AWS Whitepaper

Optimizing MySQL Running on Amazon EC2 Using Amazon EBS

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

Table of Contents

This whitepaper is for historical reference only. Some content might be outdated and some links might not be available.

Optimizing MySQL Running on Amazon EC2 Using Amazon EBS

Publication date: **December 7, 2021** (*Document history*)

This whitepaper is intended for Amazon Web Services (AWS) customers who are considering deploying their MySQL 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 MySQL databases. There are many deployment options and configurations for MySQL on Amazon EC2. This whitepaper provides performance benchmark metrics and general guidance so AWS customers can make an informed decision about whether to deploy their MySQL workloads on Amazon EC2.

Introduction

MySQL is one of the world's most popular open-source relational database engines. Its unique storage architecture provides you with many different ways of customizing database configuration according to the needs of your application. It supports transaction processing and high-volume operations. Apart from the robustness of the database engine, another benefit of MySQL is that the total cost of ownership is low. Several companies are moving their MySQL workloads into the cloud to extend the cost and performance benefits. AWS offers many compute and storage options that can help you optimize your MySQL 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 (Ops/s).
- **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 volume that provides 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 offer two types of Provisioned IOPS SSD volumes: io1 and io2.
- **Amazon EBS Throughput Optimized hard disk drive (HDD) (st1) volume** — Low- cost HDD volume designed for frequently accessed, throughput-intensive workloads.

MySQL on AWS deployment options

AWS provides various options to deploy MySQL like a fully managed database service, Amazon Relational Database Service (Amazon RDS) for MySQL. The Amazon Aurora database engine is designed to be wire-compatible with MySQL 5.6 and 5.7 using the InnoDB storage engine. You can also host MySQL on Amazon EC2 and self-manage the database, or browse the third-party MySQL offerings on the AWS Marketplace. This whitepaper explores the implementation and deployment considerations for MySQL on Amazon EC2 using Amazon EBS for storage.

Although Amazon RDS and Amazon Aurora with MySQL compatibility is a good choice for most of the use cases on AWS, deployment on Amazon EC2 might be more appropriate for certain MySQL workloads. With Amazon RDS you can connect to the database itself, which gives you access to the familiar capabilities and configurations in MySQL; however, access to the operating system (OS) isn'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 MySQL Enterprise tools. For example, enabling MySQL Enterprise Monitor requires OS-level access to gather monitoring information. As another example, MySQL Enterprise Backup requires OS-level access to access the MySQL data directory. In such cases, running MySQL on Amazon EC2 is a better alternative.

MySQL 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 your MySQL database. Amazon Aurora provides a Serverless MySQL-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 MySQL is 64 TB. The EBS volume for MySQL on Amazon EC2, conversely, supports up to 16 TB per volume.

Horizontal scaling is also an option in MySQL, where you can add MySQL secondary servers or read replicas so that you can accommodate additional read traffic into your database. 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 MySQL allows up to five read replicas. There are certain cases where you might need to enable specific MySQL replication features. Some of these features may require OS access to MySQL or advanced privileges to access certain system procedures and tables.

MySQL on Amazon EC2 is an alternative to Amazon RDS and Aurora for certain use cases. It allows you to migrate new or existing workloads that have very specific requirements. Choosing the right

compute, network, and—especially—storage configurations while taking advantage of its features plays a crucial role in achieving good performance at an optimal cost for your MySQL workloads.

Amazon EC2 block-level storage options

There are two block-level storage options for EC2 instances. The first option is an instance store, which consists of one or more instance store volumes exposed as block I/O devices. An instance store volume is a disk that is physically attached to the host computer that runs the EC2 virtual machine (VM). You must specify instance store volumes when you launch the EC2 instance. Data on instance store volumes will not persist if the instance stops, ends, or if the underlying disk drive fails.

The second option is an EBS volume, which provides off-instance storage that will persist independently from the life of the instance. The data on the EBS volume persists even if the EC2 instance that the volume is attached to shuts down or there is a hardware failure on the underlying host. 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 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 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. The remainder of this paper focuses on EBS volume-backed EC2 instances.

EBS volume features

EBS monitoring

Amazon EBS automatically sends data points to Amazon CloudWatch for one-minute intervals at no charge. Amazon CloudWatch metrics are statistical data that you can use to view, analyze, and set alarms on the operational behavior of your volumes. The EBS metrics can be viewed by selecting the monitoring tab of the volume in the Amazon EC2 console. For more information about the EBS metrics collected by CloudWatch, refer to the Amazon CloudWatch metrics for Amazon EBS.

EBS durability and availability

Durability in the storage subsystem for MySQL is especially important if you are storing user data, valuable production data, and individual data points. EBS volumes are designed for reliability with a 0.1 percent to 0.2 percent annual failure rate (AFR) compared to the typical 4 percent of commodity disk drives. EBS volumes are backed by multiple physical drives for redundancy that is replicated within the Availability Zone to protect your MySQL workload from component failure.

EBS snapshots

You can perform backups of your entire MySQL database using EBS snapshots. These snapshots are stored in Amazon Simple Storage Service (S3), which is designed for 99.999999999% (11 nines) of durability. To satisfy your recovery point and recovery time objectives, you can schedule EBS snapshots using Amazon CloudWatch Events.

Apart from providing backup, other reasons for creating EBS snapshots of your MySQL database include:

- **Set up a non-production or test environment** — You can share the EBS snapshot to duplicate the installation of MySQL in different environments and also share between different AWS accounts within the same Region. For example, you can restore a snapshot of your MySQL database that's in a production environment to a test environment to duplicate and troubleshoot production issues.
- **Disaster recovery** — EBS snapshots can be copied from one AWS Region to another for site disaster recovery.

A volume that is restored from a snapshot loads slowly in the background, which means that you can start using your MySQL database right away. When you perform a query on MySQL that finds a table that has not been downloaded yet, the data will be downloaded from Amazon S3. 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. Refer to Amazon EBS fast snapshot restore for more information. Best practices for restoring EBS snapshots are discussed in the *MySQL backups* section of this whitepaper.

EBS security

Amazon EBS supports several security features you can use from volume creation to utilization. These features prevent unauthorized access to your MySQL data.

You can use tags and resource-level permissions to enforce security on your volumes upon creation. Tags are key-value pairs that you can assign to your AWS resources as part of infrastructure management. 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. You can assign tags on EBS volumes during creation time, which allows you to enforce the management of your volume as soon as it is created.

Additionally, you can have granular control on who can create or delete tags through the IAM resource-level permissions. This granularity of control extends to the `RunInstances` and `CreateVolume` APIs where you can write IAM policies that requires the encryption of the EBS volume upon creation.

After the volume is created, you can use the IAM resource-level permissions for Amazon EC2 API actions where you can specify the authorized users or groups who can attach, delete, or detach EBS volumes to EC2 instances.

Protection of data in transit and at rest is crucial in most MySQL implementations. You can use Secure Sockets Layer (SSL) to encrypt the connection from your application to your MySQL database. To encrypt your data at rest, you can enable volume encryption during creation time. The new volume will get a unique 256-bit AES key, which is protected by the fully managed AWS Key Management Service. EBS snapshots created from the encrypted volumes are automatically encrypted.

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

The Elastic Volumes feature of EBS SSD volumes allows you to dynamically change the size, performance, and type of EBS volume in a single API call or within the AWS Management Console without any interruption of MySQL operations. This simplifies some of the administration and maintenance activities of MySQL workloads running on current generation EC2 instances.

You can call the ModifyVolume API to dynamically increase the size of the EBS volume if the MySQL database is running low on usable storage capacity. Note that decreasing the size of the EBS volume isn't supported, so AWS recommends that you do not over-allocate the 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 MySQL utilization, you can either change your volume type or add additional IOPS. The time it takes to complete these changes will depend on the size of your MySQL volume. You can monitor the progress of the volume modification either through the AWS Management Console or CLI. You can also create CloudWatch Events to send alerts after the changes are complete.

EBS volume types

General Purpose SSD volumes

General Purpose SSD volumes are designed to provide a balance of price and performance.

The General Purpose SSD (gp3) volumes offer cost-effective storage that is ideal for a broad range of database workloads. 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

The older General Purpose SSD (gp2) volume is also a good option because it also offers balanced price and performance. To maximize the performance of the gp2 volume, you need to know how the burst bucket 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. 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.

In addition to changing 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 RAID 0 configuration distributes the I/O across volumes in a stripe. Adding an additional volume also increases the throughput of your MySQL database. Throughput is the read/write transfer rate, which is the I/O block size multiplied by the IOPS rate performed on the disk. AWS recommends adding the same volume size into the stripe set since the performance of the stripe is limited to the worst performing volume in the set. Also consider fault tolerance in RAID 0. A loss of a single volume results in a complete data loss for the array. If possible, use RAID 0 in

a MySQL primary/secondary environment where data is already replicated in multiple secondary nodes.

Provisioned IOPS SSD (io1) volumes

Provisioned IOPS SSD (io1 and io2) 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, and 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 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.

The maximum ratio of Provisioned IOPS to requested volume size (in GiB) is 50:1 for io1 volumes, and 500:1 for io2 volumes. For example, a 100 GiB io1 volume can be provisioned with up to 5,000 IOPS, while a 100 GiB io2 volume can be provisioned with up to 50,000 IOPS.

To maximize the volume throughput, AWS recommends using an EBS-optimized EC2 instance type (note that most new EC2 instances are EBS-optimized by default, with no extra charge). This provides dedicated throughput between your EBS volume and EC2 instance. As instance size and type affects volume throughput, choose an instance that has more channel bandwidth than the maximum throughput of the io1 volume.

For example, an `r5.12xlarge` instance provides a maximum bandwidth of 9,500 MB/s. Therefore, it can more than handle the 1,187.5 MB/s maximum throughput of the io1 volume. Another approach to increasing io1 throughput is to configure RAID 0 on your EBS volumes. For more information about RAID configuration, refer to RAID configuration in the *Amazon EC2 User Guide*.

MySQL considerations

MySQL offers a lot of parameters that you can tune to obtain optimal performance for every type of workload. This section focuses on the MySQL InnoDB storage engine. It also looks at the MySQL parameters that you can optimize to improve performance related to the I/O of EBS volumes.

Caching

Caching is an important feature in MySQL. Knowing when MySQL will perform a disk I/O instead of accessing the cache will help you tune for performance. When you are reading or writing data, an InnoDB buffer pool caches your table and index data. This in-memory area resides between your read/write operations and the EBS volumes. Disk I/O will occur if the data you are reading isn't in the cache or when the data from dirty (that is, modified only in memory) InnoDB pages needs to be flushed to disk.

The buffer pool uses the Least Recently Used (LRU) algorithm for cached pages. When you size the buffer pool too small, the buffer pages may have to be constantly flushed to and from the disk, which affects performance and lowers the query concurrency. The default size of the buffer pool is 128 MB. You can set this value to 80 percent of your server's memory; however, be aware that there may be paging issues if other processes are consuming memory. Increasing the size of the buffer pool works well when your dataset and queries can take advantage of it. For example, if you have one GiB of data and the buffer pool is configured at 5 GiB, then increasing the buffer pool size to 10 GiB will not make your database faster. A good rule of thumb is that the buffer pool should be large enough to hold your "hot" dataset, which is composed of the rows and indexes that are used by your queries. Starting in MySQL 5.7 version, the `innodb_buffer_pool_size` can be set dynamically, which allows you to resize the buffer pool without restarting the server.

Database writes

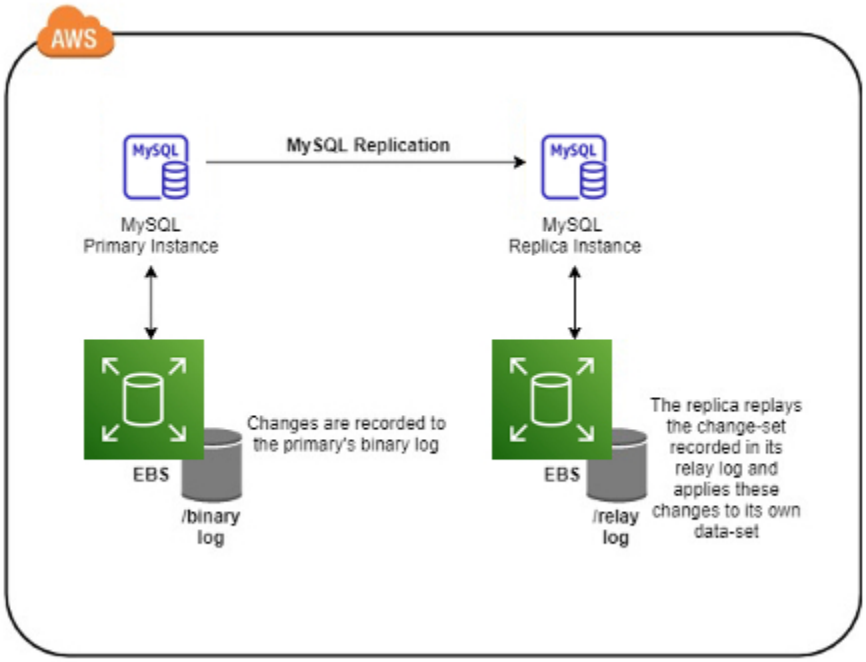
InnoDB does not write directly to disk. Instead, it first writes the data into a double write buffer. Dirty pages are the modified portion of these in-memory areas. The dirty pages are flushed if there isn't enough free space. The default setting (`innodb_flush_neighbors = 1`) results in a sequential I/O by flushing the contiguous dirty pages in the same extent from the buffer pool. This option should be turned off (by setting `innodb_flush_neighbors = 0`) so you can maximize the performance by spreading the write operations over your EBS SSD volumes.

Another parameter that can be modified for write-intensive workloads is `innodb_log_file_size`. When the size of your log file is large there are fewer data flushes, which reduces disk I/O. However, if your log file is too big, you will generally have a longer recovery time after a crash. MySQL recommends that the size of your log files should be large enough where your MySQL server will spread out the checkpoint flush activity over a longer period. The recommendation from MySQL is to size the log file to where it can accommodate an hour of write activity.

MySQL read replica configuration

MySQL allows you to replicate your data so you can scale out your read-heavy workloads with primary / secondary (read replica) configuration. You can create multiple copies of your MySQL database into one or more secondary databases so that you can increase the read throughput of your application. The availability of your MySQL database can be increased with the secondary. When a primary instance fails one of the secondary servers can be promoted, reducing the recovery time.

MySQL supports different replication methods. There is the traditional binary log file position-based replication where the primary's binary log is synchronized with the secondary's relay log. The following diagram shows the binary log file position-based replication process.



Binary log file position-based replication process

Replication between primary and secondary using global transaction identifiers (GTIDs) was introduced in MySQL 5.6. A GTID is a unique identifier created and associated with each transaction committed on the server of origin (primary). This identifier is unique not only to the server on which it originated, but is unique across all servers in a given replication setup. With GTID-based replication, it is no longer necessary to keep track of the binary log file or position on the primary to replay those events on the secondary. The benefits of this solution include a more malleable replication topology, simplified failover, and improved management of multi-tiered replication.

MySQL replication considerations

Prior to MySQL 5.6, replication was single threaded, with only one event occurring at a time. Achieving throughput in this case was usually done by pushing a lot of commands at low latency. To obtain larger I/O throughput, your storage volume requires a larger queue depth. An EBS io1 SSD volume can have up to 20,000 IOPS, which, in turn, means it has a larger queue depth. 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 EBS volumes for your MySQL database. You can join several volumes together in a RAID 0 configuration to use the available bandwidth of the EBS-optimized instances to deliver the additional network throughput dedicated to EBS. For MySQL 5.6 and above, replication is multi-threaded. This performs well on EBS volumes because it relies on parallel requests to achieve maximum I/O throughput. During replication there are sequential and random traffic patterns.

There are the sequential writes for the binary log (binlog) shipment from the primary server and sequential reads of the binlog and relay log. 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 EBS volume can provide.

Switching from a physical environment to AWS

Customers migrating from their physical MySQL Server environment into AWS usually have a battery-backed caching RAID controller, which allows data in the cache to survive a power failure. Synchronous operations are set up so that all I/O is committed to the RAID controller cache instead

of the OS main memory. Therefore, it is the controller instead of the OS that completes the write process. Due to this environment, the following MySQL parameters are used to ensure that there is no data loss:

On the Primary Side

```
sync_binlog = 1
innodb_flush_log_at_trx_commit=1
```

On the Secondary Side

```
sync_master_info = 1
sync_relay_log = 1
sync_relay_log_info = 1
innodb_flush_log_at_trx_commit=1
```

These parameters will cause MySQL to call `fsync()` to write the data from the buffer cache to the disk after any operation with the binlog and relay log. This is an expensive operation that increases the amount of disk I/O.

The immediate synchronize log to disk MySQL parameter does not provide any benefit for EBS volumes. In fact, it causes degraded performance. EBS volumes are automatically replicated within an Availability Zone, which protects them from component failures. Turning off the `sync_binlog` parameter allows the OS to determine when to flush the bin and relay log buffers to the disk, reducing I/O.

The `innodb_flush_log_at_trx_commit=1` is required for full ACID compliance. If you need to synchronize the log to disk for every transaction, then you may want to consider increasing the IOPS and throughput of the EBS volume. In this situation, you may want to separate the binlog and relay log from your data files as separate EBS volumes. You can use Provisioned IOPS SSD volumes for the binlog and relay log to have more predictable performance. You may also use the local SSD of the MySQL secondary instance if you need more throughput and IOPS.

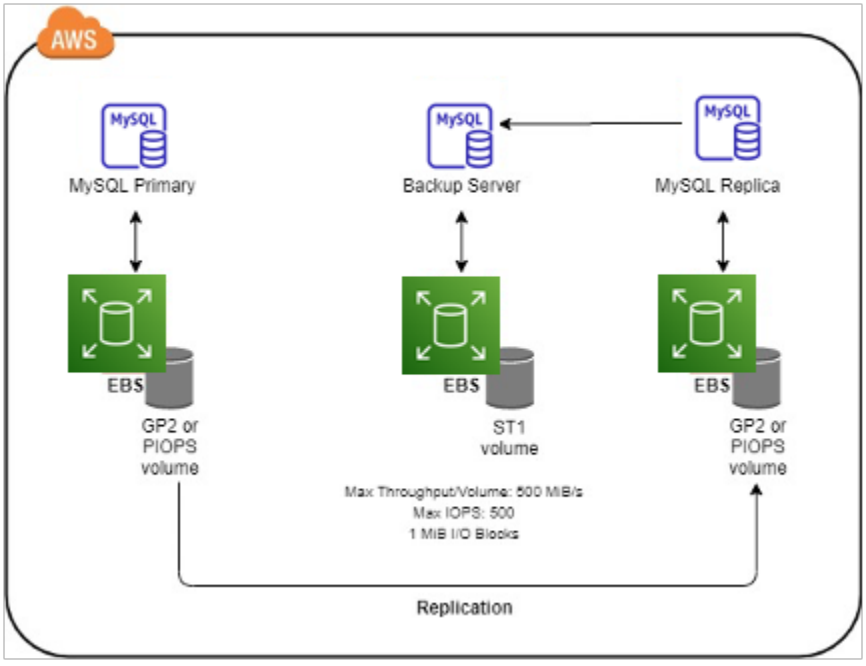
MySQL backups

Backup methodologies

There are several approaches to protecting your MySQL data depending on your Recovery Time Objective (RTO) and Recovery Point Objective (RPO) requirements. The choice of performing a hot or cold backup is based on the uptime requirement of the database. When it comes meeting your RPO, your backup approach will be based the logical database level or the physical EBS volume-level backup. This section explores the two general backup methodologies.

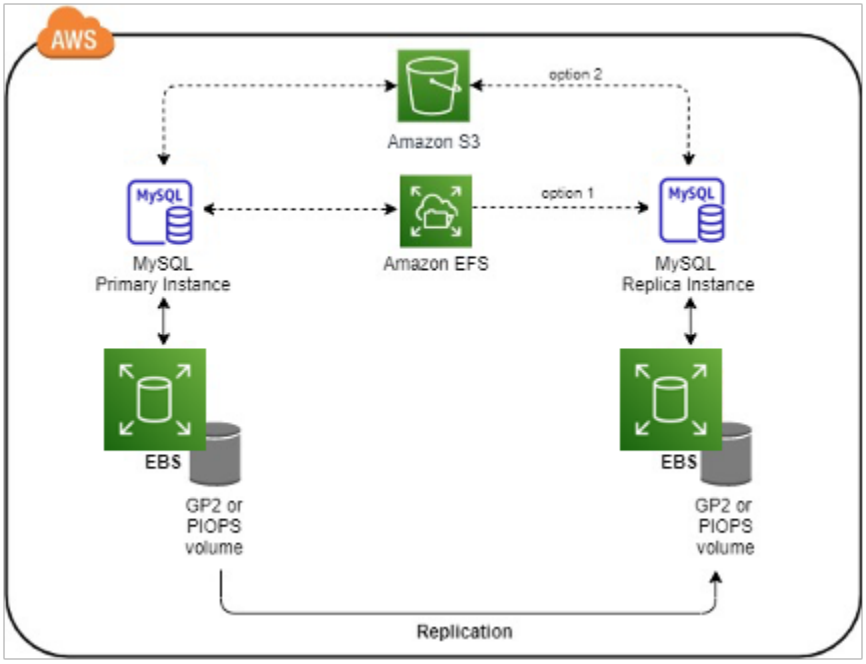
The first general approach is to back up your MySQL data using database-level methodologies. This can include making a hot backup with MySQL Enterprise Backup, making backups with mysqldump or mysqlpump, or by making incremental backups by enabling binary logging.

If the primary database server exhibits performance issues during a backup, a replication secondary database server or a read replica database server can be created to provide the source data 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 MySQL secondary server to read the backup data.



Using an st1 volume as a backup source

Another option is to have the MySQL secondary server back up the database files directly to Amazon Elastic File System (Amazon EFS) or Amazon S3. Amazon EFS is an elastic file system that stores its data redundantly across multiple Availability Zones. Both the primary and the secondary instances can attach to the EFS file system. The secondary instance can initiate a backup to the EFS file system 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

The second general approach 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 of the information needed to restore your data (from the time the snapshot was taken) to a new EBS volume.

One consideration when utilizing EBS snapshots for backups is to make sure the MySQL data remains consistent. During an EBS snapshot, any data not flushed from the InnoDB buffer cache to disk will not be captured. There is a MySQL command flush tables with read lock that will flush all the data in the tables to disk and only allow database reads but put a lock on database writes. The

lock only needs to last for a brief period of time until the EBS snapshot starts. The snapshot will take 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 also combine these approaches by using database-level backups for more granular objects, such as databases or tables, and using EBS snapshots for larger 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).

Creating snapshots of an EBS RAID array

When you take a snapshot of an attached EBS volume that is in use, the snapshot excludes data cached by applications or the operating system. For a single EBS volume, this might not be a problem. However, when cached data is excluded from snapshots of multiple EBS volumes in a RAID array, restoring the volumes from the snapshots can degrade the integrity of the array.

When creating snapshots of EBS volumes that are configured in a RAID array, it is critical that there is no data I/O to or from the volumes when the snapshots are created. RAID arrays introduce data interdependencies and a level of complexity not present in a single EBS volume configuration.

To create an *application-consistent* snapshot of your RAID array, stop applications from writing to the RAID array, and flush all caches to disk. At the database level (recommended), you can use the flush tables with read lock command. Then ensure that the associated EC2 instance is no longer writing to the RAID array by taking steps such as freezing the file system with the `sync` and `fsfreeze` commands, unmounting the RAID array, or shutting down the associated EC2 instance. After completing the steps to halt all I/O, take a snapshot of each EBS volume.

Restoring a snapshot creates a new EBS volume, then you assemble the new EBS volumes to build the RAID volumes. After that you mount the file system and then start the database. To avoid the performance degradation after the restore, AWS recommends initializing the EBS volume. The initialization of a large EBS volume can take some time to complete because data blocks have to be fetched from the S3 bucket where the snapshots are stored. To make the database available in a shorter amount of time, the initialization of the EBS volume can be done through multi-threaded reads of all the required database files for the engine recovery.

Monitoring MySQL and EBS volumes

Monitoring provides visibility into your MySQL workload. Understanding the resource utilization and performance of MySQL usually involves correlating the data from the database performance metrics gathered from MySQL and infrastructure-related metrics in CloudWatch. There are many tools that you can use to monitor MySQL, some of which include:

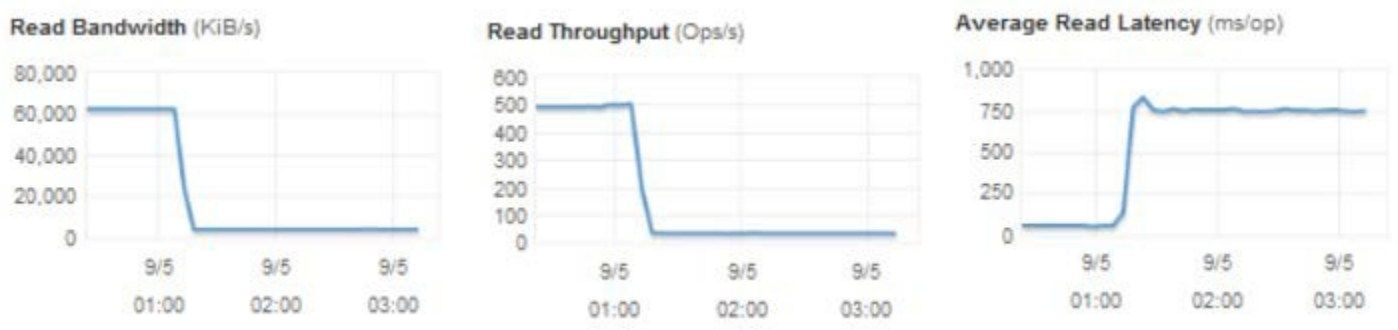
- Tools from MySQL, such as MySQL Workbench Performance
- Third-party software tools and plugins
- MySQL monitoring tools in the AWS Marketplace

When the bottleneck for MySQL performance is related to storage, database administrators usually look at latency when they run into performance issues of transactional operations. Further, if the performance is degraded due to MySQL loading or replicating data, then throughput is evaluated. These issues are diagnosed by looking at the EBS volume metrics collected by CloudWatch.

Latency

Latency is defined as the delay between request and completion. Latency is experienced by slow queries, which can be diagnosed in MySQL by enabling the MySQL performance schema. 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 EC2 console. This section covers the factors contributing to high latency.

High latency can result from exhausting the available Provisioned IOPS in your EBS volume. For gp2 volumes, the CloudWatch metric `BurstBalance` is presented so that you can determine if you have depleted the available credit for IOPS. When bandwidth (KiB/s) and throughput (Ops/s) are reduced, latency (ms/op) increases.



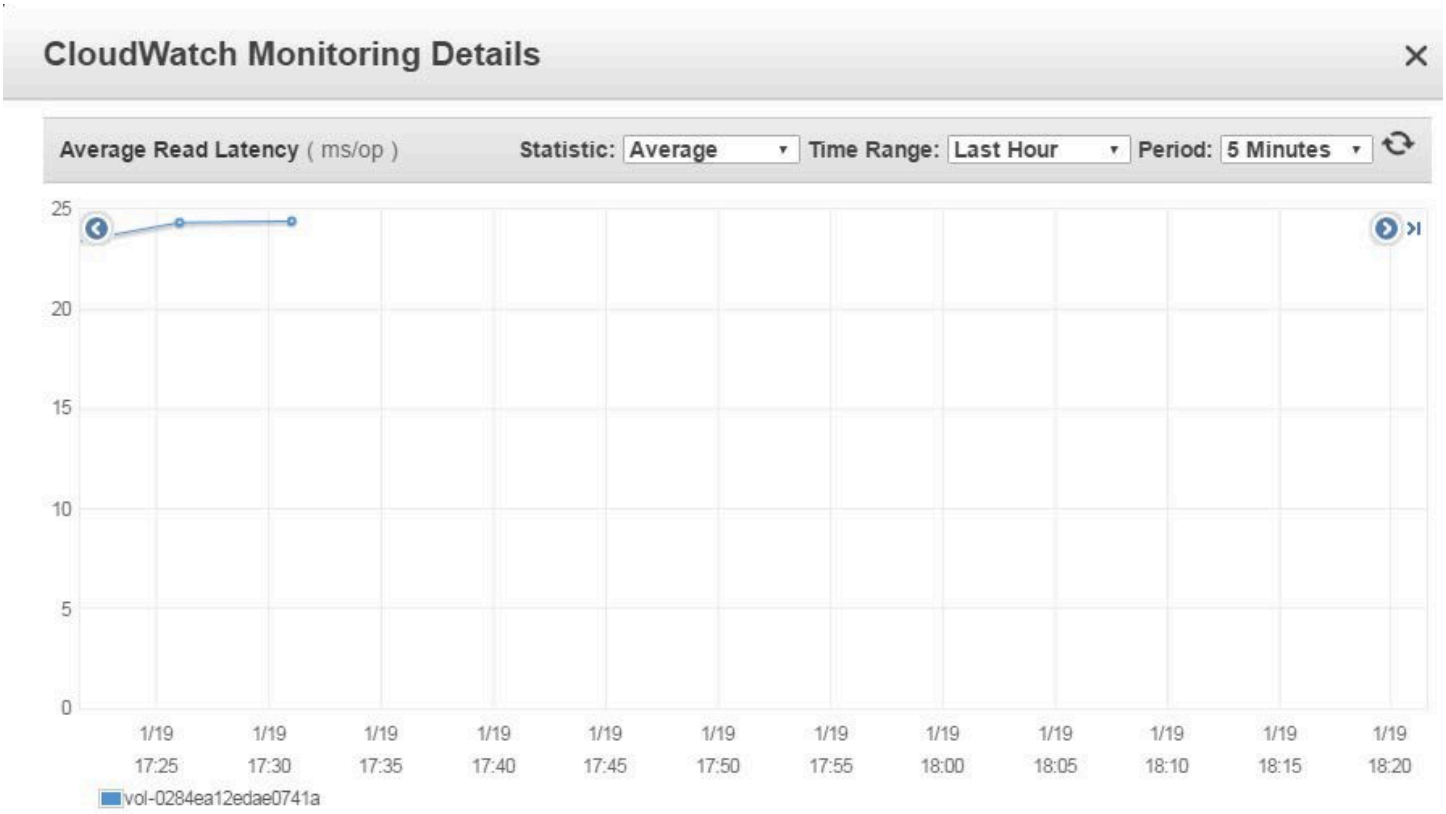
BurstBalance metric showing that when bandwidth and throughput are reduced, latency increases

Disk queue length can also contribute to high latency. Disk queue length refers to the outstanding read/write requests that are waiting for resources to be available. The CloudWatch metric `VolumeQueueLength` shows the number of pending read/write operation requests for the volume. This metric is an important measurement to monitor if you have reached the full utilization of the Provisioned IOPS on your EBS volumes. Ideally, the EBS volumes must maintain an average queue length of about one per minute for every 200 Provisioned IOPS. Use the following formula to calculate how many IOPS will be consumed based on the disk queue length:

```
Consumed IOPS = 200 IOPS * VolumeQueueLength
```

For example, say you have assigned 2,000 IOPS to your EBS volume. If the `VolumeQueueLength` increases to 10, then you consume all of your 2000 Provisioned IOPS, which results in increased latency.

Pending MySQL operations will stack up if you observe the increase of the `VolumeQueueLength` without any corresponding increase in the Provisioned IOPS, as shown in the following screenshot.



Average queue length and average read latency metrics

Throughput

Throughput is the read/write transfer rate to storage. It affects MySQL database replication, backup, and import/export activities. When considering which AWS storage option to use to achieve high throughput, you must also consider that MySQL 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 volumes on an EBS-optimized instance. In terms of throughput, io1 volumes have a maximum of 320 MB/s per volume, while gp2 volumes have a maximum of 160 MB/s per volume.

Insufficient throughput to underlying EBS volumes can cause MySQL secondary servers to lag, and can also cause MySQL 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).

In addition to using CloudWatch metrics, AWS recommends reviewing the AWS Trusted Advisor to check alerts when an EBS volume attached to an instance isn't EBS-optimized. EBS optimization ensures dedicated network throughput for your volumes. An EBS-optimized instance has segregated traffic, which is useful as many EBS volumes have significant network I/O activities. Most new instances are EBS-optimized by default, at no extra charge.

MySQL benchmark observations and considerations

Testing your MySQL database will help you determine what type of volume you need and ensure that you are choosing the most cost-effective and performant solution.

There are a couple of ways to determine the number of IOPS that you need. For an existing workload, you can monitor the current consumption of EBS volume IOPS through the CloudWatch metrics detailed in the *Monitoring MySQL and EBS volumes* section of this document.

If this is a new workload, you can do a synthetic test, which will provide you with the maximum number of IOPS that your new AWS infrastructure can achieve. If you are moving your workload to the AWS Cloud, you can run a tool such as `iostat` to profile the IOPS required by your workload. While you can use a synthetic test to estimate your storage performance needs, the best way to quantify your storage performance needs is through profiling an existing production database if that is an option.

Performing a synthetic test on the EBS volume allows you to specify the amount of concurrency and throughput that you want to simulate. Testing will allow you to determine the maximum number of IOPS and throughput needed for your MySQL workload.

There are a couple of tools that you can use:

- Mysqlslap is an application that emulates client load for MySQL Server.
- Sysbench is a popular open-source benchmark used to test open-source database management systems (DBMS).

The test environment

To simulate the MySQL client for the Sysbench tests, this example uses an r5.8xlarge instance type with a 10-gigabit network interface.

Table 1: Sysbench machine specifications

Sysbench server	
Instance type	`r5.8xlarge`

Sysbench server	
Memory	256 GB
CPU	32 vCPUs

All of the MySQL servers tested on used the `r5.8xlarge` instance type.

Table 2: MySQL server machine specifications

MySQL server	
Instance type	`r5.8xlarge`
Memory	256 GB
CPU	32 vCPUs
Storage	500 GB gp2 EBS Volume
Root volume	256 GB gp2
MySQL data volume	500 GB (gp2, gp3, io1 or io2)

To increase performance on the Sysbench Linux client, enable Receive Packet Steering (RPS) and Receive Flow Steering (RFS). RPS generates a hash to determine which CPU will process the packet. RFS handles the distribution of packets to the available CPUs.

Enable RPS with the following shell command:

```
sudo sh -c 'for x in /sys/class/net/eth0/queues/rx-*; do echo ffffffff > $x/rps_cpus;
 done'
sudo sh -c "echo 4096 > /sys/class/net/eth0/queues/rx-0/rps_flow_cnt"

sudo sh -c "echo 4096 > /sys/class/net/eth0/queues/rx-1/rps_flow_cnt
```

Enable RFS with the following shell command:

```
sudo sh -c "echo 32768 > /proc/sys/net/core/rps_sock_flow_entries"
```

Tuned compared to default configuration parameter testing

Perform a Sysbench test to compare the difference between tuned MySQL and default parameter configurations (refer to Table 3). Use a MySQL dataset of 100 tables with 10 million records per table for the test.

Table 3: MySQL parameters

Parameters	Default	Tuned
innodb_buffer_pool_size	134MB	193G
innodb_flush_method	fsync (Linux)	O_DIRECT
innodb_flush_neighbors	1	0
innodb_log_file_size	50MB	256MB

Run the following Sysbench read/write command:

```
$ sysbench ./oltp_read_write.lua <connection info>  --table_size=10000000 --max-
requests=0
--simple-ranges=0 --distinct-ranges=0 --sum-ranges=0 --order-ranges=0 --point-
selects=0
--time=3600 --threads=1024 --rand-type=uniform run
```

Results of the Sysbench test are presented in Table 4. Under optimized conditions, the MySQL server processed approximately 12 times the number of transactions per section compared to the default configuration.

Table 4: Sysbench results

Sysbench metrics	Default	Tuned
Queries:		
Read	17511928	223566532
Write	5003408	63876152

Sysbench metrics	Default	Tuned
Other	2501704	31938076
Total	25017040	319380760
Transactions	1250852 (347.37 per sec.)	15969038 (4434.57 per sec.)
Queries	25017040 (6947.43 per sec.)	319380760 (88691.33 per sec.)
ignored errors:	0 (0.00 per sec.)	0 (0.00 per sec.)
reconnects:	0 (0.00 per sec.)	0 (0.00 per sec.)
General statistics:		
Total time	3600.9046s	3601.0355s
Total number of events	1250852	15969038
Latency (ms):		
min	7.72	4.843
avg	2947.65	230.90
max	95885.04	6158.04
95^{th} percentile	9284.15	1258.08
sum	3687074024.45	3687189581.27
Thread fairness:		
events (avg/stddev):	1221.5352/48.86	15594.7637/45.63
runtime (avg/stddev):	3600.6582/0.11	3600.7711/0.04

Other InnoDB configuration options to consider for better performance of heavy I/O MySQL workloads are detailed in the MySQL Optimizing InnoDB Disk I/O documentation. When

considering these configurations, AWS suggests performing a test after deployment to ensure that it will be safe for your application.

Comparative analysis of different storage types

Conduct the test across four different MySQL server configurations with the following configurations:

- MySQL Server - EBS General Purpose SSD (gp2)
  - 500 GB SQL data drive
  - 1,500 baseline IOPS / 3,000 burstable IOPS

- MySQL Server - EBS Provisioned IOPS SSD (gp3)
  - 500 GB SQL data drive
  - 3,000 Provisioned IOPS

- MySQL Server - EBS Provisioned IOPS SSD (io1)
  - 500 GB SQL data drive
  - 3,000 Provisioned IOPS

- MySQL Server - EBS Provisioned IOPS SSD (io2)
  - 500 GB SQL data drive
  - 3,000 Provisioned IOPS

> **Note**
>
> Unless specified, all EBS volumes are unencrypted.

Sysbench client and MySQL server setup

Table 5: Server setup for MySQL database and Sysbench client

Use case	Instance type	vCPUs	Memory	Instance storage	EBS-optimized	Network
MySQL database	r5.8xlarge	32	256	EBS only	Yes	10 Gigabit
Sysbench client (AWS Cloud9)	r5.8xlarge	32	256	EBS only	Yes	10 Gigabit

Tests were performed using Sysbench read/write OLTP test by running the following Sysbench command below over a one-hour period.

```
$ sysbench ./oltp_read_write.lua <connection info>  --table_size=10000000
--max-requests=0 --simple-ranges=0 --distinct-ranges=0 --sum-ranges=0
--order-ranges=0 --point-selects=0 --time=3600 --threads=1024
--rand-type=uniform run
```

Results

The various tests of the four different volume configurations yielded similar results, with each server processing approximately 3,600 Sysbench transactions per second. There was no discernible workload difference is noticed while running performance consistency test in all four volumes. Upon closer examination, you observe that the minimum latency is offered by the IO2 volume and less than one millisecond latency is observed for the same workload.

Table 6: Performance analysis of same MySQL workload on different EBS volume types

Sysbench metrics	gp2	gp3	io1	io2
SQL statistics				
read queries	17511928	181507690	188343428	186051460
write queries	5003408	51859340	53812408	53157560

Sysbench metrics	gp2	gp3	io1	io2
other queries	2501704	25929670	26906204	26578780
total queries	25017040	259296700	269062040	265787800
transactions	12508520 (3470.37 per sec.)	12964835 (3600.93 per sec.)	13453102 (3733.12 per sec.)	13289390 (3690.20 per sec.)
queries	250170400 (69470.43 per sec.)	259296700 (72018.53 per sec.)	269062040 (74662.42 per sec.)	265787800 (73803.92 per sec.)
Latency (ms)				
min	7.72	6.82	6.1	6.02
avg	294.65	284.35	274.24	277.45
max	95885.04	43718.24	33179.31	34803.75
95th percentile	928.15	816.63	943.16	861.95
sum	3687074024.45	3686559158.83	3689386834.2	3687138536.08
EBS statistics				
Write latency (ms)	1.1	1.01	0.994	0.824
Volume queue length (count)	3.49	3.01	3.227	2.71

Conclusion

The AWS Cloud provides several options for deploying MySQL and the infrastructure supporting it. Amazon RDS for MySQL provides a very good platform to operate, scale, and manage your MySQL database in AWS. It removes much of the complexity of managing and maintaining your database, allowing you to focus on improving your applications.

However, there are some cases where MySQL on Amazon EC2 and Amazon EBS that work better for some workloads and configurations. It is important to understand your MySQL workload and test it. This can help you decide which EC2 server 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 will help you 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 MySQL workloads that need more consistent IOPS, then you should use Provisioned IOPS volumes (io1 or io2).

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

Contributors

Contributors to this document include:

- Marie Yap, Enterprise Solutions Architect, Amazon Web Services
- Ricky Chang, Cloud Infrastructure Architect, Amazon Web Services
- Kehinde Otubamowo, Database Partner Solutions Architect, Amazon Web Services
- Arnab Saha, Cloud Support DBA, Amazon Web Services
- Chi Dinjors, Cloud Support Engineer, Amazon Web Services

Further reading

For additional information, refer to:

- AWS Architecture Center
- MySQL Performance Tuning and Optimization Resources
- MySQL 5.7 Performance Tuning Immediately After Installation
- MySQL Database Backup Methods

Document history

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

Change	Description	Date
Whitepaper updated	Updated for technical accuracy.	December 7, 2021
Initial publication	Whitepaper first published.	November 1, 2017

Note

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

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

AWS Glossary

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