**Best Practices for Amazon RDS**

Learn best practices for working with Amazon RDS. As new best practices are identified, we will keep this section up to date.

## Amazon RDS Basic Operational Guidelines

The following are basic operational guidelines that everyone should follow when working with Amazon RDS. Note that the Amazon RDS Service Level Agreement requires that you follow these guidelines:

* Monitor your memory, CPU, and storage usage. Amazon CloudWatch can be setup to notify you when usage patterns change or when you approach the capacity of your deployment, so that you can maintain system performance and availability.
* Scale up your DB instance when you are approaching storage capacity limits. You should have some buffer in storage and memory to accommodate unforeseen increases in demand from your applications.
* Enable automatic backups and set the backup window to occur during the daily low in write IOPS.
* If your database workload requires more I/O than you have provisioned, recovery after a failover or database failure will be slow. To increase the I/O capacity of a DB instance, do any or all of the following:
  + Migrate to a DB instance class with High I/O capacity.
  + Convert from standard storage to either General Purpose or Provisioned IOPS storage, depending on how much of an increase you need. For information on available storage types, see [Amazon RDS Storage Types](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Storage.html#Concepts.Storage).

If you convert to Provisioned IOPS storage, make sure you also use a DB instance class that is optimized for Provisioned IOPS. For information on Provisioned IOPS, see [Provisioned IOPS Storage](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Storage.html#USER_PIOPS).

* + If you are already using Provisioned IOPS storage, provision additional throughput capacity.
* If your client application is caching the Domain Name Service (DNS) data of your DB instances, set a time-to-live (TTL) value of less than 30 seconds. Because the underlying IP address of a DB instance can change after a failover, caching the DNS data for an extended time can lead to connection failures if your application tries to connect to an IP address that no longer is in service.
* Test failover for your DB instance to understand how long the process takes for your use case and to ensure that the application that accesses your DB instance can automatically connect to the new DB instance after failover.

## DB Instance RAM Recommendations

An Amazon RDS performance best practice is to allocate enough RAM so that your working set resides almost completely in memory. To tell if your working set is almost all in memory, check the ReadIOPS metric (using Amazon CloudWatch) while the DB instance is under load. The value of ReadIOPS should be small and stable. If scaling up the DB instance class—to a class with more RAM—results in a dramatic drop in ReadIOPS, your working set was not almost completely in memory. Continue to scale up until ReadIOPS no longer drops dramatically after a scaling operation, or ReadIOPS is reduced to a very small amount. For information on monitoring a DB instance's metrics, see [Viewing DB Instance Metrics](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Monitoring.html#USER_Monitoring).

## Amazon RDS Security Best Practices

Use AWS IAM accounts to control access to Amazon RDS API actions, especially actions that create, modify, or delete RDS resources such as DB instances, security groups, option groups, or parameter groups, and actions that perform common administrative actions such as backing up and restoring DB instances, or configuring Provisioned IOPS storage.

* Assign an individual IAM account to each person who manages RDS resources. Do not use AWS root credentials to manage Amazon RDS resources; you should create an IAM user for everyone, including yourself.
* Grant each user the minimum set of permissions required to perform his or her duties.
* Use IAM groups to effectively manage permissions for multiple users.
* Rotate your IAM credentials regularly.

For more information about IAM, go to [AWS Identity and Access Management](http://docs.aws.amazon.com/IAM/latest/UserGuide/Welcome.html). For information on IAM best practices, go to [IAM Best Practices](http://docs.aws.amazon.com/IAM/latest/UserGuide/IAMBestPractices.html).

## Using Enhanced Monitoring to Identify Operating System Issues

Amazon RDS provides metrics in real time for the operating system (OS) that your DB instance runs on. You can view the metrics for your DB instance using the console, or consume the Enhanced Monitoring JSON output from Amazon CloudWatch Logs in a monitoring system of your choice. For more information about Enhanced Monitoring, see [Enhanced Monitoring](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_Monitoring.OS.html)

Enhanced Monitoring is available for the following database engines:

* Amazon Aurora
* MariaDB
* Microsoft SQL Server
* MySQL version 5.5 or later
* Oracle
* PostgreSQL

Enhanced monitoring is available for all DB instance classes except for db.m1.small. Enhanced Monitoring is available in all regions except for AWS GovCloud (US).

## Using Metrics to Identify Performance Issues

To identify performance issues caused by insufficient resources and other common bottlenecks, you can monitor the metrics available for your Amazon RDS DB instance.

### Viewing Performance Metrics

You should monitor performance metrics on a regular basis to see the average, maximum, and minimum values for a variety of time ranges. If you do so, you can identify when performance is degraded. You can also set Amazon CloudWatch alarms for particular metric thresholds so you are alerted if they are reached.

In order to troubleshoot performance issues, it’s important to understand the baseline performance of the system. When you set up a new DB instance and get it running with a typical workload, you should capture the average, maximum, and minimum values of all of the performance metrics at a number of different intervals (for example, one hour, 24 hours, one week, two weeks) to get an idea of what is normal. It helps to get comparisons for both peak and off-peak hours of operation. You can then use this information to identify when performance is dropping below standard levels.

**To view performance metrics**

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the left navigation pane, select **Instances**, and then select a DB instance.
3. Select **Show Monitoring**. The first eight performance metrics display. The metrics default to showing information for the current day.
4. Use the numbered buttons at top right to page through the additional metrics, or select **Show All** to see all metrics.
5. Select a performance metric to adjust the time range in order to see data for other than the current day. You can change the **Statistic**, **Time Range**, and **Period** values to adjust the information displayed. For example, to see the peak values for a metric for each day of the last two weeks, set **Statistic** to **Maximum**, **Time Range** to **Last 2 Weeks**, and **Period** to **Day**.

**Note**

Changing the **Statistic**, **Time Range**, and **Period** values changes them for all metrics. The updated values persist for the remainder of your session or until you change them again.

You can also view performance metrics using the CLI or API. For more information, see [Viewing DB Instance Metrics](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Monitoring.html#USER_Monitoring).

**To set a CloudWatch alarm**

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the left navigation pane, select **Instances**, and then select a DB instance.
3. Select **Show Monitoring**, and then select a performance metric to bring up the expanded view.
4. Select **Create Alarm**.
5. On the **Create Alarm** page, identify what email address should receive the alert by selecting a value in the **Send a notification to** box. Select **create topic** to the right of that box to create a new alarm recipient if necessary.
6. In the **Whenever** list, select the alarm statistic to set.
7. In the **of** box, select the alarm metric.
8. In the **Is** box and the unlabeled box to the right of it, set the alarm threshold, as shown following:


                                Create Alarm dialog box
                            

1. In the **For at least** box, enter the number of times that the specified threshold must be reached in order to trigger the alarm.
2. In the **consecutive period(s) of** box, select the period during which the threshold must have been reached in order to trigger the alarm.
3. In the **Name of alarm** box, enter a name for the alarm.
4. Select **Create Alarm**.

The performance metrics page appears, and you can see the new alarm in the **CloudWatch Alarms** status bar. If you don't see the status bar, refresh your page.

### Evaluating Performance Metrics

A DB instance has a number of different categories of metrics, and how to determine acceptable values depends on the metric.

**CPU**

* CPU Utilization – Percentage of computer processing capacity used.

**Memory**

* Freeable Memory – How much RAM is available on the DB instance, in megabytes. The red line in the Monitoring tab metrics is marked at 75% for CPU, Memory and Storage Metrics. If instance memory consumption frequently crosses that line, then this indcates that you should check your workload or upgrade your instance.
* Swap Usage – How much swap space is used by the DB instance, in megabytes.

**Disk space**

* Free Storage Space – How much disk space is not currently being used by the DB instance, in megabytes.

**Input/output operations**

* Read IOPS, Write IOPS – The average number of disk read or write operations per second.
* Read Latency, Write Latency – The average time for a read or write operation in milliseconds.
* Read Throughput, Write Throughput – The average number of megabytes read from or written to disk per second.
* Queue Depth – The number of I/O operations that are waiting to be written to or read from disk.

**Network traffic**

* Network Receive Throughput, Network Transmit Throughput – The rate of network traffic to and from the DB instance in megabytes per second.

**Database connections**

* DB Connections – The number of client sessions that are connected to the DB instance.

For more detailed individual descriptions of the performance metrics available, see [Amazon RDS Dimensions and Metrics](http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/rds-metricscollected.html).

Generally speaking, acceptable values for performance metrics depend on what your baseline looks like and what your application is doing. Investigate consistent or trending variances from your baseline. Advice about specific types of metrics follows:

* **High CPU or RAM consumption –** High values for CPU or RAM consumption might be appropriate, provided that they are in keeping with your goals for your application (like throughput or concurrency) and are expected.
* **Disk space consumption –**Investigate disk space consumption if space used is consistently at or above 85 percent of the total disk space. See if it is possible to delete data from the instance or archive data to a different system to free up space.
* **Network traffic –** For network traffic, talk with your system administrator to understand what expected throughput is for your domain network and Internet connection. Investigate network traffic if throughput is consistently lower than expected.
* **Database connections –** Consider constraining database connections if you see high numbers of user connections in conjunction with decreases in instance performance and response time. The best number of user connections for your DB instance will vary based on your instance class and the complexity of the operations being performed. You can determine the number of database connections by associating your DB instance with a parameter group where the User Connectionsparameter is set to other than 0 (unlimited). You can either use an existing parameter group or create a new one. For more information, see [Working with DB Parameter Groups](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithParamGroups.html).
* **IOPS metrics –** The expected values for IOPS metrics depend on disk specification and server configuration, so use your baseline to know what is typical. Investigate if values are consistently different than your baseline. For best IOPS performance, make sure your typical working set will fit into memory to minimize read and write operations.

For issues with any performance metrics, one of the first things you can do to improve performance is tune the most used and most expensive queries to see if that lowers the pressure on system resources. For more information, see [Tuning Queries](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_BestPractices.html#CHAP_BestPractices.TuningQueries)

If your queries are tuned and an issue persists, consider upgrading your Amazon RDS [DB Instance Class](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.DBInstanceClass.html) to one with more of the resource (CPU, RAM, disk space, network bandwidth, I/O capacity) that is related to the issue you are experiencing.

### Tuning Queries

One of the best ways to improve DB instance performance is to tune your most commonly used and most resource-intensive queries to make them less expensive to run.

MySQL Query Tuning

Go to [Optimizing SELECT Statements](http://dev.mysql.com/doc/refman/5.5/en/select-optimization.html) in the MySQL documentation for more information on writing queries for better performance. You can also go to [MySQL Performance Tuning and Optimization Resources](http://www.mysql.com/why-mysql/performance/) for additional query tuning resources.

Oracle Query Tuning

Go to the [Database SQL Tuning Guide](https://docs.oracle.com/database/121/TGSQL/toc.htm) in the Oracle documentation for more information on writing and analyzing queries for better performance.

SQL Server Query Tuning

Go to [Analyzing a Query](http://technet.microsoft.com/en-us/library/ms191227.aspx) in the SQL Server documentation to improve queries for SQL Server DB instances. You can also use the execution-, index- and I/O-related data management views (DMVs) described in the [Dynamic Management Views and Functions](http://msdn.microsoft.com/en-us/library/ms188754%28v=sql.110%29.aspx) documentation to troubleshoot SQL Server query issues.

A common aspect of query tuning is creating effective indexes. You can use the [Database Engine Tuning Advisor](http://msdn.microsoft.com/en-us/library/hh231122%28v=sql.110%29.aspx) to get potential index improvements for your DB instance. For more information, see [Analyzing Your Database Workload on an Amazon RDS DB Instance with SQL Server Tuning Advisor](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.SQLServer.CommonDBATasks.Workload.html).

PostgreSQL Query Tuning

Go to [Using EXPLAIN](http://www.postgresql.org/docs/8.2/static/using-explain.html) in the PostgreSQL documentation to learn how to analyze a query plan. You can use this information to modify a query or underlying tables in order to improve query performance. You can also go to [Controlling the Planner with Explicit JOIN Clauses](http://www.postgresql.org/docs/8.2/static/explicit-joins.html) to get tips about how to specify joins in your query for the best performance.

MariaDB Query Tuning

Go to [Query Optimizations](https://mariadb.com/kb/en/mariadb/query-optimizations/) in the MariaDB documentation for more information on writing queries for better performance.

## Best Practices for Working with Amazon Aurora

You have several different options for improving performance and stability in Amazon Aurora, depending on the database engine used by your Aurora DB cluster and DB instances. For more information about best practices with Amazon Aurora, see [Best Practices with Amazon Aurora](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.BestPractices.html).

## Best Practices for Working with MySQL Storage Engines

On a MySQL DB instance, observe the following table creation limits:

* You're limited to 10,000 tables if you are either using Provisioned IOPS storage, or using General Purpose storage and the DB instance is 200 GiB or larger in size.
* You’re limited to 1000 tables if you are either using standard storage, or using General Purpose storage and the DB instance is less than 200 GiB in size.

We recommend these limits because having large numbers of tables significantly increases database recovery time after a failover or database crash. If you need to create more tables than recommended, set the innodb\_file\_per\_table parameter to 0. For more information, seeWorking with InnoDB Tablespaces to Improve Crash Recovery Times and [Working with DB Parameter Groups](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithParamGroups.html).

For MySQL DB instances that use version 5.7 or later, you can exceed these table creation limits due to improvements in InnoDB crash recovery. However, we still recommend that you take caution due to the potential performance impact of creating very large numbers of tables.

On a MySQL DB instance, avoid tables in your database growing too large. Provisioned storage limits restrict the maximum size of a MySQL table file to 16 TB. Instead, partition your large tables so that file sizes are well under the 16 TB limit. This approach can also improve performance and recovery time. For more information, see [MySQL File Size Limits](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/MySQL.KnownIssuesAndLimitations.html#MySQL.Concepts.Limits.FileSize).

The Point-In-Time Restore and snapshot restore features of Amazon RDS for MySQL require a crash-recoverable storage engine and are supported for the InnoDB storage engine only. Although MySQL supports multiple storage engines with varying capabilities, not all of them are optimized for crash recovery and data durability. For example, the MyISAM storage engine does not support reliable crash recovery and might prevent a Point-In-Time Restore or snapshot restore from working as intended. This might result in lost or corrupt data when MySQL is restarted after a crash.

InnoDB is the recommended and supported storage engine for MySQL DB instances on Amazon RDS. InnoDB instances can also be migrated to Aurora, while MyISAM instances can't be migrated. However, MyISAM performs better than InnoDB if you require intense, full-text search capability. If you still choose to use MyISAM with Amazon RDS, following the steps outlined in [Automated Backups with Unsupported MySQL Storage Engines](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithAutomatedBackups.html#Overview.BackupDeviceRestrictions) can be helpful in certain scenarios for snapshot restore functionality.

If you want to convert existing MyISAM tables to InnoDB tables, you can use the process outlined in the [MySQL documentation](http://dev.mysql.com/doc/refman/5.0/en/converting-tables-to-innodb.html). MyISAM and InnoDB have different strengths and weaknesses, so you should fully evaluate the impact of making this switch on your applications before doing so.

In addition, Federated Storage Engine is currently not supported by Amazon RDS for MySQL.

## Best Practices for Working with MariaDB Storage Engines

The point-in-time restore and snapshot restore features of Amazon RDS for MariaDB require a crash-recoverable storage engine. Although MariaDB supports multiple storage engines with varying capabilities, not all of them are optimized for crash recovery and data durability. For example, although Aria is a crash-safe replacement for MyISAM, it might still prevent a point-in-time restore or snapshot restore from working as intended. This might result in lost or corrupt data when MariaDB is restarted after a crash. InnoDB (for version 10.2 and higher) and XtraDB (for version 10.0 and 10.1) are the recommended and supported storage engines for MariaDB DB instances on Amazon RDS. If you still choose to use Aria with Amazon RDS, following the steps outlined in [Automated Backups with Unsupported MariaDB Storage Engines](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_WorkingWithAutomatedBackups.html#Overview.BackupDeviceRestrictionsMariaDB) can be helpful in certain scenarios for snapshot restore functionality.

## Best Practices for Working with PostgreSQL

Two important areas where you can improve performance with PostgreSQL on Amazon RDS are when loading data into a DB instance and when using the PostgreSQL autovacuum feature. The following sections cover some of the practices we recommend for these areas.

### Loading Data into a PostgreSQL DB Instance

When loading data into an Amazon RDS PostgreSQL DB instance, you should modify your DB instance settings and your DB parameter group values to allow for the most efficient importing of data into your DB instance.

Modify your DB instance settings to the following:

* Disable DB instance backups (set backup\_retention to 0)
* Disable Multi-AZ

Modify your DB parameter group to include the following settings. You should test the parameter settings to find the most efficient settings for your DB instance:

* Increase the value of the maintenance\_work\_mem parameter. For more information about PostgreSQL resource consumption parameters, see the [PostgreSQL documentation](http://www.postgresql.org/docs/9.4/static/runtime-config-resource.html).
* Increase the value of the checkpoint\_segments and checkpoint\_timeout parameters to reduce the number of writes to the wal log.
* Disable the synchronous\_commit parameter (do not turn off FSYNC).
* Disable the PostgreSQL autovacuum parameter.
* Make sure none of the tables you are importing are unlogged. Data stored in unlogged tables can be lost during a failover. For more information see, [CREATE TABLE UNLOGGED](https://www.postgresql.org/docs/current/static/sql-createtable.html)

Use the pg\_dump -Fc (compressed) or pg\_restore -j (parallel) commands with these settings.

### Working with the fsync and full\_page\_writes database parameters

In PostgreSQL 9.4.1 on Amazon RDS, the fsync and full\_page\_writes database parameters are not modifiable. Disabling the fsync and full\_page\_writes database parameters can lead to data corruption, so we have enabled them for you. We recommend that customers with other 9.3 DB engine versions of PostgreSQL not disable the fsync and full\_page\_writes parameters.

### Working with the PostgreSQL Autovacuum Feature

The autovacuum feature for PostgreSQL databases is a feature that we strongly recommend you use to maintain the health of your PostgreSQL DB instance. Autovacuum automates the execution of the VACUUM and ANALYZE command; using autovacuum is required by PostgreSQL, not imposed by Amazon RDS, and its use is critical to good performance. The feature is enabled by default for all new Amazon RDS PostgreSQL DB instances, and the related configuration parameters are appropriately set by default.

Your database administrator needs to know and understand this maintenance operation. For the PostgreSQL documentation on autovacuum, see <http://www.postgresql.org/docs/current/static/routine-vacuuming.html#AUTOVACUUM>.

Autovacuum is not a “resource free” operation, but it works in the background and yields to user operations as much as possible. When enabled, autovacuum checks for tables that have had a large number of updated or deleted tuples. It also protects against loss of very old data due to [transaction ID wraparound](http://www.postgresql.org/docs/9.3/static/routine-vacuuming.html).

Autovacuum should not be thought of as a high-overhead operation that can be reduced to gain better performance. On the contrary, tables that have a high velocity of updates and deletes will quickly deteriorate over time if autovacuum is not run.

**Important**

Not running autovacuum can result in an eventual required outage to perform a much more intrusive vacuum operation. When an Amazon RDS PostgreSQL DB instance becomes unavailable because of an over conservative use of autovacuum, the PostgreSQL database will shut down to protect itself. At that point, Amazon RDS must perform a single-user–mode full vacuum directly on the DB instance , which can result in a multi-hour outage. Thus, we strongly recommend that you do not turn off autovacuum, which is enabled by default.

The autovacuum parameters determine when and how hard autovacuum works. Theautovacuum\_vacuum\_threshold and autovacuum\_vacuum\_scale\_factor parameters determine when autovacuum is run. The autovacuum\_max\_workers, autovacuum\_nap\_time, autovacuum\_cost\_limit, and autovacuum\_cost\_delay parameters determine how hard autovacuum works. For more information about autovacuum, when it runs, and what parameters are required, see the [PostgreSQL documentation](http://www.postgresql.org/docs/9.3/static/routine-vacuuming.html).

The following query shows the number of "dead" tuples in a table named table1 :

PROMPT> select relname, n\_dead\_tup, last\_vacuum, last\_autovacuum from

pg\_catalog.pg\_stat\_all\_tables

where n\_dead\_tup > 0 and relname =  ’table1' order by n\_dead\_tup desc;

The results of the query will resemble the following:

relname | n\_dead\_tup | last\_vacuum | last\_autovacuum

---------+------------+-------------+-----------------

tasks | 81430522 | |

(1 row)

## Best Practices for Working with SQL Server

Best practices for a Multi-AZ deployment with a SQL Server DB instance include the following:

* Use Amazon RDS DB events to monitor failovers. For example, you can be notified by text message or email when a DB instance fails over. For more information about Amazon RDS events, see [Using Amazon RDS Event Notification](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_Events.html).
* If your application caches DNS values, set time to live (TTL) to less than 30 seconds. Setting TTL as so is a good practice in case there is a failover, where the IP address might change and the cached value might no longer be in service.
* We recommend that you do not enable the following modes because they turn off transaction logging, which is required for Multi-AZ:
  + Simple recover mode
  + Offline mode
  + Read-only mode
* Test to determine how long it takes for your DB instance to failover. Failover time can vary due to the type of database, the instance class, and the storage type you use. You should also test your application's ability to continue working if a failover occurs.
* To shorten failover time, you should do the following:
  + Ensure that you have sufficient Provisioned IOPS allocated for your workload. Inadequate I/O can lengthen failover times. Database recovery requires I/O.
  + Use smaller transactions. Database recovery relies on transactions, so if you can break up large transactions into multiple smaller transactions, your failover time should be shorter.
* Take into consideration that during a failover, there will be elevated latencies. As part of the failover process, Amazon RDS automatically replicates your data to a new standby instance. This replication means that new data is being committed to two different DB instances, so there might be some latency until the standby DB instance has caught up to the new primary DB instance.
* Deploy your applications in all Availability Zones. If an Availability Zone does go down, your applications in the other Availability Zones will still be available.

When working with a Multi-AZ deployment of SQL Server, remember that Amazon RDS mirrors all SQL Server databases on your instance. If you don't want particular databases to be mirrored, set up a separate DB instance that doesn't use Multi-AZ for those databases.

## Working with DB Parameter Groups

We recommend that you try out DB parameter group changes on a test DB instance before applying parameter group changes to your production DB instances. Improperly setting DB engine parameters in a DB parameter group can have unintended adverse effects, including degraded performance and system instability. Always exercise caution when modifying DB engine parameters and back up your DB instance before modifying a DB parameter group. For information about backing up your DB instance, see [Backing Up and Restoring Amazon RDS DB Instances](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_CommonTasks.BackupRestore.html).