# Managing the Performance of RDBMs in Amazon Cloud: A Deep Dive


Thanks to a relatively simple database-scaling capability and optimized storage for database workloads, you might assume that managing their performance in the cloud has become a simple task. But this is far from the truth. Database performance and management is, even in the cloud, still one of the most important responsibilities and challenging task for any DBA, and requires close attention.

In this article, we will dive deep into the different options for database performance and management, including a self-hosted database run on a specific EC2 instance or using AWS DBaaS solutions, such as RDS and Aurora. We will include several examples and performance benchmarks of self-hosted vs. RDS.

## Let's Start: Performance Benchmarks

For a database on the Amazon cloud, you can choose between a self-hosted AWS EC2 (compute) and an Amazon RDS instance. Keep in mind that if you choose RDS as a solution, you lose access to the operating system running the database.

We used the sysbench tool and a MySQL database with 50 million rows (approximately 12 GB of data) to compare the performance of a self-hosted database EC2 instance with that of an RDS instance.

| | **EC2 Instance**<br>m4.2xlarge; 1TB io1 storage; 3000 IOPS | **Amazon RDS**<br>db.m4.2xlarge; 1TB io1 storage; 3000 IOPS |
|---|---|---|
| **Transactions/sec** | 398.46 | 2136.41 |

| | | |
|---|---|---|
| **Read/Write (sec)** | 7571.15 | 40595.05 |
| **Min (ms)** | 2.54 | 8.71 |
| **Avg (ms)** | 125.47 | 23.4 |
| **Max (ms)** | 2015.02 | 679.96 |
| **95th percentile (ms)** | 508.23 | 41.22 |

As the test results show, the RDS instance performed better. Nonetheless, we would get an improvement in performance on the EC2 instance if we were to change the database configuration file to adjust to the workload — an option not available on an RDS instance.

Now let's go over each database option.

# Option 1: Self-Hosted on EC2 Instance

If you decide to go for a self-hosted database, you will have full control over your operating system, database configuration files, and infrastructure. To achieve the best performance, however, you should pay close attention to compute resources, memory, network throughput, and storage.

AWS makes it possible – with the click of a button – to alter compute and memory resources by changing the instance type for your database. This makes selecting the optimal instance type an easy task. You'll achieve ideal performance if you use memory-optimized instances, as your goal is to have as much data in memory as possible. By using these types of instances, you'll also reduce the number of direct disc reads and writes.

Due to a large number of transactions and IO, storage is usually the bottleneck for databases. When launching an EC2 instance, we can choose from two storage options — Instance Store and Elastic Block Storage. In case of stopping then restarting, the instance data is not persistent on an Instance Store volume. You can improve database performance by using it as a cache node, swap partition, or to store temporary files.

Elastic Block Storage (EBS) is a network-attached storage. Two types of EBS storage – General Purpose SSD (gp2) and Provisioned IOPS (io1) – are key for database performance improvement. Both types support manual definition of IOPS, which also enable improved database performance in the cloud. It's important to select the EBS-optimized instance because it enables dedicated bandwidth capacity between EBS-optimized instances and EBS Storage volume.

The maximum number of IOPS per gp2 volume is 10000 IOPS, and the maximum throughput is 160 MB/s. This is plenty in the beginning, but as the load and number of IO calls to a database increase, you will likely need a larger number of IOPS. To that end, and to increase the performance, you can stripe your volumes using RAID 0 technology. By striping your volumes, you increase the total number of IOPS, as well as the throughput.

If a large number of reads from your database affects performance, you can redirect the read traffic to a slave instance (read replica). You can also take advantage of placement groups. Placement groups enable low network latency and high network throughput. To provide the lowest latency, and the highest packet-per-second network performance for your placement group, choose an instance type that supports enhanced networking. It's important to use private IP addresses for communication between nodes, so you stay on an internal network.

# Option 2: RDS

Amazon RDS is a managed database service that takes care of the underlying hardware and the operating system, as well as software updates and backups. As you don't have access to the database host operating system, your options for optimizing database performance are reduced. That's because some of the properties are predefined and cannot be changed.

You can use read replicas to enhance your database performance on RDS. Using read replicas in different regions is a very convenient feature. You can improve performance by redirecting reads to read replicas located in the region closest to the end user. To ensure a sufficient number of IOPS, you can use Provisioned IOPS Storage. It supports up to 30000 IOPS per volume.

# Option 3: Aurora

In order to improve the performance inside the Amazon cloud, you can use Amazon Aurora. Aurora is a fully managed, MySQL-compatible relational database engine. You can scale an Aurora DB instance horizontally up to 15 read replicas, and with MySQL on Amazon RDS you can have up to 5 read replicas. Aurora also supports cross-region replication. To replicate the information across the different storage, Aurora only replicates FRM files and data coming from IB_LOGS, which is a significant advantage over other methods of replications.

Another advantage of Aurora is that it doesn't use a double write buffer. Writes in Aurora are organised by filling its commit queue and pushing the changes as a group commit to the storage. Aurora uses thread pool with multiplexed connections and has the ability to handle over 5000 concurrent sessions, which means that Aurora doesn't use connection pooling, thus enabling much better CPU utilization. Also, unlike MySQL where query cache was often the cause of server stalls, Aurora uses an improved version of query cache.

# Monitoring and Performance

Monitoring and benchmarking are both imperative if you want to proactively prevent performance issues in the cloud. Noting patterns in database performance equips you with the information to properly react to and address these issues. But, first you need a plan. When it comes to monitoring, you need to define goals, identify resources to track, and decide how often to do so. For benchmarking, you need a baseline for normal database performance in your environment by measuring performance at various times and under different load conditions.

## How to Monitor

You can use Amazon CloudWatch for monitoring and benchmarking your database, AWS Cloudtrail for log monitoring, and RDS for monitoring event and database log files. Third-party monitoring tools can also be used.

Thanks to the Enhanced Monitoring feature, RDS and Aurora provides real-time metrics for the OS hosting your database instance. Enhanced Monitoring gathers its metrics from an agent on the instance. Since you don't have access to the operating system of the database instance, you can't manually install monitor agents and plugins, nor can you run batch commands or powershell scripts. In addition, SQL Server Instant file initialization is not enabled on RDS and you can't enable it. This means autogrowth events need to be avoided.

### Key Metrics

Whether the performance value metrics are acceptable depends on the baseline. As previously mentioned, a database requires four main resources: CPU, memory, storage, and network. Metrics on those resources are the key factor for performance monitoring. If you're monitoring IO operations, keep an eye on VolumeReadIOPS and VolumeWriteIOPS. If you're using gp2 storage, monitoring load spikes is very important. VolumeReadBytes and VolumeWriteBytes will show you the amount of bytes during I/O operations.

We highly recommend checking Amazon documentation on RDS monitoring and Aurora DB cluster monitoring.

## Database Scalability in the Cloud

If database performance is poor, then scaling is the first, though not necessarily the best choice. If you don't have your performance metrics properly configured, and if you haven't identified the performance bottlenecks, then it's likely that database scaling won't solve the performance issues in the long run.

As we already mentioned, changing the database instance can be done by a single click of a mouse. That enables simple vertical scale up of your database. You can choose from 18 different EC2 instances to vertically scale your database. As database storage and instance type are decoupled, you can scale them independently. You can use read replicas to horizontally scale your database in cases when there's heavy read on the database. Another option is to scale out your database by sharding it into several independent pieces, each running on its own host.

## Improve and Optimize Performance

Long-running queries, poorly designed database schema, and bad indexes are usually bigger issues for database performance than bottlenecks caused by the hardware. In order to optimize and improve database performance, your first step is to discover what is causing the problems. Database testing is of paramount importance for optimal performance. You can use various third-party tools to track performance on different workloads.



You don't have to optimize the placement of tablespace files to optimize read and write operations at the physical device level, however it's still probable that storage is causing the bottleneck. Therefore, divide the data files from log files onto separate EBS volumes. GP2 volumes have the ability to burst to 3,000 IOPS per volume, independent of volume size, to meet the occasional spike in performance needs. This ability is very useful for a database, as you can predict normal IOPS needs well, but you might still find a higher spike from time to time based on specific workloads.

If you need more IOPS and throughput than GP2 can provide, then PIOPS is the right choice. If you select an IOPS number based on average IOPS used by your existing database, then you will have sufficient IOPS to service the database in most cases, but database performance will suffer at peak load.

You should query the system tables over time and identify the peak IOPS usage by the existing database. That is the best way to estimate actual IOPS needed for your database. Throughput is the measure of the transfer of bits across the network between the Amazon EC2 instance running your database and the Amazon EBS volumes.

The throughput might be directly related to the network bandwidth available to the Amazon EC2 instance and the capability of Amazon EBS to receive data. Bandwidth and throughput to the storage subsystem are crucial for good database performance. You should choose instances with higher network performance for better database performance.

# A Final Note

There is no silver bullet when it comes to database performance.

Every database requires a different approach. One of the best ways to improve database instance performance is to tune your most commonly used and most resource-intensive queries to make them less expensive to run. Proper monitoring and testing under different workloads can help you optimize and enhance database performance.