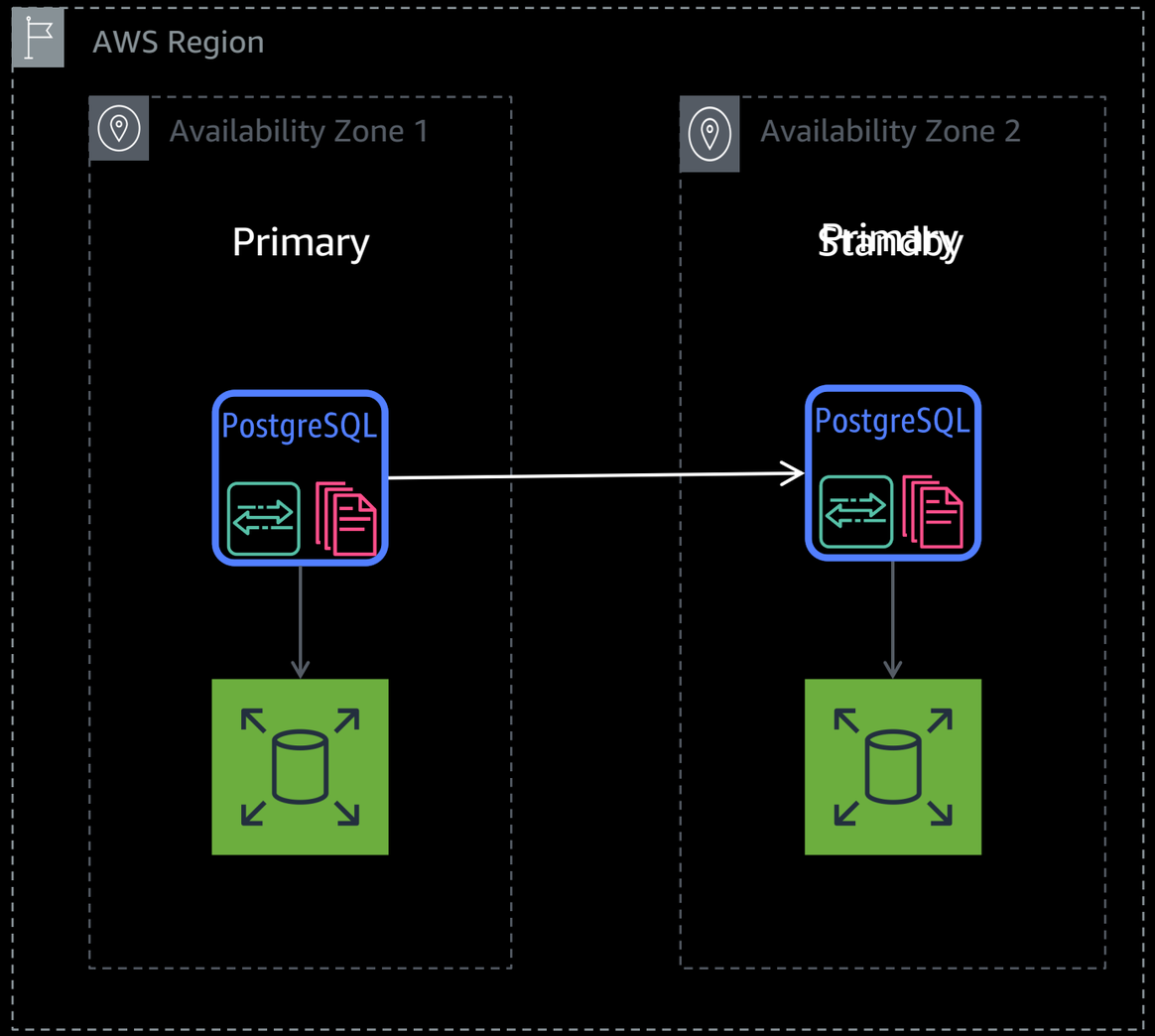


Pg_prewarm

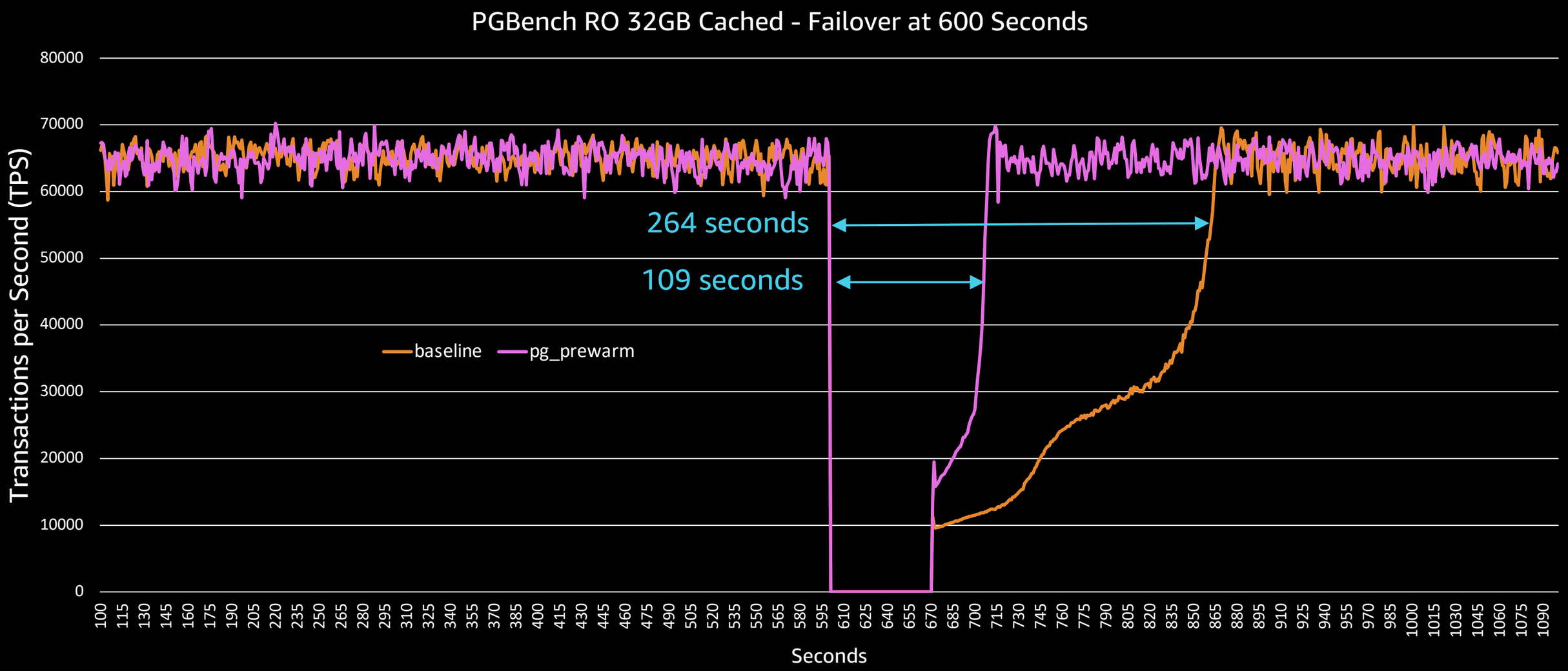
Extension available in all supported versions of PostgreSQL

Can manually load tables and indexes into cache

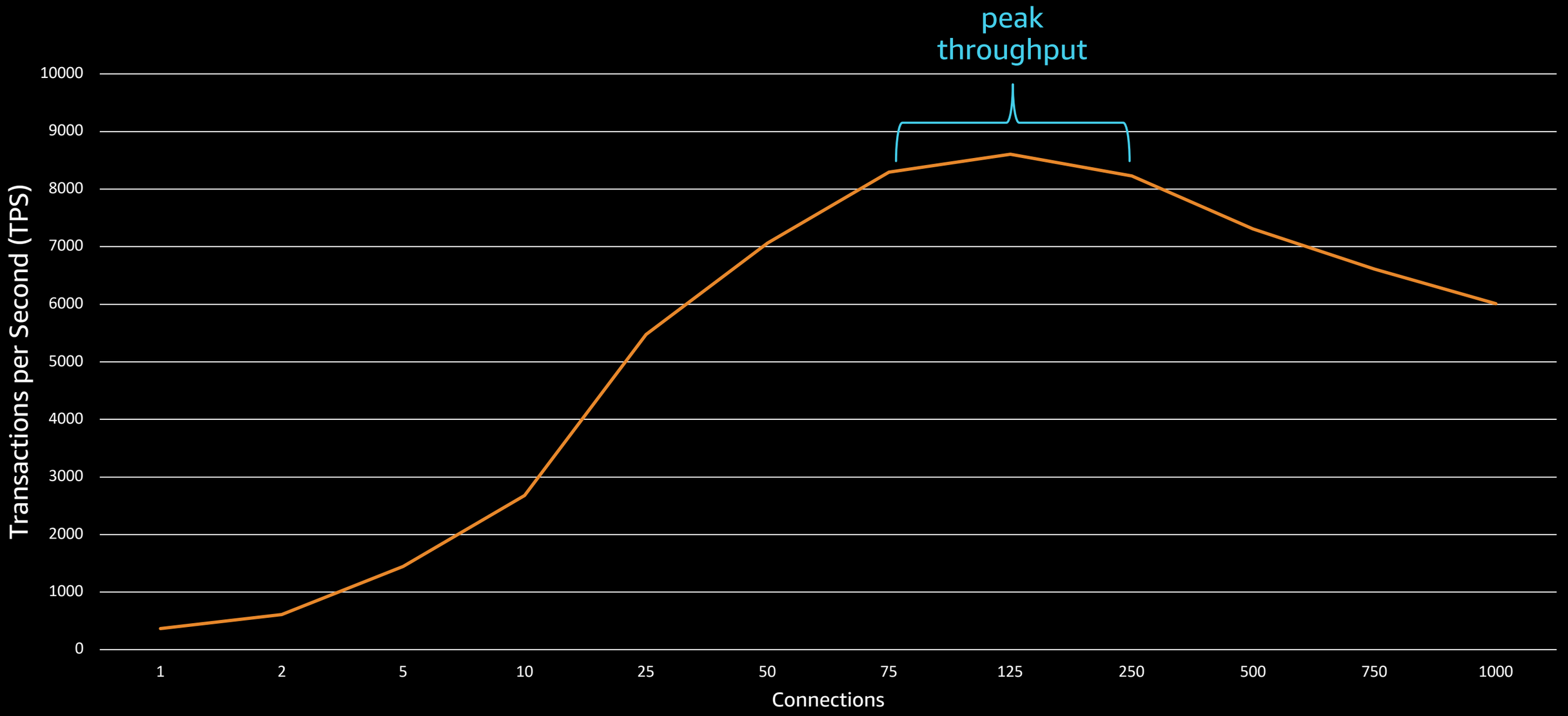
PostgreSQL 11 introduced auto prewarm to restore the cache after a restart or failover



Automatic cache warming



Connection scaling



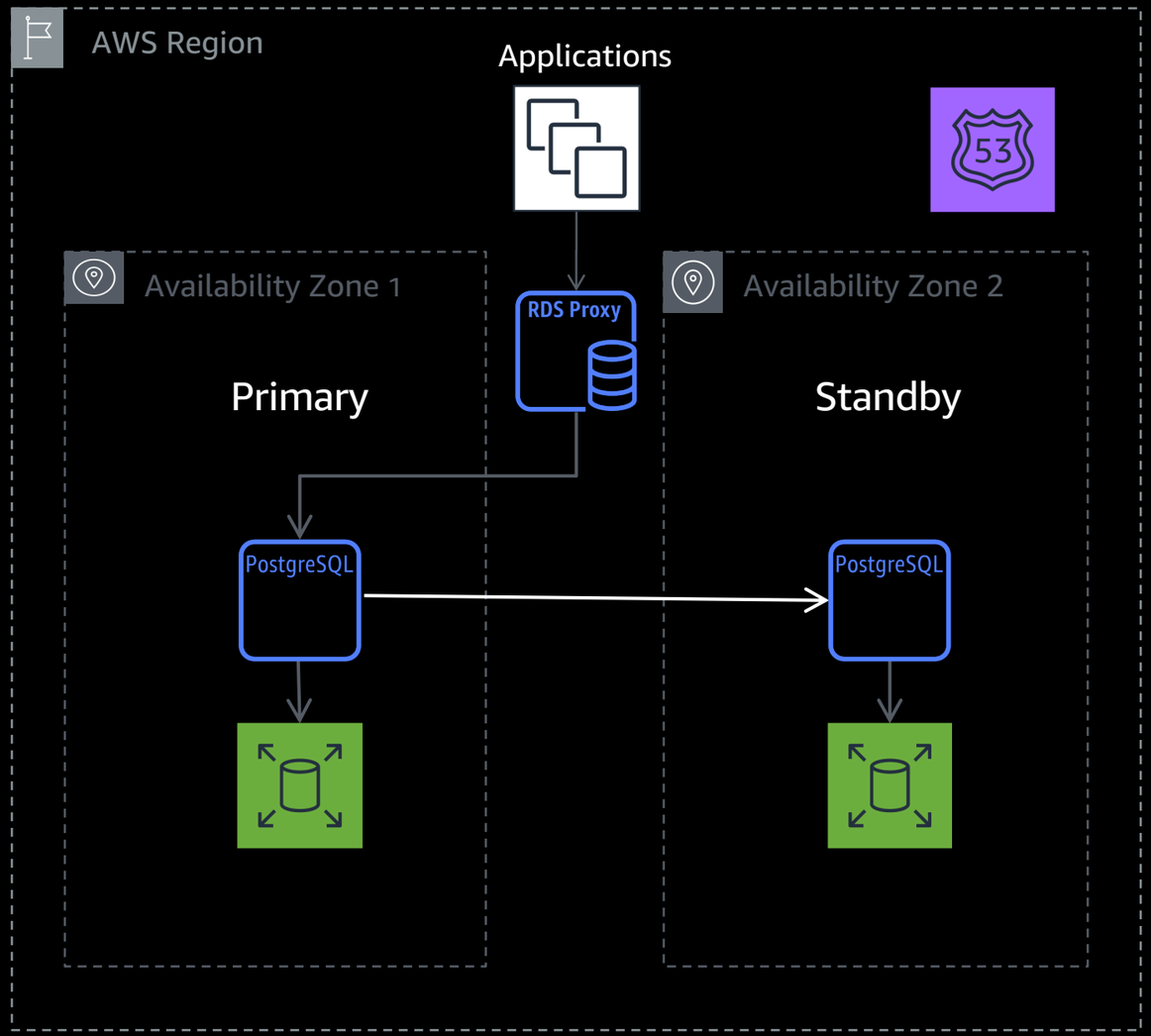
Amazon RDS Proxy

Fully managed database proxy

Pools and shares database connections

Increases application availability

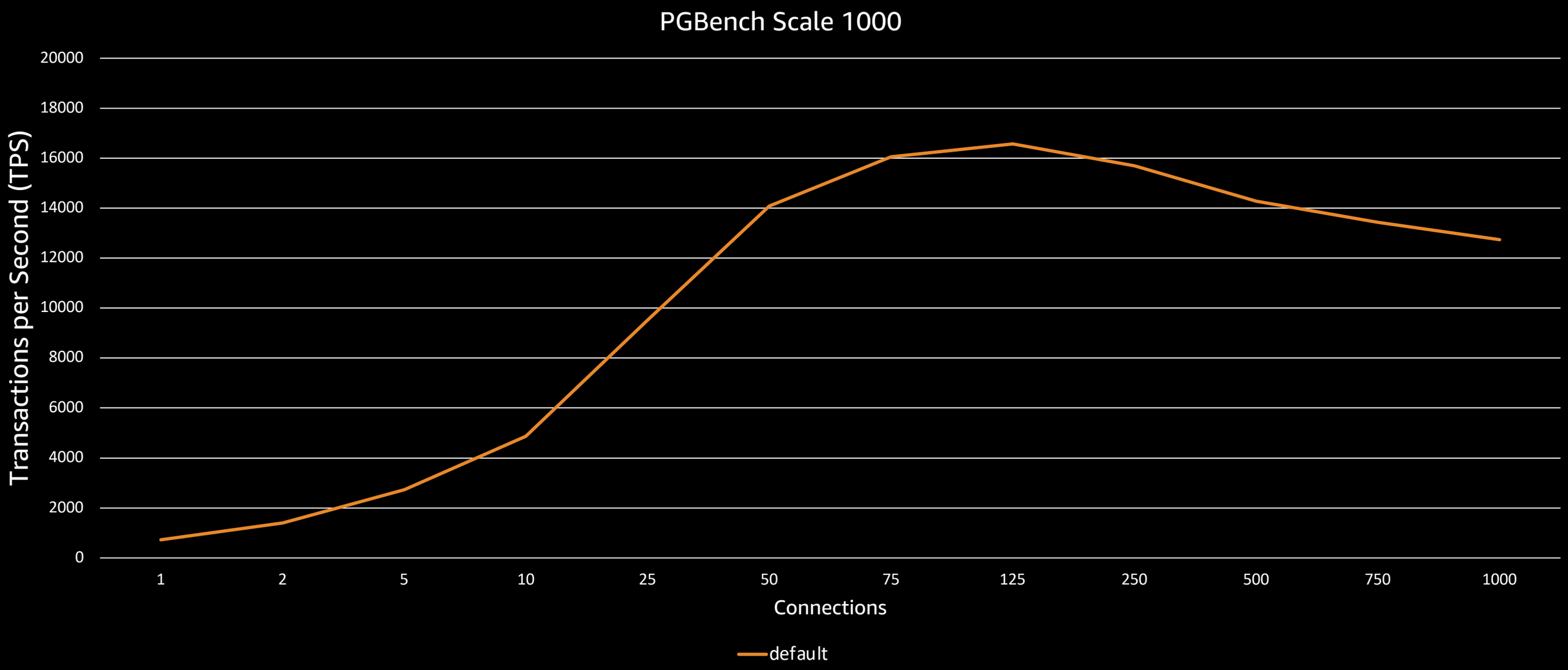
Reduces database failover times



Performance

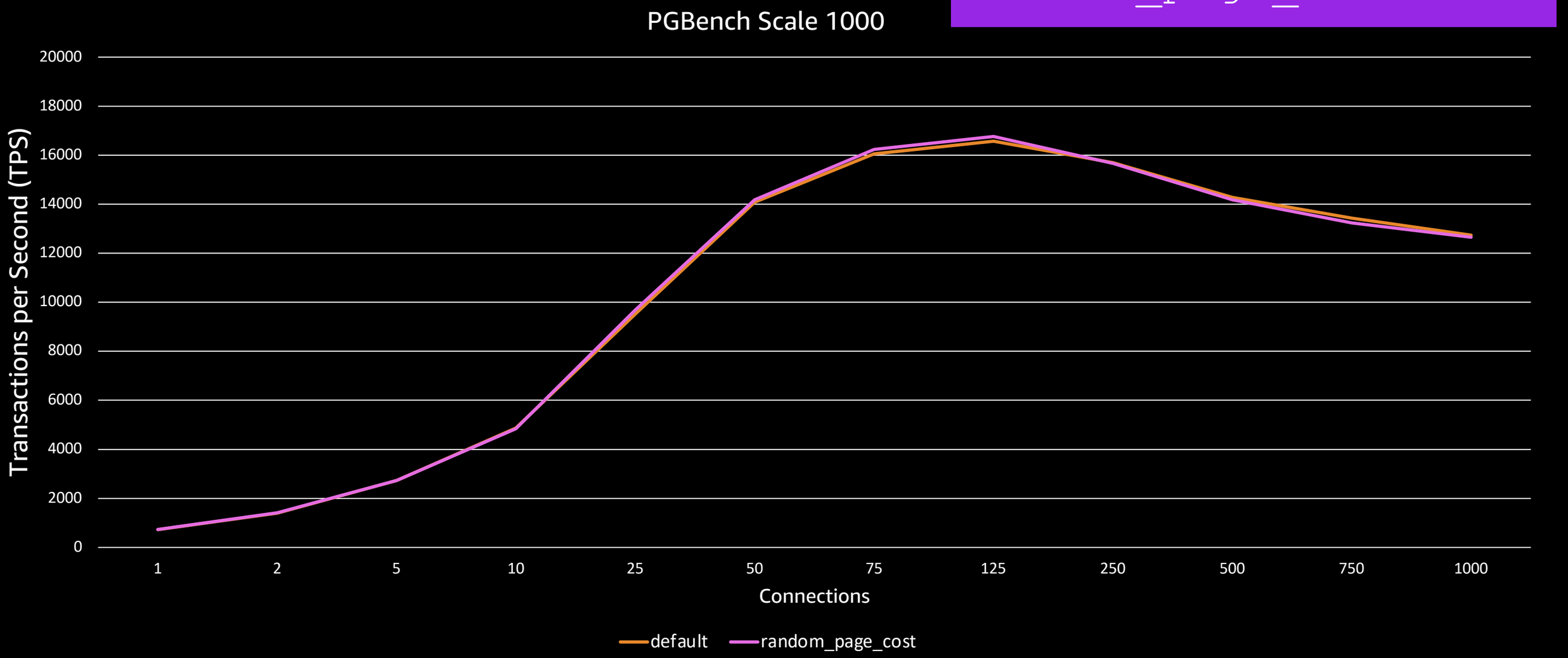
AWS re:Invent

Parameter changes



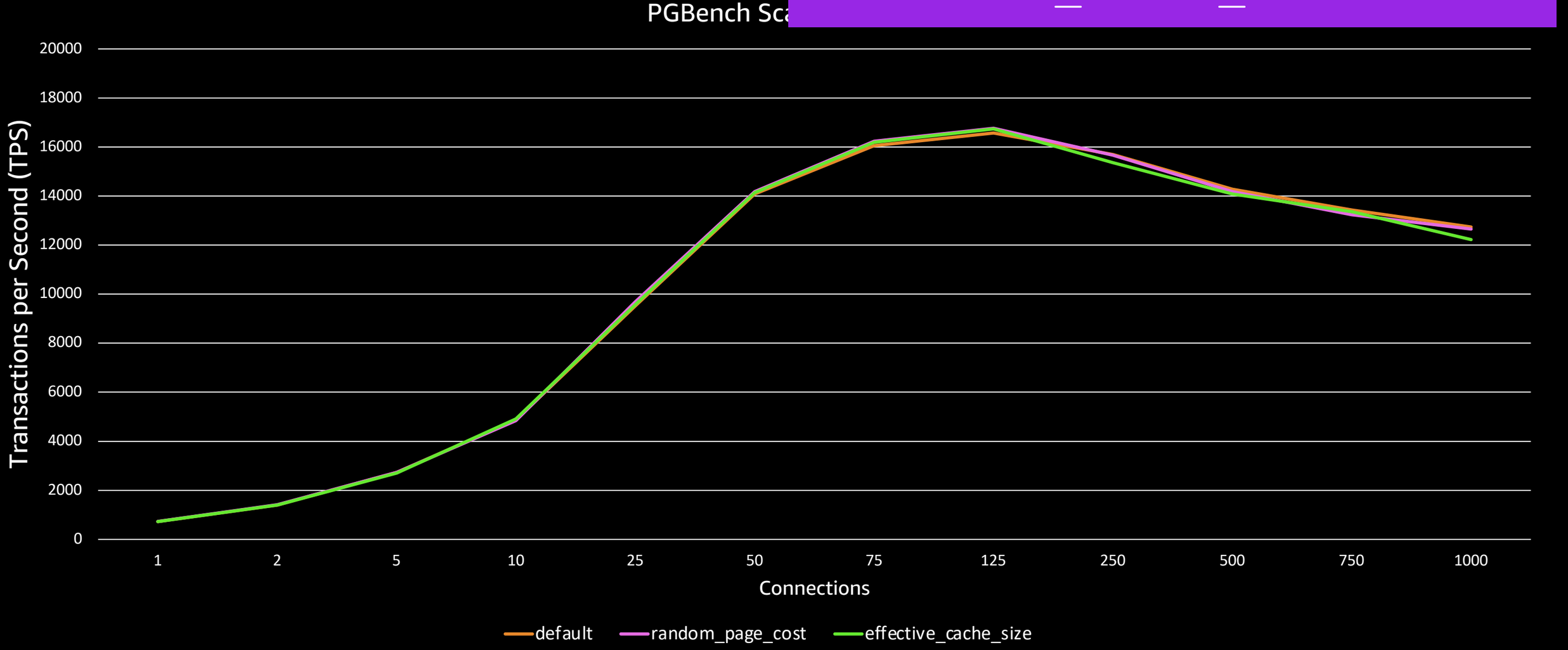
Parameter changes

```
random_page_cost = 1
```



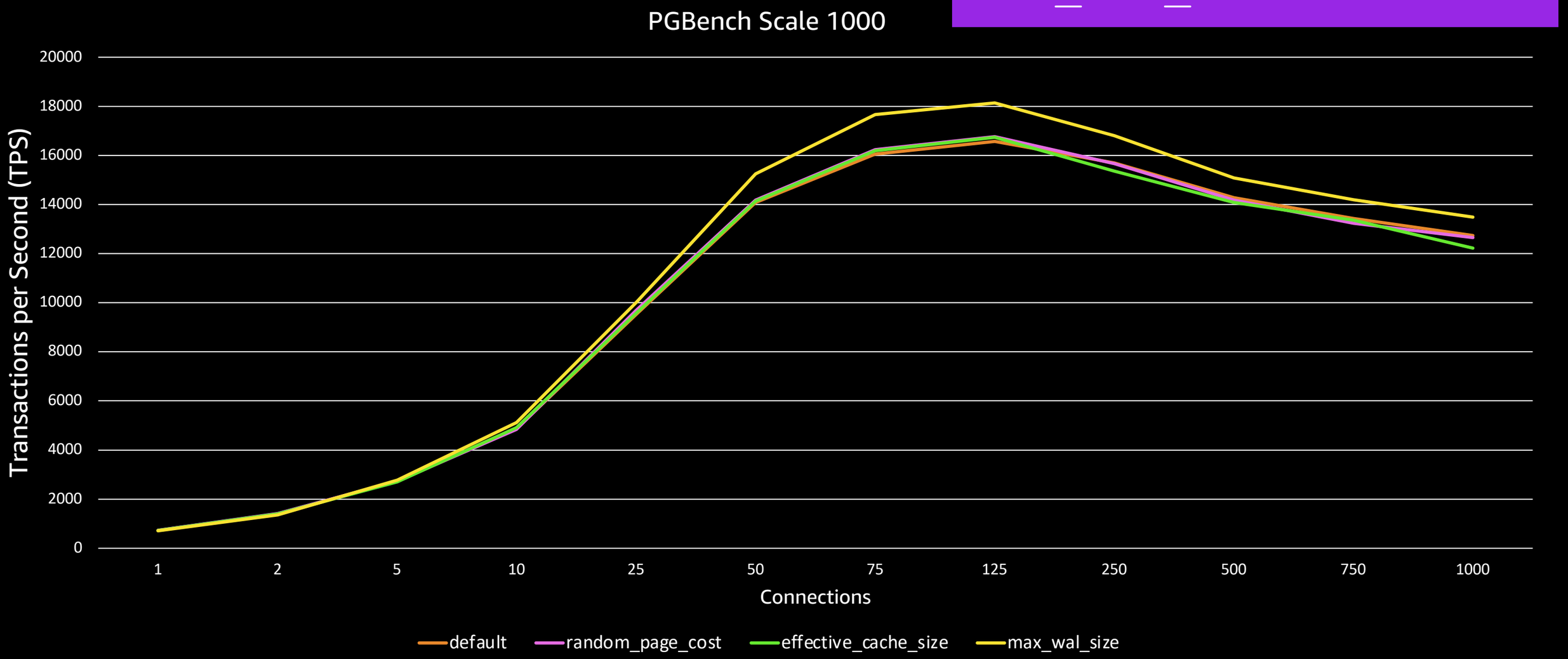
Parameter changes

```
effective_cache_size = 1TB
```



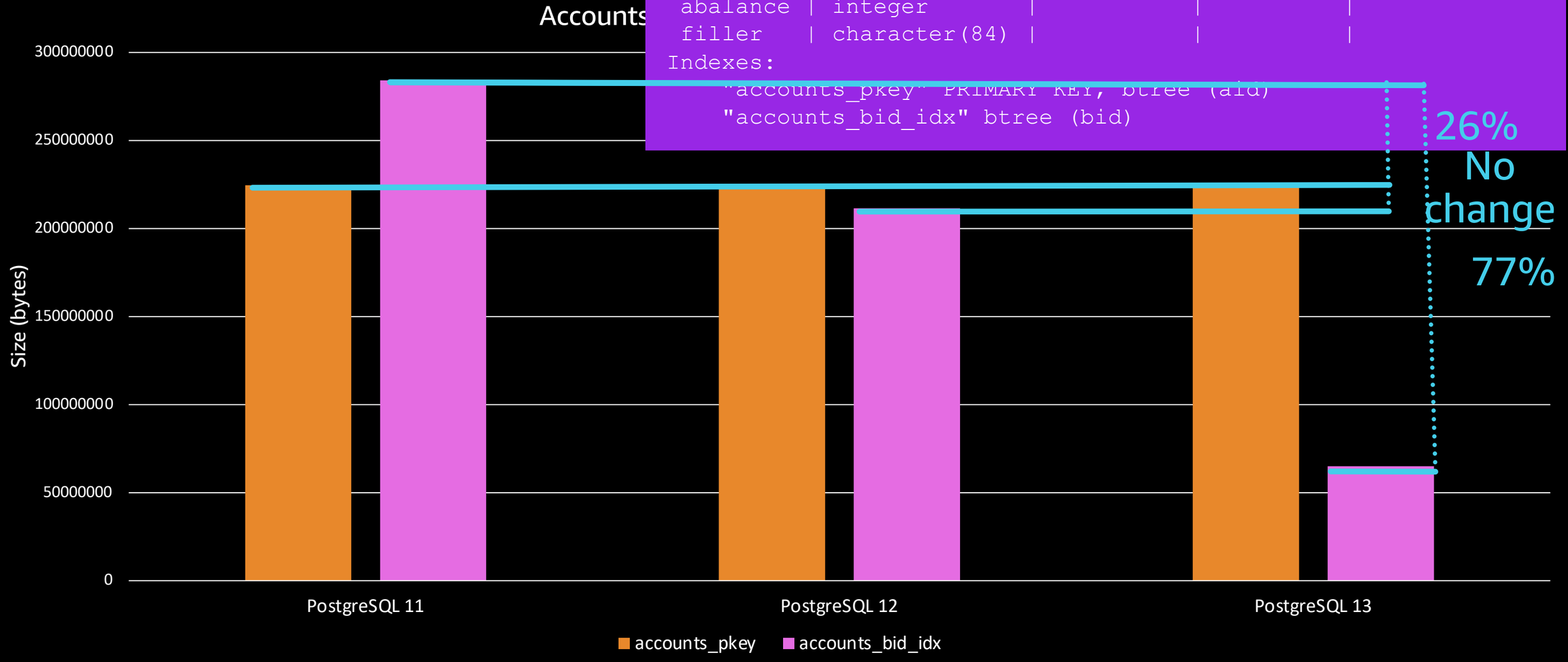
Parameter changes

```
max_wal_size = 20GB
```



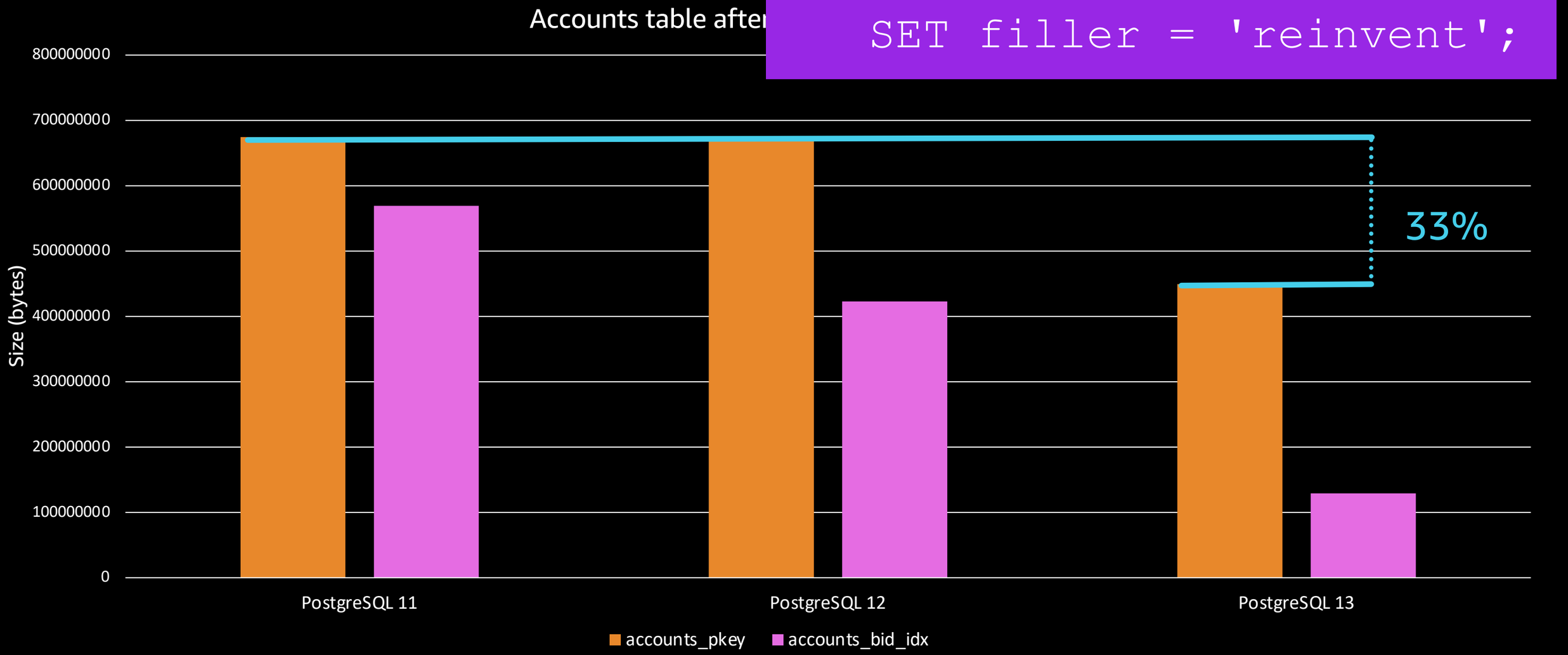
Index sizes

```
 Table "public.accounts"
  Column   |     Type      | Collation | Nullable | Default
-----------+---------------+-----------+----------+---------
 aid       | integer       |           | not null |
 bid       | integer       |           |          |
 abalance  | integer       |           |          |
 filler    | character(84) |           |          |
Indexes:
    "accounts_pkey" PRIMARY KEY, btree (aid)
    "accounts_bid_idx" btree (bid)
```



Index sizes

```
UPDATE accounts
   SET filler = 'reinvent';
```



Monitoring

Enhanced monitoring for Amazon RDS

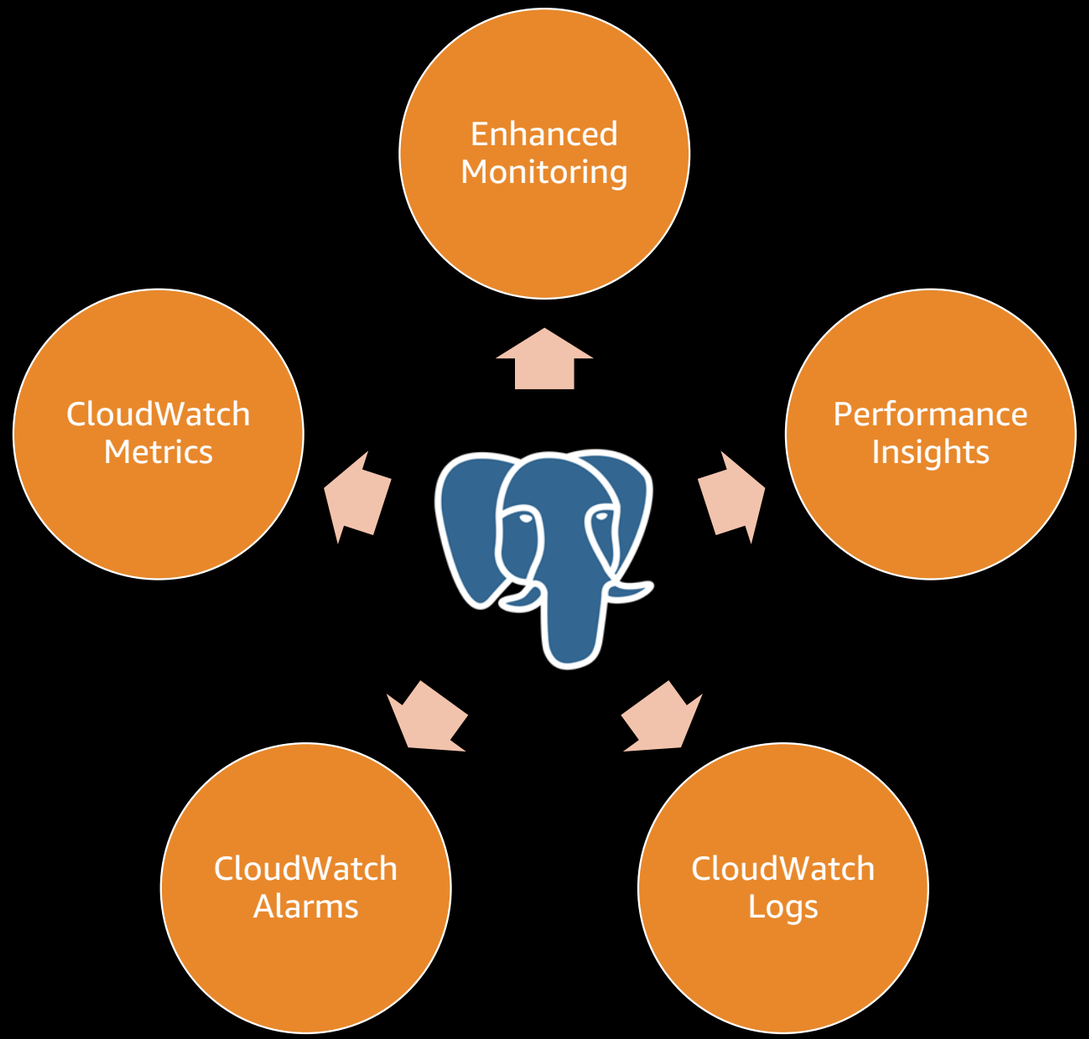
- Access to over 50 CPU, memory, file system, and disk I/O metrics

Amazon CloudWatch Metrics

- Displayed in the Amazon RDS console or in personalized CloudWatch dashboards

Amazon CloudWatch Alarms

- Alarms triggered based on metric values crossing configurable thresholds



CPU intensive functions

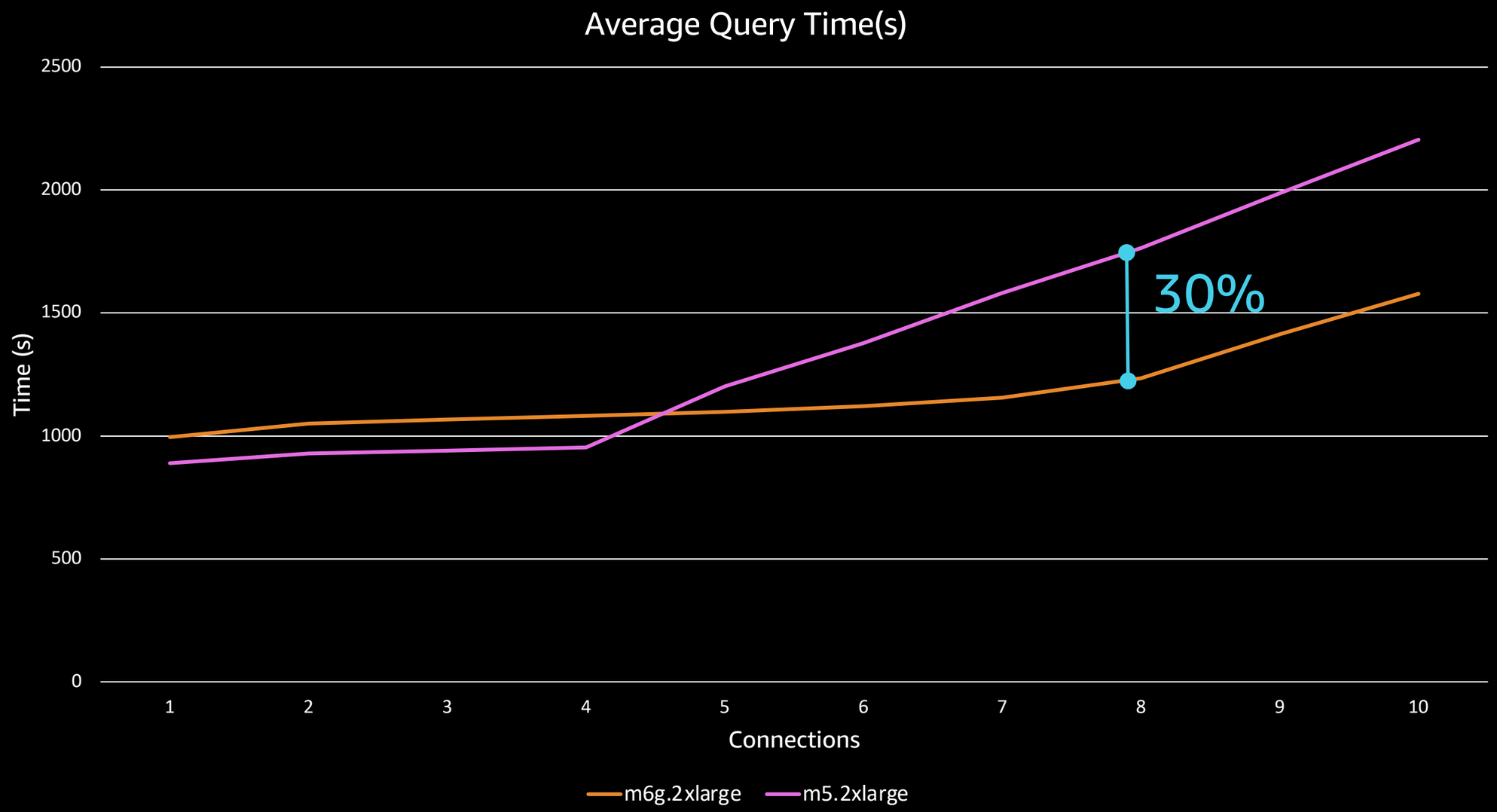
Many applications put complex logic in stored procedures

Some are CPU intensive

- Geospatial
- Full-text search

```
CREATE FUNCTION fib(n int)
  RETURNS int AS
$$
BEGIN
  IF n <= 1 THEN
    RETURN n;
  END IF;

  RETURN fib(n-1) + fib(n-2);
END;
$$ LANGUAGE plpgsql;
```

CPU intensive functions on Graviton



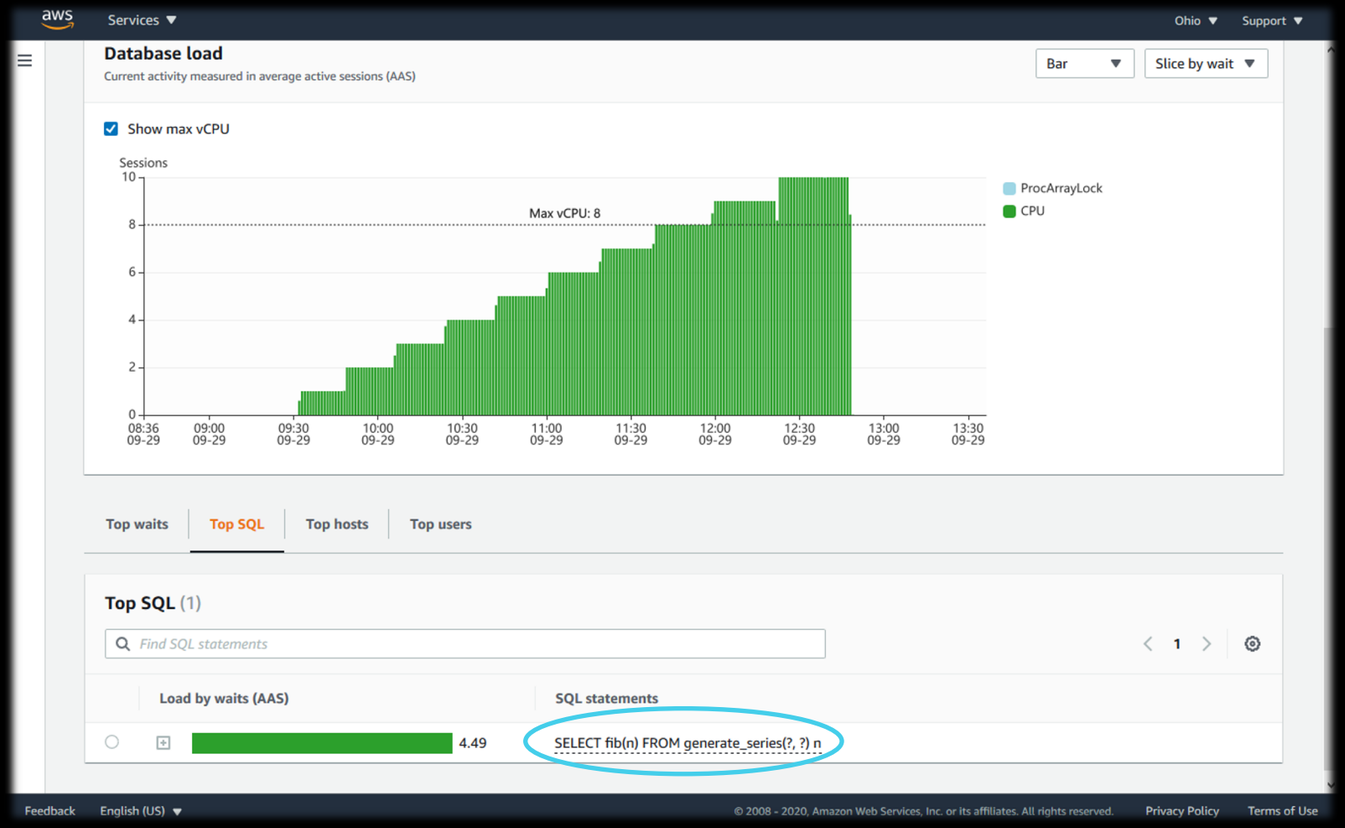
Amazon RDS performance insights

Dashboard shows database load over time

Identifies bottlenecks

- Sort by top SQL
- Slice by wait events, host, user

Store up to 2 years of metrics



OS level visibility

Pg_proctab exposes the OS /proc information through SQL

Available as an extension

Can tie individual queries to CPU, memory and IO usage

```
postgres=> SELECT substr(sa.query, 0, 14) as query, sa.wait_event,
       pt.state, pt.utime, pt.stime, pt.rss,
       pt.rchar, pt.wchar
  FROM pg_proctab() pt, pg_stat_activity sa
 WHERE sa.pid = pt.pid
   AND sa.pid != pg_backend_pid();
     query     |     wait_event      | state | utime  | stime |  rss  |   rchar   |    wchar
---------------+---------------------+-------+--------+-------+-------+-----------+-------------
               | AutoVacuumMain      | S     |    939 |  1510 |  8020 | 461888028 |      231830
               | LogicalLauncherMain | S     |      5 |     5 |  6832 |       562 |           2
 SELECT value  | ClientRead          | S     |    897 |   684 | 15944 |   2094218 |   546611799
 SELECT fib(n) |                     | R     | 129222 |    45 | 16192 |    172295 |           0
 SELECT fib(n) |                     | R     | 129380 |    45 | 16184 |    172295 |           0
 SELECT fib(n) |                     | R     | 131358 |    45 | 16300 |    172295 |           0
 SELECT fib(n) |                     | R     | 131427 |    33 | 16276 |    172295 |           0
 SELECT fib(n) |                     | R     | 130163 |    51 | 16252 |    172295 |           0
 SELECT fib(n) |                     | R     | 130397 |    47 | 16232 |    172295 |           0
 SELECT fib(n) |                     | R     | 131418 |    47 | 16252 |    172295 |           0
 SELECT fib(n) |                     | R     | 131615 |    49 | 16276 |    172295 |           0
 SELECT fib(n) |                     | R     | 130999 |    43 | 16320 |    172295 |           0
 SELECT fib(n) |                     | R     | 131542 |    46 | 16396 |    172295 |           0
               | BgWriterHibernate   | S     |    150 |   123 |  4332 |        69 |          21
               | CheckpointerMain    | S     |   1702 |  4080 |  6260 |        52 | 77683956205
               | WalWriterMain       | S     |   1550 |   241 |  4332 |      2439 |    20261207
(16 rows)
```

Profiling stored procedures

Plprofiler is an extension that creates performance profiles of stored procedure code

Provides the execution count and timing of individual code lines

Helps pinpoint code bottlenecks

```
SHOW shared_preload_libraries ;
        shared_preload_libraries
-----------------------------------------
 rdsutils,pg_stat_statements,plprofiler
(1 row)

CREATE EXTENSION plprofiler;

SELECT pl_profiler_set_enabled_local(true);
SELECT pl_profiler_set_collect_interval(0);

SELECT fib(35);
```

Profiling stored procedures

```
SELECT (regexp_split_to_array(
              prosrc,
              '\n'))[line_number]
              as line,
       exec_count,
       total_time
  FROM pl_profiler_linestats_local(),
       pg_proc
 WHERE func_oid = oid
 ORDER BY line_number;
```

line	exec_count	total_time
	29860703	2189639410
	0	0
BEGIN	29860703	2188447361
IF n <= 1 THEN	29860703	16213
RETURN n;	14930352	1089
END IF;	0	0
	0	0
RETURN fib(n-1) + fib(n-2);	14930351	2184767917
END;	0	0
	0	0
	0	0

2184 seconds

Profiling stored procedures

77 microseconds

```
CREATE OR REPLACE FUNCTION fib(n int)
  RETURNS int AS
$$
DECLARE
  f int[];
BEGIN
  f[0] = 0;
  f[1] = 1;

  FOR i IN 2..n LOOP
    f[i] = f[i-1] + f[i-2];
  END LOOP;

  RETURN f[n];
END;
$$ LANGUAGE plpgsql;
```

line	exec_count	total_time
	1	77
	0	0
DECLARE	0	0
f int[];	0	0
BEGIN	1	76
f[0] = 0;	1	8
f[1] = 1;	1	5
	0	0
FOR i IN 2..n LOOP	1	58
f[i] = f[i-1] + f[i-2];	34	43
END LOOP;	0	0
	0	0
RETURN f[n];	1	2
END;	0	0
	0	0
	0	0

Upgrades

AWS re:Invent

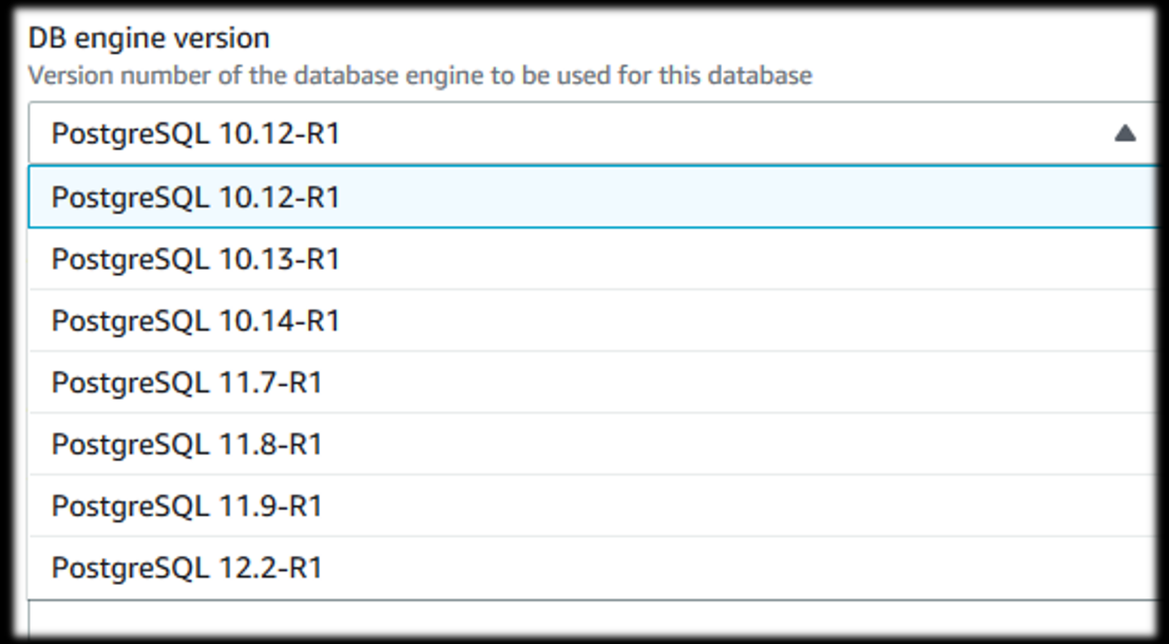
Upgrades

Minor version upgrades

- Patches to the binaries
- No new functionality
- May contain important security fixes

Major version upgrades

- Tracks the community yearly release cycle
- Introduces new functionality
- May change system catalogs and page formats

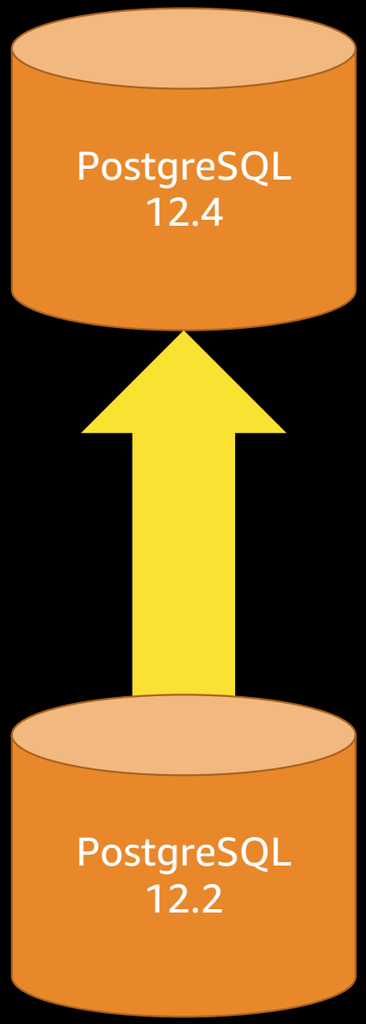


Minor version upgrades

The minor version upgrade process typically takes less than 5 to 10 minutes

- Normal shutdown of the instance
  - The time can be extended with long in-flight transactions
- Replace version binaries
- Start the instance

Can be performed manually or automatically

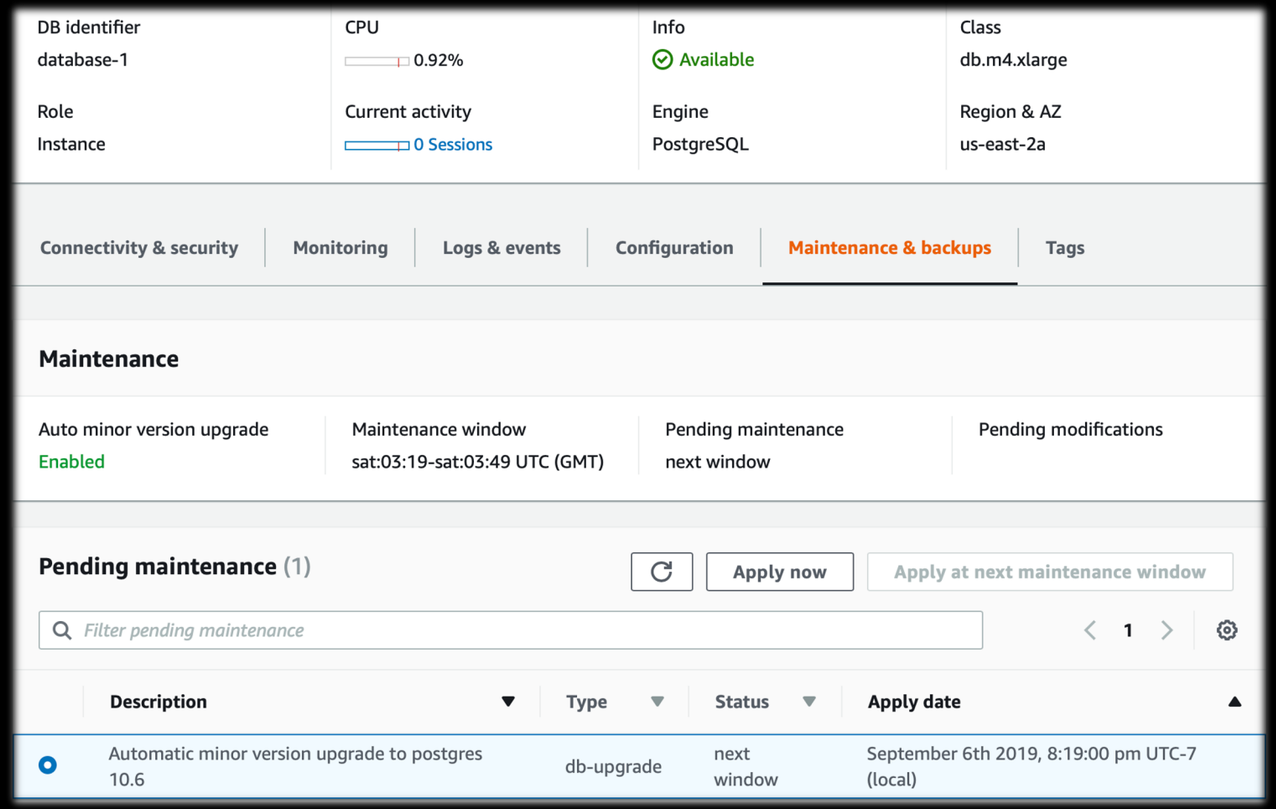


Auto minor version upgrade (AmVU)

Scheduled when the newer minor version is marked as preferred

By default, AmVU is ON

If a database is on a lower version, the instance is upgraded to the preferred version in the next maintenance window



Major version upgrades

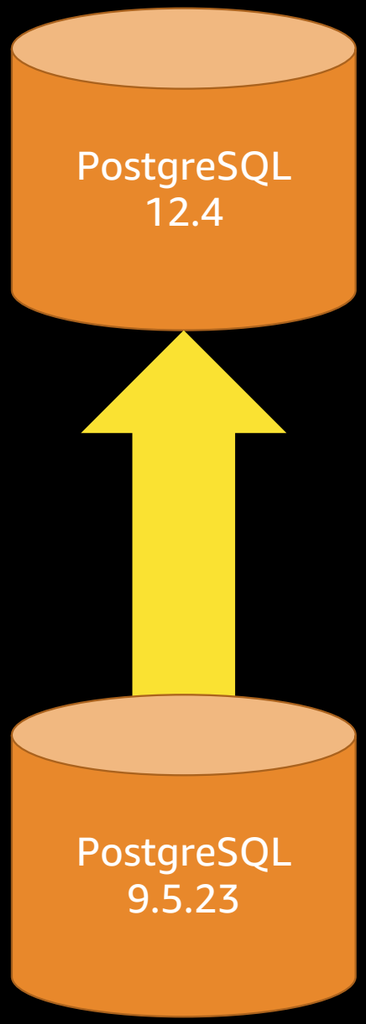
Upgrades can skip major versions with multi major version upgrade

The time needed to upgrade depends more on the number of database objects than the database size

Snapshots taken before and after the upgrade

Statistics are not upgraded so ANALYZE is needed on the new version

A new parameter group is needed for the new version



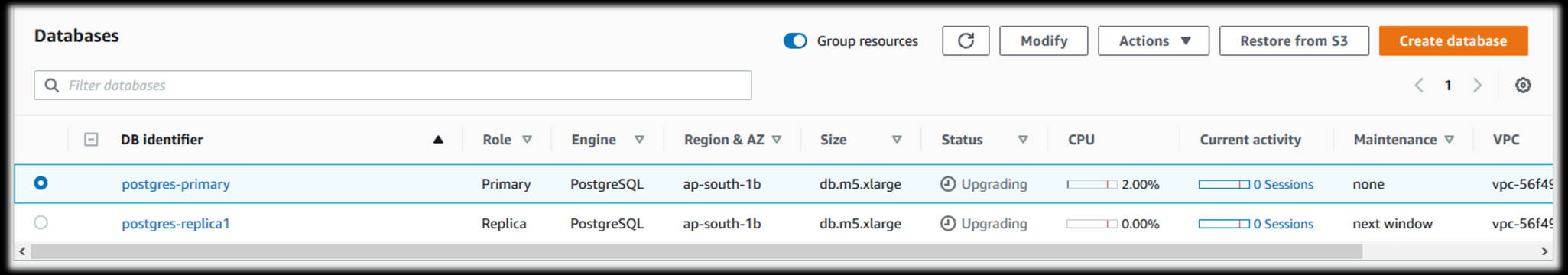
Major version upgrades with read replicas

Replicas within the same region are upgraded with the primary

Replicas are upgraded after the primary has successfully upgraded

Cross region replicas are disconnected from the primary

To prevent replicas from being upgraded, promote them before the upgrade

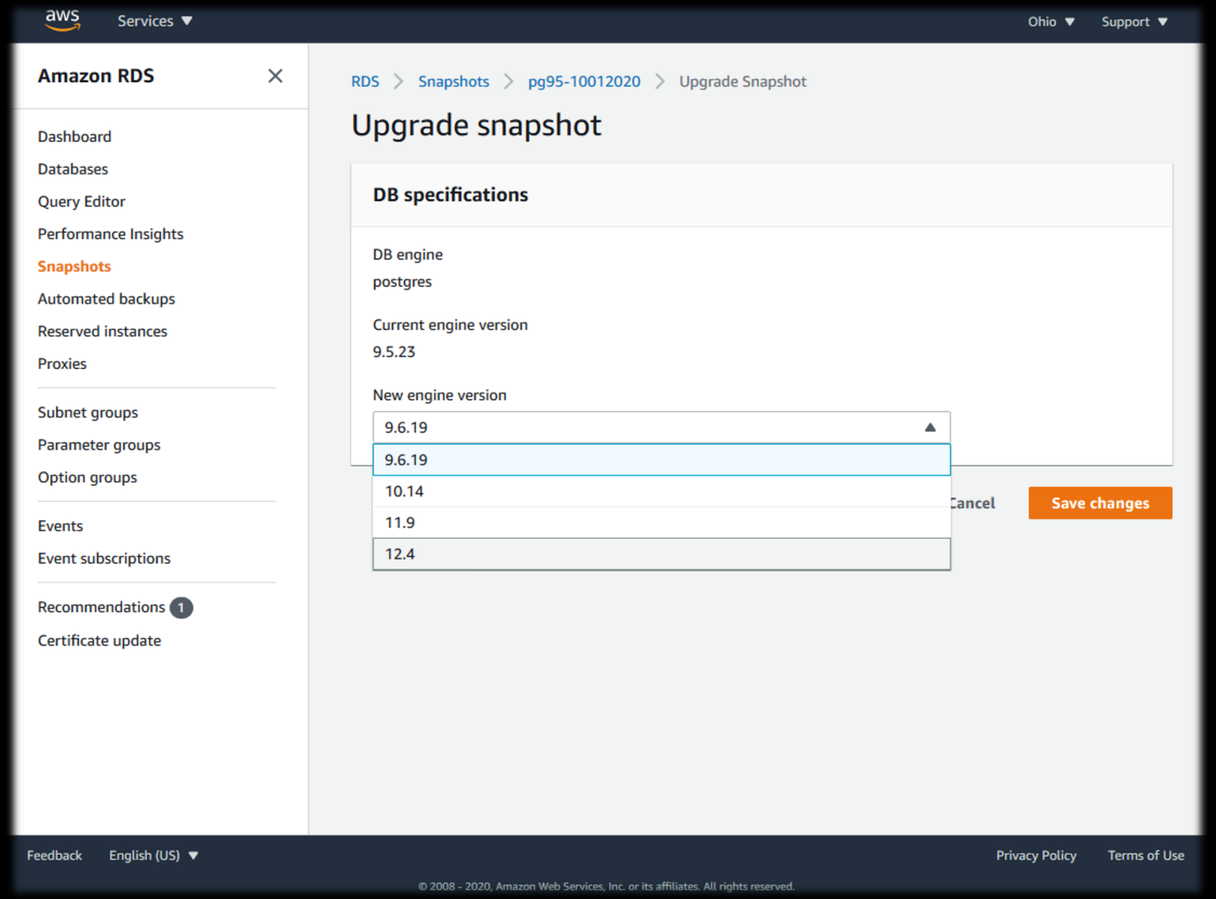


Snapshot upgrades

Upgrades the engine version of user-initiated snapshots to a higher supported version

Useful when the original database version is no longer supported

Copies of snapshots can be upgraded keeping the original



Moving data

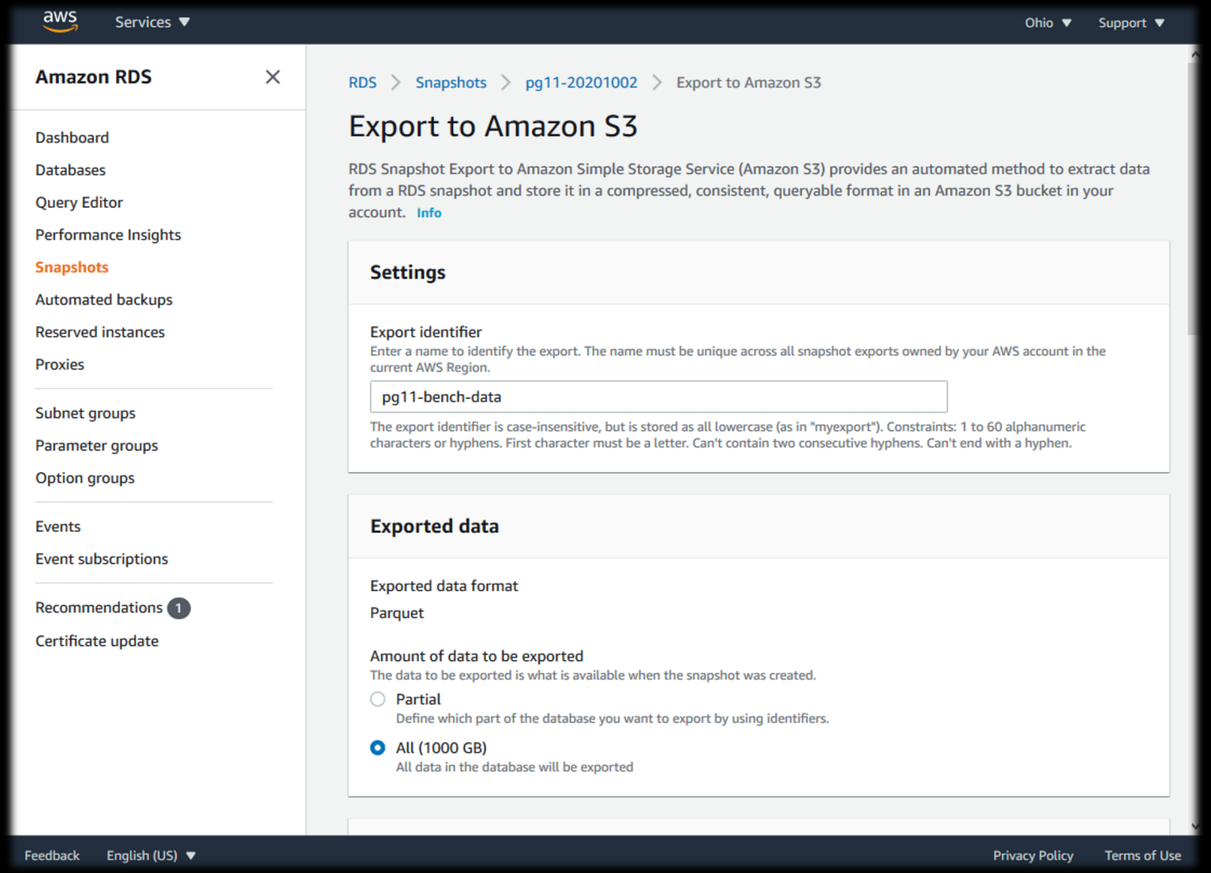
AWS re:Invent

Snapshot exports to Amazon S3

Exports a whole or partial database snapshot to an Amazon S3 bucket

Actions are offline from the database so there is no performance impact

Files created with a Parquet format to be consumed by Amazon Athena, Apache Spark (on Amazon EMR) or Amazon Redshift Spectrum



Amazon S3 query export

The aws_s3 extension adds a function to export to S3

Permissions are managed by AWS Identity and Access Management (IAM) roles and polices

The function follows the COPY command options

```
CREATE EXTENSION aws_s3 CASCADE;

SELECT *
  FROM aws_s3.query_export_to_s3(
         'SELECT *
            FROM pgbench_branches',
        'pg-query-bucket',
        'branches',
        'us-east-2',
        'format csv');
```

Amazon S3 table import

Loads a file from Amazon S3 directly into a PostgreSQL table using the COPY syntax

Available using the aws_s3 extension

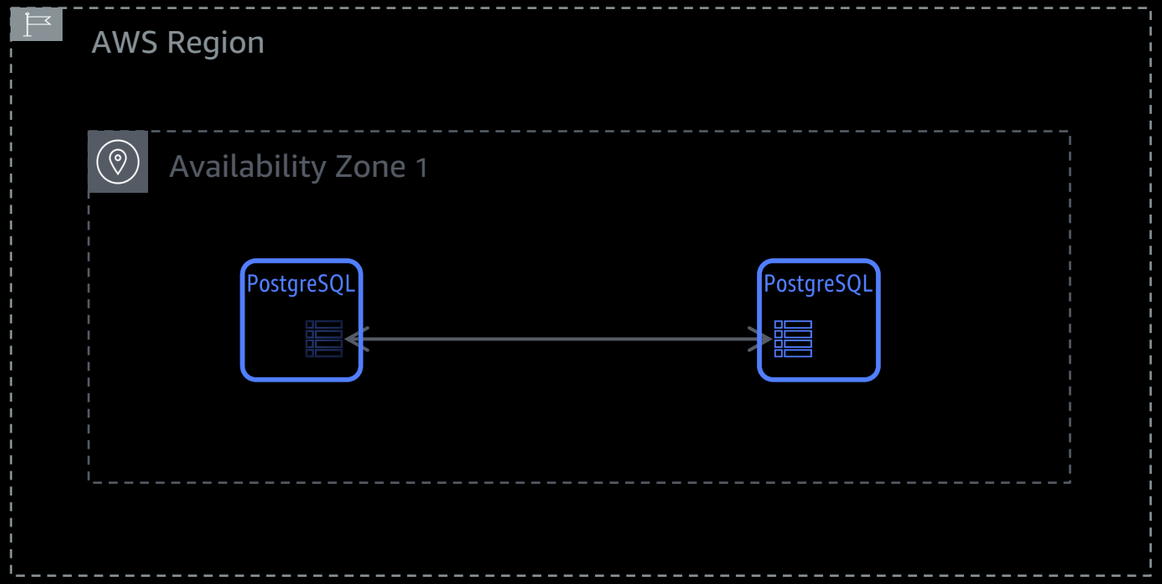
```
CREATE EXTENSION aws_s3 CASCADE;

SELECT aws_s3.table_import_from_s3(
             'table1',
             'col1,col2,col3',
             '(format csv)',
          aws_commons.create_s3_uri(
                       p_bucket,
                       p_event_file,
                       p_region));
```

PostgreSQL foreign data wrappers

Creates a database link to another PostgreSQL database

Allows SELECT, INSERT, UPDATE and DELETE

Can push down many operations like joins, aggregates and sorts

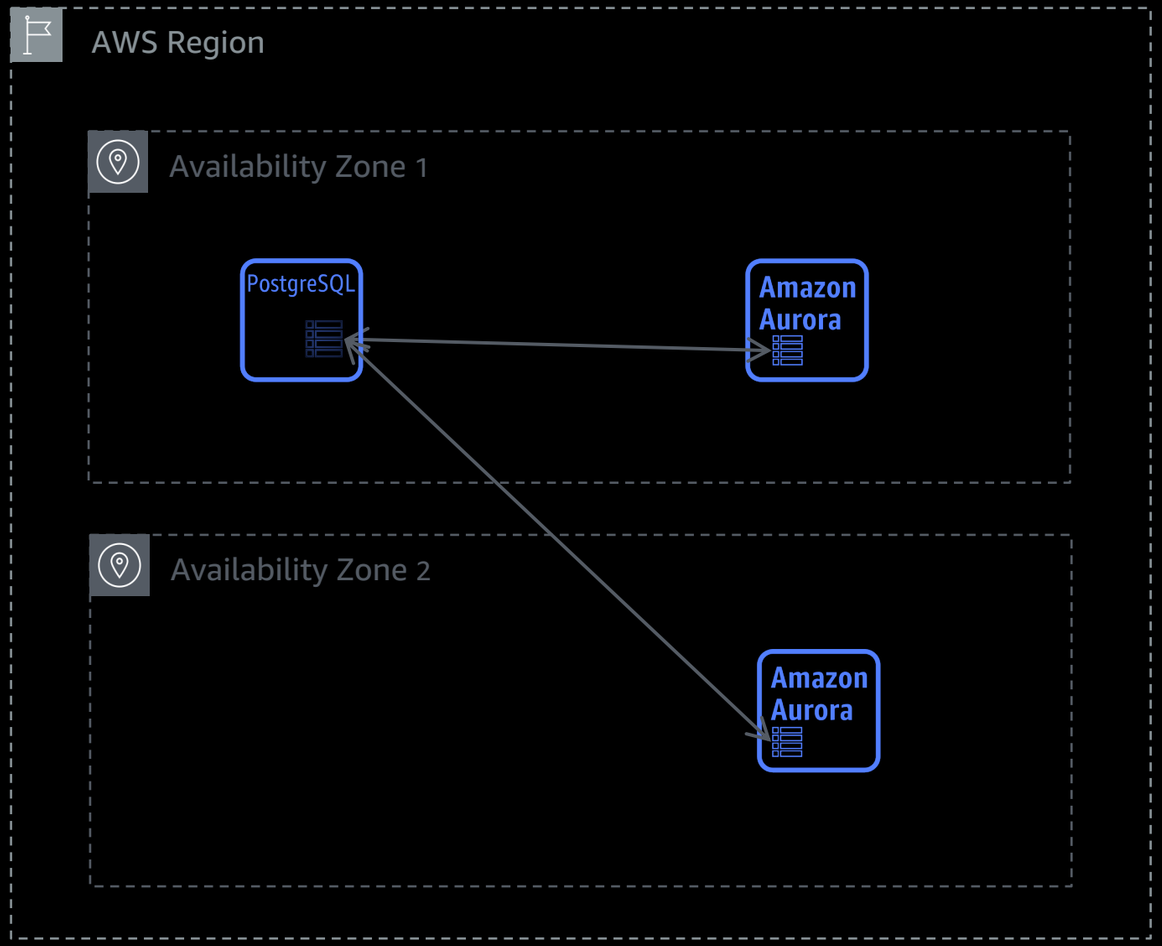


PostgreSQL foreign data wrappers

Creates a database link to another PostgreSQL database

Allows SELECT, INSERT, UPDATE and DELETE

Can push down many operations like joins, aggregates and sorts

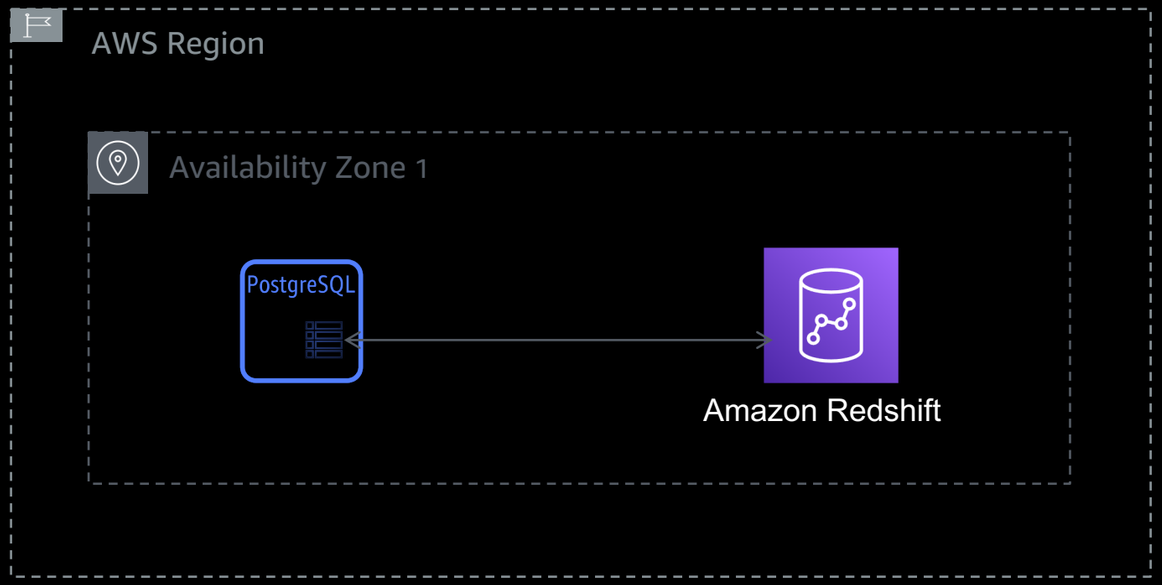


PostgreSQL foreign data wrappers

Creates a database link to another PostgreSQL database

Allows SELECT, INSERT, UPDATE and DELETE

Can push down many operations like joins, aggregates, and sorts



PostgreSQL foreign data wrappers

```
CREATE EXTENSION postgres_fdw;

CREATE SERVER redshift
 FOREIGN DATA WRAPPER postgres_fdw
 OPTIONS
  (dbname 'dw',
   host 'redshift-dw.us-east-2.redshift.amazonaws.com',
   port '5432');

CREATE USER MAPPING for CURRENT_USER
SERVER redshift
OPTIONS (user 'dwuser', password 'secret');
```

PostgreSQL foreign data wrappers

```
CREATE SCHEMA redshift;

CREATE FOREIGN TABLE redshift.transaction_history (
    tid     bigint,
    bid     integer,
    aid     integer,
    delta   integer OPTIONS (column_name 'amount'),
    mtime   timestamp,
    filler  varchar
)
SERVER redshift
OPTIONS (schema_name 'public',
         table_name 'transaction_history');
```

PostgreSQL foreign data wrappers

```
INSERT INTO redshift.transaction_history
SELECT *
  FROM pgbench_history;
```

```
postgres=> EXPLAIN VERBOSE SELECT avg(delta) FROM
redshift.transaction_history;
                              QUERY PLAN
-----------------------------------------------------------------
 Foreign Scan  (cost=107.31..146.59 rows=1 width=32)
   Output: (avg(delta))
   Relations: Aggregate on (redshift.transaction_history)
   Remote SQL: SELECT avg(amount) FROM public.transaction_history
(4 rows)
```

Related sessions

DAT301

Deep dive on Amazon Aurora with PostgreSQL compatibility

DAT202

What's new in Amazon RDS

AWS re:Invent

Thank you!

AWS re:Invent

Please complete the session survey

AWS re:Invent