AWS Whitepaper

Optimizing PostgreSQL Running on Amazon EC2 Using Amazon EBS

Optimizing PostgreSQL Running on Amazon EC2 Using Amazon EBS: AWS Whitepaper

Table of Contents

Optimizing PostgreSQL Running on Amazon EC2 Using Amazon EBS

Publication date: **October 19, 2023** (*Document history*)

This whitepaper is intended for Amazon Web Services (AWS) customers who are considering deploying their PostgreSQL database on Amazon Elastic Compute Cloud (Amazon EC2) using Amazon Elastic Block Store (Amazon EBS) volumes. This whitepaper describes the features of EBS volumes and how they can affect the security, availability, durability, cost, and performance of PostgreSQL databases. There are many deployment options and configurations for PostgreSQL on Amazon EC2. This whitepaper provides performance benchmark metrics and general guidance so AWS customers can make an informed decision about deploying their PostgreSQL workloads on Amazon EC2.

Introduction

PostgreSQL is an advanced, enterprise class open-source relational database that supports storing and querying relational and non-relational data. It is a highly stable database management system, backed by over 25 years of community development which has contributed to its high levels of resiliency, integrity, and performance. PostgreSQL is used as the primary data store or data warehouse for many web, mobile, geospatial, and analytics applications. It supports variety of data types including geographic data, key-value, one-dimensional arrays, geolocation-based data and multidimensional points. PostgreSQL also supports full-text search and vector similarity search. It's Multi version Concurrency Control (MVCC) architecture allows multiple transactions to read and write to the database concurrently without interfering each other. Apart from the robustness of the database engine, another benefit of PostgreSQL is that the total cost of ownership is low compared to commercial database engines. Several organizations are moving their PostgreSQL workloads into the cloud to extend its cost and performance benefits. AWS offers many compute and storage options that can help optimize PostgreSQL deployments.

Terminology

The following definitions are for the common terms that will be referenced throughout this paper:

- **IOPS** — Input/output (I/O) operations per second (IOPS)

- **Throughput** — Read/write transfer rate to storage (MB/s).
- **Latency** — Delay between sending an I/O request and receiving an acknowledgment (ms).
- **Block size** — Size of each I/O (KB).
- **Page size** — Internal basic structure to organize the data in the database files (KB).
- **Amazon Elastic Block Store (Amazon EBS) volume** — Persistent block-level storage devices for use with Amazon Elastic Compute Cloud (Amazon EC2) instances. This whitepaper focuses on solid state drive (SSD) EBS volume types optimized for transactional workloads involving frequent read/write operations with small I/O size, where the dominant performance attribute is IOPS.
  - **Amazon EBS General Purpose SSD volume** — General Purpose SSD volumes provide a balance of price and performance. AWS recommends these volumes for most workloads. Currently, AWS offer two types of General Purpose SSD volumes: `gp2` and `gp3`.
  - **Amazon EBS Provisioned IOPS SSD volume** — Highest performance SSD volume designed for high performance for mission-critical, low-latency, or high-throughput workloads. Currently AWS offers three types of Provisioned IOPS SSD volumes: `io1`, `io2` and `io2` Block Express (bx).
  - **Amazon EBS Throughput Optimized hard disk drive (HDD) volume** — Low-cost HDD volume designed for frequently accessed, throughput-intensive workloads: `st1`, `sc1`.

PostgreSQL on AWS deployment options

AWS provides various options to deploy fully managed PostgreSQL database service. Amazon Aurora for PostgreSQL database engine is designed to be wire-compatible with PostgreSQL versions. Amazon Aurora is a fully managed MySQL and PostgreSQL compatible service that has several times faster performance than the typical high-end implementations in those community editions. Moreover, it's durable, performant, and available as the commercial-grade databases, but at one tenth of the cost. Alternatively, customers can choose Amazon RDS for PostgreSQL as it gives them access to capabilities of the familiar PostgreSQL database engine. You can also host PostgreSQL on Amazon EC2 and self-manage the database, or browse the third-party PostgreSQL offerings on the AWS Marketplace. This whitepaper explores the implementation and deployment considerations for PostgreSQL on Amazon EC2 using Amazon EBS for storage.

Although Amazon RDS and Amazon Aurora with PostgreSQL compatibility is a good choice for most of the use cases on AWS, deployment on Amazon EC2 might be more appropriate for certain PostgreSQL workloads. With Amazon RDS, you can connect to the database itself using SQL interface and it gives you access to the familiar capabilities and configurations in PostgreSQL.

However, with managed database models, access to the operating system (OS) and certain system catalogs aren't available. This is an issue when you need OS-level access due to specialized configurations that rely on low-level OS settings, such as when using PostgreSQL extensions. For example, enabling PostgreSQL extension pg_top requires OS-level access to gather monitoring information. As another example, Slony-I an asynchronous replication system for PostgreSQL that provides support for cascading and failover requires access to `rdsadmin` owned system catalogs. In such cases, running PostgreSQL on Amazon EC2 is a better alternative.

PostgreSQL can be scaled vertically by adding additional hardware resources (CPU, memory, disk, network) to the same server. For both Amazon RDS and Amazon EC2, you can change the EC2 instance type to match the resources required by PostgreSQL database. Amazon Aurora provides a Serverless PostgreSQL Compatible Edition that allows compute capacity to be auto scaled on demand based on application needs. Both Amazon RDS and Amazon EC2 have an option to use EBS General Purpose SSD and EBS Provisioned IOPS volumes. The maximum provisioned storage limit for Amazon RDS database (DB) instances running PostgreSQL is 64 TB. The EBS volume for PostgreSQL on Amazon EC2, conversely, supports up to 16 TB per volume for General Purpose (`gp2`, `gp3`) and Provisioned IOPS volumes (`io1`, `io2`). EBS also offers io2 Block Express volumes which are suited for workloads that benefit from a single volume that provides sub-millisecond latency, and supports higher IOPS, higher throughput, and larger capacity than `io2` volumes. `io2bx` volumes can support up to 64 TiB.

Horizontal scaling is also an option in PostgreSQL, where you can add PostgreSQL read replicas to accommodate additional read traffic to separate database instance. With Amazon RDS, you can easily enable this option through the AWS Management Console with click of a button, Command Line Interface (CLI), or REST API. Amazon RDS for PostgreSQL allows up to fifteen read replicas. It also supports cascading replication, a series of up to three read replicas in a chain from a source database instance. There are certain cases where you might need to enable specific PostgreSQL replication features. Some of these features such as delayed replication and streaming replication outside of RDS may require OS access to PostgreSQL or advanced privileges to access certain system procedures and tables. Delayed replication is the concept of applying time-delayed changes from the WAL. That is, a transaction that is committed at physical time `X` is only going to be visible on a standby with delay `d` at time `X + d`. This is useful for disaster recovery.

PostgreSQL on Amazon EC2 is an alternative to Amazon RDS and Aurora for certain use cases. It allows you to migrate new or existing databases that have very specific requirements for workloads and business needs. Choosing the right compute, network, and storage configurations plays a crucial role in achieving good performance at an optimal cost for PostgreSQL workloads.

Amazon EBS block-level storage options

Amazon EBS provides flexible, cost effective, and easy-to-use data storage options for Amazon EC2 instances. Each option has a unique combination of performance and durability. These storage options can be used independently or in combination to suit workloads' requirements.

There are two primary block-level storage options for Amazon EC2 instances.

- Amazon EC2 instance store
- Amazon Elastic Block Store

Amazon EC2 instance store

The instance store volume consists of one or more storage volumes exposed as block I/O devices. It provides temporary block-level storage for instances. An instance store volume is a disk that is physically attached to the Amazon EC2 instance. You must specify instance store volumes when you launch the Amazon EC2 instance. Data on instance store volumes does not persist if the instance stops, hibernates, or terminates; or if the underlying disk drive fails.

Amazon Elastic Block Store

Amazon EBS provides durable, block-level storage volumes that can be attached to a running instance. An Amazon EBS volume behaves like a raw, unformatted, external block device that you can attach to an instance. The volume persists independently from the running life of an instance. The data on the Amazon EBS volume persists even if the associated Amazon EC2 instance shuts down or goes through a hardware failure. The data persists on the volume until the volume is explicitly deleted. Refer to Solid state drives (SSD) in the AWS documentation for the details about SSD-backed Amazon EBS volumes.

Due to the immediate proximity of the instance to the instance store volume, the I/O latency to an instance store volume tends to be lower than to an Amazon EBS volume. Use cases for instance store volumes include acting as a layer of cache or buffer, storing temporary database tables or logs, or providing storage for read replicas. For a list of the instance types that support instance store volumes, refer to Amazon EC2 instance store within the Amazon EC2 User Guide for Linux instances. Unlike Amazon EBS volumes, you can't detach an instance store volume from one instance and attach it to a different instance. An instance store volume exists only during the

lifetime of the instance to which it is attached. You can't configure an instance store volume to persist beyond the lifetime of its associated instance.

The remainder of this paper focuses on Amazon EBS SSD volumes:

- General Purpose (gp2, gp3)
- Provisioned IOPS (io1, io2, io2bx)

Amazon EBS volume features

Amazon EBS monitoring

Amazon EC2 allows different types of metrics and logs to be collected, viewed, and analyzed. The metrics deal with the Amazon EC2 instance, storage, network and application level. Amazon EBS automatically sends data points to Amazon CloudWatch for one-minute intervals at no charge. Amazon CloudWatch metrics are statistical data to view, analyze, and set alarms on the operational behavior of storage volumes. The Amazon EBS metrics can be viewed by selecting the monitoring tab of the volume in the Amazon EC2 console. For more information about the Amazon EBS metrics collected by CloudWatch, refer to the Amazon CloudWatch metrics for Amazon EBS.

Amazon EBS durability and availability

Amazon EBS general purpose volumes are designed for reliability with a 0.1 percent to 0.2 percent annual failure rate (AFR) compared to the typical 2% of commodity disk drives. These storage volumes are backed by multiple physical drives for redundancy that is replicated within an Availability Zone to protect database workload from component failure. Amazon EBS also offers a higher durability volume (io2), that is designed to provide 99.999% durability with an annual failure rate (AFR) of 0.001%, where failure refers to a complete or partial loss of the volume. For example, if you have 100,000 Amazon EBS io2 volumes running for one year, you should expect only one io2 volume to experience a failure. This makes io2 ideal for business-critical PostgreSQL applications. For more details, see the Amazon EBS Service Level Agreement.

Amazon EBS snapshots

Amazon EBS snapshots back up the data on Amazon EBS volumes by taking point-in-time snapshots to Amazon Simple Storage Service (Amazon S3) which is designed for 99.999999999% (11 nines) of durability. Apart from providing backup, other reasons for creating Amazon EBS snapshots include:

- **Set up a non-production or test environment** — You can share the Amazon EBS snapshot to duplicate the installation of PostgreSQL in different environments. You can also share Amazon EBS snapshots among different AWS accounts within the same AWS Region. For example, you can restore a snapshot of your PostgreSQL database that's in a production environment to a test environment to duplicate and troubleshoot production issues.

- **Disaster recovery** — Amazon EBS's ability to copy snapshots across AWS Regions makes it easier to leverage multiple AWS Regions for geographical expansion, data center migration and disaster recovery. Amazon EBS Snapshots can be copied from one AWS Region to another for site disaster recovery.
- **Meet compliance and regulatory obligations -** Certain industries require periodic archival of key data including PostgreSQL databases. Amazon EBS Snapshots enable you to leverage Amazon EBS Snapshots archive which is a lower storage cost tier that stores a full copy of your point-in-time Amazon EBS Snapshots, and can be restored as needed.

In addition, you can also leverage Data Lifecycle Manager (DLM) which provides a mechanism to automate creation, retention, archival and deletion of Amazon EBS Snapshots. This facilitates simple and automated way to manage backup of PostgreSQL data stored on Amazon EBS volumes. You can define backup and retention schedules for Amazon EBS snapshots by creating lifecycle policies based on tags. With this feature, there are no dependencies to rely on custom scripts to create and manage your backups.

Also, note that a volume that is restored from a snapshot is lazily loaded in the background, which means that you can start using PostgreSQL database right away. When you perform a query on a PostgreSQL data that has not been downloaded yet, the data will be downloaded from Amazon S3 directly. You also have the option of enabling Amazon EBS fast snapshot restore to create a volume from a snapshot that is fully initialized at creation. For an additional hourly charge, you can enable Fast Snapshot Restore (FSR) capability for low latency access to data restored from snapshots. You can enable FSR on snapshots you own or those shared with you. Amazon EBS volumes restored from FSR-enabled snapshots instantly receive their full performance. Refer to Amazon EBS fast snapshot restore for more information.

Amazon EBS security

Amazon EBS encryption offers seamless encryption of Amazon EBS data volumes, boot volumes and snapshots, eliminating the need to build and manage a secure key management infrastructure. Amazon EBS supports several security features to use from volume creation to utilization. These features prevent unauthorized access to PostgreSQL databases. You can use tags and resource-level permissions to enforce security on volumes upon creation. These tags are typically used to track resources, control cost, implement compliance protocols, and control access to resources through AWS Identity and Access Management (IAM) policies. Tags can be assigned on Amazon EBS volumes during creation time for efficient volume management. After the volume is created, you can use the IAM resource-level permissions for Amazon EC2 API actions where only

authorized IAM users; or groups who can attach, delete, or detach Amazon EBS volumes to Amazon EC2 instances.

Protection of data in transit and at rest is crucial in most PostgreSQL implementations. You can use Secure Sockets Layer (SSL) to encrypt the connection from application to PostgreSQL database. To encrypt data at rest, Amazon EBS volumes should have encryption enabled at the time of creation. The new volume gets an unique 256-bit AES key, which is protected by the fully managed AWS Key Management Service. Amazon EBS snapshots created from the encrypted volumes are automatically encrypted. Encryption operations occur on the servers that host Amazon EC2 instances, ensuring the security of both data-at-rest and data-in-transit between an instance and its attached Amazon EBS storage. The Amazon EBS encryption feature is available on all current generation instance types. For more information on the supported instance types, refer to the Amazon EBS Encryption documentation.

Elastic volumes

Elastic volumes is a feature that allows to easily adapt Amazon EBS volumes as per the needs of application's requirements. The elastic feature of Amazon EBS SSD volumes allows dynamically change the size, performance, and type of Amazon EBS volume in a single API call or within the AWS Management Console without any interruption of PostgreSQL operations. This simplifies some of the administration and maintenance activities of PostgreSQL workloads running on current generation Amazon EC2 instances.

You can call the ModifyVolume API to dynamically increase the size of the Amazon EBS volume if the PostgreSQL database is running low on usable storage capacity. Note that decreasing the size of the Amazon EBS volume isn't supported, so AWS recommends not to over-allocate the Amazon EBS volume size any more than necessary to avoid paying for extra resources that you do not use.

In situations where there is a planned increase in your PostgreSQL utilization, you can either change your volume type or add additional IOPS. The time it takes to complete these changes depends on the size of Amazon EBS volume. The progress of the volume modification can be monitored by either through the AWS Management Console or CLI. You can also create CloudWatch Events to send alerts after the changes are complete.

Amazon EBS–optimized instances

Amazon EBS-optimized instances deliver dedicated throughput between Amazon EC2 and Amazon EBS. The dedicated throughput minimizes contention between Amazon EBS I/O and other traffic

from Amazon EC2 instance, providing the best performance for PostgreSQL workloads. It is recommended to choose an Amazon EBS–optimized instance that provides more dedicated Amazon EBS throughput than application needs; otherwise, the connection between Amazon EBS and Amazon EC2 can become a performance bottleneck. For more information about the instance types that can be launched as Amazon EBS-Optimized instances, see Amazon EC2 Instance Types.

Amazon EBS volume types

General purpose SSD volumes

General purpose SSD (gp2 and gp3) volumes are backed by solid-state drives (SSDs). These storge options balance price and performance for a wide variety of database workloads. Both gp2 and gp3 volumes provide single-digit millisecond latency and 99.8 percent to 99.9 percent volume durability with an annual failure rate (AFR) no higher than 0.2 percent, which translates to a maximum of two volume failures per 1,000 running volumes over a one-year period.

gp2

The general purpose SSD (gp2) volume offers balanced price and performance. To maximize the performance of the gp2 volume, you need to know how the burst capability works. The size of the gp2 volume determines the baseline performance level of the volume and how quickly it can accumulate I/O credits. Depending on the volume size, baseline performance ranges between a minimum of 100 IOPS up to a maximum of 16,000 IOPS per volume. Volumes earn I/O credits at the baseline performance rate of 3 IOPS/GiB of volume size. The larger the volume size, the higher the baseline performance and the faster I/O credits accumulate. Refer to General purpose SSD volumes (gp2) for more information related to I/O characteristics and burstable performance of gp2 volumes.

gp3

These volumes deliver a consistent baseline rate of 3,000 IOPS and 125 MiB/s, included with the price of storage. You can provision additional IOPS (up to 16,000) and throughput (up to 1,000 MiB/s) for an additional cost. The maximum ratio of Provisioned IOPS to provisioned volume size is 500 IOPS per GiB. The maximum ratio of provisioned throughput to Provisioned IOPS is .25 MiB/s per IOPS. The following volume configurations support provisioning either maximum IOPS or maximum throughput:

- **32 GiB or larger:** 500 IOPS/GiB x 32 GiB = 16,000 IOPS
- **8 GiB or larger and 4,000 IOPS or higher:** 4,000 IOPS x 0.25 MiB/s/IOPS = 1,000 MiB/s

In addition to change the volume type, size and provisioned throughput (for gp3 only); you can also use RAID 0 to stripe multiple gp2 or gp3 volumes together to achieve greater I/O

performance. The I/O is striped across volumes in the RAID 0 configuration. The throughput of your PostgreSQL database also rises with the addition of an extra volume. The read/write transfer rate, or throughput, is calculated by multiplying the I/O block size by the IOPS rate that is carried out on the disk. Since the performance of the stripe is only as good as the lowest performing volume in the set, AWS advises adding the same volume size to the stripe set. Think about RAID 0's failure tolerance as well as a single volume loss results in a total loss of data for the array. If possible, use RAID 0 in a PostgreSQL primary/secondary environment where data is already replicated in multiple secondary nodes.

Provisioned IOPS SSD volumes

Provisioned IOPS SSD (`io1`, `io2`, `io2` Block Express) volumes are designed to meet the needs of I/O-intensive workloads, particularly database workloads that are sensitive to storage performance and consistency. Provisioned IOPS SSD volumes use a consistent IOPS rate, which you specify when you create the volume. Amazon EBS delivers the provisioned performance 99.9 percent of the time.

- `io1` volumes are designed to provide 99.8 to 99.9 percent volume durability with an annual failure rate (AFR) no higher than 0.2 percent, which translates to a maximum of two volume failures per 1,000 running volumes over a one-year period.
- `io2` and `io2` Block Express volumes are designed to provide 99.999 percent volume durability with an AFR no higher than 0.001 percent, which translates to a single volume failure per 100,000 running volumes over a one-year period.

Provisioned IOPS SSD `io1` and `io2` volumes are available for all Amazon EC2 instance types. Provisioned IOPS SSD `io2` volumes attached to `c6in`, `c7g`, `m6in`, `m6idn`, `r5b`, `r6in`, `r6idn`, `trn1`, `x2idn`, and `x2iedn` instances run on Amazon EBS Block Express.

Provisioned IOPS SSD volumes can range in size from 4 GiB to 16 TiB, and you can provision from 100 IOPS up to 64,000 IOPS per volume. You can achieve up to 64,000 IOPS only on Instances built on the Nitro System. This provides dedicated throughput between Amazon EBS volume and Amazon EC2 instance. On other instance families you can achieve performance up to 32,000 IOPS. The maximum ratio of provisioned IOPS to requested volume size (in GiB) is 50:1 for `io1` volumes, and 500:1 for `io2` volumes.

`io2` Block Express volumes is the next generation of Amazon EBS storage server architecture. It has been built for the purpose of meeting the performance requirements of the most demanding I/O intensive applications that run on Nitro-based Amazon EC2 instances. You can provision IOPS up to

256,000, with an IOPS:GiB ratio of 1,000:1. Maximum IOPS can be provisioned with volumes 256 GiB in size and larger (1,000 IOPS × 256 GiB = 256,000 IOPS) while volume throughput can go up to 4,000 MiB/s.

PostgreSQL considerations

PostgreSQL offers many settings that can be tuned to obtain optimal performance for every type of workload. This section focuses on the PostgreSQL database engine settings. It also looks at the PostgreSQL parameters that can be optimized to improve performance related to Amazon EBS volumes I/Os.

Caching

PostgreSQL tracks the access patterns of data and keeps frequently accessed data in cache. While PostgreSQL does have a few parameter settings that directly affect memory allocation for the purposes of caching, most of the cache that PostgreSQL uses is provided by the underlying operating system. PostgreSQL also utilizes caching of its data in a space called `shared_buffers`. Knowing when PostgreSQL will perform a disk I/O instead of accessing the cache helps performance tuning. While reading data, `shared_buffers` caches recent accessed data. While writing, `shared_buffers` hosts dirty pages. This in-memory area resides between read/write operations and the Amazon EBS volumes. Disk I/O occurs if the data is not in the cache for read requests, or when the data from dirty pages are flushed to disk.

The `shared_buffers` uses the Least Recently Used (LRU) algorithm for cached pages. When the size of `shared_buffers` is too small, it could result frequent Read IOs to get data from storage and the buffer pages may have to be constantly flushed to and from the disk. This affects performance and lowers the query concurrency. The default size of the shared buffers is 128 MB. Increasing the size of the `shared_buffers` works well when the dataset and queries take advantage of it. For example, if you have 1 GiB of data and the `shared_buffers` is configured at 5 GiB, then increasing the `shared_buffers` size to 10 GiB doesn't make database faster. A good rule of thumb is that the `shared_buffers` should be large enough to hold working data-set, which is composed of the rows and indexes that are used by the queries. For most PostgreSQL workloads, 20%-25% of total RAM as `shared_buffers` is a good practice. While modifying `shared_buffers` parameter, you should consider that enough memory is left for other database operations such as sorting, hashing, auto-vacuum, temp_buffers, and wal_buffers.

Database writes

PostgreSQL does not write directly to disk. Instead, PostgreSQL writes all modifications into a persistent storage to prepare for failures. In PostgreSQL, Write Ahead Logging (WAL) is the

standard method for ensuring data integrity. It makes sure that changes to data files must be written only after those changes have been logged, that is, after log records describing the changes have been flushed to permanent storage. WAL data records are written into the in-memory WAL buffer by change operations such as insertion, deletion, or commit actions. Then, they are written into WAL segment files on the storage when a transaction commits or aborts. Now, background writer process keeps flushing modified data pages to Amazon EBS volumes permanent storage. Checkpoints make sure that all modified data pages have been flushed to storage after point of time.

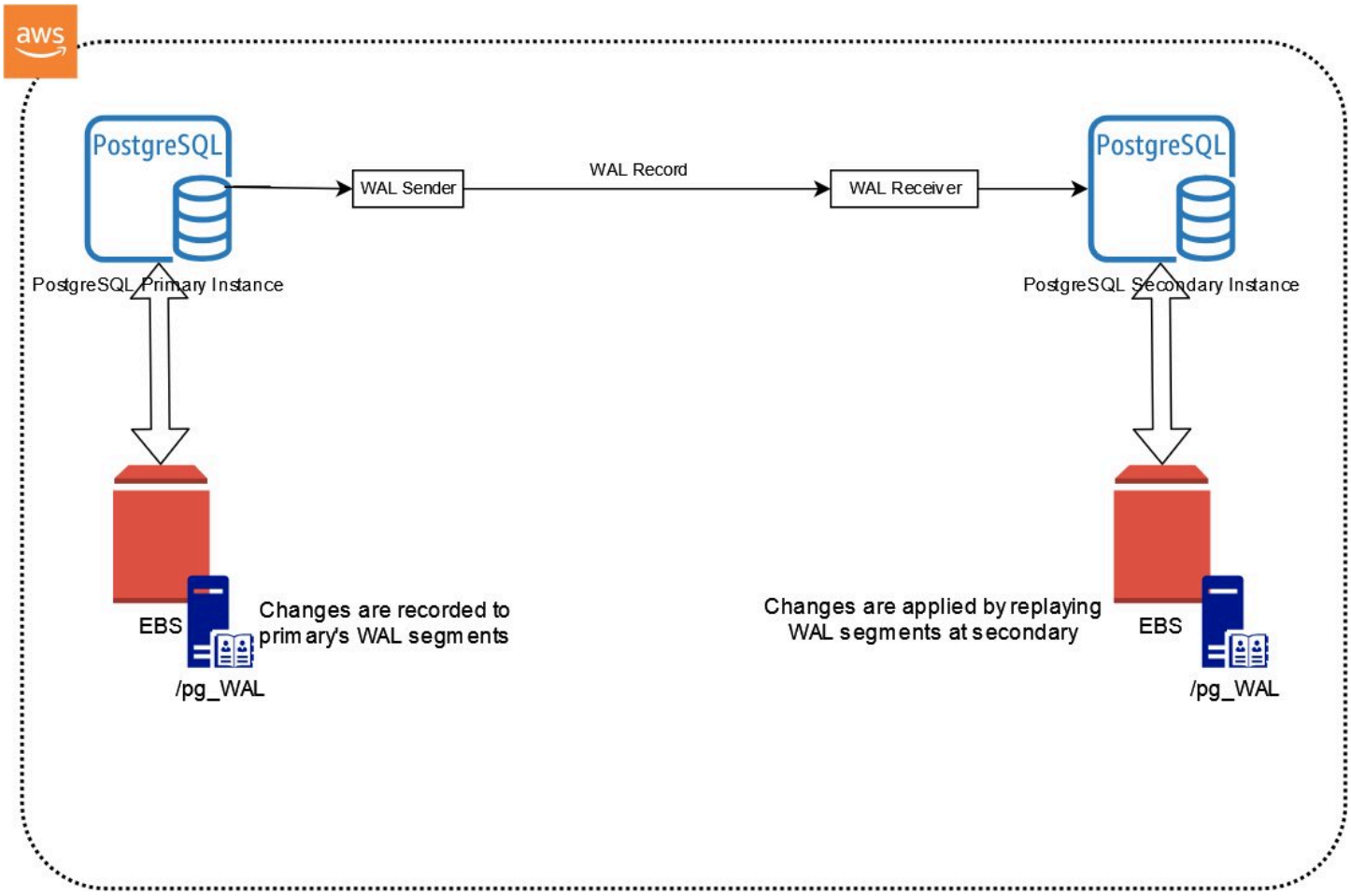
Frequent checkpoints can cause high IOs, high commit latency and low throughput. In practice, checkpoints should happen infrequently not to affect the users, but frequently enough to reasonably limit time for recovery and disk space requirements. A checkpoint begins after every checkpoint_timeout seconds, or if max_wal_size is about to be exceeded, whichever comes first. The default settings are 5 minutes and 1 GB respectively. Reducing these parameters allows faster after-crash recovery, since less work will need to be redone. However, this could increase the cost of flushing modified data pages more often. Checkpoint_completion_target parameter specifies the target of checkpoint completion, as a fraction of total time between checkpoints. Reducing this parameter is not recommended because it causes the checkpoint to complete faster. This results in a higher rate of I/O during the checkpoint followed by a period of less I/O between the checkpoint completion and the next scheduled checkpoint.

PostgreSQL read replica configuration

PostgreSQL allows to replicate data so you can scale out read-heavy workloads with source/replica configuration. You can create multiple copies of PostgreSQL databases into one or more replica instances to increase the read throughput for application. The availability of PostgreSQL database can be increased with the replicated instances. When a source instance fails, one of the replica instances servers can be promoted, reducing the recovery time.

In PostgreSQL, there are two types of replication methods: Physical replication that collectively replicates a database cluster, and logical replication that replicates given database objects such as tables, schemas and databases. PostgreSQL built-in streaming replication (physical replication) continuously sends data changes from primary instance to secondary instance. In cascaded replication, secondary instances can also be senders as well as receivers. `max_wal_senders`, `max_replication_slots`, `wal_keep_size`, `max_slot_wal_keep_size`, `wal_sender_timeout` are some of the parameters that can be set at primary server to send replication data to one or more secondary instances. The

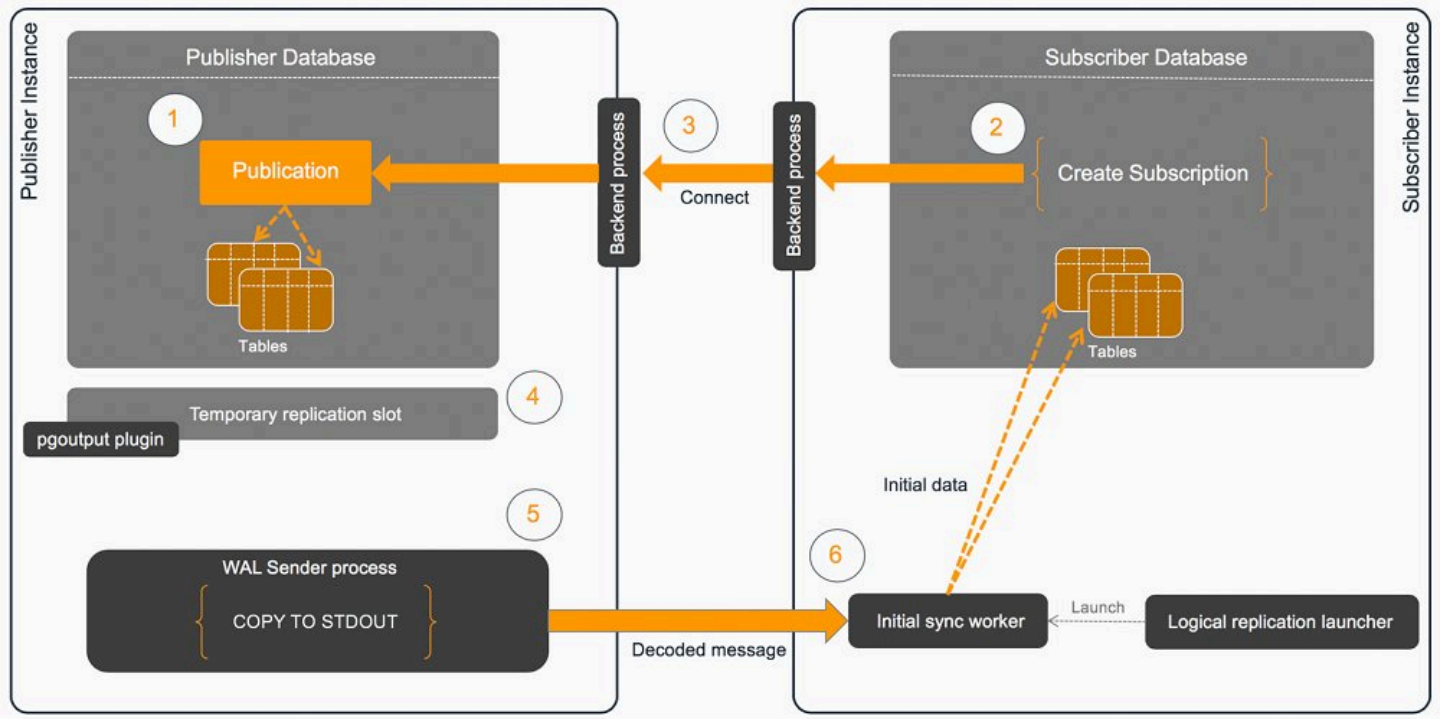
`max_wal_senders` parameter specifies the maximum number of concurrent connections from secondary instances. The default is 10. The value 0 means replication is disabled. The `wal_keep_size` parameter specifies the minimum size of WAL segments kept in the `pg_wal` directory, in case a standby server needs to fetch them for streaming replication. If a secondary instance connected to the sending server falls behind by more than `wal_keep_size` megabytes, the sending server might remove a WAL segment still needed by the standby, in which case the replication connection will be terminated. However, the standby server can recover by fetching the segment from archive, if WAL archiving is in use. `wal_keep_size` should be set high enough that spiky write workloads don't terminate replication. The following diagram illustrates how PostgreSQL performs streaming replication:



PostgreSQL streaming replication

In PostgreSQL, logical replication is a method of replicating data objects and their changes, based upon their replication identity, usually a primary key. Unlike physical replication which uses exact block addresses and byte-by-byte replication, logical replication uses a *publish* and *subscribe* model with one or more *subscribers* subscribing to one or more *publications* on a *publisher* node.

Subscribers pull data from the publications they subscribe to and may subsequently re-publish data to allow cascading replication or more complex configurations. Logical replication of a table typically starts with taking a snapshot of the data on the publisher database and copying that to the subscriber. Once that is done, the changes on the publisher are sent to the subscriber as they occur in real-time. The subscriber applies the data in the same order as the publisher so that transactional consistency is guaranteed for publications within a single subscription. This method of data replication is sometimes referred to as transactional replication. The following diagram represents the data flow for initial data copy and synchronization.



Data flow for initial data copy and sychronization

PostgreSQL logical replication

PostgreSQL replication considerations

PostgreSQL has single process for replaying Write Ahead Logs (WAL) file. Running out of IOs can cause replication lag. To obtain larger I/O throughput, storage volume requires a larger queue depth. An Amazon EBS io1 or io2 can provide up to 64,000 IOPS/volume, which, in turn, means it has a larger queue depth. An Amazon EBS io2 Block Express SSD volumes can provide up to 256,000 IOPS/volume. AWS recommends using this volume type on workloads that require heavy replication.

As mentioned in the Provisioned IOPS SSD volumes section of this document, RAID 0 increases the performance and throughput of Amazon EBS volumes for your PostgreSQL database. You can join several volumes together in a RAID 0 configuration to use the available bandwidth of the Amazon EBS-optimized instances to deliver the additional network throughput dedicated to Amazon EBS.

There are the sequential writes for the WAL shipment from the primary server and sequential reads of the WAL. Additionally, there is the traffic of regular random updates to your data files. Using RAID 0 in this case improves the parallel workloads since it spreads the data across the disks and their queues. However, you must be aware of the penalty from the sequential and single-threaded workloads because extra synchronization is needed to wait for the acknowledgments from all members in the stripe. Only use RAID 0 if you need more throughput than that which the single Amazon EBS volume can provide. As RAID0 has no data redundancy, for high read and write throughput, RAID10 (mirrored striped sets) should be considered.

Migrating PostgreSQL from on-premises to Amazon EC2

Migrating databases requires strategy, resources, and downtime maintenance. If you have already running PostgreSQL databases out of AWS, you have several options to migrate to Amazon EC2 hosted PostgreSQL. If downtime is affordaable, you can use `pg_dump/ pg_restore` to migrate PostgreSQL databases from on-premises to Amazon EC2.

Single database backup:

```
pg_dump –host <hostname> –format=directory –create –jobs 5 –dbname <database name> –
username <username> –file /home/db11.dump
pg_restore –host <hostname> –format=directory –create --jobs 5 –dbname <database name>
 –username <username> –file /home/ec2-user/db11.dump
```

`pg_dumpall` can be used for migrating all databases along with globals.

Backup all databases:

```
pg_dumpall > alldb.dump
```

Restore all databases:

```
psql -f alldb.dump postgres
```

If downtime is not permissible, you can setup physical replication between on-prem PostgreSQL and Amazon EC2 hosted PostgreSQL. Once data is in sync with no replica lag, you can promote

Amazon EC2 PostgreSQL and set it up as new primary. For setting up physical replication, customers need to do these changes:

On the on-prem PostgreSQL side:

```
echo "listen_addresses = * >> $PGDATA/postgresql.conf
echo "wal_level = replica" >> $PGDATA/postgresql.conf
systemctl restart postgresql
postgres=# CREATE USER migration_replication WITH REPLICATION ENCRYPTED PASSWORD
 'secret';
echo "host replication migration_replication 192.1111.11.11/32 md5" >> $PGDATA/
pg_hba.conf
psql -c "select pg_reload_conf()"
```

On the EC2 PostgreSQL side:

```
pg_ctl -D $PGDATA start
pg_basebackup -h <primary IP> -U migration_replication -p 5432 -D $PGDATA  -Fp -Xs -R
```

PostgreSQL backups

Backup methodologies

There are several approaches to protect PostgreSQL data depending on recovery time objective (RTO) and recovery point objective (RPO) requirements. PostgreSQL production databases should be backed up regularly. There are three fundamentally different approaches to back up PostgreSQL data:

- **SQL dump**

  This is the method to generate a file with SQL commands that, when fed back to the server, will recreate the database in the same state as it was at the time of the backup. PostgreSQL provides the utility program pg_dump for this purpose. The basic usage of this command is:

  ```
  pg_dump dbname > dumpfile
  ```

  pg_dump is a PostgreSQL client application that perform backup procedure from any remote host that has access to the database. The user running pg_dump command should have read access to the objects being backed up. pg_dump can be used to back up given tables, schemas or databases. The pg_dump file can be restored by using psql client or `pg_restore` utility. `pg_dumpall` backs up all databases in the given cluster, and also preserves cluster-wide data such as roles and tablespace definitions. The basic usage of this command is:

  ```
  pg_dumpall > dumpfile
  ```

  The resulting dump can be restored with `psql`:

  ```
  psql -f dumpfile postgres
  ```

- **File system level backup:**

  An alternative backup strategy is to directly copy the files that PostgreSQL uses to store the data in the database. You can use your preferred method for file system backups. For example:

```
tar -cf backup.tar /usr/local/pgsql/data
```

There are two restrictions, however, which make this method impractical, or at least inferior to the pg_dump method.

- The database server must be shut down in order to get a usable backup. Half-way measures such as disallowing all connections will not work.
- File system backups only work for complete backup and restoration of an entire database cluster.

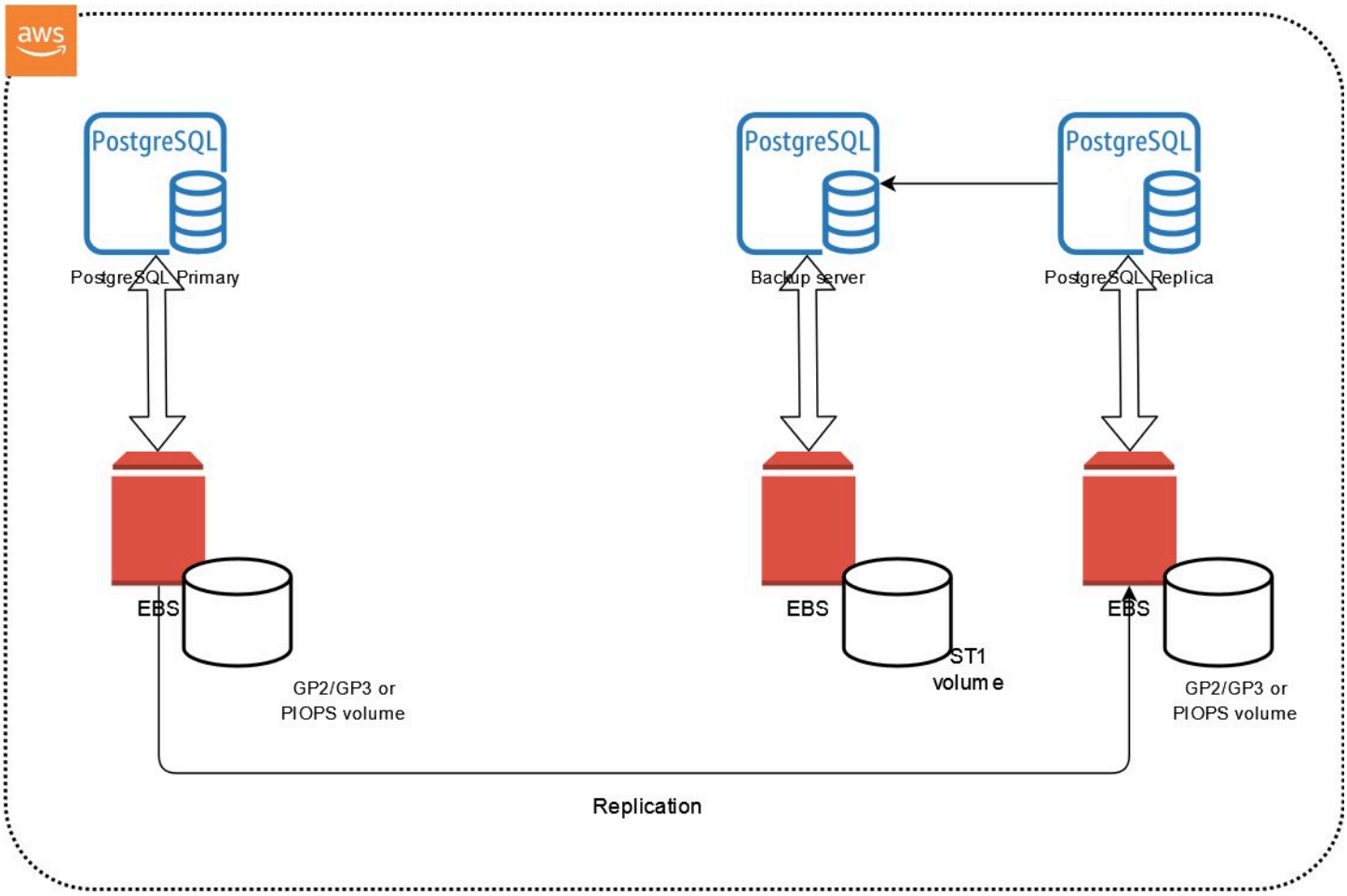
Unlike pg_dump, it can't be used to back up given tables, schemas or databases. Alternatively, you can use pg_basebackup, a PostgreSQL native utility to take a base backup of a PostgreSQL cluster. The following is the basic command to use pg_basebackup:

```
pg_basebackup -h <host> -D <PostgreSQL data directory>
```

- **Continuous archiving**

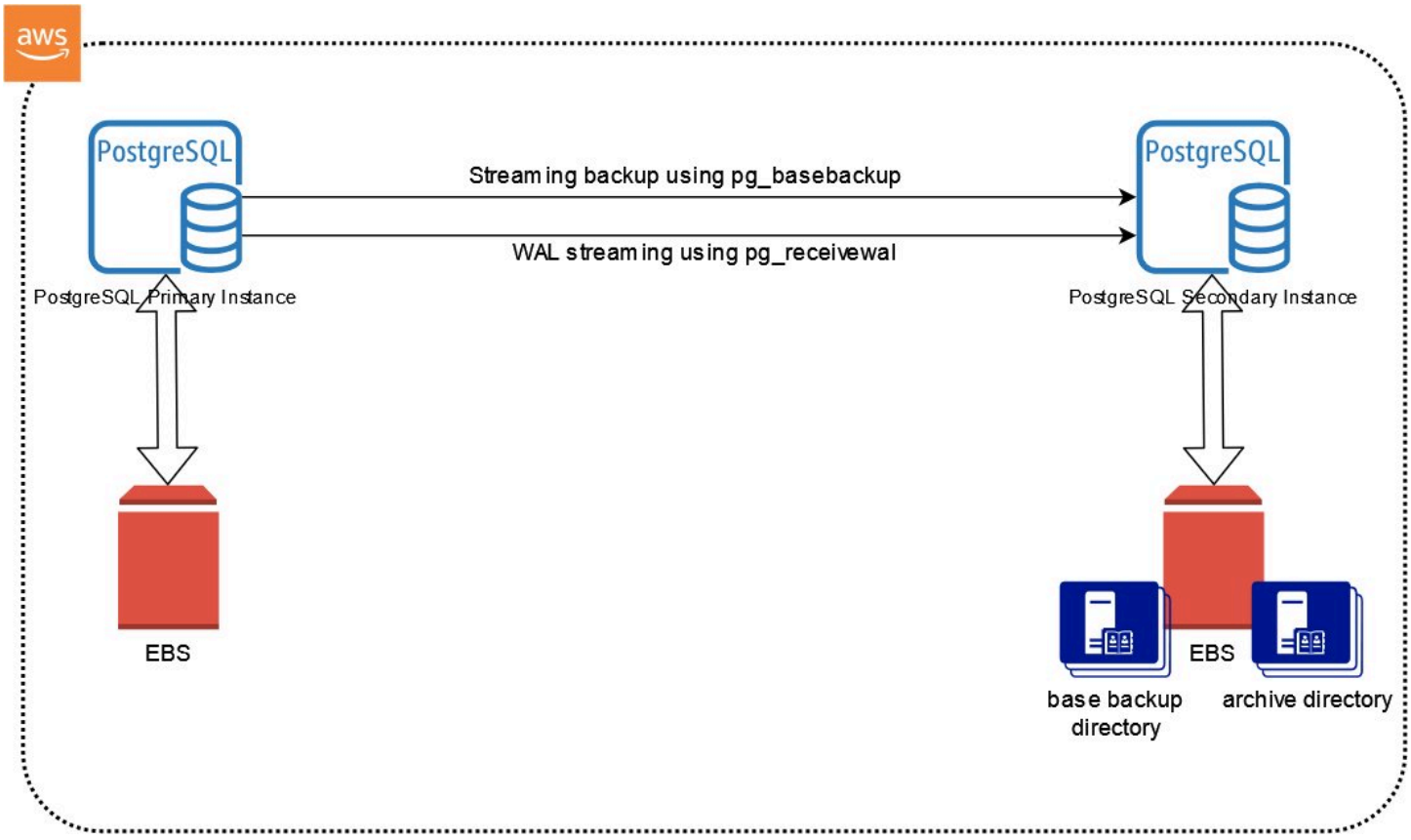
  The WAL log records every change made to the database's data files. This log exists on primarily for crash-safety purposes. If the system crashes, the database can be restored to consistency by "replaying" the log entries made since the last checkpoint. However, the existence of the log makes it possible to use a third strategy for backing up databases. We can combine a file-system-level backup with backup of the WAL files. If recovery is needed, we restore the file system backup and then replay from the backed-up WAL files to bring the system to a current state. This PostgreSQL document discusses about setting up continuous archiving.

If the primary database server exhibits performance issues during a backup, a replicated secondary database server can be used for the backups to alleviate the backup load from the primary database server. One approach can be to back up from a secondary server's SSD data volume to a backup server's Throughput Optimized HDD (st1) volume. The high throughput of 500 MiB/s per volume and large 1 MiB I/O block size make it an ideal volume type for sequential backups meaning it can use the larger I/O blocks. The following diagram shows a backup server using the PostgreSQL secondary server to read the backup data.



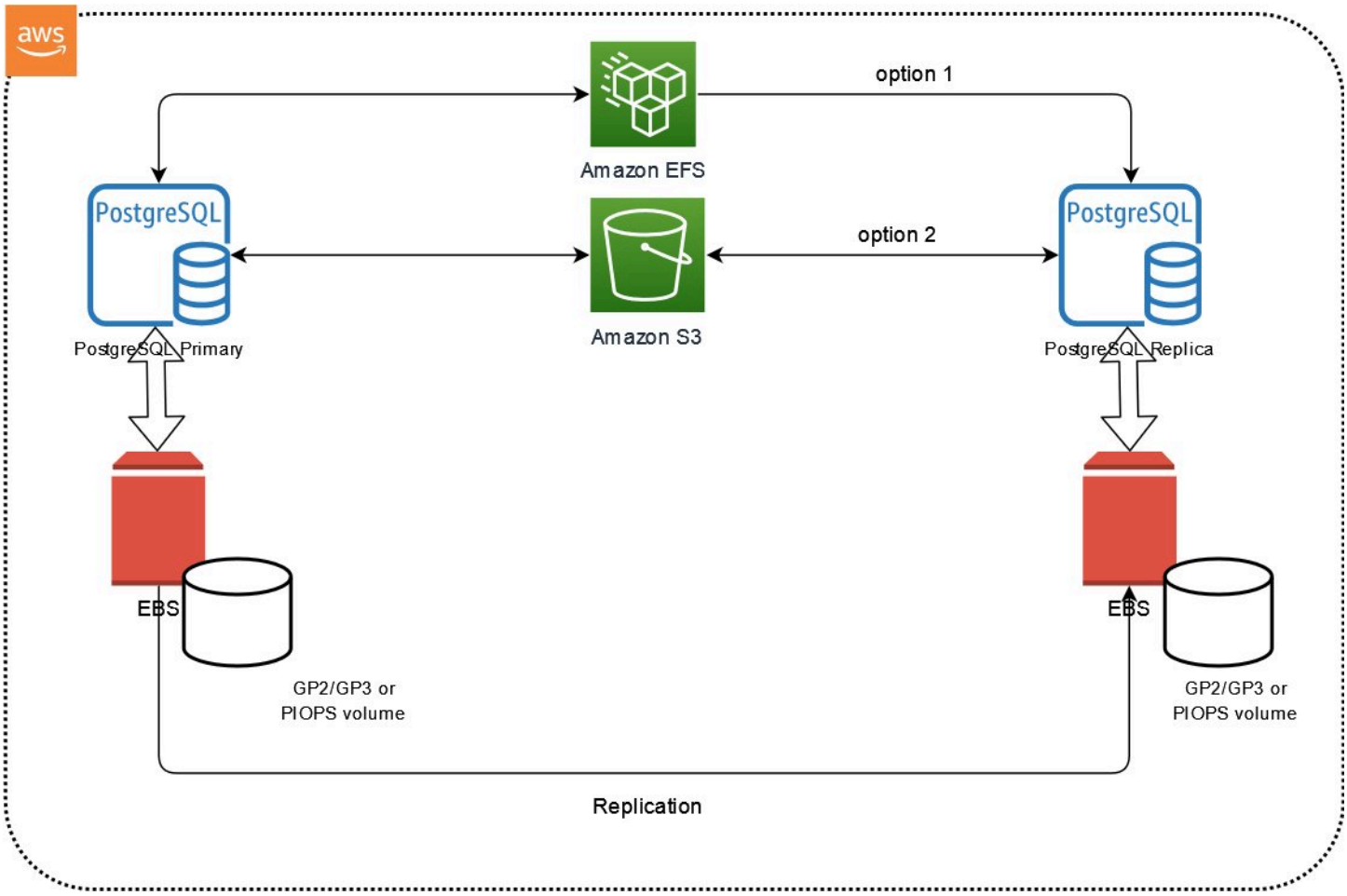
PostgreSQL backup using secondary server

Regarding file system level backup, pg_basebackup is a widely used PostgreSQL backup tool that allows us to take an online and consistent file system level backup. These backups can be used for point-in-time-recovery or to set up a secondary PostgreSQL server. The following image shows PostgreSQL files system backup using pg_basebackup and streaming the write-ahead logs from a running PostgreSQL cluster:



PostgreSQL files system backup using pg_basebackup and streaming the WAL

Another option is to have the PostgreSQL secondary server back up the database files directly to Amazon Elastic File System (Amazon EFS) or Amazon S3. Amazon EFS stores its data redundantly across multiple Availability Zones. Both the primary and the secondary instances can attach to Amazon EFS. The secondary instance can initiate a backup to Amazon EFS from where the primary instance can do a restore. Amazon S3 can also be used as a backup target. Amazon S3 can be used in a manner similar to Amazon EFS except that Amazon S3 is object-based storage rather than a file system. The following diagram depicts the option of using Amazon EFS or Amazon S3 as a backup target.



Using Amazon EFS or Amazon S3 as a backup target for PostgreSQL database

Second approach of backing up PostgreSQL database is to use volume-level EBS snapshots. Snapshots are incremental backups, which means that only the blocks on the device that have changed after your most recent snapshot are saved. This minimizes the time required to create the snapshot and saves on storage costs. When you delete a snapshot, only the data unique to that snapshot is removed. Active snapshots contain all the information needed to restore your data to a new Amazon EBS volume.

One consideration when utilizing Amazon EBS snapshots for backups is to make sure the PostgreSQL data remains consistent. During an Amazon EBS snapshot, any data not flushed from cache to disk will not be captured. There is a PostgreSQL command CHECKPOINT that flushes all the modified data pages from buffers to the disk. The snapshot takes a point-in-time capture of all the content within that volume. The database lock needs to be active until the snapshot process starts, but it doesn't have to wait for the snapshot to complete before releasing the lock. You can use Amazon Data Lifecycle Manager to automate the creation, retention, and deletion of Amazon EBS snapshots and Amazon EBS-backed AMIs.

You can also combine these approaches for an effective backup strategy. You can use database-level backups for more granular objects, such as databases or tables. You can leverage Amazon EBS snapshots for large scale operations such as recreating the database server, restoring the entire volume or migrating a database server to another Availability Zone or another Region for disaster recovery (DR).

Multi-Volume Crash-Consistent Snapshots

Amazon Elastic Block Store (Amazon EBS) enables to back up volumes at any time using Amazon EBS snapshots. Snapshots retain the data from all completed I/O operations, allowing you to restore the volume to its exact state at the moment before backup (referred to as crash-consistency). There is a one-click solution to take backups across multiple Amazon EBS volumes while ensuring that the data on each volume is in sync. Refer to this user guide for details.

Additionally, you can use AWS Backup, which creates crash-consistent backups of Amazon EBS volumes that are attached to an Amazon EC2 instance. Crash consistency means that the snapshots for every Amazon EBS volume attached to the same Amazon EC2 instance are taken at the exact same moment. You no longer have to stop your instances or coordinate between multiple Amazon EBS volumes to ensure crash-consistency of your application state.

Refer to the *AWS Backup Developer Guide* for details.

You can evaluate PostgreSQL performance related to storage by looking at latency when you run into performance issues of transactional operations. Further, if the performance is degraded due to PostgreSQL loading or replicating data, then throughput is evaluated. These issues are diagnosed by looking at the Amazon EBS volume metrics collected by CloudWatch.

Throughput

Throughput is the measure of the amount of data transferred from/to a storage. It affects PostgreSQL database workload, replication, backup, and import/export activities. When considering which AWS storage option to use to achieve high throughput, you must also consider that PostgreSQL has random I/O caused by small transactions that are committed to the database. To accommodate these two different types of traffic patterns, our recommendation is to use io1, io2/io2bx volumes on an Amazon EBS-optimized instance.

Amazon EBS calculates throughput using the equation:

```
Throughput = Number of IOPS * size per I/O operation
```

Insufficient throughput to underlying Amazon EBS volumes can cause PostgreSQL secondary servers to lag, and can also cause PostgreSQL backups to take longer to complete. To diagnose throughput issues, CloudWatch provides the metrics Volume Read/Write Bytes (the amount of data being transferred) and Volume Read/ Write Ops (the number of I/O operations).

You can use the CloudWatch metric to monitor the Amazon EBS volume level throughput:

```
Read Bandwidth (KiB/s) = Sum(VolumeReadBytes) / Period / 1024
Write Bandwidth (KiB/s) = Sum(VolumeWriteBytes) / Period / 1024
```

Latency

Latency is the round-trip time elapsed between sending a PostgreSQL I/O request to an Amazon EBS volume and receiving an acknowledgement from the volume that the I/O read or write is complete. Latency is experienced by slow queries, which can be diagnosed in PostgreSQL by enabling the parameter log_min_duration_statement to log slow queries.

Latency can also occur at the disk I/O-level, which can be viewed in the "Average Read Latency (ms/op)" and "Average Write Latency (ms/op)" in the monitoring tab of the Amazon EC2 console.

Average Read Latency (ms/op) is defined as blow:

```
Avg(VolumeTotalReadTime) × 1000
```

For Nitro-based instances, the following formula derives Average Read Latency using CloudWatch Metric Math:

```
(Sum(VolumeTotalReadTime) / Sum(VolumeReadOps)) × 1000
```

The `VolumeTotalReadTime` and `VolumeReadOps` metrics are available in the Amazon EBS CloudWatch console.

Average Write Latency (ms/op) is defined as follows:

```
Avg(VolumeTotalWriteTime) × 1000
```

For Nitro-based instances, the following formula derives Average Write Latency using CloudWatch Metric Math:

```
(Sum(VolumeTotalWriteTime) / Sum(VolumeWriteOps)) * 1000
```

The `VolumeTotalWriteTime` and `VolumeWriteOps` metrics are available in the Amazon EBS CloudWatch console.

This section covers the factors contributing to high latency for gp2, gp3, `io1`, and `io2` Amazon EBS volumes. High latency can result from exhausting the baseline IOPS in Amazon EBS volumes. However, gp2 volumes comes with burstable performance. If the workload is driving I/O traffic beyond its baseline performance, then burst credit gets depleted. If burst credit reaches zero, then these volume types get throttled at their baseline IOPS or throughput. The CloudWatch metric `BurstBalance` indicates if you have depleted the available burstable credit for IOPS for gp2 volumes.

Burst duration can be calculated as follows:

```
                        (I/O credit balance)
Burst duration = -----------------------------
                  (Burst IOPS) - (Baseline IOPS)
```

For example, for 100 GB gp2 volume, the baseline IOPS is 100*3=300 IOPS. However, any gp2 volume upto 1000 GB comes with burstable capability upto 3,000 IOPS. Hence, this particular volume will be able to sustain a workload upto 3,000 IOPS for the burst duration of 2000 secs as calculated from the above formula. After the burst credit are depleted, the volume performance will be back to baseline of 300 IOPS.

Disk queue length can also contribute to high latency. Disk queue length refers to the outstanding read/write requests that are waiting for instance and volume resources to be available. The CloudWatch metric `VolumeQueueLength` shows the number of pending read/write operation requests for the volume.

```
Avg Queue Length (Operations) = Avg(VolumeQueueLength)
```

This metric is an important measurement to monitor if you have reached the full utilization of the allocated IOPS on your Amazon EBS volumes. A value of average queue length greater than one for every 1000 IOPS can cause latency.

It is recommended to monitor the CloudWatch metrics for IOPS and throughput so it that does not go beyond the provisioned limits. If it goes beyond that limit, it causes the I/O to be queued up a disk level which in turn increase the Round-trip time for the I/O. This increases the latency, and thus impacts the PostgreSQL database performance.

PostgreSQL benchmark observations and considerations

Testing PostgreSQL database helps determine what type of volume is needed and ensures the most cost-effective and performant solution. For new workload, you can do a synthetic test, which provides the maximum number of IOPS that the new AWS infrastructure can achieve. If you are moving your workloads to the AWS Cloud, you can run a tool such as `iostat` to profile the IOPS required by your workloads. While you can use a synthetic test to estimate your storage performance needs, the best way to quantify storage performance needs is through profiling an existing production database if that is an option.

Performing a synthetic test on the Amazon EBS volume allows you to specify the amount of concurrency and throughput that you want to simulate. It also helps determine the maximum number of IOPS and throughput needed for PostgreSQL workloads. pgbench is an open-source benchmark utility to run benchmark tests on PostgreSQL.

The test environment

To simulate the PostgreSQL client, we are using pgbench as the benchmarking tool. In this example, we are using an `r5.4xlarge` instance type.

Table 1: Amazon EC2 machine specifications

Configuration	Value
Instance type	`r5.4xlarge`
Operating system	Amazon Linux 6.1.41-63.114.amzn2023.x86_64
Memory	128 GB
CPU	16 vCPUs
Root volume	100 GB gp2
PostgreSQL data volume	500 GB (gp2, gp3, `io1`, or `io2`)

As part of the testing, four Amazon EC2 instances are provisioned for each of the volume type (gp2, gp3, io1, io2) with `r5.4xlarge` each having 100 GB gp2 as root volume and 500 GB data volume.

The below pgbench settings are used for the performance benchmarking test for each of the Amazon EBS volumes.

Install pgbench in each of the PostgreSQL database server:

```
sudo yum install postgresql15-contrib
```

Load test data in the each of the PostgreSQL databases:

```
pgbench -i –fillfactor=90 –scale=10000 postgres

bash-5.2$ pgbench -i –fillfactor=90 –scale=10000 postgres
dropping old tables...
NOTICE: table "pgbench_accounts" does not exist, skipping
NOTICE: table "pgbench_branches" does not exist, skipping
NOTICE: table "pgbench_history" does not exist, skipping
NOTICE: table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
1000000000 of 1000000000 tuples (100%) done (elapsed 2441.43 s, remaining 0.00 s))
vacuuming...
creating primary keys...
done in 3096.86 s (drop tables 0.01 s, create tables 0.03 s, client-side generate
 2445.44 s, vacuum 0.86 s, primary keys 650.52 s).
```

Set the max_connection parameter to 1000 in the PostgreSQL configuration for each of the servers:

```
echo "max_connections=1000">>/data/pgsql15/data/postgresql.conf
pg_ctl -D /data/pgsql15/data/ restart
```

Run the below command to conduct the performance testing in each of the PostgreSQL servers. pgbench uses the `-S` and `-N` options to execute a workload comprising an equal distribution of read and write operations, with 50% of the test involving `UPDATE` queries, and the remaining 50% consisting of `SELECT` queries. The variables that can be adjusted to align with the available system resources are the number of clients and threads. In this benchmarking, we used 5000 number of worker threads, and 950 clients.

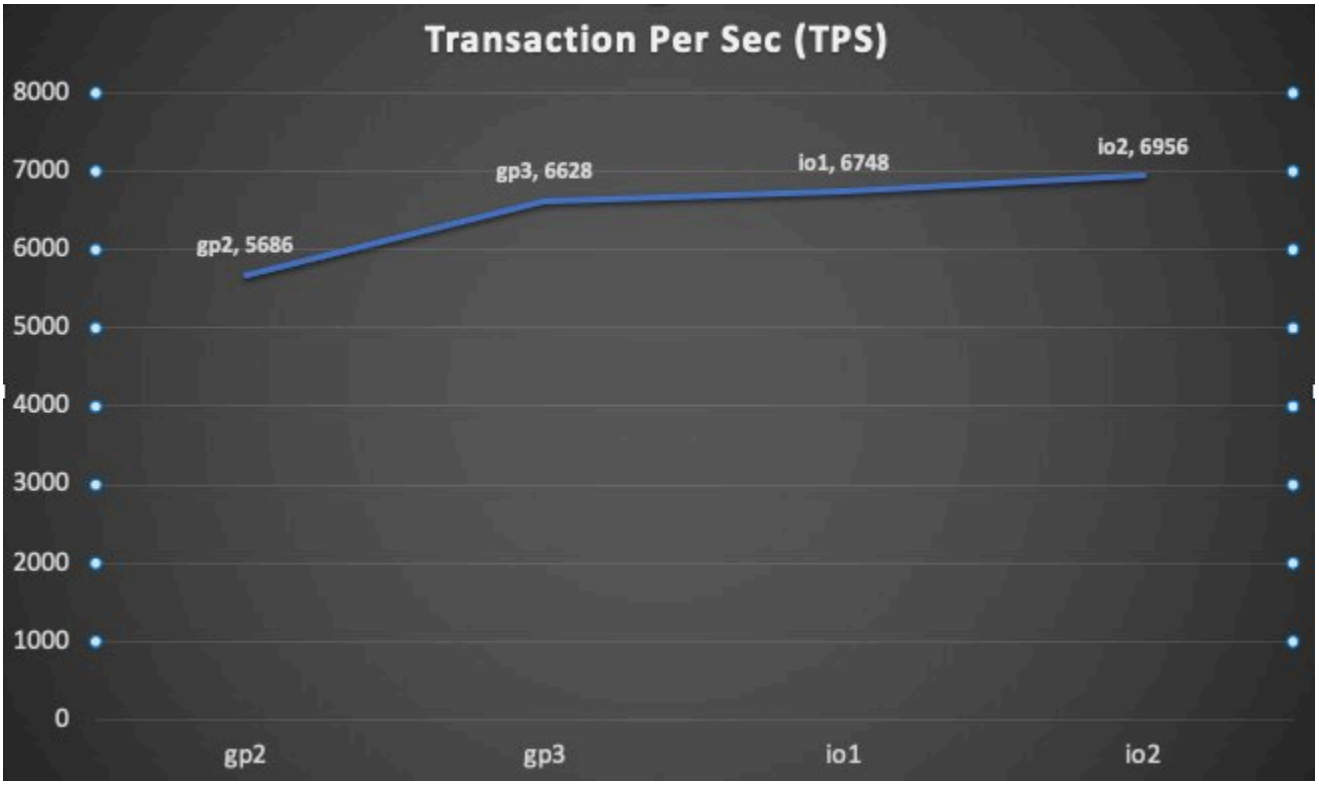
```
pgbench -U postgres -P 60 -c 950 -j 5000 -T 1800 -S -N postgres
```

Results

The various tests of the four different volume configurations yielded different results. The result set we focused are: latency average and transaction per second (TPS). The workload was run for 1,800 seconds and below are the trends in our result set. After closer observation, we observe that the minimum latency is offered by `io2` volume, that is 136ms. See the following performance analysis of same PostgreSQL workload on different Amazon EBS volume types:

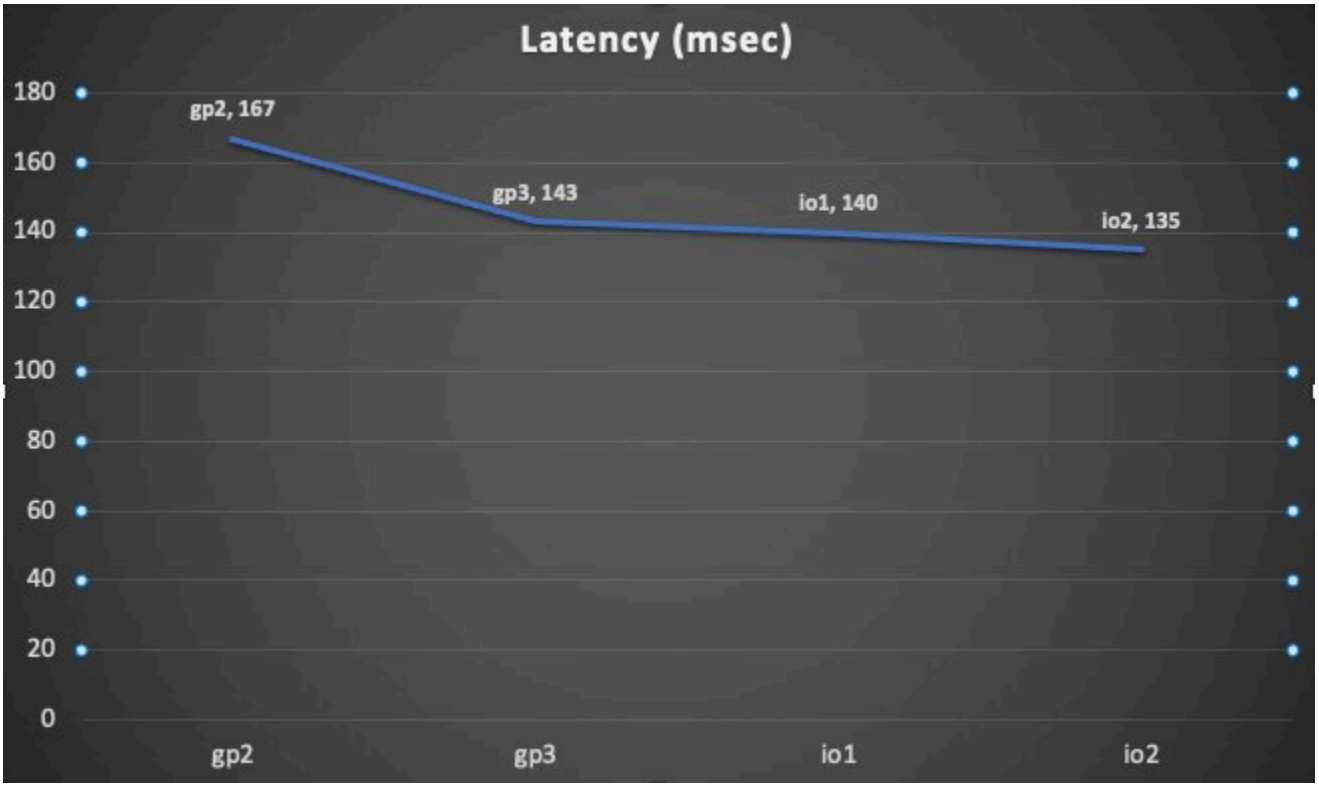
pgbench metrics	**gp2**	**gp3**	**io1**	**io2**
Initial connection time(ms)	444	420	384	381
Number of transactions actually processed	10238208	11931975	10926661	11267355
Transaction Per Sec (TPS)	5686	6628	6748	6956

The following diagram shows the gradual increase in TPS.



Gradual increase in TPS

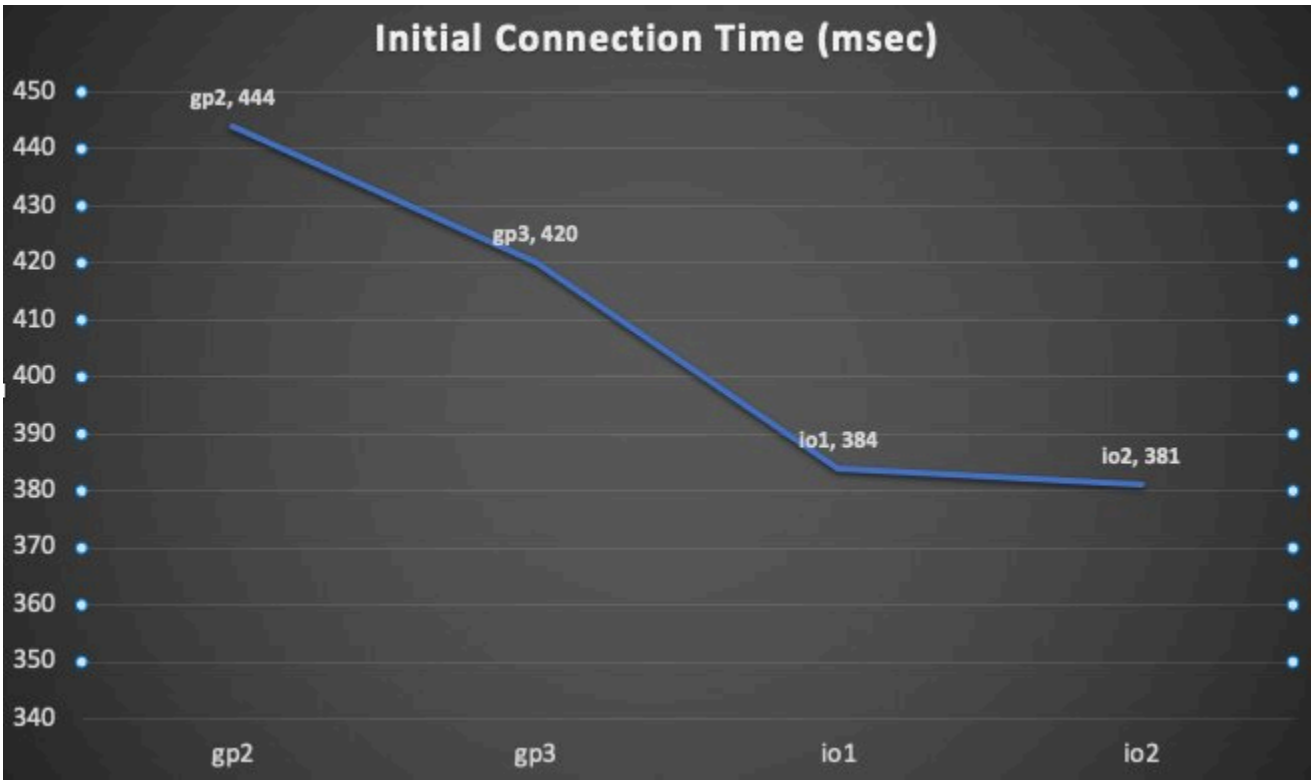
The following graph shows the gradual decrease in latency.



Gradual decrease in latency

Latency comparison for various Amazon EBS storage types

The below graph shows the gradual decrease in initial connection time.



Gradual decrease in initial connection time

Conclusion

The AWS cloud provides several options for deploying PostgreSQL and the infrastructure supporting it. Amazon Aurora for PostgreSQL and RDS for PostgreSQL provide a platform to operate, scale, and manage PostgreSQL database in AWS. It removes much of the complexity of managing and maintaining databases, allowing you to focus on improving applications. However, there are some cases where hosting PostgreSQL on Amazon EC2 and Amazon EBS brings operational flexibility and maintenance easiness. It's important to understand PostgreSQL workload and test it. This can help you decide which Amazon EC2 instance and storage to use for optimal performance and cost.

For a balanced performance and cost consideration, General Purpose SSD Amazon EBS volumes (gp2 and gp3) are good options. To maximize the benefit of gp2, you need to understand and monitor the burst credit. This helps determine whether you should consider other volume types. On the other hand, gp3 provides predictable 3,000 IOPS baseline performance and 125 MiB/s, regardless of volume size. With gp3 volumes, you can provision IOPS and throughput independently, without increasing storage size, at costs up to 20 percent lower per GB compared to gp2 volumes. If you have mission critical PostgreSQL workloads that need more consistent IOPS, then you should use Provisioned IOPS volumes (`io1`, `io2`, `io2` Block Express).

To maximize the benefit of both General Purpose and Provisioned IOPS volume types, AWS recommends using Amazon EBS-optimized Amazon EC2 instances and tuning database parameters to optimize storage consumption. This ensures dedicated network bandwidth for Amazon EBS volumes. You can cost effectively operate your PostgreSQL database in AWS without sacrificing performance by taking advantage of the durability, availability, and elasticity of the Amazon EBS volumes.

Contributors

Contributors to this document include:

- Vivek Singh, Principal Specialist Technical Account Manager - Databases, Amazon Web Services
- Arnab Saha, Senior Database Specialist Solutions Architect, Amazon Web Services

Further reading

For additional information, refer to:

- AWS Architecture Center
- PostgreSQL Performance Tuning
- PostgreSQL Database Backup Methods

Document history

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Initial publication	Whitepaper first published.	October 19, 2023

Note

To subscribe to RSS updates, you must have an RSS plug-in enabled for the browser that you are using.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.