Optimizing MySQL Running on Amazon EC2 Using Amazon EBS

September 2017

Notices

Contents

Abstract

This whitepaper is intended for AWS customers who are considering deploying their MySQL database on Amazon Elastic Compute Cloud (EC2) using Amazon Elastic Block Store (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 will provide performance benchmark metrics and general guidance so AWS customers can make an informed decision about whether to deploy their MySQL workloads in AWS.

Introduction

MySQL is one of the world's most popular open source relational database engine. 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. Amazon Web Services (AWS) offers many compute and storage options that can help you optimize your MySQL deployments.

Terminology

Below are definitions 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 acknowledgement (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 (EBS) Volume**: Persistent block storage volumes for use with Amazon Elastic Compute Cloud (EC2) instances
- **Amazon EBS General Purpose SSD (gp2) Volume**: General purpose SSD volume that balances price and performance for a wide variety of transactional workloads
- **Amazon EBS Provisioned IOPS SSD (io1) Volume**: Highest performance SSD volume designed for latency-sensitive transactional workloads
- **Amazon EBS Throughput Optimized HDD (st1) Volume**: Low-cost HDD volume designed for frequently accessed, throughput-intensive workloads

MySQL on AWS Deployment Options

There are two common options for deploying MySQL on AWS. The first option is deploying MySQL on Amazon EC2. The second option is leveraging Amazon Relational Database Service (Amazon RDS). Amazon RDS for MySQL is a managed relational database service that makes it easy to set up, operate, and scale MySQL deployments in AWS. This section explores the implementation and deployment considerations for MySQL on Amazon EC2.

While Amazon RDS is a good choice for most MySQL on AWS use cases, deployment on Amazon EC2 may 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. 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 6 TB. The EBS volume for MySQL on Amazon EC2, on the other hand, supports up to 16 TB per volume.

Horizontal scaling is also an option in MySQL, where you can add MySQL slaves 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, Command Line Interface (CLI), or REST API. Amazon RDS for MySQL allows up to 5 read replicas. There are certain cases where you may 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. Examples of features that you can implement in MySQL on Amazon EC2 are replication with Global Transaction

Identifiers (ID), filtered or partial replication, and semisynchronous replication, which is implemented by plugins.

MySQL on Amazon EC2 is an alternative to Amazon RDS 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. You must specify instance store volumes when you launch the EC2 instance. Data on instance store volumes will not persist if the instance stops, terminates, or if the underlying disk drive fails.

The second option is an EBS volume, which provides off-instance storage that can persist independently from the life of the instance. The data on the EBS volume will persist 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 deleted explicitly.

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, please see Instance Store Volumes within the Amazon EC2 User Guide.[1] The remainder of this paper will focus on EBS volume-backed EC2 instances.

EBS Volume Features

EBS Monitoring

Amazon EBS automatically sends data points to Amazon CloudWatch at five-minute intervals for General Purpose SSD (gp2), Throughput Optimized HDD (st1), and Cold HDD (sc1) volumes. Provisioned IOPS SSD (io1) volumes send data points to CloudWatch at one-minute intervals. 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, see the Amazon EBS Metrics and Dimensions.[2]

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% to 0.2% annual failure rate (AFR) compared to the typical 4% 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.[3]

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 another durable storage service. To satisfy your recovery point and recovery time objectives, you can schedule EBS snapshots using Amazon CloudWatch Events.[4]

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

- Resize MySQL storage - You can restore a snapshot of the EBS volume to a larger volume size.
- 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 hits a table that has not been downloaded yet, the data will be downloaded from Amazon S3. Best practices for restoring EBS snapshots will be discussed in the Backup and Restore section.

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 via 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 [5] where you can specify the authorized IAM 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.[6]

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.[7]

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 we recommend that you do not over-allocate the EBS volume size any more than you have to in order 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 (gp2) Volumes

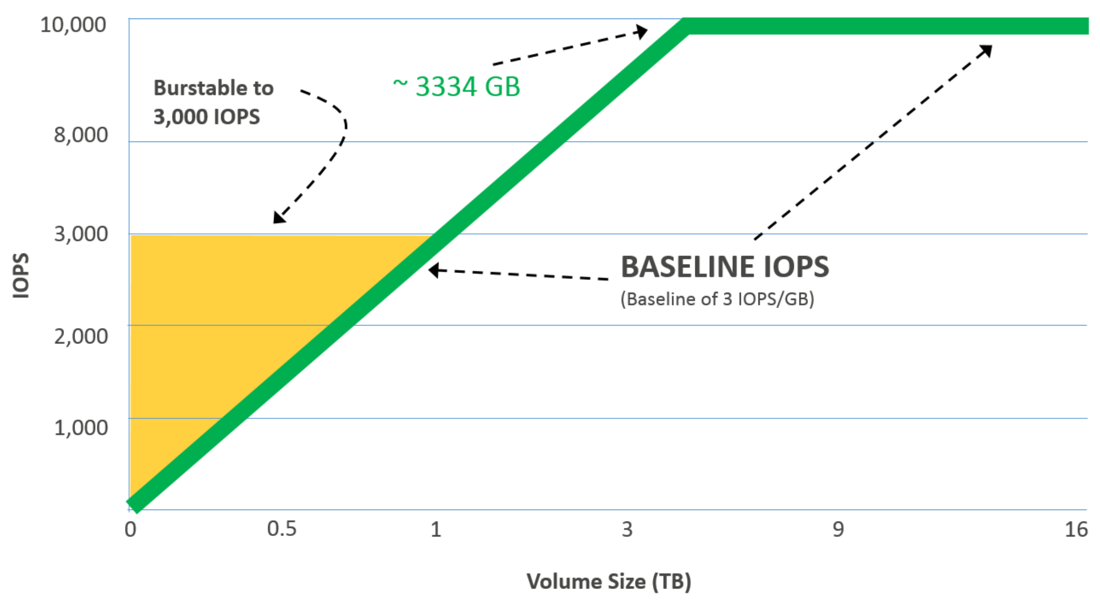
For most standalone MySQL workloads, the General Purpose SSD (gp2) volume is a good option because it offers balanced price and performance. To maximize the performance of the gp2 volume, you need to know how the burst bucket works.

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 10,000 IOPS (see the following figure). 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.

Upon creation, each gp2 volume starts with an initial baseline I/O credit balance of 5.4 million I/O credits. When additional performance is needed, the volume will draw on its I/O credits to burst to the required performance level. Examples of when you might need the additional performance include the following:

- Performing MySQL backups
- Restarting MySQL or Amazon EC2
- Large import and export of MySQL data
- Additional traffic from your application to the MySQL database

Figure 1 shows the correlation between volume size and baseline and burst performance (IOPS). Each volume receives an initial I/O credit balance of 5.4 million I/O credits, which is enough to sustain the maximum burst performance of 3,000 IOPS for 30 minutes. Each burst I/O credit pays for one read or one write. Note that burst performance applies only to volumes under 1 TiB. Volumes larger than 1,000 GiB have a baseline performance that is greater than or equal to the maximum burst performance, and their I/O credit balance never depletes.



Note: Bursting and I/O credits are only relevant to volumes under 1,000 GiB, where burst performance exceeds baseline performance

Figure 1: gp2 volume burst characteristics by size

For information on how to calculate burst duration, refer to the I/O Credits and Burst Performance section of the Amazon EBS Volume User Guide.[8]

When using gp2 volumes, it is important to monitor your I/O credit balance when running sustained heavy IOPS on your MySQL workload, because latency will increase if you exhaust all of your I/O credits. On the other hand, if the utilization rate of your I/O credits is less than the refill rate, any unused I/O credits will be added to your I/O credit balance.

Although in practice EBS volumes very rarely exhaust their burst I/O credit balance, the following diagram shows how high, sustained IOPS utilization can impact your gp2 volume's burst I/O credits. You can monitor the burst-bucket level for your gp2 volume using the Burst Balance metric on CloudWatch. For more information about I/O metrics, see I/O Characteristics and Monitoring.[9]

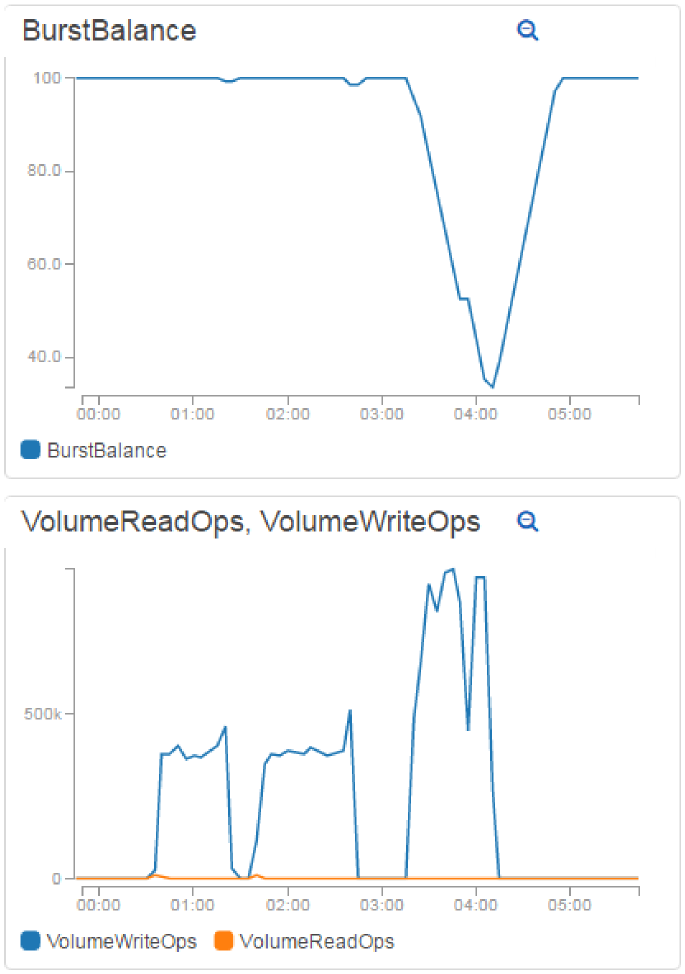


Figure 2: Impact of high, sustained IOPS utilization on gp2 volume burst I/O credits

As mentioned earlier, the EBS Elastic Volumes feature can be used to increase the size and IOPS of a volume or to switch from a General Purpose SSD (gp2) volume to a Provisioned IOPS SSD (io1) volume without interrupting the MySQL operations. This feature can be used when there is an anticipated and sustained increase in the application traffic to your MySQL database. Before committing any changes to your EBS volume, it is recommended that you plan and monitor your EBS volume usage. This will allow enough time for the EBS volume to finish the optimization state after the modification event.

Other than changing the volume type and size, you can also use RAID 0 to stripe multiple gp2 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.)

We recommend adding the same gp2 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 master/slave environment where data is already replicated in multiple slave nodes.

Provisioned IOPS SSD (io1) Volumes

EBS Provisioned IOPS SSD (io1) volumes are optimized for Online Transaction Processing (OLTP) workloads that have high transaction and throughput requirements. Generally, MySQL performs random reads and write, where IOPS plays a major part in the workload’s performance. However, there are times when sequential, larger IO read and write operations occur where high throughput is needed. This is in the event when indexes are created, databases are being restored or backed up, and logs are replicated in a master/slave configuration.

To maximize both gp2 and io1 volume throughput we recommend 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 r4.8xlarge instance provides a maximum bandwidth of 7,000 Mbps. Therefore, it can more than handle the 320 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 EC2 User Guide.[10]

Deciding whether to use an io1 volume or a gp2 volume depends primarily on your price/performance needs. While io1 is the higher cost option, these volumes consistently deliver performance within 10% of the provisioned IOPS 99.9% of the time within a given year. In contrast, gp2, the lower cost option, delivers within 10% of the expected IOPS 99% of the time. If performance consistency is more important than cost, then io1 is the right volume type to use. To determine what storage option to use, we recommend monitoring the performance of your current MySQL workload and conducting a stress test for the new databases, and comparing the cost of both io1 and gp2 volumes.

MySQL Configuration

MySQL offers a lot of parameters that you can tune to obtain optimal performance for every type of workload. In this section, we focus on the MySQL InnoDB storage engine. We also look at the MySQL parameters that you can optimize to improve performance that is 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 LRU (Least Recently Used) 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 be 80% of your server's memory;[11] 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 data set and queries can take advantage of it. For example, if you have 1 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" data set, 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 IO 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 that 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.[12]

MySQL Master/Slave (Read Replica) Configuration

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

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

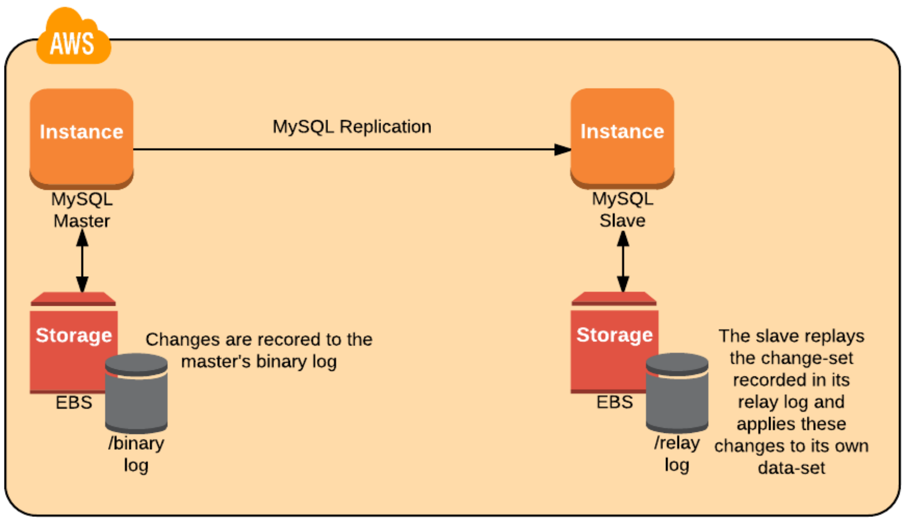

AWS
Instance
MySQL Master
MySQL Replication
Instance
MySQL Slave
Storage
EBS
Changes are recored to the master's binary log
/binary log
Storage
EBS
/relay log
The slave replays the change-set recorded in its relay log and applies these changes to its own data-set

Figure 3: Binary log file position-based replication process

Replication between master and slave 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 (master). 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 master to replay those events on the slave. The benefits of this solution include a more malleable replication topology, simplified failover, and improved management of multi-tiered replication.

Types of Replication

Prior to MySQL 5.6, replication was single threaded, with only one event executing 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. We recommend using this volume type on workloads that require heavy replication.

As mentioned in the Provisioned IOPS SSD (io1) Volumes section, RAID 0 increases the performance and throughput of the io1 volumes if 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 bin log (binary log) shipment from the master server and sequential reads of the bin log 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 acknowledgements from all members in the stripe. Only use RAID 0 if you need more throughput than that which the single EBS io1 SSD 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 Master Side
sync_binlog = 1
innodb_flush_log_at_trx_commit=1

On the Slave Slide
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 bin log 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, thus reducing IO.

The innodb_flush_log_at_trx_commit=1 is required for full ACID compliance. If you do 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 bin log and relay log from your data files as separate EBS volumes. You can use io1 for the bin log and relay log to have more predictable performance. You may also use the local SSD of the MySQL slave 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. In this section, we explore 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,[13] making backups with mysqldump[14] or mysqlpump,[15] or by making incremental backups by enabling binary logging.

If the primary database server exhibits performance issues during a backup, a replication slave 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 slave 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 leverage the larger I/O blocks. The following diagram shows a backup server leveraging the MySQL slave server to read the backup data.

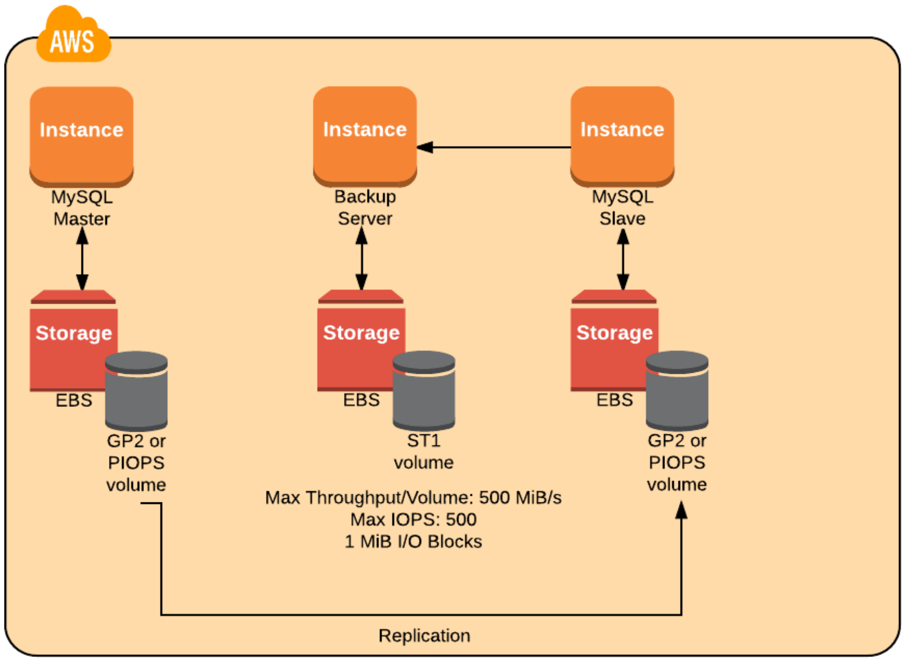


Figure 4: Using an st1 volume as a backup source

Another option is to have the MySQL slave 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 master and the slave instances can attach to the EFS file system. The slave instance can initiate a backup to the EFS file system from where the master instance can do a restore. Amazon S3 can also be used as a backup target. Amazon S3 can be leveraged 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.

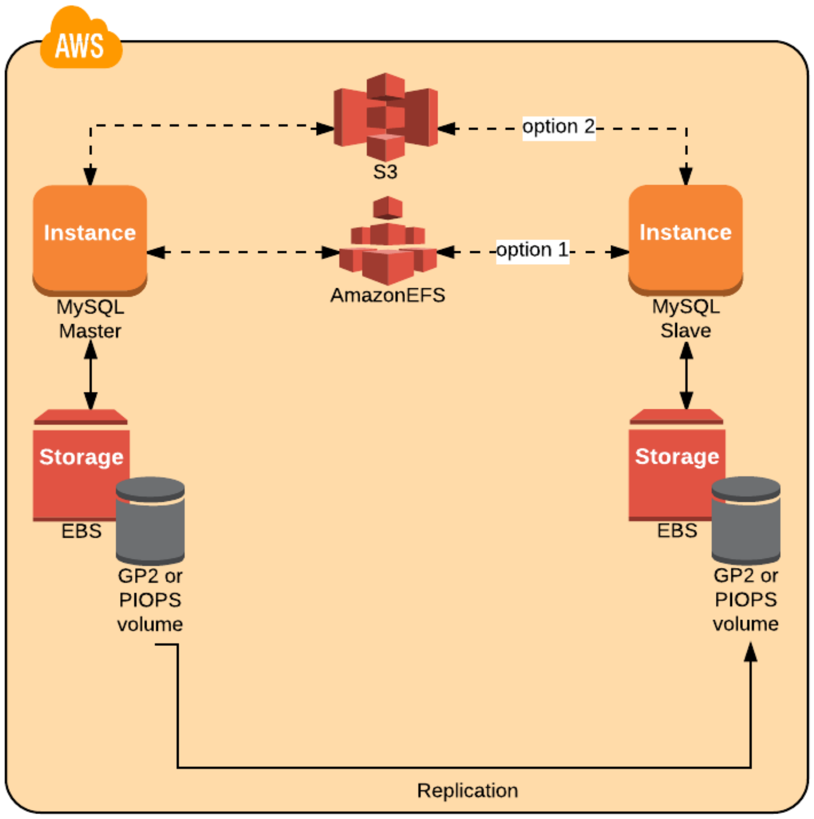


Figure 5: Using Amazon EFS or Amazon S3 as a backup target

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 contents 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.

We 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.

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, we recommend 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 are listed here:

- Tools from MySQL such as MySQL Enterprise Monitor,[16] MySQL Workbench Performance,[17] and MySQL Query Analyzer[18]
- Third party software tools and plugins
- MySQL Monitoring tools at the AWS Marketplace[19]

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. We'll show you how we diagnose these issues by looking at the EBS volume metrics collected by CloudWatch.[20]

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.[21] 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. In this section, we look at 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.

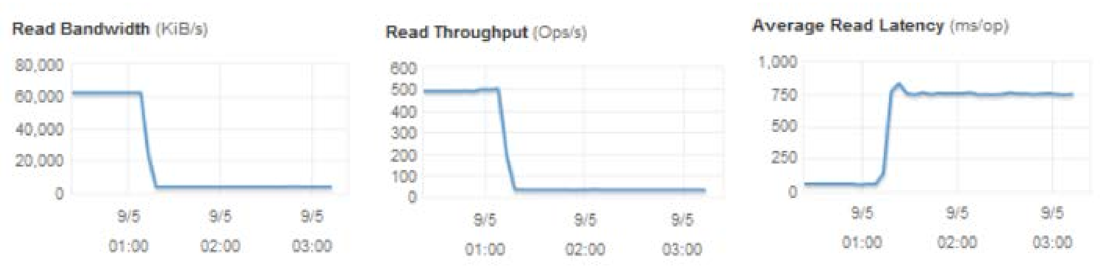


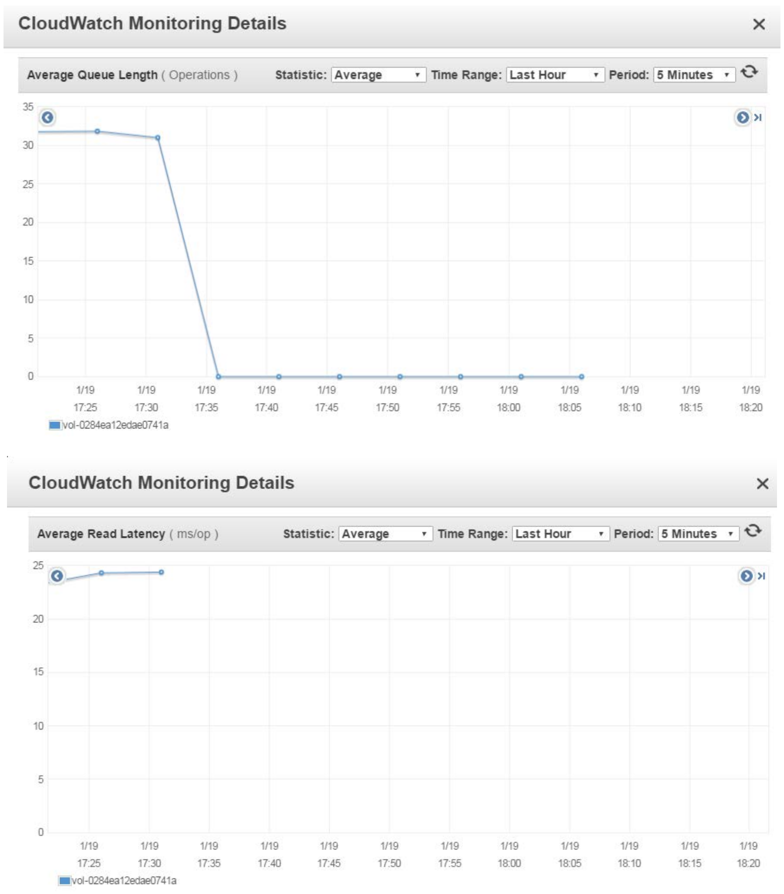
Figure 6: BurstBalance metric showing that when bandwidth and throughput are reduced, latency increases

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 1 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 will be consuming all of your 2,000 provisioned IOPS, which will result in increased latency.

Pending MySQL operations will stack up if you see the increase of the **VolumeQueueLength** without any corresponding increase in the provisioned IOPS as shown in the following screen shot.



CloudWatch Monitoring Details

Average Queue Length (Operations) Statistic: Average Time Range: Last Hour Period: 5 Minutes

CloudWatch Monitoring Details

Average Read Latency (ms/op) Statistic: Average Time Range: Last Hour Period: 5 Minutes

Figure 7: 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 slaves 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, we recommend reviewing the AWS Trusted Advisor to check alerts when an io1 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 section. If this is a brand new workload, you can do a synthetic test, which will provide you 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 like 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 several tools that you can use:

- Mysqlslap is an application that emulates client load for MySQL Server.[22]
- Sysbench is a popular open source benchmark used to test open source database management systems (DBMS).[23]

The Test Environment

To simulate the MySQL client for our Sysbench tests, we used an r3.8xlarge instance type with a 10-gigabit network interface.

Table 1: Sysbench machine specifications

Sysbench Server	
Instance type	r3.8xlarge
Memory	244 GB
CPU	32 vCPUs

All of the MySQL servers we tested on used the r4.8xlarge instance type.

Table 2: MySQL server machine specifications

MySQL Server	
Instance type	r4.8xlarge
Memory	244 GB
CPU	32 vCPUs
Storage	500 GB gp2 EBS Volume
Root volume	256 GB gp2
MySQL data volume	500 GB (gp2 or io1)
MySQL data volume 1	500 GB (gp2 or io1) RAID 0 config only
MySQL data volume 2	500 GB (gp2 or io1) RAID 0 config only

To increase performance on the Sysbench Linux client we enabled 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.

We enabled 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
```

We enabled RFS with the following shell command:

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

Tuned vs. Default Configuration Parameter Testing

We performed a Sysbench test to compare the difference between tuned MySQL and default parameter configurations (see Table 3). We used 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	134MB
table_open_cache_instances	1	16

We ran the following Sysbench read/write command:

```
./sysbench --test=tests/db/oltp.lua <connection info> --oltp-tablesize=10000000 --MySQL-
db=<dbname> --max-requests=0 –oltp-simple-ranges=0 --oltp-distinct-ranges=0 --oltp-sum-
ranges=0 --oltp-order-ranges=0 --max-time=3600 --oltp-point-selects=0 --num-threads=1024 --
rand-type=uniform run
```

Results of the Sysbench test are presented in Table 4. Under optimized conditions, the MySQL server processed approximately 16 times the number of transactions per section compared to the default configuration.

Table 4: Sysbench results

Sysbench Metrics	Default	Tuned
Queries		
Read	7,361,120	118,531,040
Write	2,943,676	47,412,407
Other	1,471,983	23,706,205
Total	11,776,779	189,649,652
Transactions	735,871(203.44 / sec)	11,853,101(3292.27 / sec)
Deadlocks	241(0.07 / sec)	1(0.00 / sec)
Read/write requests	10,304,796 (2848.85 / sec)	165,943,447 (46091.9 / sec)
Other operations	1,471,983 (406.94 / sec)	23,706,205 (6584.56 / sec)
General statistics		
Total time	3,616.77 s	3,600.2189 s
Total number of events	735,871	1,185,3101
Total time taken by event execution	1,802,967.6954 s	1,800,050.1108 s
Response time		
min	7.35 ms	4.937 ms
avg	7,617.46 ms	455.60 ms
max	276,522.87 ms	9758.44 ms
approx.95percentile	36,395.63 ms	861.75 ms

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.[24] When considering these configurations, we suggest performing a test after deployment to ensure that it will be safe for your application.

MySQL Transactions Performance Testing

Sysbench Client Setup

To simulate the more common usage scenario, we did not use EC2 Placement Groups.

We performed a Sysbench read/write test by running the following Sysbench command that generates 6 read transactions for every 4 write transactions:

```
./sysbench --test=tests/db/oltp.lua <connection info> --oltp-
tablesize=10000000 --MySQL-db=<dbname> --max-requests=0 –oltp-
simple-ranges=0 --oltp-distinct-ranges=0 --oltp-sum-ranges=0 --
oltp-order-ranges=0 --max-time=3600 --oltp-point-selects=0 --
num-threads=1024 --rand-type=uniform run
```

MySQL Server Setup

We conducted 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 (io1)
  - 500 GB SQL data drive
  - 5,000 provisioned IOPS
- MySQL Server - 3 RAID 0 Striped EBS Provisioned IOPS SSD (io1) Volumes
  - 3 x 500 GB SQL data drives striped
  - 5,000 provisioned IOPS per EBS volume*
  - Raid 0 configuration using mdadm
- Encrypted MySQL Server - EBS Provisioned IOPS SSD (io1)
  - 500 GB SQL data drive

- 5,000 provisioned IOPS on encrypted EBS volume

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

Results

The various tests of the four different server configurations yielded similar results, with each server processing approximately 6,000 Sysbench transactions per second. There was no discernable difference in performance between running gp2 or io1, although we did not run the test for long enough to measure performance consistency over time. There was also no noticeable difference in performance between the encrypted EBS and unencrypted EBS volumes.

Upon closer examination, we found that Read Throughput measured for the EBS volume in all test cases was fairly low at about 25 read Ops/s compared to the 36,000 read Ops/s as reported by Sysbench. This difference can be attributed to queries retrieving pages from the InnoDB buffer cache rather than from the EBS volume.

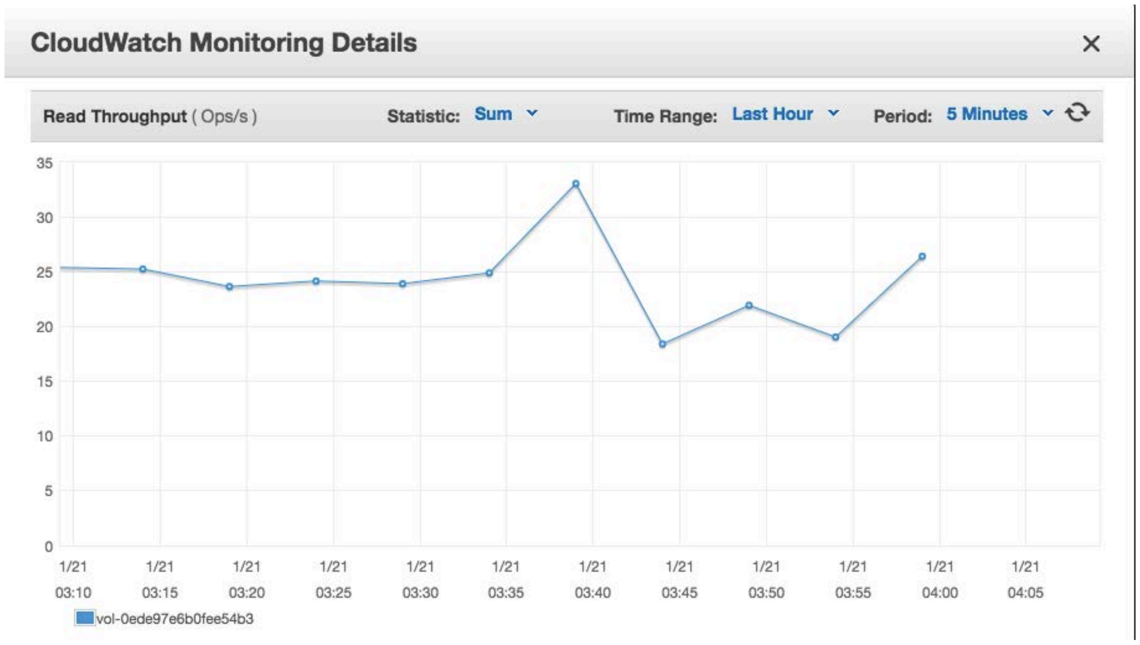


Figure 8: Read Throughput measured for EBS volume

The Write Throughput was around 1,250 write Ops/s measured on the EBS volume for all non-RAID single-volume test cases. The Write Throughput for the three striped EBS io1 volumes test case produced approximately 350 write

Ops/s, meaning that the write operations were being split amongst the three disks.

We observed that the Average Read Size in EBS was 16 KiB/op, meaning that read operations matched the MySQL page size.

The Average Write Size in EBS was approximately 22 KiB/op, which was larger than the 16 KiB/op page size of MySQL resulting in less write operations. This illustrates the benefit of capturing the metrics at both the storage and database layers so you can properly map the corresponding EBS configuration that is needed to cost-effectively support the expected performance of your MySQL database.

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, gp2 is a good option. To maximize the benefit of gp2, you need to understand and monitor the burst credit. This will help you determine whether you really need io1 or not. And if your MySQL workload needs more consistent IOPS, then you should use io1. To maximize the benefit of both io1 and gp2, we recommend using EBS-optimized EC2 instances. 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

The following individuals and organizations contributed to this document:

- Marie Yap, Enterprise Solutions Architect, Amazon Web Services
- Ricky Chang, Cloud Infrastructure Architect, Amazon Web Services

Further Reading

For additional information, see the following:

MySQL Performance Tuning 101: https://www.mysql.com/why-mysql/presentations/mysql-performance-tuning101/

MySQL 5.7 Peformance Tuning: https://dzone.com/articles/mysql-57-performance-tuning-immediately-after-inst

MySQL on EC2: Consistent Backup and Log Purging Using EBS Snapshots and CPM:
http://www.n2ws.com/blog/mysql_backup_log_purging_ebs_snapshots.html

MySQL Database Backup Methods:
http://dev.mysql.com/doc/refman/5.7/en/backup-methods.html

Notes

1 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/InstanceStorage.html

2 http://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/ebs-metricscollected.html

3 https://aws.amazon.com/ebs/details/

4 http://docs.aws.amazon.com/AmazonCloudWatch/latest/events/TakeScheduledSnapshot.html

5 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-supported-iam-actions-resources.html

6 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSEncryption.html#EBSEncryption_supported_instances

7 http://alpha-docs-aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html#current-gen-instances

8 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumeTypes.html

9 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-io-characteristics.html

10 http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/raid-config.html

11 http://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html

12 http://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html#sysvar_innodb_log_file_size

13 https://www.mysql.com/products/enterprise/backup.html

14 https://dev.mysql.com/doc/refman/5.7/en/mysqldump.html

15 https://dev.mysql.com/doc/refman/5.7/en/mysqlpump.html

16 https://www.mysql.com/products/enterprise/monitor.html

17 https://www.mysql.com/products/workbench/performance/

18 https://www.mysql.com/products/enterprise/query.html

19 https://aws.amazon.com/marketplace/search/results?searchTerms=mysql&category=2649280011&page=1

20 https://aws.amazon.com/cloudwatch/

21 http://dev.mysql.com/doc/refman/5.7/en/performance-schema.html

22 http://dev.mysql.com/doc/refman/5.7/en/mysqlslap.html

23 https://dev.mysql.com/downloads/benchmarks.html

24 https://dev.mysql.com/doc/refman/5.7/en/optimizing-innodb-diskio.html