aws

调整亚马逊RDS和亚马逊 Aur SQL ora 中的 Postgre 参数

AWS 规范性指导

AWS 规范性指导: 调整亚马逊RDS和亚马逊 Aur SQL ora 中的 Postgre 参数

Table of Contents

调整亚马逊 RDS 和亚马逊 Aurora 中的 PostgreSQL 参数

亚马逊 Web Services 的 Sumana Yanamandra、Ramu Jagini 和 Rohit Kapoor ()AWS

2024 年 2 月（文档历史记录）

Amazon Aurora PostgreSQL 兼容版和适用于 PostgreSQL 的亚马逊关系数据库服务（Amazon RDS）是复杂的开源关系数据库服务，提供全方位的功能。您可以使用这些服务在各种平台和应用程序上设置 PostgreSQL 数据库。

Aurora 和 Amazon RDS 提供了一种管理和操作 PostgreSQL 数据库的简化方法。它们旨在管理数据库基础架构，在您专注于应用程序开发的同时提供高可用性、持久性和可扩展性。但是，这些服务的默认配置可能不是所有工作负载的最佳配置。默认情况下，这些服务配置为以尽可能少的资源在任何地方运行，并且不会引入漏洞。调整参数可以帮助您提高性能、减少停机时间并提高数据库的整体效率。通过优化特定工作负载的参数，您可以充分利用 Amazon RDS 和 Aurora 提供的功能并最大限度地发挥其优势。

例如，您可以通过优化 Aurora 和 Amazon RDS for PostgreSQL 并配置它们的参数来提高性能。在创建数据库查询时，还应考虑性能。即使您优化了数据库设置，如果您的查询执行完整的表扫描、使用索引或者运行昂贵的联接或聚合操作，系统也会表现不佳。

本指南适用于想要调整其 PostgreSQL 数据库的内存、自动清理、日志和逻辑复制参数的数据库开发人员、数据库工程师和管理员。该指南还涵盖了特定于亚马逊 RDS for PostgreSQL 和兼容 Aurora PostgreSQL 的参数。调整这些参数可以帮助您优化数据库性能并减少特定工作负载的资源使用量，从而提高性能并节省成本。

使用数据库和数据库集群参数组

Amazon RDS 和 Aurora 可以根据您的数据库实例大小自动为某些设置确定最合适的参数值。它们还支持参数自定义，以便通过数据库实例和集群的参数组进行性能优化。

您可以使用数据库和数据库集群参数组来修改控制数据库引擎行为的各个方面（例如内存使用情况、磁盘 I/O、网络和锁定）的参数。通过调整这些参数，您可以针对您的特定工作负载优化数据库引擎并提高性能。

您可以使用、 AWS Command Line Interface (AWS CLI) 或 Amazon RDS API 创建和配置数据库和数据库集群参数组。 AWS Management Console本指南假设您使用的是 AWS CLI. 有关控制台和 API 的说明，请参阅 Amazon RDS 文档中的使用数据库参数组和使用数据库集群参数组。

> **Important**
>
> 要使用本指南中提供的 AWS CLI 命令，必须先安装和配置 AWS CLI。

要创建和配置数据库参数组，请执行以下操作：

```
# Create a new DB parameter group
aws rds create-db-parameter-group \
  --db-parameter-group-name mydbparamgroup \
  --db-parameter-group-family postgres13 \
  --description "My DB Parameter Group"

# Modify a parameter on the DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <param group name> \
  --parameters "ParameterName=max_connections<parameter-
name>,ParameterValue=<value>,ApplyMethod=immediate"

# Verify DB parameters
aws rds describe-db-parameters \
  --db-parameter-group-name aurora-instance-1
```

要创建和配置数据库集群参数组，请执行以下操作：

```
# Create a new DB cluster parameter group
aws rds create-db-cluster-parameter-group \
```

```
  --db-cluster-parameter-group-name myparametergroup \
  --db-parameter-group-family postgres12 \
  --description "My new parameter group"

# Modify a parameter on the DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name aws-guide-cluster \
  --parameters "ParameterName=<parameter-name>,ParameterValue=,ApplyMethod=immediate"

# Allocate the new DB cluster parameter to your cluster
aws rds modify-db-cluster  \
  --db-cluster-identifier  \
  --db-cluster-parameter-group-name=-cluster

# Verify cluster parameters
aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name=-cluster
```

> **Note**
>
> Aurora 和 Amazon RDS 提供了一个默认参数组，其中包含无法更改的预配置值。

参数组可以设置为静态或动态。无论是否启用该`ApplyMethod=immediate`选项，都会立即应用动态参数。静态参数需要手动重启才能生效。

调整内存参数

调整内存参数是优化与 Amazon RDS 和 Aurora PostgreSQL 兼容的数据库性能的一项基本任务。为各种数据库操作（例如运行查询、排序、索引和缓存）正确分配内存可以显著提高数据库性能。本节介绍兼容 Amazon RDS 和 Aurora PostgreSQL 的一些最关键的内存参数，包括它们的默认值、计算适当值的公式以及如何更改它们。有关参数的完整列表，请参阅亚马逊 RDS 文档中的使用适用于 PostgreSQL 的 RDS 数据库实例上的参数和 Aurora 文档中的 A mazon Aurora PostgreS Q L 参数。

优化这些参数需要深入了解您的数据库工作负载以及 Aurora 或 Amazon RDS 数据库实例上的可用资源。系统性能受两大类参数的影响：重要参数和偶然参数。

重要参数是不可或缺的参数，对系统的性能产生重大而直接的影响，对于实现最佳结果至关重要。

- 共享缓冲区
- 临时缓冲区
- 有效缓存大小
- work_mem
- maintenance_work_mem

或有参数是特定于方案和业务的参数。它们取决于其他因素，在支持重要参数和最大限度地提高整体系统性能方面起着间接但关键的作用。

- 随机页面成本
- seq_page_cost
- 曲目活动查询大小
- 闲置交易会话超时
- 语句超时
- 搜索路径
- 最大连接数

以下各节将更详细地讨论这些参数。

shared_buffers

该shared_buffers参数控制 PostgreSQL 用于在内存中缓存数据的内存量。将此参数设置为适当的值有助于提高查询性能。

对于 Amazon RDS，的默认值设置shared_buffers为{DBInstanceClassMemory/32768}字节，具体取决于数据库实例的可用内存。对于Aurora{DBInstanceClassMemory/12038,-50003}，根据数据库实例的可用内存，默认值设置为。此参数的最佳值取决于多个因素，包括数据库的大小、并发连接数和可用实例内存。

AWS CLI 语法

shared_buffers对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用参数组的所有实例或集群。

```
# Modify shared_buffers on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=shared_buffers,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify shared_buffers on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters "ParameterName=shared_buffers,ParameterValue=<new-
value>,ApplyMethod=immediate"
```

类型：静态（应用更改需要重新启动）

默认值：亚马逊 RDS for PostgreSQL 中的{DBInstanceClassMemory/32768}字节，与 Aurora PostgreSQL 兼{DBInstanceClassMemory/12038,-50003}容。在大多数情况下，这个方程大约占系统内存的25%。遵循此指南，参数组中的shared_buffers设置是使用 PostgreSQL 的 8K 缓冲区默认单位而不是字节或千字节来设置的。

shared_buffers参数设置可能会对性能产生重大影响，因此我们建议您对更改进行全面测试，以确保该值适合您的工作负载。

示例

假设你有一个金融服务应用程序在亚马逊 RDS 或 Aurora 上运行 PostgreSQL 数据库。该数据库用于存储客户交易数据。它有大量的表，并且可以由大量服务器上的多个应用程序访问。应用程序遇到查询性能缓慢和 CPU 使用率过高的问题。您确定调整shared_buffers参数可能有助于提高性能。

在 Amazon RDS for PostgreSQL db.r5.xlarge 中，shared_buffers的默认值设置{DBInstanceClassMemory/32768}为可用内存的字节数（例如 3 GB）。要确定合适的值shared_buffers，请运行一系列具有不同值的测试shared_buffers，从可用内存的默认值开始，然后逐渐增加该值。对于每项测试，您都要测量数据库的查询性能和 CPU 使用率。

根据测试结果，您可以确定将的值设置为 8 GB 会shared_buffers使您的工作负载获得最佳的整体查询性能和 CPU 使用率。该值是通过测试和分析工作负载特征来确定的，这些特征包括数据库的大小、查询的数量和复杂性、并发用户的数量以及可用的系统资源。进行更改后，您的监控系统会检查数据库的性能，以确保新值适合您的工作负载。然后，您可以根据需要微调其他参数以进一步提高性能。

temp_buffers

temp_buffers是兼容 Aurora PostgreSQL 和 Amazon RDS for PostgreSQL 中的一个关键配置参数，它可能会显著影响涉及对临时表进行排序、哈希和聚合操作的工作负载的性能。此参数决定为临时缓冲区分配的内存量，这反过来又会影响此类操作的效率和速度。如果分配的内存不足temp_buffers，则系统可能不得不使用速度较慢、效率较低的方法对临时表进行排序、哈希和聚合操作，从而导致性能不理想。

AWS CLI 语法

temp_buffers对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用参数组的所有实例或集群。

```
# Modify temp_buffers on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=temp_buffers,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify temp_buffers on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=temp_buffers,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：8 MB

有关此参数的更多信息，请参阅 PostgreSQL 文档中的资源消耗。

示例

如果您的工作负载涉及对临时表的大量排序、哈希和聚合操作，则temp_buffers可能无法分配足够的内存。在这种情况下，系统可能必须对临时表执行排序操作，这会导致基于磁盘的方法变慢，而不是内存中的排序、哈希和聚合操作。这可能会导致查询性能显著降低，对于涉及大型数据集的查询尤其如此。增加的值temp_buffers可以确保有足够的内存可用于在内存中执行此类操作，从而显著提高性能。

要找到的最佳值temp_buffers，请监控您的系统性能并确定性能不佳的区域。如果您发现查询响应时间较慢或 CPU 使用率较高，请考虑进行调整temp_buffers。例如，如果您的工作负载涉及大量临时表，则增加的值temp_buffers可以帮助确保这些表存储在内存中。这可能比使用存储中的读/写 I/O 快得多。

以较小的增量尝试不同的值，并在每次更改后仔细监控系统性能。temp_buffers分析不同值对性能的影响，并根据工作负载的具体特征对设置进行微调。

effective_cache_size

该effective_cache_size参数指定 PostgreSQL 应假设可用于缓存数据的内存量。正确设置此参数可以让 PostgreSQL 更好地利用可用内存，从而提高性能。

AWS CLI 语法

effective_cache_size对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用参数组的所有实例或集群。

```
# Modify effective_cache_size on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=effective_cache_size,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify effective_cache_size on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=effective_cache_size,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：SUM(DBInstanceClassMemory/12038,-50003)KB

示例

在线学习平台拥有一个庞大的数据库，其中包含课程资料、学生数据和其他用户经常访问的内容。该应用程序在内存为 32 GB 的 Amazon RDS for db.r5.xlarge PostgreSQL 实例上运行。当用户尝试阅读经常访问的内容时，应用程序会遇到性能低下的问题。分析数据库服务器的资源使用情况后，可以确定 PostgreSQL 没有充分利用可用内存。

Amazon RDS for PostgreSQL 中的effective_cache_size参数控制服务器用于磁盘缓存的内存量。实例类的默认值设置为 SUM({DBInstanceClassMemory/12038},-50003) KBdb.r5.xlarge，但此默认值可能不适用于所有工作负载。在此示例中，数据库服务器可能存储了大量经常访问的课程资料和学生数据。增加effective_cache_size参数的值会导致内存中缓存更多的数据，从而减少所需的磁盘读取次数并提高查询性能。

运行查询时，Amazon RDS for PostgreSQL 会首先检查查询所需的数据是否已在缓存中。如果是，则可以从内存中读取数据，而不是从磁盘读取数据。如果数据不在缓存中，则必须从磁盘读取，这可能是一个缓慢的操作。

对于在线学习平台，经过测试和分析，您可能会决定effective_cache_size将其设置为 16 GB（可用内存的一半）。此值允许 PostgreSQL 更好地利用可用内存，从而减少所需的磁盘读取次数并提高查询性能。

work_mem

该work_mem参数控制查询用于排序和哈希操作的内存量。它的默认值为 4 MB。如果查询包含多个操作，则每个操作最多可使用 4 MB。增加的值work_mem可以提高需要排序或哈希处理的查询的性能，因为这些操作需要更多的内存。但是，将此参数设置得过高可能会导致内存使用过多，从而导致性能下降。

要计算的最佳值work_mem，可以使用以下公式：

```
work_mem = (available_memory / (max_connections * work_mem_fraction))
```

其中available_memory是服务器上可用的内存总量，max_connections 是允许的最大连接数，work_mem_fraction 是决定应为每个连接分配多少可用内存的分数。

AWS CLI 语法

work_mem对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用参数组的所有实例或集群。

```
# Modify work_mem on a DB parameter group
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify work_mem on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改`ApplyMethod=immediate`）

默认值：4 MB

示例

社交媒体分析工具处理大量数据，而涉及复杂排序和联接操作的查询会导致高磁盘 I/O 并溢出到磁盘。如果将的值`work_mem`从 4 MB 增加到 16 MB，PostgreSQL 可以为这些操作使用更多内存。这减少了 I/O 量并提高了查询性能。

maintenance_work_mem

该`maintenance_work_mem`参数控制维护操作（例如`VACUUMANALYZE`、和索引创建）使用的内存量。在 Amazon RDS 和 Aurora 中，此参数的默认值为 64 MB。

要计算此参数的相应值，可以使用以下公式：

```
maintenance_work_mem = (total_memory - shared_buffers) / (max_connections * 5)
```

Aurora PostgreSQL 兼容版和亚马逊 RDS for PostgreSQL 使用以下公式来设置最佳值：

```
GREATEST({DBInstanceClassMemory/63963136*1024},65536)
```

AWS CLI 语法

`maintenance_work_mem`对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用参数组的所有实例或集群。

```
# Modify maintenance_work_mem on a DB parameter group
```

```
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=maintenance_work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify maintenance_work_mem on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=maintenance_work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：64 MB

示例

您的大型应用程序使用托管在 Aurora 或 Amazon RDS 上的 PostgreSQL 数据库。您会注意到，在清理和索引编制等维护活动期间，数据库运行缓慢且无响应。您可以监控内存使用率、维护操作时间和 CPU 使用率等指标，以确定当前值maintenance_work_mem是否导致问题。

要确定的最佳值maintenance_work_mem，您可以调整参数并监控其影响。如果在维护操作期间内存使用率一直很高或操作时间长于预期，则增加内存使用率maintenance_work_mem可能会有所帮助。相反，如果维护操作期间的 CPU 使用率一直很高，则降低使用率maintenance_work_mem可能会有所帮助。通过反复调整和测试，您可以找到最佳值maintenance_work_mem，从而在内存使用率、维护操作时间和 CPU 使用率之间取得最佳平衡。

在调查期间，假设您确定默认值 64 MB maintenance_work_mem 对于您的数据库大小来说太低了。因此，维护操作需要更长的时间才能完成，导致停机时间过长，并降低应用程序的性能。要解决此问题，您可以决定调整maintenance_work_mem参数，将其从 64 MB 增加到 512 MB（您认为这是最佳值）。应用更改可以将维护操作时间缩短三分之二。例如，以前需要 30 分钟才能完成的真空操作现在可能只需 10 分钟。通过这种优化，您的数据库现在可以更有效地处理维护活动。

random_page_cost

该random_page_cost参数有助于确定执行随机页面访问的成本。Amazon RDS 和 Aurora 中的查询计划器使用此参数以及有关该表的其他统计数据来确定运行查询的最有效计划。

seq_page_cost和random_page_cost参数密切相关，通常由计划者一起使用，以比较不同访问方法的成本，并确定哪种方法最有效。因此，如果您更改其中一个参数，则还应考虑是否需要调整其他参数。

通常，查询计划器会尽量减少运行查询的成本。成本是通过使用磁盘页读取次数和值的组合来确定的random_page_cost。值越高random_page_cost往往有利于顺序扫描，而较低的值往往有利于索引扫描。较低的值也倾向于使用嵌套循环联接而不是哈希联接。

除非在random_page_cost参数组或本地会话中设置了值，否则该参数使用默认的 PostgreSQL 引擎值 (4)。您可以根据服务器和工作负载的具体特征调整此值。如果工作负载中使用的大多数索引都适合内存或 Aurora 分层缓存，则random_page_cost将的值更改为接近的值seq_page_cost是合适的。

AWS CLI 语法

random_page_cost对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用参数组的所有实例或集群。

```
# Modify random_page_cost on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=random_page_cost,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify random_page_cost on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=random_page_cost,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：4

示例

假设您的数据库将大量数据存储在一个表中，该表经常使用非索引列上的筛选器进行查询。查询需要很长时间才能完成，而且查询计划器不会选择最有效的数据访问计划。

提高性能的一种方法是减少random_page_cost参数。如果将其设置为 1，则随机页面访问的成本将比默认值低四倍。如果保留random_page_cost其默认值 4，则随机页面访问的成本将是顺序页面访问的四倍（由seq_page_cost参数确定，默认为 1.0）。但是，在这种特殊情况下，随机页面访问实际上可能要昂贵得多，具体取决于存储类型。

减小random_page_cost参数值可以使查询计划器更有可能选择基于索引的计划或使用更适合表特定特征的不同访问方法。

我们建议您在更改参数后监控查询性能，并根据需要进行调整。您还应该使用EXPLAIN语句检查查询计划器，以检查它是否选择了有效的计划。

这只是一个例子。最佳设置取决于工作负载的具体特征。此外，这只是性能调整的一个方面；您还应该考虑其他可能影响查询性能的参数和配置选项。

seq_page_cost

该seq_page_cost参数有助于确定执行顺序页面访问的成本。Amazon RDS 和 Aurora 中的查询计划器使用此参数以及有关该表的其他统计数据来确定运行查询的最有效计划。

seq_page_cost和random_page_cost 参数密切相关，通常由计划者一起使用，以比较不同访问方法的成本，并确定哪种方法最有效。因此，如果您更改其中一个参数，则还应考虑是否需要调整其他参数。

当按顺序访问表时，PostgreSQL 可以使用操作系统的文件系统缓存来提供更快的访问速度。默认情况下，设置seq_page_cost为 1.0，这假设顺序页面访问与读取单个磁盘块一样便宜。如果您的表主要是按顺序访问的，但由于受到较少 IOPS 的限制，因此磁盘访问速度很慢，则可能需要增加的值seq_page_cost以反映访问磁盘的额外成本。

更改此参数的值会影响系统中运行的所有查询，因此我们建议您使用不同的值测试查询，以确定特定用例的最佳值。

AWS CLI 语法

seq_page_cost对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用参数组的所有实例或集群。

```
# Modify seq_page_cost on a DB parameter group
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=seq_page_cost,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify seq_page_cost on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=seq_page_cost,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：1.0

示例

假设你有一个数据库，它在一个主要按顺序访问的表中存储了大量数据。该表主要用于报告，查询执行速度非常慢，而且查询计划器无法选择最有效的计划来访问数据。

提高性能的一种方法是减少seq_page_cost参数。默认值为 1.0，它假设顺序页面访问与读取单个磁盘块一样便宜。但是，在这种特定情况下，由于存储类型（取决于 I/O），顺序页面访问实际上可能比这更昂贵。如果设置为 seq_page_cost 0.5，则顺序页面访问的成本是默认值的一半。此更改会使查询计划器更有可能选择使用更适合表特定特征的顺序访问方法的计划。

我们建议您在更改参数后监控查询性能，并根据需要进行调整。您还应该使用EXPLAIN语句检查查询计划，以检查它是否选择了有效的计划。

这只是一个例子。最佳设置取决于工作负载的具体特征。此外，这只是性能调整的一个方面；您还应该考虑影响查询性能的其他参数和配置选项。

track_activity_query_size

该track_activity_query_size参数控制pg_stat_activity视图中每个活动会话记录的查询字符串的大小。默认情况下，只有查询字符串的前 1,024 字节记录在 Amazon RDS for PostgreSQL 中，4,096 字节记录在兼容 Aurora PostgreSQL 的环境中。如果要记录更长的查询，可以将此参数设置为更高的值。

AWS CLI 语法

track_activity_query_size对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用参数组的所有实例或集群。

```
# Modify track_activity_query_size on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=track_activity_query_size,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify track_activity_query_size on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=track_activity_query_size,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：静态（应用更改需要重新启动）

默认值：1,024 字节（适用于 PostgreSQL 的亚马逊 RDS）、4,096 字节（兼容 Aurora PostgreSQL）

示例

您的 Amazon RDS for PostgreSQL 数据库的查询性能很慢，您怀疑该问题可能与长时间运行的查询有关。您可以通过将较长的查询记录到pg_stat_activity视图中来进一步调查。

增加track_activity_query_size参数的值可能会导致日志记录增加，从而对数据库的性能产生影响。我们建议您在问题解决后将该参数重新设置为其默认值 1,024。

idle_in_transaction_session_timeout

该idle_in_transaction_session_timeout参数控制闲置事务在停止之前等待的时间。

在兼容 Aurora PostgreSQL 和兼容 Aurora PostgreSQL 的亚马逊 RDS 中，此参数的默认值为 86,400,000 毫秒。

AWS CLI 语法

idle_in_transaction_session_timeout对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用参数组的所有实例或集群。

```
# Modify idle_in_transaction_session_timeout on a DB parameter group
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=idle_in_transaction_session_timeout,ParameterValue=<new_value>,ApplyMethod=imme

# Modify idle_in_transaction_session_timeout on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=idle_in_transaction_session_timeout,ParameterValue=<new_value>,ApplyMethod=imme
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：86,400,000 毫秒（兼容 Aurora PostgreSQL）

示例

您有一个处理在线订单的电子商务应用程序。该应用程序使用托管在 Amazon RDS 或 Aurora 上的 PostgreSQL 数据库。每当客户下订单时，应用程序都会启动新的交易以更新库存和订单记录。

如果事务长时间处于闲置状态，则可能会阻止其他事务访问相同的记录，从而导致性能问题，甚至可能导致应用程序停机。此外，未正确停止的闲置事务可能会消耗宝贵的系统资源，例如内存和 CPU。

为避免此类问题，您可以将`idle_in_transaction_session_timeout`参数设置为对您的应用程序有意义的值。例如，您可以将其设置为 5 分钟（300 秒），这样任何闲置时间超过 5 分钟的事务都将自动停止。这有助于确保系统资源得到有效利用，并且应用程序可以在不减慢速度的情况下处理大量订单。通过为设置适当的值`idle_in_transaction_session_timeout`，您可以帮助确保您的应用程序在 Amazon RDS 或 Aurora 上以最佳方式运行。

statement_timeout

该`statement_timeout`参数设置查询在停止之前可以运行的最大时间。

AWS CLI 语法

`statement_timeout`对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用参数组的所有实例或集群。

```
# Modify statement_timeout on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=statement_timeout,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify statement_timeout on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=statement_timeout,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改`ApplyMethod=immediate`）

默认值：0 毫秒（无超时）

示例

您的 Web 应用程序允许用户在大型产品数据库中进行搜索。搜索查询有时可能需要很长时间才能完成，从而导致用户响应时间变慢。要解决此问题，您可以将`statement_timeout`参数设置为较低的值，例如 10 秒，这将强制停止任何耗时超过 10 秒的查询。

这可能看起来像是一个严厉的措施，但实际上它在提高性能方面可能非常有效。在许多情况下，长时间运行的查询是由于 SQL 查询优化不佳或索引效率低下造成的。通过设置较低的`statement_timeout`值，您可以识别这些有问题的查询并采取措施对其进行优化。

例如，假设您发现某个搜索查询一直处于超时状态。您可以使用`EXPLAIN`和之类的工具`EXPLAIN ANALYZE`来分析查询并确定任何性能瓶颈。发现问题后，您可以采取措施通过添加新索引、重写查询或使用其他搜索算法来优化查询。通过以这种方式持续分析和优化 SQL 查询，可以显著提高应用程序的性能。

search_path

该`search_path` 参数确定在 SQL 语句中在架构中搜索对象的顺序。默认值为`$user, public`，这意味着 PostgreSQL 首先在与用户名匹配的架构中搜索对象，然后在公共架构中搜索对象。

如果您有大量架构，或者需要访问特定架构中的对象，则更改`search_path`参数有助于提高性能。当你设置`search_path`为特定架构时，PostgreSQL 可以更快地找到对象，而不必搜索多个架构。

要更改 Amazon RDS 和 Aurora 中的`search_path`参数，您可以对`ROLE`级别使用以下命令：

```
ALTER ROLE <username> SET search_path = <schema>;
```

AWS CLI 语法

`search_path`对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用参数组的所有实例或集群。

```
# Modify search_path on a DB parameter group
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=search_path,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify search_path on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=search_path,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改`ApplyMethod=immediate`）

默认值：`$user, public`

示例

你有一个多租户应用程序，每个租户使用适用于 PostgreSQL 的 Amazon RDS 或兼容 Aurora PostgreSQL 的数据库，每个租户都有不同的架构，并且你需要运行一个涉及联接来自多个架构的数据的查询。

默认情况下，Amazon RDS 和 Aurora 使用搜索路径来确定给定表要使用哪个架构。搜索路径是 PostgreSQL 在不限定架构名称的情况下引用表时按顺序搜索的架构名称列表。默认情况下，Amazon RDS 和 Aurora 首先在架构中查找与当前用户同名的表，然后在公共架构中查找。

假设您要运行一个查询，该查询涉及联接来自多个名为`tenant1`、`tenant2`和`tenant3`的架构中的表。要使用租户架构，可以在查询中包含架构名称：

```
SELECT *
FROM tenant1.table1
JOIN tenant2.table2 ON tenant1.table1.id = tenant2.table2.id
JOIN tenant3.table3 ON tenant2.table2.id = tenant3.table3.id;
```

但是，更有效的方法是使用AWS CLI 语法部分中的命令更改`search_path`参数以包含租户架构。你也可以在 PostgreSQL 会话中使用`SET`以下命令：

```
SET search_path = tenant1, tenant2, tenant3, public;
```

然后，您可以在不限定架构名称的情况下编写查询：

```
SELECT *
FROM table1
JOIN table2 ON table1.id = table2.id
JOIN table3 ON table2.id = table3.id;
```

这可以使您的查询更加简洁、更易于阅读，如果您有许多查询涉及连接来自多个架构的表，它还可以简化您的应用程序代码。

max_connections

该`max_connections`参数设置您的 PostgreSQL 数据库的最大并发连接数。

AWS CLI 语法

max_connections对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用参数组的所有实例或集群。

```
# Modify max_connections on a DB parameter group
aws rds modify-db-parameter-group \
--db-parameter-group-name <parameter_group_name> \
--parameters
 "ParameterName=max_connections,ParameterValue=<new_value>,ApplyMethod=pending-reboot"

# Modify max_connections on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name <parameter_group_name> \
--parameters
 "ParameterName=max_connections,ParameterValue=<new_value>,ApplyMethod=pending-reboot"
```

类型：静态（应用更改需要重新启动）

默认值：LEAST(DBInstanceClassMemory/9531392,5000)连接

要优化在 Amazon RDS 或 Aurora max_connections 中的使用并最大限度地减少其对性能的影响，请考虑以下最佳实践：

- 根据可用的系统资源设置参数值。
- 监控连接使用情况，防止快速达到限制。
- 使用连接池减少所需的连接数。
- 使用 Amazon RDS 代理进行连接池。

max_connections在调整亚马逊 RDS for PostgreSQL 或兼容 Amazon Aurora PostgreSQL 时，请考虑可用的实例类型及其分配的资源，并关注内存和 CPU 容量。存储和 I/O 详细信息由管理 AWS，因此您可以监控一般工作负载特征和系统指标，例如FreeableMemory通过 Amazon CloudWatch 或 Amazon RDS 控制台，以确认是否有足够的内存用于连接。监视高CPUUtilization值，这可能表明需要进行调整。避免设置max_connections得太高，因为它会影响内存使用量，并可能间接影响 I/O。请记住，每个连接都会消耗内存。要找到适当的平衡，请慢慢增加max_connections，看看它如何影响您的系统。注意性能降低或 CPU 使用率较高的迹象。检查您的应用程序是否仍然运行良好。使用 Aurora 中的只读副本等功能来分配读取流量并减少主实例的负载。根据观察到的使用模式max_connections定期进行审查和调整，以确保在给定的资源限制内实现最佳数据库性能。

调整自动吸尘参数

适用于 PostgreSQL 的亚马逊 RDS 数据库和兼容 Aurora PostgreSQL 的数据库需要定期维护，即清理。Autovacuum 是一个内置的 PostgreSQL 实用程序，它可以删除过时或不必要的数据，以释放数据库中的空间。autovacuum 进程定期在后台运行该VACUUM命令。

调整自动真空设置是保持亚马逊 RDS for PostgreSQL 或兼容 Aurora PostgreSQL 的数据库系统的性能、稳定性和可用性的关键步骤。通过调整 autovacuum 参数以适应您的工作负载和数据库大小，可以优化 autovacum 进程的性能并减少其对系统资源的影响，从而改善数据库的整体运行状况。

除了调整自动清理设置外，还必须使用 Amazon RDS 和 Aurora 中提供的工具和指标来监控数据库及其组件的性能。通过监控性能指标（例如膨胀、可用空间和查询运行时间），您可以在潜在问题变成严重问题之前将其识别出来，并采取适当的措施来解决这些问题。

本节讨论以下自动真空吸尘主题和参数：

- 抽真空和分析命令
- 检查是否有膨胀
- 自动吸尘器
- Autovacuum_work_mem
- Autovacuum_naptime
- Autovacuum_max_workers
- autovacuum_vacuuum_scale_
- 自动抽真空_真空_阈值
- autovacuum_analyze_scale_f
- 自动抽真空分析阈值
- autovacuum_vacuum_cost_limi

有关 autovacuum 的更多信息，请参阅以下链接：

- 了解 Amazon RDS for PostgreSQL 环境中的自动真空（博客文章）
- 使用适用于 PostgreSQL 的亚马逊 RDS 上的 PostgreSQL 自动真空吸尘器（亚马逊 RDS 文档）
- 在亚马逊 RDS 中进行并行清理 PostgreSQL 和亚马逊 Aurora PostgreSQL（博客文章）

抽真空和分析命令

VACUUM对数据库进行垃圾收集和分析（可选）。对于大多数应用程序来说，让 autovacum 守护程序执行真空处理就足够了。但是，有些管理员可能想要修改自动真空的数据库参数，或者通过使用可以根据调度程序运行的手动管理VACUUM命令来补充或替换守护程序的活动。

VACUUM回收被死元组占用的存储空间。在标准的 PostgreSQL 操作中，当元组因更新而被删除或过时时，在执行操作之前VACUUM，它们不会从表中实际删除。因此，我们建议您VACUUM定期运行，尤其是在经常更新的表上。

在 Amazon RDS for PostgreSQL 和兼容 Aurora PostgreSQL 版本中，调整VACUUM参数尤其重要，因为与自我管理的 PostgreSQL 数据库相比，这些托管数据库服务具有不同的特征。这些差异可能会影响真空操作的性能。调整VACUUM参数对于优化资源使用并确保 vacum 操作不会对数据库系统的性能和可用性产生负面影响至关重要。

以下是你可以在兼容 Aurora PostgreSQL 和亚马逊 RDS for PostgreSQL 中使用该VACUUM命令的一些参数：

- FULL
- FREEZE
- VERBOSE
- ANALYZE
- DISABLE_PAGE_SKIPPING
- table_name
- column_name

VACUUM ANALYZE对每个选定的表执行一个VACUUMANALYZE操作，然后执行一个操作。它为执行例行维护提供了一种有效的方法。

使用不带FULL选项的VACUUM命令可以回收空间以供重复使用。它不需要对表进行排他锁，因此你可以在标准的读写操作中运行这个命令。但是，在大多数情况下，该命令不会向操作系统返回额外的空间，而是将其保留在同一个表中供重复使用。 VACUUM FULL将表的全部内容重写为没有额外空间的新磁盘文件，并允许将未使用的空间返回给操作系统。这种形式要慢得多，需要在每张桌子上都加ACCESS EXCLUSIVE锁。

有关这些参数的完整信息，请参阅 PostgreSQL 文档。

在 Aurora 和 Amazon RDS 中，autovacuum 是一个守护程序（后台实用程序）进程，它定期运行VACUUM和ANALYZE命令以清理数据库和服务器中的冗余数据。即使您依赖自动吸尘，我们也建议您查看并调整以下各节中讨论的自动吸尘设置，以确保获得最佳性能。

检查是否有膨胀

以下 SQL 查询会检查 XML 架构中的每个表，并识别浪费磁盘空间的死行（元组）：

```
SELECT schemaname || '.' || relname as tuplename,
    n_dead_tup,
    (n_dead_tup::float / n_live_tup::float) * 100 as pfrag
FROM pg_stat_user_tables
WHERE schemaname = 'xml' and n_dead_tup > 0 and n_live_tup > 0 order by pfrag desc;
```

如果此查询返回的死元组百分比很高 (pfrag)，则可以使用VACUUM命令回收空间。

要监控事务前后的数据大小，请在连接到特定数据库后在 shell 中运行以下查询：

```
SELECT pg_size_pretty(pg_relation_size('table_name'));
```

autovacuum

您可以使用autovacuum配置参数全局设置 autovacuum，也可以通过将表的autovacuum_enabled列设置false为true或针对特定pg_class表来针对每个表进行更改。

在表上启用 autovacuum 时，数据库服务器会定期扫描表中的死行和元组，并在后台将其删除，无需数据库管理员进行任何干预。这有助于保持表小、提高查询性能并减小备份的大小。

AWS CLI 语法

以下命令autovacuum为特定的数据库参数组启用。此更改适用于使用该参数组的所有实例或集群。

```
# Modify autovacuum on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters "ParameterName=autovacuum,ParameterValue=true,ApplyMethod=immediate"

# Modify autovacuum on a DB cluster parameter group
```

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters "ParameterName=autovacuum,ParameterValue=true,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：启用

你也可以使用 p sql 在特定表上禁用或启用自动清理：

```
ALTER TABLE <table_name> SET (autovacuum_enabled = true);
```

清理过多会影响性能，因此监控自动真空处理的性能以及数据库的性能并根据需要调整设置非常重要。

示例

您的 PostgreSQL 数据库有一个表，该表可以接收大量的写入和删除操作。如果没有 autovacum，该表最终将充满死行（即已标记为删除但尚未从表中实际删除的行）。这些死行会占用磁盘上的空间，减慢查询速度，并增加备份的大小。您可以在桌子上启用 autovacuum 以自动扫描死行并将其删除以缓解这些问题。

Autovacuum_work_mem

autovacuum_work_mem是一个 PostgreSQL 配置参数，用于控制自动真空进程在执行表维护任务（例如清理或分析）时使用的内存量。

在 Aurora 和 Amazon RDS 中，您可以调整的值autovacuum_work_mem以优化性能。

AWS CLI 语法

以下命令autovacuum_work_mem 为特定的数据库参数组启用。此更改适用于使用该参数组的所有实例或集群。

```
# Modify autovacuum_work_mem on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_work_mem on a DB cluster parameter group
```

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_work_mem,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：兼容 Aurora PostgreSQL 的 GREATEST({DBInstanceClassMemory/32768},131072) KB，亚马逊 RDS for PostgreSQL 中的 64 MB。但是，默认值可能会有所不同，具体取决于您使用的 Amazon RDS 或 Aurora 的特定版本。

示例

您的 Amazon RDS for PostgreSQL 数据库有一个经常更新的大表。随着时间的推移，您会注意到数据库变慢，并且您怀疑 autovacum 需要太长时间才能完成。

作为调查的一部分，您可以检查系统日志，使用pg_stat_activity视图查看当前正在运行哪些查询和进程，pg_stat_user_tables查看视图以查看每个表的统计信息，使用pg_settings视图将的值与系统上的可用内存进行比较，并监控内存使用率是否出现峰值。autovacuum_work_mem收集到这些信息后，您可以autovacuum_work_mem将其设置为工作负载所需的最佳值。为了在内存使用量和性能之间找到适当的平衡，您可以决定将其设置为系统可用内存的四分之一。更改该值后，您可以监视数据库的性能，可能会发现 autovacuum 的完成速度比以前快得多，而且数据库的总体执行速度更快。

Autovacuum_naptime

该autovacuum_naptime参数控制自动真空过程连续运行之间的时间间隔。适用于 PostgreSQL 的亚马逊 RDS 的默认值为 15 秒，兼容 Aurora PostgreSQL 的默认值为 5 秒。

例如，假设您的 Amazon RDS for PostgreSQL 数据库中的一个表可以接收大量的写入和删除操作。如果您保留默认设置，则频繁的自动真空扫描将破坏此高交易量的表。如果将此参数设置为较高的值，则连续扫描之间的间隔将更长，并且删除死行的频率将降低。

您可以使用autovacuum_naptime来管理 vacuum 进程造成的负载，尤其是在您的服务器繁忙且其 CPU 或 I/O 负载已经很高时。设置的午睡时间越长，autovacuum 运行的频率就越低，从而减少服务器的负载。但是，设置autovacuum_naptime为非常高的值可能会导致 PostgreSQL 表增长和死行积累，从而导致性能下降。我们建议您监控自动吸尘过程的性能并根据需要调整autovacuum_naptime设置。

AWS CLI 语法

autovacuum_naptime对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify autovacuum_naptime on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_naptime,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_naptime on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_naptime,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：15 秒（适用于 PostgreSQL 的亚马逊 RDS）、5 秒（兼容 Aurora PostgreSQL）

Autovacuum_max_workers

该autovacuum_max_workers参数控制 autovacum 进程可以创建的最大工作进程数。每个工作进程都负责清理或分析单个表。

例如，假设您有一个大型数据库，其中包含许多经常更新和删除的表。如果将设置autovacuum_max_workers为较低的值（例如 1），则一次只能清理一个表，并且需要更长的时间才能清理所有表。如果设置autovacuum_max_workers为高值（例如 8），则最多可以同时清理八张表。这可以加快包含许多表的数据库的清理过程。

AWS CLI 语法

autovacuum_max_workers对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify autovacuum_max_workers on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_max_workers,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_max_workers on a DB cluster parameter group
```

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_max_workers,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：静态（应用更改需要重新启动）

默认值：GREATEST(DBInstanceClassMemory/64371566592,3)工作人员

增加该autovacuum_max_workers设置会增加服务器的负载，如果您没有足够的资源，则可能会影响性能。最佳设置取决于数据库的具体要求、数据库大小和包含的表数量。我们建议您尝试不同的值并监控性能，以找到适合您的用例的最佳设置。

autovacuum_vacuum_scale_factor

autovacuum_vacuum_scale_factor配置参数控制在清理桌子时自动吸尘过程的激进程度。

真空比例因子是表中元组总数的一小部分，在 autovacum 清理表之前必须对其进行修改。默认值为 0.1（也就是说，必须修改 10% 的元组）。例如，如果一个表有 1,000,000 个元组，其中有 100,000 个元组被标记为失效或已删除，则 autovacuum 会清空该表，具体取决于作为控制因子的值。autovacuum_vacuum_threshold

该autovacuum_vacuum_scale_factor参数可帮助您控制真空过程的运行频率。如果表收到大量写入操作，则可能需要降低真空缩放系数，以便 autovacuum 更频繁地运行，并保持表较小。相反，如果表收到的写入操作很少，则可能需要提高真空缩放系数，以降低 autovacuum 的运行频率并节省资源。

AWS CLI 语法

autovacuum_vacuum_scale_factor对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify autovacuum_vacuum_scale_factor on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_vacuum_scale_factor,ParameterValue=<new_value>,ApplyMethod=immediate

# Modify autovacuum_vacuum_scale_factor on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
```

```
  --parameters
 "ParameterName=autovacuum_vacuum_scale_factor,ParameterValue=<new_value>,ApplyMethod=immediate
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：0.1

该autovacuum_vacuum_scale_factor参数与autovacuum_vacuum_threshold、autovacuum_vacuum_cost_limit, and autovacuum_naptime参数配合使用。有关此参数的更多信息，请参阅 AWS 博客文章了解 Amazon RDS for PostgreSQL 环境中的自动真空。

autovacuum_vacuum_threshold

该autovacuum_vacuum_threshold参数控制在 autovacuum 清理表之前必须对表执行的最小元组更新或删除操作次数。此设置可用于防止对这些操作率不高的表进行不必要的清理。适用于 PostgreSQL 的亚马逊 RDS 和兼容 Aurora PostgreSQL 的默认值均为 50，这是 PostgreSQL 引擎的默认值。

例如，假设你有一个包含 100,000 行且设置autovacuum_vacuum_threshold为 50 的表。如果该表只收到 49 次更新或删除，则 autovacuum 不会将其清空。如果表收到 50 个或更多更新或删除，autovacuum 会将其清空，具体取决于的值autovacuum_vacuum_scale_factor乘以表格行数作为控制因素。

将此参数设置得过高可能会导致表增长和死行累积，从而影响性能。

AWS CLI 语法

autovacuum_vacuum_threshold对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify autovacuum_vacuum_threshold on a DB parameter group
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=autovacuum_vacuum_threshold,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_vacuum_threshold on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
   --db-cluster-parameter-group-name <parameter_group_name> \
```

```
   --parameters
 "ParameterName=autovacuum_vacuum_threshold,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：50 个操作

该autovacuum_vacuum_threshold参数与autovacuum_vacuum_scale_factor、autovacuum_vacuum_cost_limit, and autovacuum_naptime参数配合使用。最佳设置取决于数据库和表大小的具体要求。

有关此参数的更多信息，请参阅 AWS 博客文章了解 Amazon RDS for PostgreSQL 环境中的自动真空。

autovacuum_analyze_scale_factor

该autovacuum_analyze_scale_factor参数控制在分析（收集有关表中数据分布的统计数据）时自动真空处理的积极程度。

autovacuum 过程使用此参数根据表中的元组数量计算阈值。如果插入、更新或删除元组的数量超过此阈值，则 autovacuum 会分析该表。适用于亚马逊 RDS for PostgreSQL 和兼容 Aurora PostgreSQL 的默认值均为 0.05（也就是说，必须修改 5% 的元组）。

例如，假设你的表有 1,000,000 个元组，而你将默认autovacuum_analyze_scale_factor值保持在 0.05。如果表收到 50,000 个或更多的更新或删除，autovacuum 会对其进行清理，具体取决于autovacuum_analyze_threshold值并将表行数相加作为控制因素。

AWS CLI 语法

autovacuum_analyze_scale_factor对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify autovacuum_analyze_scale_factor on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_analyze_scale_factor,ParameterValue=<new_value>,ApplyMethod=immediat

# Modify autovacuum_analyze_scale_factor on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
```

```
  --parameters
 "ParameterName=autovacuum_analyze_scale_factor,ParameterValue=<new_value>,ApplyMethod=immediat
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：0.05（5%）

查询计划器必须收集统计数据，以便做出明智的决定，例如如何访问数据以及如何组织数据，因此我们建议您监控自动真空处理的性能，并根据需要调整设置，以确保统计数据是最新的。

该autovacuum_analyze_scale_factor参数与autovacuum_analyze_threshold、autovacuum_analyze_cost_limit, and autovacuum_naptime参数配合使用。最佳设置取决于您的数据库和表大小以及更新频率的具体要求。有关此参数的更多信息，请参阅 AWS 博客文章了解 Amazon RDS for PostgreSQL 环境中的自动真空。

autovacuum_analyze_threshold

该autovacuum_analyze_threshold参数类似于autovacuum_vacuum_threshold。它控制在 autovacuum 分析表之前必须对表进行的最小元组插入、更新或删除次数。此设置可用于防止对这些操作率不高的表进行不必要的清理。适用于 PostgreSQL 的亚马逊 RDS 和兼容 Aurora PostgreSQL 的默认值均为 50，这是 PostgreSQL 引擎的默认值。

例如，假设你有一个包含 100,000 行的表，并且将autovacuum_analyze_threshold默认值保持在 50。如果该表仅接收 49 次插入、更新或删除，则 autovacuum 将不会对其进行分析。如果表收到 50 次或更多次插入、更新或删除，autovacuum 将对其进行分析，将的值autovacuum_analyze_scale_factor乘以表格行数作为控制因素。

AWS CLI 语法

autovacuum_analyze_threshold对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify autovacuum_analyze_threshold on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_analyze_threshold,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify autovacuum_analyze_threshold on a DB cluster parameter group
```

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_analyze_threshold,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：50 个操作

此参数与参数配合使用，因此在配置 auto autovacuum_analyze_scale_factor vacuum 时，请同时考虑这两个设置。

查询计划者必须收集统计数据，以便做出明智的决定，例如如何访问数据以及如何组织数据。设置autovacuum_analyze_threshold得太高可能会导致统计数据过时，从而导致性能不佳。我们建议您监控自动吸尘过程的性能并根据需要调整设置。

有关此参数的更多信息，请参阅 AWS 博客文章了解 Amazon RDS for PostgreSQL 环境中的自动真空。

autovacuum_vacuum_cost_limit

该autovacuum_vacuum_cost_limit参数控制自动真空工作器可以消耗的 CPU 和 I/O 资源量。

限制 autovacum 进程的资源使用有助于防止它们消耗过多 CPU 或磁盘 I/O，这可能会影响在同一系统上运行的其他查询的性能。该参数指定了成本限制，这是允许工作人员执行的工作单位，然后必须停下来检查是否仍低于该限制。例如，如果将参数设置为 2,000，则允许工作人员在暂停之前处理 2,000 个工作单元。

您可以在 PostgreSQL 会话中使用SET命令来设置autovacuum_vacuum_cost_limit参数，也可以使用命令来设置参数。 AWS CLI

AWS CLI 语法

autovacuum_vacuum_cost_limit对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify autovacuum_vacuum_cost_limit on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_vacuum_cost_limit,ParameterValue=<new_value>,ApplyMethod=immediate"
```

```
# Modify autovacuum_vacuum_cost_limit on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=autovacuum_vacuum_cost_limit,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：工作GREATEST({log(DBInstanceClassMemory/21474836480)*600},200)单位

如果将值设置autovacuum_vacuum_cost_limit得太高，autovacuum 进程可能会消耗太多资源并减慢其他查询的速度。如果将其设置得太低，autovacuum 过程可能无法回收足够的空间，这会导致表格随着时间的推移而变大。找到适合您系统的适当平衡点至关重要。

此参数仅影响自动真空处理过程，不影响手动VACUUM命令。此外，它仅适用于自动吸尘过程，VACUUM但不适用于ANALYZE自动吸尘过程。

调整日志参数

在 PostgreSQL 中调整日志参数有助于确保您收集到正确的信息，而不会生成使系统不堪重负的大型日志。

优化日志参数对于平衡日志详细信息与系统性能和磁盘使用情况至关重要。您可以自定义以下日志记录参数，以便在日志中捕获适当的详细级别、诊断问题和有效地调查事件，同时最大限度地减少对系统性能和磁盘使用率的影响。

- rds.force_autovacuum_log
- rds.force_admin_logging_level
- 日志持续时间
- log_min_duration_语句
- log_error_verbosity
- 日志语句
- log_statement_stats
- log_min_error_语句
- log_min_mins 消息
- log_temp_files
- 日志连接
- 记录_断开连接

以下各节将更详细地讨论这些参数。

> **Warning**
>
> 这些参数的最佳设置取决于贵组织的政策和合规性要求。但是，启用日志记录参数可能会导致大量日志和消息，这可能会耗尽存储空间并影响性能，对于繁忙的数据库来说尤其如此。我们建议您谨慎使用这些参数。例如，您可以决定暂时启用它们以缩小性能缓慢的 SQL 语句的问题范围，并在监视期结束后将其关闭。

rds.force_autovacuum_log

该rds.force_autovacuum_logging参数（仅在 Amazon RDS for PostgreSQL 中可用）控制是否将自动清理操作记录在服务器日志中。它的值是disabled、、debug5、debug4、debug3、debug2、debug1、info、notice、warning、、error默认值为 warning。

启用后rds.force_autovacuum_logging，将记录自动清理过程的所有操作，例如进程何时启动、何时结束以及清理了多少行。这对于调试或排除自动真空性能问题很有帮助。

AWS CLI 语法

rds.force_autovacuum_logging对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify rds.force_autovacuum_logging on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=rds.force_autovacuum_logging,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify rds.force_autovacuum_logging on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=rds.force_autovacuum_logging,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：warning

示例

您可以使用rds.force_autovacuum_logging参数在写入速率非常高的表上分析 autovacuum 的性能。例如，如果您的表每秒收到大量的写入和删除操作，并且您遇到性能低下的问题，则可以启用参数来记录每次自动真空运行的开始和结束时间，并确定清空了多少行。这可以提供有关autovacum运行频率、运行时间以及吸尘多少行的宝贵信息。然后，您可以使用此信息来微调 autovacum设置autovacuum_vacuum_scale_factor，例如autovacuum_vacuum_threshold、和，autovacuum_naptime以优化性能。

rds.force_admin_logging_level

该rds.force_admin_logging_level参数（仅在 Amazon RDS for PostgreSQL 中可用）控制由清理、分析和重新索引等管理操作生成的日志的详细程度。它接受值debug5、、debug4、debug3、debug2、debug1、loginfo、notice、warning、error、logfata和off （默认）。最佳设置取决于您的用例。例如，如果您要对问题进行故障排除，则可能需要将参数设置为调试级别。否则，您可以使用loginfo、或warning设置。

如果设置rds.force_admin_logging_level为debug1，则可以记录重新索引操作的详细信息，例如开始和结束时间、处理的行数以及在此过程中出现的任何错误或警告。这可以提供有关重新索引过程的执行情况的宝贵信息，并帮助您解决出现的任何问题。

AWS CLI 语法

rds.force_admin_logging_level对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify rds.force_admin_logging_level on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=rds.force_admin_logging_level,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify rds.force_admin_logging_level on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=rds.force_admin_logging_level,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：off

示例

您可以使用rds.force_admin_logging_level监控和分析大型数据库中多个表的管理操作性能。例如，假设你有一个包含许多表的大型数据库，你想通过定期对这些表进行清理和分析操作来优化这些表的性能。通过将rds.force_admin_logging_level参数设置为info或log，您可以记录每个操作的开始和结束时间以及受影响的表。您可以使用此信息来跟踪不同表的管理操作性能，并确定哪些表可能需要更频繁或更积极的维护。

有些日志记录级别会生成大量日志文件和消息，这些文件和消息会很快填满磁盘空间，尤其是在数据库繁忙的情况下。我们建议您谨慎使用此参数，并在监控周期结束后将其关闭。

log_duration

该log_duration参数控制是否在查询中记录每个查询的持续时间（即运行所花费的时间）。将此参数设置为时on，运行每个查询所需的时间将与查询文本一起包含在日志输出中。时间以毫秒为单位。

该log_duration参数的主要用例是帮助进行性能调整和故障排除。通过记录每个查询的持续时间，您可以识别运行时间最长的查询，然后将精力集中在优化这些查询上。这可以帮助您识别和修复性能瓶颈，并有助于提高数据库的整体性能。

AWS CLI 语法

log_duration对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify log_duration on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_duration,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_duration on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_duration,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：off

示例

如果您怀疑某一或一组查询导致性能问题，则可以使用此参数。通过启用log_duration参数并检查日志输出，您可以查看哪些查询的运行时间最长，然后采取适当的措施，例如优化索引、添加新索引或重写查询。

启用log_duration可以增加日志输出量。我们建议您仅在需要时使用它，并在标准操作期间将其关闭，以免占满存储空间或使日志难以读取。

log_min_duration_statement

该log_min_duration_statement参数控制 SQL 语句在记录之前运行的最短时间（以毫秒为单位）。

此参数可帮助您识别可能导致性能问题的长时间运行的查询。您可以将其设置为阈值（对于特定工作负载而言，运行时间被认为过长），以捕获超过该阈值的查询并识别潜在的性能瓶颈。有关示例用例，请参阅本指南后面的使用日志参数捕获绑定变量。

AWS CLI 语法

log_min_duration_statement对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify log_min_duration_statement on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_duration_statement,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_ min_duration_statement on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_duration_statement,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：1（禁用，这是 PostgreSQL 引擎的默认值）

示例

以下命令记录所有运行时间超过 100 毫秒的语句：

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name <parameter_group_name> \
   --parameters
 "ParameterName=log_min_duration_statement,ParameterValue=100,ApplyMethod=immediate"
```

log_error_verbosity

该log_error_verbosity参数控制日志输出中包含的错误和错误级别或更高级别的消息的详细程度。此参数可以取三个值之一：tersedefault、或verbose。

- terse仅包括消息文本、错误级别以及发生错误的文件和行号。
- default包括消息文本、错误级别、文件和行号以及错误上下文。
- verbose包括消息文本、错误级别、文件和行号、错误上下文以及完整的错误消息。

将参数设置为，verbose以获取有关在非生产环境中进行故障排除和调试的最详细信息。在生产环境中，您可能需要将其设置为terse左default右，它仅提供基本信息，并且不会在日志存储空间中填满太多细节。

AWS CLI 语法

log_error_verbosity对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify log_error_verbosity on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_error_verbosity,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_error_verbosity on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_error_verbosity,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：default

log_statement

该log_statement 参数控制在服务器日志中记录哪些 SQL 语句。该参数可以采用以下值之一：

- none（默认）不记录任何语句

- ddl仅记录数据定义语言 (DDL) 语句，例如和 CREATE TABLE ALTER TABLE
- mod仅记录数据修改语句INSERT，例如UPDATE、和 DELETE
- all记录所有 SQL 语句

您可以使用log_statement参数通过仅记录与您的用例相关的特定类型的语句来控制写入日志的信息量。

AWS CLI 语法

log_statement对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify log_statement on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_statement,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_statement on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_statement,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：none

示例

在生产环境中，您可能需要将设置log_statementddl为仅记录 DDL 语句并跟踪对数据库架构所做的任何更改。在开发环境中，您可能需要将参数设置为以记录所有语句，all以帮助进行调试和故障排除。有关另一个示例用例，请参阅本指南后面的使用日志参数捕获绑定变量。

启用log_statement可以增加日志输出量，因此请仅在需要时使用它并将其关闭，以免占用存储空间或使日志难以读取。

我们建议您监控系统并调整此参数的值，以便在记录的信息量与系统的存储和性能之间取得适当的平衡。

log_statement_stats

该log_statement_stats参数控制与运行 SQL 语句相关的统计信息是否与该语句一起记录。启用此参数后，诸如受影响的行数、读取和写入的磁盘块数以及运行该语句所需的时间之类的统计信息将包含在日志输出中。

您可以使用log_statement_stats参数收集有关单个语句的性能和总体工作负载的其他信息。通过记录语句统计信息，您可以识别查询性能和资源使用情况的模式，并使用这些信息来优化数据库并提高整体性能。

AWS CLI 语法

log_statement_stats对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify log_statement_stats on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_statement_stats,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_statement_stas on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_statement_stats,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：off（PostgreSQL 引擎默认）；使用 0 或 1（布尔值）在参数组中进行设置

示例

您可以使用log_statement_stats来分析特定查询的行为，查看其如何使用 CPU、内存和磁盘 I/O 等资源，并确定是否可以优化查询。您还可以使用此参数来查看是否经常读取特定表（这可能表明需要在特定列上创建索引），或者扫描表的频率是否过高。

启用log_statement_stats可以增加日志输出量，因此请仅在需要时使用它并将其关闭，以免占用存储空间或使日志难以读取。

log_min_error_statement

该log_min_error_statement参数控制将记录哪些导致错误的 SQL 语句。它的值是debug5、debug4、debug3、debug2、debug1、、info、notice、warning、error、log、fatal和panic。这些设置控制写入日志的信息量，因此您可以筛选出严重性较低的消息。您可以将此参数设置为更高的严重性级别，以减少日志输出量并更轻松地查找重要消息。

AWS CLI 语法

log_min_error_statement对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify log_min_error_statement on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_error_statement,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_min_error_statement on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_error_statement,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：error（PostgreSQL 引擎默认）

示例

在对特定问题进行故障排除并希望查看导致错误的 SQL 语句中的错误消息log_min_error_statement时，可以考虑使用。

log_min_messages

该log_min_messages参数控制写入日志的严重性级别。您可以将参数设置为debug5、、debug4、debug3debug2、debug1、info、notice、warning、error、log、fatal、或panic。这些设置控制写入日志的信息量，因此您可以筛选出严重性较低的消息。您可以将此参数设置为更高的严重性级别，以减少日志输出量并更轻松地查找重要消息。

AWS CLI 语法

log_min_messages对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify log_min_messages on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_messages,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_min_messages on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_min_messages,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：notice

示例

如果您要对特定问题进行故障排除，并且想要查看所有错误消息，则可以将此参数设置error为仅记录错误和更高级别的严重性问题。如果您有兴趣监控系统的性能，可以将此参数设置为，info以查看更详细的信息，例如每条语句的持续时间和统计信息。

设置log_min_messages为更高的严重性级别会减少日志量。我们建议您根据您的具体用例、要检查的日志大小以及您拥有的磁盘空间量来调整此参数。

log_temp_files

该log_temp_files参数控制临时文件名和大小的记录。它适用于为排序、哈希和临时查询结果等目的而创建的临时文件。启用此参数后，将在删除时为每个临时文件生成一个日志条目，包括其文件大小（以字节为单位）。您可以将此参数设置为 0（零）以全面记录所有临时文件信息，或者将此参数设置为正值以记录超过该大小的文件（如果未指定单位，则以千字节为单位）。这对于识别和解决性能瓶颈或其他与临时存储相关的问题非常有用。默认情况下，临时文件的日志记录处于禁用状态。

AWS CLI 语法

log_temp_files对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify log_temp_files on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_temp_files,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_temp_files on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_temp_files,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：-1（PostgreSQL 引擎默认）

示例

如果您怀疑系统使用了过多的临时存储空间，或者临时文件未被正确删除，则可以启用此参数。当您检查日志输出时，可以看到生成临时文件的查询或操作以及这些文件的使用情况。

某些查询或操作会创建大量临时文件，因此启用log_temp_files可能会影响系统的整体性能。

log_connections

该log_connections参数控制是否记录与数据库的连接。将此参数设置为时on，日志将包含有关每次成功连接数据库的信息，例如客户机的 IP 地址、用户名、数据库名称以及连接的日期和时间。

您可以使用log_connections参数来监控数据库连接并对其进行故障排除。您可以看到连接到数据库的用户、应用程序、终端和机器人、他们从哪里连接以及连接频率。此信息可用于识别和解决与连接相关的问题或跟踪使用模式。

AWS CLI 语法

log_connections对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify log_connections on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_connections,ParameterValue=<new_value>,ApplyMethod=immediate"
```

```
# Modify log_connections on a DB cluster parameter group
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_connections,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：off（PostgreSQL 引擎默认）

示例

如果您怀疑数据库连接过多，或者某个特定用户或 IP 地址的连接频率过高，会影响性能，则可以使用此参数。通过启用该log_connections参数并检查日志输出，您可以查看所有连接的数量和详细信息。

在启用此参数之前，请检查贵组织的政策，并考虑记录 IP 地址和用户名对安全的影响。

log_disconnections

该log_disconnections参数控制与数据库断开连接的记录。将此参数设置为时on，它会记录有关每个会话结束的信息，例如客户机的 IP 地址、用户名、数据库名称以及断开连接的日期和时间。

您可以使用log_disconnections参数来监控数据库会话终止并对其进行故障排除。您可以看到与数据库断开连接的用户、应用程序、终端和机器人、何时以及为何断开连接。例如，您可以查看意外终止情况，例如崩溃或管理员启动的断开连接。此信息可用于识别和解决与断开连接相关的问题或跟踪使用模式。

AWS CLI 语法

log_disconnections对于特定的数据库参数组，以下命令会发生变化。此更改适用于使用该参数组的所有实例或集群。

```
# Modify log_disconnections on a DB parameter group
aws rds modify-db-parameter-group \
  --db-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_disconnections,ParameterValue=<new_value>,ApplyMethod=immediate"

# Modify log_disconnections on a DB cluster parameter group
```

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name <parameter_group_name> \
  --parameters
 "ParameterName=log_disconnections,ParameterValue=<new_value>,ApplyMethod=immediate"
```

类型：动态（如果您设置了更改，则会立即应用更改ApplyMethod=immediate）

默认值：off（PostgreSQL 引擎默认）

示例

log_disconnections如果您怀疑有太多用户断开与数据库的连接，或者某个特定用户或 IP 地址断开连接的频率过高，则可以使用。通过启用该log_disconnections参数并检查日志输出，您可以查看所有断开连接的数量和详细信息，包括断开连接之前出现了谁、何时以及是否出现了任何错误。

在启用此参数之前，请检查贵组织的政策，并考虑记录 IP 地址和用户名对安全的影响。

使用日志参数捕获绑定变量

在 PostgreSQL 中捕获绑定变量的典型用例是调试和性能优化 SQL 查询。绑定变量允许您在运行查询时将数据传递给查询。通过捕获绑定变量，您可以看到传递给查询的输入数据，这可以帮助您识别数据或查询性能方面的任何问题。捕获绑定变量还可以帮助您审计输入数据并检测潜在的安全风险或恶意活动。

有几种方法可以捕获 PostgreSQL 的绑定变量。一种方法是启用debug_print_parse和debug_print_rewritten参数。这会导致 PostgreSQL 将 SQL 语句的解析和重写版本以及绑定变量发送到服务器日志。

- debug_print_parse：启用此参数后，传入查询的解析树将打印到服务器日志中。这对于理解查询的结构和任何绑定参数的值很有用。
- debug_print_rewritten：启用此参数后，传入查询的重写形式将打印到服务器日志中。这对于了解查询计划器如何解释查询以及任何绑定参数的值很有用。

您可以在 Amazon RDS 和 Aurora 中使用另外两个参数来捕获 PostgreSQL 数据库中的绑定变量：

- log_min_duration_statement：此参数设置语句在记录之前的最短持续时间，以毫秒为单位。当一条语句花费的时间超过指定的持续时间时，其绑定值将包含在日志输出中。
- log_statement：此参数控制记录哪些 SQL 语句。将此参数设置为all或绑定以在日志中包含绑定值。提高日志级别会影响性能，因此我们建议您在故障排除后恢复更改。

您也可以使用pg_stat_statements扩展程序，它为服务器运行的所有 SQL 语句提供性能统计信息，包括查询文本和绑定值。此扩展允许您使用 pgadmin 或类似工具来监控和分析查询性能。

另一种选择是使用该pg_bind_parameter_status()函数从准备好的语句中获取绑定参数的值，或者使用该pg_get_parameter_status (paramname)函数来检索特定运行时参数的状态或值。

此外，您可以使用 P gBadger 等第三方工具来分析 P ostgreSQL 日志，提取绑定变量和其他信息以供进一步分析。

调整复制参数

在 PostgreSQL 中，您可以使用逻辑复制而不是基于文件的物理复制，将数据更改从一个 PostgreSQL 数据库复制到另一个数据库。逻辑复制使用预写日志 (WAL) 来捕获更改，并支持复制选定表或整个数据库。

适用于 PostgreSQL 的 Amazon RDS 和兼容 Aurora PostgreSQL 的数据库都支持逻辑复制，因此您可以设置一个高度可用且可扩展的数据库架构，该架构可以处理来自多个来源的读取和写入流量。这些服务使用 pglogical（一种开源 PostgreSQL 扩展）来实现逻辑复制。

在 Aurora 和 Amazon RDS 中调整逻辑复制对于实现最佳性能、可扩展性和可用性非常重要。您可以调整 pglogical 扩展中的参数来管理逻辑复制的性能。例如，您可以：

- 通过增加工作进程的数量或调整其内存分配来提高复制性能。
- 通过调整源数据库和副本数据库之间的同步频率，降低复制延迟的风险。
- 通过调整工作进程的内存和 CPU 分配来优化资源的使用。
- 确保复制过程不会对源数据库的性能造成不当影响。

您可以在 Aurora 和 Amazon RDS 中使用以下参数来控制和配置逻辑复制：

- `max_replication_slots`设置可在服务器上创建的最大复制槽数。复制槽是复制连接的命名永久预留，用于将 WAL 数据发送到副本。
- `max_wal_senders`设置同时连接的 WAL 发送器进程的最大数量。WAL 发送者进程用于将 WAL 从主服务器流式传输到副本。
- `wal_sender_timeout`设置 WAL 发送者在放弃并重新连接之前等待副本响应的最长时间（以毫秒为单位）。
- `wal_receiver_timeout`设置副本在超时之前等待来自主数据库的 WAL 数据的最长时间（以毫秒为单位）。
- `log_replication_commands`，如果设置为`on`，则运行与复制相关的 SQL 语句。

启用该`rds.logical_replication`参数（将其设置为 1）时，该`wal_level`参数将设置为`logical`，这意味着对数据库所做的所有更改都以一种可以读取并应用于副本的格式写入 WAL。此设置是启用逻辑复制所必需的。此设置还允许复制`SELECT`语句。

设置`wal_level`为`logical`会增加写入 WAL 的数据量，从而增加写入磁盘的数据量，这可能会影响系统性能。我们建议您在启用逻辑复制时考虑可用磁盘空间和系统性能。

示例

您想将数据从主数据库复制到辅助数据库以进行备份和灾难恢复。但是，辅助数据库的读取操作量很大，因此您需要在不影响数据完整性的前提下确保复制过程尽可能快和高效。

Amazon RDS 和 Aurora 中逻辑复制的默认值优先考虑一致性而不是性能，因此它们可能不是此用例的最佳选择。要优化逻辑复制设置以提高速度和效率，可以按以下方式自定义参数：

- `max_replication_slots`从 10（Amazon RDS 的默认值）或 20（Aurora 的默认值）增加到 30，以适应未来的潜在增长和复制需求。
- `max_wal_senders`从 10（默认）增加到 20，以确保有足够的 WAL 发送器进程来满足复制需求。
- `wal_sender_timeout`从 30 秒（默认）减少到 15 秒，以确保更快地终止空闲的 WAL 发送器进程，从而腾出资源用于主动复制。
- `wal_receiver_timeout`从 30 秒（默认）减少到 15 秒，以确保更快地终止空闲的 WAL 接收器进程，从而腾出资源用于主动复制。
- `max_logical_replication_workers`从 4（默认）增加到 8，以确保有足够的逻辑复制工作进程来满足复制需求。

这些优化提供了更快、更高效的数据复制，同时保持了数据的完整性和安全性。

例如，如果发生灾难而主数据库不可用，则由于复制过程经过优化，辅助数据库将已经有最新的可用数据。这将使您的业务运营能够不间断地继续提供关键服务。

最佳实践

调整具有巨大工作负载的逻辑复制可能是一项复杂的任务，这取决于多种因素，包括数据集的大小、要复制的表的数量、副本的数量和可用资源。以下是一些针对庞大工作负载调整逻辑复制的常规技巧：

- 监控复制延迟。复制延迟是主服务器和备用服务器之间的时间差。监控复制延迟可以帮助您识别潜在的瓶颈并采取措施提高复制性能。您可以使用该`pg_current_wal_lsn()` 功能来检查当前的复制延迟。
- 调整 WAL 设置。该`pg_logical`扩展使用 WAL 将更改从主服务器传输到备用服务器。如果未正确调整 WAL 设置，复制可能会变得缓慢且不可靠。根据您的工作负载，请务必将`max_wal_senders`和`max_replication_slots`参数设置为足够的值。
- 制定索引策略。在主服务器上设置适当的索引有助于提高逻辑复制的性能、减少主服务器上的 I/O 以及减轻系统的负载。

- 使用并行复制。使用并行复制允许多个并行工作进程复制数据，从而有助于提高复制速度。此功能在 PostgreSQL 12 及更高版本中可用。

后续步骤

在优化了适用于 Amazon RDS for PostgreSQL 或兼容 Aurora PostgreSQL 的数据库的内存、复制、自动清理和日志参数之后，请考虑以下步骤来进一步提高数据库的性能：

- 监控您的数据库。使用内置监控工具或第三方解决方案，随时跟踪数据库性能。监控 CPU 利用率、磁盘 I/O、内存使用率和查询运行时等关键性能指标，以确定潜在的瓶颈和需要改进的领域。
- 持续调整参数。随着工作负载的变化，请继续监控和调整数据库参数以确保最佳性能。定期检查系统日志、错误消息和性能指标，以发现新的调整机会。
- 实现缓存。使用缓存来减少访问数据库的查询数量。您可以使用 Memcached 或 Redis 等工具在应用程序级别实现缓存，也可以使用 Amazon ElastiCache 为您的数据库提供内存缓存。
- 优化您的查询。设计不当的查询会严重影响数据库性能。使用`EXPLAIN`和其他查询调整工具来识别慢速查询，对其进行优化，并消除任何不必要的查询。

通过遵循这些指南，您可以优化 Aurora 或 Amazon RDS for Amazon RDS for PostgreSQL 数据库的性能，并通过提高数据库性能、提高可靠性、减少停机时间、提高安全性并节省成本来确保其满足应用程序的需求。通过优化配置参数以适应您的工作负载，您可以确保数据库高效运行并有效使用资源，从而提高性能和应用程序的响应速度。此外，正确配置的参数可以降低出现错误和漏洞的可能性，从而提高可靠性和安全性。这可以转化为成本节约，减少维护和停机时间，以及改善整体用户体验和满意度。

资源

- Amazon Aurora PostgreSQL 参数，第 1 部分：内存和查询计划管理AWS （博客文章）
- Amazon Aurora PostgreSQL 参数，第 2 部分：复制、安全和日志（博客AWS 文章）
- 亚马逊 Aurora PostgreSQL 参数，第 3 部分：优化器参数（博客文章）AWS
- 亚马逊 Aurora PostgreSQL 参数，第 4 部分：ANSI 兼容性选项（博客文章）AWS
- 使用亚马逊 Aurora Postgre AWS SQL（文档）
- 使用适用于 Postgre AWS SQL 的亚马逊 RDS（文档）
- 在 Amazon RDS 上使用 Performance Insights 监控数据库负载（AWS 文档）
- 使用 Amazon CloudWatch 指标（AWS 文档）
- pg_stats_statements（PostgreSQL 文档）

文档历史记录

下表介绍了本指南的一些重要更改。如果您希望收到有关未来更新的通知，可以订阅 RSS 源。

变更	说明	日期
更新了有关内存和自动吸尘参数的信息	更新了 random_page_cos t 参数的描述；在内存和自动真空参数的默认值中添加了缺失的单位；更新了 max_conne ctions 参数的 AWS CLI 语法。	2024 年 2 月 27 日
有关的最新信息 autovacuu m	更正了自动吸尘器的默认设置（已启用）。	2023 年 12 月 27 日
有关的最新信息 max_conne ctions	使用有关调整此参数的新指南更新了 max_connec tions 部分。	2023 年 11 月 15 日
初次发布	—	2023 年 10 月 31 日

AWS 规范性指导词汇表

以下是 AWS 规范性指导提供的策略、指南和模式中的常用术语。若要推荐词条，请使用术语表末尾的提供反馈链接。

数字

7 R

将应用程序迁移到云中的 7 种常见迁移策略。这些策略以 Gartner 于 2011 年确定的 5 R 为基础，包括以下内容：

- 重构/重新架构 - 充分利用云原生功能来提高敏捷性、性能和可扩展性，以迁移应用程序并修改其架构。这通常涉及到移植操作系统和数据库。示例：将您的本地 Oracle 数据库迁移到兼容 Amazon Aurora PostgreSQL 的版本。
- 更换平台 - 将应用程序迁移到云中，并进行一定程度的优化，以利用云功能。示例：在中将您的本地 Oracle 数据库迁移到适用于 Oracle 的亚马逊关系数据库服务 (Amazon RDS) AWS Cloud。
- 重新购买 - 转换到其他产品，通常是从传统许可转向 SaaS 模式。示例：将您的客户关系管理 (CRM) 系统迁移到 Salesforce.com。
- 更换主机（直接迁移）- 将应用程序迁移到云中，无需进行任何更改即可利用云功能。示例：在中的 EC2 实例上将您的本地 Oracle 数据库迁移到 Oracle AWS Cloud。
- 重新定位（虚拟机监控器级直接迁移）：将基础设施迁移到云中，无需购买新硬件、重写应用程序或修改现有操作。您可以将服务器从本地平台迁移到同一平台的云服务。示例：将Microsoft Hyper-V应用程序迁移到 AWS。
- 保留（重访）- 将应用程序保留在源环境中。其中可能包括需要进行重大重构的应用程序，并且您希望将工作推迟到以后，以及您希望保留的遗留应用程序，因为迁移它们没有商业上的理由。
- 停用 - 停用或删除源环境中不再需要的应用程序。

A

ABAC

请参阅基于属性的访问控制。

抽象服务

参见托管服务。

酸

参见原子性、一致性、隔离性、耐久性。

主动-主动迁移

一种数据库迁移方法，在这种方法中，源数据库和目标数据库保持同步（通过使用双向复制工具或双写操作），两个数据库都在迁移期间处理来自连接应用程序的事务。这种方法支持小批量、可控的迁移，而不需要一次性割接。与主动-被动迁移相比，它更灵活，但需要更多的工作。

主动-被动迁移

一种数据库迁移方法，在这种方法中，源数据库和目标数据库保持同步，但在将数据复制到目标数据库时，只有源数据库处理来自连接应用程序的事务。目标数据库在迁移期间不接受任何事务。

聚合函数

一个 SQL 函数，它对一组行进行操作并计算该组的单个返回值。聚合函数的示例包括SUM和MAX。

AI

参见人工智能。

AIOps

参见人工智能操作。

匿名化

永久删除数据集中个人信息的过程。匿名化可以帮助保护个人隐私。匿名化数据不再被视为个人数据。

反模式

一种用于解决反复出现的问题的常用解决方案，而在这类问题中，此解决方案适得其反、无效或不如替代方案有效。

应用程序控制

一种安全方法，仅允许使用经批准的应用程序，以帮助保护系统免受恶意软件的侵害。

应用程序组合

有关组织使用的每个应用程序的详细信息的集合，包括构建和维护该应用程序的成本及其业务价值。这些信息是产品组合发现和分析过程的关键，有助于识别需要进行迁移、现代化和优化的应用程序并确定其优先级。

人工智能（AI）

计算机科学领域致力于使用计算技术执行通常与人类相关的认知功能，例如学习、解决问题和识别模式。有关更多信息，请参阅什么是人工智能？

人工智能运营（AIOps）

使用机器学习技术解决运营问题、减少运营事故和人为干预以及提高服务质量的过程。有关如何在 AWS 迁移策略中使用 AIOps 的更多信息，请参阅运营集成指南。

非对称加密

一种加密算法，使用一对密钥，一个公钥用于加密，一个私钥用于解密。您可以共享公钥，因为它不用于解密，但对私钥的访问应受到严格限制。

原子性、一致性、隔离性、持久性（ACID）

一组软件属性，即使在出现错误、电源故障或其他问题的情况下，也能保证数据库的数据有效性和操作可靠性。

基于属性的访问权限控制（ABAC）

根据用户属性（如部门、工作角色和团队名称）创建精细访问权限的做法。有关更多信息，请参阅 AWS Identity and Access Management (I AM) 文档 AWS中的 AB AC。

权威数据源

存储主要数据版本的位置，被认为是最可靠的信息源。您可以将数据从权威数据源复制到其他位置，以便处理或修改数据，例如对数据进行匿名化、编辑或假名化。

可用区

中的一个不同位置 AWS 区域 ，不受其他可用区域故障的影响，并向同一区域中的其他可用区提供低成本、低延迟的网络连接。

AWS 云采用框架 (AWS CAF)

该框架包含指导方针和最佳实践 AWS ，可帮助组织制定高效且有效的计划，以成功迁移到云端。AWS CAF将指导分为六个重点领域，称为视角：业务、人员、治理、平台、安全和运营。业务、人员和治理角度侧重于业务技能和流程；平台、安全和运营角度侧重于技术技能和流程。例如，人员角度针对的是负责人力资源（HR）、人员配置职能和人员管理的利益相关者。从这个角度来看，AWS CAF 为人员发展、培训和沟通提供了指导，以帮助组织为成功采用云做好准备。有关更多信息，请参阅 AWS CAF 网站和 AWS CAF 白皮书。

AWS 工作负载资格框架 (AWS WQF)

一种评估数据库迁移工作负载、推荐迁移策略和提供工作估算的工具。 AWS WQF 包含在 AWS Schema Conversion Tool (AWS SCT) 中。它用来分析数据库架构和代码对象、应用程序代码、依赖关系和性能特征，并提供评测报告。

B

坏机器人

旨在破坏个人或组织或对其造成伤害的机器人。

BCP

参见业务连续性计划。

行为图

一段时间内资源行为和交互的统一交互式视图。您可以使用 Amazon Detective 的行为图来检查失败的登录尝试、可疑的 API 调用和类似的操作。有关更多信息，请参阅 Detective 文档中的行为图中的数据。

大端序系统

一个先存储最高有效字节的系统。另请参见字节顺序。

二进制分类

一种预测二进制结果（两个可能的类别之一）的过程。例如，您的 ML 模型可能需要预测诸如“该电子邮件是否为垃圾邮件？”或“这个产品是书还是汽车？”之类的问题

bloom 筛选条件

一种概率性、内存高效的数据结构，用于测试元素是否为集合的成员。

蓝/绿部署

一种部署策略，您可以创建两个独立但完全相同的环境。在一个环境中运行当前的应用程序版本（蓝色），在另一个环境中运行新的应用程序版本（绿色）。此策略可帮助您在影响最小的情况下快速回滚。

自动程序

一种通过互联网运行自动任务并模拟人类活动或互动的软件应用程序。有些机器人是有用或有益的，例如在互联网上索引信息的网络爬虫。其他一些被称为恶意机器人的机器人旨在破坏个人或组织或对其造成伤害。

僵尸网络

被恶意软件感染并受单方（称为机器人牧民或机器人操作员）控制的机器人网络。僵尸网络是最著名的扩展机器人及其影响力的机制。

分支

代码存储库的一个包含区域。在存储库中创建的第一个分支是主分支。您可以从现有分支创建新分支，然后在新分支中开发功能或修复错误。为构建功能而创建的分支通常称为功能分支。当功能可以发布时，将功能分支合并回主分支。有关更多信息，请参阅关于分支（GitHub 文档）。

破碎的玻璃通道

在特殊情况下，通过批准的流程，用户 AWS 账户 可以快速访问他们通常没有访问权限的内容。有关更多信息，请参阅 Well -Architected 指南中的 “实施破碎玻璃程序” 指示 AWS 器。

棕地策略

您环境中的现有基础设施。在为系统架构采用棕地策略时，您需要围绕当前系统和基础设施的限制来设计架构。如果您正在扩展现有基础设施，则可以将棕地策略和全新策略混合。

缓冲区缓存

存储最常访问的数据的内存区域。

业务能力

企业如何创造价值（例如，销售、客户服务或营销）。微服务架构和开发决策可以由业务能力驱动。有关更多信息，请参阅在 AWS上运行容器化微服务白皮书中的围绕业务能力进行组织部分。

业务连续性计划（BCP）

一项计划，旨在应对大规模迁移等破坏性事件对运营的潜在影响，并使企业能够快速恢复运营。

C

CAF

参见AWS 云采用框架。

金丝雀部署

向最终用户缓慢而渐进地发布版本。当你有信心时，你可以部署新版本并全部替换当前版本。

CCoE

参见云卓越中心。

CDC

参见变更数据捕获。

更改数据捕获（CDC）

跟踪数据来源（如数据库表）的更改并记录有关更改的元数据的过程。您可以将 CDC 用于各种目的，例如审计或复制目标系统中的更改以保持同步。

混沌工程

故意引入故障或破坏性事件来测试系统的弹性。您可以使用 AWS Fault Injection Service (AWS FIS) 来执行实验，对您的 AWS 工作负载施加压力并评估其响应。

CI/CD

查看持续集成和持续交付。

分类

一种有助于生成预测的分类流程。分类问题的 ML 模型预测离散值。离散值始终彼此不同。例如，一个模型可能需要评估图像中是否有汽车。

客户端加密

在目标 AWS 服务 收到数据之前，对数据进行本地加密。

云卓越中心（CCoE）

一个多学科团队，负责推动整个组织的云采用工作，包括开发云最佳实践、调动资源、制定迁移时间表、领导组织完成大规模转型。有关更多信息，请参阅 AWS Cloud 企业战略博客上的 CCoE 帖子。

云计算

通常用于远程数据存储和 IoT 设备管理的云技术。云计算通常与边缘计算技术相关。

云运营模型

在 IT 组织中，一种用于构建、完善和优化一个或多个云环境的运营模型。有关更多信息，请参阅构建您的云运营模型。

云采用阶段

组织迁移到以下阶段时通常会经历四个阶段 AWS Cloud：

- 项目 - 出于概念验证和学习目的，开展一些与云相关的项目
- 基础 - 进行基础投资以扩大云采用率（例如，创建登录区、定义 CCoE、建立运营模型）

- 迁移 - 迁移单个应用程序
- 重塑 - 优化产品和服务，在云中创新

Stephen Orban在 AWS Cloud 企业战略博客的博客文章《云优先之旅和采用阶段》中定义了这些阶段。有关它们与 AWS 迁移策略的关系的信息，请参阅迁移准备指南。

CMDB

参见配置管理数据库。

代码存储库

通过版本控制过程存储和更新源代码和其他资产（如文档、示例和脚本）的位置。常见的云存储库包括 GitHub 或 AWS CodeCommit。每个版本的代码都称为一个分支。在微服务结构中，每个存储库都专门用于一个功能。单个 CI/CD 管道可以使用多个存储库。

冷缓存

一种空的、填充不足或包含过时或不相关数据的缓冲区缓存。这会影响性能，因为数据库实例必须从主内存或磁盘读取，这比从缓冲区缓存读取要慢。

冷数据

很少访问的数据，且通常是历史数据。查询此类数据时，通常可以接受慢速查询。将这些数据转移到性能较低且成本更低的存储层或类别可以降低成本。

计算机视觉 (CV)

人工智能领域，使用机器学习来分析和提取数字图像和视频等视觉格式中的信息。例如， AWS Panorama 提供将 CV 添加到本地摄像机网络的设备，而 Amazon 则为 CV SageMaker 提供图像处理算法。

配置偏差

对于工作负载，配置会从预期状态发生变化。这可能会导致工作负载变得不合规，而且通常是渐进的，不是故意的。

配置管理数据库（CMDB）

一种存储库，用于存储和管理有关数据库及其 IT 环境的信息，包括硬件和软件组件及其配置。您通常在迁移的产品组合发现和分析阶段使用来自 CMDB 的数据。

合规性包

一系列 AWS Config 规则和补救措施，您可以汇编这些规则和补救措施，以自定义合规性和安全性检查。您可以使用 YAML 模板将一致性包作为单个实体部署在 AWS 账户 和区域或整个组织中。有关更多信息，请参阅 AWS Config 文档中的一致性包。

持续集成和持续交付（CI/CD）

自动执行软件发布过程的源代码、构建、测试、暂存和生产阶段的过程。CI/CD 通常被描述为管道。CI/CD 可以帮助您实现流程自动化、提高工作效率、改善代码质量并加快交付速度。有关更多信息，请参阅持续交付的优势。CD 也可以表示持续部署。有关更多信息，请参阅持续交付与持续部署。

CV

参见计算机视觉。

D

静态数据

网络中静止的数据，例如存储中的数据。

数据分类

根据网络中数据的关键性和敏感性对其进行识别和分类的过程。它是任何网络安全风险管理策略的关键组成部分，因为它可以帮助您确定对数据的适当保护和保留控制。数据分类是 Well-Architecte AWS d Framework 中安全支柱的一个组成部分。有关详细信息，请参阅数据分类。

数据漂移

生产数据与用来训练机器学习模型的数据之间的有意义差异，或者输入数据随时间推移的有意义变化。数据漂移可能降低机器学习模型预测的整体质量、准确性和公平性。

传输中数据

在网络中主动移动的数据，例如在网络资源之间移动的数据。

数据网格

一种架构框架，可提供分布式、去中心化的数据所有权以及集中式管理和治理。

数据最少化

仅收集并处理绝对必要数据的原则。在中进行数据最小化 AWS Cloud 可以降低隐私风险、成本和分析碳足迹。

数据边界

AWS 环境中的一组预防性防护措施，可帮助确保只有可信身份才能访问来自预期网络的可信资源。有关更多信息，请参阅在上构建数据边界。 AWS

数据预处理

将原始数据转换为 ML 模型易于解析的格式。预处理数据可能意味着删除某些列或行，并处理缺失、不一致或重复的值。

数据溯源

在数据的整个生命周期跟踪其来源和历史的过程，例如数据如何生成、传输和存储。

数据主体

正在收集和处理其数据的个人。

数据仓库

一种支持商业智能（例如分析）的数据管理系统。数据仓库通常包含大量历史数据，通常用于查询和分析。

数据库定义语言（DDL）

在数据库中创建或修改表和对象结构的语句或命令。

数据库操作语言（DML）

在数据库中修改（插入、更新和删除）信息的语句或命令。

DDL

参见数据库定义语言。

深度融合

组合多个深度学习模型进行预测。您可以使用深度融合来获得更准确的预测或估算预测中的不确定性。

深度学习

一个 ML 子字段使用多层人工神经网络来识别输入数据和感兴趣的目标变量之间的映射。

defense-in-depth

一种信息安全方法，经过深思熟虑，在整个计算机网络中分层实施一系列安全机制和控制措施，以保护网络及其中数据的机密性、完整性和可用性。当你采用这种策略时 AWS，你会在 AWS Organizations 结构的不同层面添加多个控件来帮助保护资源。例如，一种 defense-in-depth 方法可以结合多因素身份验证、网络分段和加密。

委托管理员

在中 AWS Organizations，兼容的服务可以注册 AWS 成员帐户来管理组织的帐户并管理该服务的权限。此账户被称为该服务的委托管理员。有关更多信息和兼容服务列表，请参阅 AWS Organizations 文档中使用 AWS Organizations的服务。

部署

使应用程序、新功能或代码修复在目标环境中可用的过程。部署涉及在代码库中实现更改，然后在应用程序的环境中构建和运行该代码库。

开发环境

参见环境。

侦测性控制

一种安全控制，在事件发生后进行检测、记录日志和发出警报。这些控制是第二道防线，提醒您注意绕过现有预防性控制的安全事件。有关更多信息，请参阅在 AWS上实施安全控制中的侦测性控制。

开发价值流映射 (DVSM)

用于识别对软件开发生命周期中的速度和质量产生不利影响的限制因素并确定其优先级的流程。DVSM 扩展了最初为精益生产实践设计的价值流映射流程。其重点关注在软件开发过程中创造和转移价值所需的步骤和团队。

数字孪生

真实世界系统的虚拟再现，如建筑物、工厂、工业设备或生产线。数字孪生支持预测性维护、远程监控和生产优化。

维度表

在星型架构中，一种较小的表，其中包含事实表中定量数据的数据属性。维度表属性通常是文本字段或行为类似于文本的离散数字。这些属性通常用于查询约束、筛选和结果集标注。

灾难

阻止工作负载或系统在其主要部署位置实现其业务目标的事件。这些事件可能是自然灾害、技术故障或人为操作的结果，例如无意的配置错误或恶意软件攻击。

灾难恢复 (DR)

您用来最大限度地减少灾难造成的停机时间和数据丢失的策略和流程。有关更多信息，请参阅 Well-Architected Fr ame AWS work 中的 “工作负载灾难恢复：云端 AWS 恢复”。

DML

参见数据库操作语言。

领域驱动设计

一种开发复杂软件系统的方法，通过将其组件连接到每个组件所服务的不断发展的领域或核心业务目标。Eric Evans 在其著作领域驱动设计：软件核心复杂性应对之道（Boston: Addison-Wesley Professional, 2003）中介绍了这一概念。有关如何将领域驱动设计与 strangler fig 模式结合使用的信息，请参阅使用容器和 Amazon API Gateway 逐步将原有的 Microsoft ASP.NET（ASMX）Web 服务现代化。

DR

参见灾难恢复。

漂移检测

跟踪与基准配置的偏差。例如，您可以使用 AWS CloudFormation 来检测系统资源中的偏差，也可以使用 AWS Control Tower 来检测着陆区中可能影响监管要求合规性的变化。

DVSM

参见开发价值流映射。

E

EDA

参见探索性数据分析。

边缘计算

该技术可提高位于 IoT 网络边缘的智能设备的计算能力。与云计算相比，边缘计算可以减少通信延迟并缩短响应时间。

加密

一种将人类可读的纯文本数据转换为密文的计算过程。

加密密钥

由加密算法生成的随机位的加密字符串。密钥的长度可能有所不同，而且每个密钥都设计为不可预测且唯一。

字节顺序

字节在计算机内存中的存储顺序。大端序系统先存储最高有效字节。小端序系统先存储最低有效字节。

端点

参见服务端点。

端点服务

一种可以在虚拟私有云（VPC）中托管，与其他用户共享的服务。您可以使用其他 AWS 账户 或 AWS Identity and Access Management (IAM) 委托人创建终端节点服务， AWS PrivateLink 并向其授予权限。这些账户或主体可通过创建接口 VPC 端点来私密地连接到您的端点服务。有关更多信息，请参阅 Amazon Virtual Private Cloud（Amazon VPC）文档中的创建端点服务。

企业资源规划 (ERP)

一种自动化和管理企业关键业务流程（例如会计、MES 和项目管理）的系统。

信封加密

用另一个加密密钥对加密密钥进行加密的过程。有关更多信息，请参阅 AWS Key Management Service (AWS KMS) 文档中的信封加密。

environment

正在运行的应用程序的实例。以下是云计算中常见的环境类型：

- 开发环境 — 正在运行的应用程序的实例，只有负责维护应用程序的核心团队才能使用。开发环境用于测试更改，然后再将其提升到上层环境。这类环境有时称为测试环境。
- 下层环境 — 应用程序的所有开发环境，比如用于初始构建和测试的环境。
- 生产环境 — 最终用户可以访问的正在运行的应用程序的实例。在 CI/CD 管道中，生产环境是最后一个部署环境。
- 上层环境 — 除核心开发团队以外的用户可以访问的所有环境。这可能包括生产环境、预生产环境和用户验收测试环境。

epic

在敏捷方法学中，有助于组织工作和确定优先级的功能类别。epics 提供了对需求和实施任务的总体描述。例如， AWS CAF 安全史诗包括身份和访问管理、侦探控制、基础设施安全、数据保护和事件响应。有关 AWS 迁移策略中 epics 的更多信息，请参阅计划实施指南。

ERP

参见企业资源规划。

探索性数据分析（EDA）

分析数据集以了解其主要特征的过程。您收集或汇总数据，并进行初步调查，以发现模式、检测异常并检查假定情况。EDA 通过计算汇总统计数据和创建数据可视化得以执行。

F

事实表

星形架构中的中心表。它存储有关业务运营的定量数据。通常，事实表包含两种类型的列：包含度量的列和包含维度表外键的列。

失败得很快

一种使用频繁和增量测试来缩短开发生命周期的理念。这是敏捷方法的关键部分。

故障隔离边界

在中 AWS Cloud，诸如可用区 AWS 区域、控制平面或数据平面之类的边界，它限制了故障的影响并有助于提高工作负载的弹性。有关更多信息，请参阅AWS 故障隔离边界。

功能分支

参见分支。

特征

您用来进行预测的输入数据。例如，在制造环境中，特征可能是定期从生产线捕获的图像。

特征重要性

特征对于模型预测的重要性。这通常表示为数值分数，可以通过各种技术进行计算，例如 Shapley 加法解释（SHAP）和积分梯度。有关更多信息，请参阅机器学习模型的可解释性：AWS。

功能转换

为 ML 流程优化数据，包括使用其他来源丰富数据、扩展值或从单个数据字段中提取多组信息。这使得 ML 模型能从数据中获益。例如，如果您将“2021-05-27 00:15:37”日期分解为“2021”、“五月”、“星期四”和“15”，则可以帮助学习与不同数据成分相关的算法学习精细模式。

FGAC

请参阅精细的访问控制。

精细访问控制 (FGAC)

使用多个条件允许或拒绝访问请求。

快闪迁移

一种数据库迁移方法，它使用连续的数据复制，通过更改数据捕获在尽可能短的时间内迁移数据，而不是使用分阶段的方法。目标是将停机时间降至最低。

G

地理封锁

请参阅地理限制。

地理限制（地理阻止）

在 Amazon 中 CloudFront，一种阻止特定国家/地区的用户访问内容分发的选项。您可以使用允许列表或阻止列表来指定已批准和已禁止的国家/地区。有关更多信息，请参阅 CloudFront 文档中的限制内容的地理分布。

GitFlow 工作流程

一种方法，在这种方法中，下层和上层环境在源代码存储库中使用不同的分支。Gitflow 工作流程被认为是传统的，而基于主干的工作流程是现代的首选方法。

全新策略

在新环境中缺少现有基础设施。在对系统架构采用全新策略时，您可以选择所有新技术，而不受对现有基础设施（也称为棕地）兼容性的限制。如果您正在扩展现有基础设施，则可以将棕地策略和全新策略混合。

防护机制

一种高级规则，用于跨组织单位（OU）管理资源、策略和合规性。预防性防护机制会执行策略以确保符合合规性标准。它们是使用服务控制策略和 IAM 权限边界实现的。侦测性防护机制会检测策略违规和合规性问题，并生成警报以进行修复。它们通过使用 AWS Config、、Amazon、 AWS Security Hub GuardDuty AWS Trusted Advisor、Amazon Inspector 和自定义 AWS Lambda 支票来实现。

H

HA

参见高可用性。

异构数据库迁移

将源数据库迁移到使用不同数据库引擎的目标数据库（例如，从 Oracle 迁移到 Amazon Aurora）。异构迁移通常是重新架构工作的一部分，而转换架构可能是一项复杂的任务。AWS 提供了 AWS SCT 来帮助实现架构转换。

高可用性 (HA)

在遇到挑战或灾难时，工作负载无需干预即可连续运行的能力。HA 系统旨在自动进行故障转移、持续提供良好性能，并以最小的性能影响处理不同负载和故障。

历史数据库现代化

一种用于实现运营技术 (OT) 系统现代化和升级以更好满足制造业需求的方法。历史数据库是一种用于收集和存储工厂中各种来源数据的数据库。

同构数据库迁移

将源数据库迁移到共享同一数据库引擎的目标数据库（例如，从 Microsoft SQL Server 迁移到 Amazon RDS for SQL Server）。同构迁移通常是更换主机或更换平台工作的一部分。您可以使用本机数据库实用程序来迁移架构。

热数据

经常访问的数据，例如实时数据或近期的转化数据。这些数据通常需要高性能存储层或存储类别才能提供快速的查询响应。

修补程序

针对生产环境中关键问题的紧急修复。由于其紧迫性，修补程序通常是在典型的 DevOps 发布工作流程之外进行的。

hypercare 周期

割接之后，迁移团队立即管理和监控云中迁移的应用程序以解决任何问题的时间段。通常，这个周期持续 1-4 天。在 hypercare 周期结束时，迁移团队通常会将应用程序的责任移交给云运营团队。

I

IaC

参见基础架构即代码。

基于身份的策略

附加到一个或多个 IAM 委托人的策略，用于定义他们在 AWS Cloud 环境中的权限。

空闲应用程序

90 天内平均 CPU 和内存使用率在 5% 到 20% 之间的应用程序。在迁移项目中，通常会停用这些应用程序或将其保留在本地。

IIoT

参见工业物联网。

不可变的基础架构

一种为生产工作负载部署新基础架构，而不是更新、修补或修改现有基础架构的模型。不可变基础架构本质上比可变基础架构更一致、更可靠、更可预测。有关更多信息，请参阅 Well-Architected Framework 中的使用不可变基础架构 AWS 部署最佳实践。

入站（入口）VPC

在 AWS 多账户架构中，一种接受、检查和路由来自应用程序外部的网络连接的 VPC。AWS 安全参考架构建议使用入站、出站和检查 VPC 设置网络账户，保护应用程序与广泛的互联网之间的双向接口。

增量迁移

一种割接策略，在这种策略中，您可以将应用程序分成小部分进行迁移，而不是一次性完整割接。例如，您最初可能只将几个微服务或用户迁移到新系统。在确认一切正常后，您可以逐步迁移其他微服务或用户，直到停用遗留系统。这种策略降低了大规模迁移带来的风险。

工业 4.0

该术语由克劳斯·施瓦布（Klaus Schwab）于2016年推出，指的是通过连接、实时数据、自动化、分析和人工智能/机器学习的进步实现制造流程的现代化。

基础设施

应用程序环境中包含的所有资源和资产。

基础设施即代码（IaC）

通过一组配置文件预置和管理应用程序基础设施的过程。IaC 旨在帮助您集中管理基础设施、实现资源标准化和快速扩展，使新环境具有可重复性、可靠性和一致性。

工业物联网（IIoT）

在工业领域使用联网的传感器和设备，例如制造业、能源、汽车、医疗保健、生命科学和农业。有关更多信息，请参阅制定工业物联网（IIoT）数字化转型策略。

检查 VPC

在 AWS 多账户架构中，一种集中式 VPC，用于管理 VPC（相同或不同 AWS 区域）、互联网和本地网络之间的网络流量检查。AWS 安全参考架构建议使用入站、出站和检查 VPC 设置网络账户，保护应用程序与广泛的互联网之间的双向接口。

物联网（IoT）

由带有嵌入式传感器或处理器的连接物理对象组成的网络，这些传感器或处理器通过互联网或本地通信网络与其他设备和系统进行通信。有关更多信息，请参阅什么是 IoT？

可解释性

它是机器学习模型的一种特征，描述了人类可以理解模型的预测如何取决于其输入的程度。有关更多信息，请参阅使用 AWS 实现机器学习模型的可解释性。

IoT

参见物联网。

IT 信息库（ITIL）

提供 IT 服务并使这些服务符合业务要求的一套最佳实践。ITIL 是 ITSM 的基础。

IT 服务管理（ITSM）

为组织设计、实施、管理和支持 IT 服务的相关活动。有关将云运营与 ITSM 工具集成的信息，请参阅运营集成指南。

ITIL

请参阅 IT 信息库。

ITSM

请参阅 IT 服务管理。

L

基于标签的访问控制 (LBAC)

强制访问控制 (MAC) 的一种实施方式，其中明确为用户和数据本身分配了安全标签值。用户安全标签和数据安全标签之间的交集决定了用户可以看到哪些行和列。

登录区

landing zone 是一个架构精良的多账户 AWS 环境，具有可扩展性和安全性。这是一个起点，您的组织可以从这里放心地在安全和基础设施环境中快速启动和部署工作负载和应用程序。有关登录区的更多信息，请参阅设置安全且可扩展的多账户 AWS 环境。

大规模迁移

迁移 300 台或更多服务器。

LBAC

参见基于标签的访问控制。

最低权限

授予执行任务所需的最低权限的最佳安全实践。有关更多信息，请参阅 IAM 文档中的应用最低权限许可。

直接迁移

见 7 R。

小端序系统

一个先存储最低有效字节的系统。另请参见字节顺序。

下层环境

参见环境。

M

机器学习（ML）

一种使用算法和技术进行模式识别和学习的人工智能。ML 对记录的数据（例如物联网（IoT）数据）进行分析和学习，以生成基于模式的统计模型。有关更多信息，请参阅机器学习。

主分支

参见分支。

恶意软件

旨在危害计算机安全或隐私的软件。恶意软件可能会破坏计算机系统、泄露敏感信息或获得未经授权的访问。恶意软件的示例包括病毒、蠕虫、勒索软件、特洛伊木马、间谍软件和键盘记录器。

托管服务

AWS 服务 它 AWS 运行基础设施层、操作系统和平台，您可以访问端点来存储和检索数据。亚马逊简单存储服务 (Amazon S3) Service 和 Amazon DynamoDB 就是托管服务的示例。这些服务也称为抽象服务。

制造执行系统 (MES)

一种软件系统，用于跟踪、监控、记录和控制在车间将原材料转化为成品的生产过程。

MAP

参见迁移加速计划。

机制

一个完整的过程，在此过程中，您可以创建工具，推动工具的采用，然后检查结果以进行调整。机制是一种在运行过程中自我增强和改进的循环。有关更多信息，请参阅在 Well-Architect AWS ed 框架中构建机制。

成员账户

AWS 账户 除属于组织中的管理账户之外的所有账户 AWS Organizations。一个账户一次只能是一个组织的成员。

MES

参见制造执行系统。

消息队列遥测传输 (MQTT)

一种基于发布/订阅模式的轻量级 machine-to-machine (M2M) 通信协议，适用于资源受限的物联网设备。

微服务

一种小型独立服务，通过明确定义的 API 进行通信，通常由小型独立团队拥有。例如，保险系统可能包括映射到业务能力（如销售或营销）或子域（如购买、理赔或分析）的微服务。微服务的好处包括敏捷、灵活扩展、易于部署、可重复使用的代码和恢复能力。有关更多信息，请参阅使用 AWS 无服务器服务集成微服务。

微服务架构

一种使用独立组件构建应用程序的方法，这些组件将每个应用程序进程作为微服务运行。这些微服务使用轻量级 API 通过明确定义的接口进行通信。该架构中的每个微服务都可以更新、部署和扩展，以满足对应用程序特定功能的需求。有关更多信息，请参阅在上实现微服务。 AWS

迁移加速计划（MAP）

AWS 该计划提供咨询支持、培训和服务，以帮助组织为迁移到云奠定坚实的运营基础，并帮助抵消迁移的初始成本。MAP 提供了一种以系统的方式执行遗留迁移的迁移方法，以及一套用于自动执行和加速常见迁移场景的工具。

大规模迁移

将大部分应用程序组合分波迁移到云中的过程，在每一波中以更快的速度迁移更多应用程序。本阶段使用从早期阶段获得的最佳实践和经验教训，实施由团队、工具和流程组成的迁移工厂，通过自动化和敏捷交付简化工作负载的迁移。这是 AWS 迁移策略的第三阶段。

迁移工厂

跨职能团队，通过自动化、敏捷的方法简化工作负载迁移。迁移工厂团队通常包括运营、业务分析师和所有者、迁移工程师、开发 DevOps 人员和冲刺专业人员。20% 到 50% 的企业应用程序组合由可通过工厂方法优化的重复模式组成。有关更多信息，请参阅本内容集中有关迁移工厂的讨论和云迁移工厂指南。

迁移元数据

有关完成迁移所需的应用程序和服务器的信息。每种迁移模式都需要一套不同的迁移元数据。迁移元数据的示例包括目标子网、安全组和 AWS 账户。

迁移模式

一种可重复的迁移任务，详细列出了迁移策略、迁移目标以及所使用的迁移应用程序或服务。示例：使用 AWS 应用程序迁移服务重新托管向 Amazon EC2 的迁移。

迁移组合评测（MPA）

一种在线工具，可提供信息，用于验证迁移到的业务案例。 AWS Cloud MPA 提供了详细的组合评测（服务器规模调整、定价、TCO 比较、迁移成本分析）以及迁移计划（应用程序数据分析和数据收集、应用程序分组、迁移优先级排序和波次规划）。所有 AWS 顾问和 APN 合作伙伴顾问均可免费使用 MPA 工具（需要登录）。

迁移准备情况评测（MRA）

使用 AWS CAF 深入了解组织的云就绪状态、确定优势和劣势以及制定行动计划以缩小已发现差距的过程。有关更多信息，请参阅迁移准备指南。MRA 是 AWS 迁移策略的第一阶段。

迁移策略

用于将工作负载迁移到的方法 AWS Cloud。有关更多信息，请参阅此词汇表中的 7 R 条目和动员组织以加快大规模迁移。

ML

参见机器学习。

现代化

将过时的（原有的或单体）应用程序及其基础设施转变为云中敏捷、弹性和高度可用的系统，以降低成本、提高效率和利用创新。有关更多信息，请参阅中的应用程序现代化策略。 AWS Cloud

现代化准备情况评估

一种评估方式，有助于确定组织应用程序的现代化准备情况；确定收益、风险和依赖关系；确定组织能够在多大程度上支持这些应用程序的未来状态。评估结果是目标架构的蓝图、详细说明现代化进程发展阶段和里程碑的路线图以及解决已发现差距的行动计划。有关更多信息，请参阅中的评估应用程序的现代化准备情况 AWS Cloud。

单体应用程序（单体式）

作为具有紧密耦合进程的单个服务运行的应用程序。单体应用程序有几个缺点。如果某个应用程序功能的需求激增，则必须扩展整个架构。随着代码库的增长，添加或改进单体应用程序的功能也会变得更加复杂。若要解决这些问题，可以使用微服务架构。有关更多信息，请参阅将单体分解为微服务。

MPA

参见迁移组合评估。

MQTT

请参阅消息队列遥测传输。

多分类器

一种帮助为多个类别生成预测（预测两个以上结果之一）的过程。例如，ML 模型可能会询问“这个产品是书、汽车还是手机？”或“此客户最感兴趣什么类别的产品？”

可变基础架构

一种用于更新和修改现有生产工作负载基础架构的模型。为了提高一致性、可靠性和可预测性，Well-Architect AWS ed Framework 建议使用不可变基础设施作为最佳实践。

O

OAC

请参阅源站访问控制。

OAI

参见源访问身份。

OCM

参见组织变更管理。

离线迁移

一种迁移方法，在这种方法中，源工作负载会在迁移过程中停止运行。这种方法会延长停机时间，通常用于小型非关键工作负载。

OI

参见运营集成。

OLA

参见运营层协议。

在线迁移

一种迁移方法，在这种方法中，源工作负载无需离线即可复制到目标系统。在迁移过程中，连接工作负载的应用程序可以继续运行。这种方法的停机时间为零或最短，通常用于关键生产工作负载。

OPC-UA

参见开放流程通信-统一架构。

开放流程通信-统一架构 (OPC-UA)

一种用于工业自动化的 machine-to-machine （M2M）通信协议。OPC-UA 提供了数据加密、身份验证和授权方案的互操作性标准。

运营级别协议（OLA）

一项协议，阐明了 IT 职能部门承诺相互交付的内容，以支持服务水平协议（SLA）。

运营准备情况审查 (ORR)

一份问题清单和相关的最佳实践，可帮助您理解、评估、预防或缩小事件和可能的故障的范围。有关更多信息，请参阅 Well-Architecte AWS d Frame work 中的运营准备情况评估 (ORR)。

操作技术 (OT)

与物理环境配合使用以控制工业运营、设备和基础设施的硬件和软件系统。在制造业中，OT 和信息技术 (IT) 系统的集成是工业 4.0 转型的重点。

运营整合（OI）

在云中实现运营现代化的过程，包括就绪计划、自动化和集成。有关更多信息，请参阅运营整合指南。

组织跟踪

由 AWS CloudTrail 此创建的跟踪记录组织 AWS 账户 中所有人的所有事件 AWS Organizations。该跟踪是在每个 AWS 账户 中创建的，属于组织的一部分，并跟踪每个账户的活动。有关更多信息，请参阅 CloudTrail文档中的为组织创建跟踪。

组织变革管理（OCM）

一个从人员、文化和领导力角度管理重大、颠覆性业务转型的框架。OCM 通过加快变革采用、解决过渡问题以及推动文化和组织变革，帮助组织为新系统和战略做好准备和过渡。在 AWS 迁移策略中，该框架被称为人员加速，因为云采用项目需要变更的速度。有关更多信息，请参阅 OCM 指南。

来源访问控制（OAC）

在中 CloudFront，一个增强的选项，用于限制访问以保护您的亚马逊简单存储服务 (Amazon S3) 内容。OAC 全部支持所有 S3 存储桶 AWS 区域、使用 AWS KMS (SSE-KMS) 进行服务器端加密，以及对 S3 存储桶的动态PUT和DELETE请求。

来源访问身份（OAI）

在中 CloudFront，一个用于限制访问权限以保护您的 Amazon S3 内容的选项。当您使用 OAI 时，CloudFront 会创建一个 Amazon S3 可以对其进行身份验证的委托人。经过身份验证的委托人只能通过特定 CloudFront 分配访问 S3 存储桶中的内容。另请参阅 OAC，其中提供了更精细和增强的访问控制。

或者

参见运营准备情况审查。

OT

参见运营技术。

出站（出口）VPC

在 AWS 多账户架构中，一种处理从应用程序内部启动的网络连接的 VPC。AWS 安全参考架构建议使用入站、出站和检查 VPC 设置网络账户，保护应用程序与广泛的互联网之间的双向接口。

P

权限边界

附加到 IAM 主体的 IAM 管理策略，用于设置用户或角色可以拥有的最大权限。有关更多信息，请参阅 IAM 文档中的权限边界。

个人身份信息 (PII)

直接查看其他相关数据或与之配对时可用于合理推断个人身份的信息。PII 的示例包括姓名、地址和联系信息。

PII

查看个人身份信息。

playbook

一套预定义的步骤，用于捕获与迁移相关的工作，例如在云中交付核心运营功能。playbook 可以采用脚本、自动化运行手册的形式，也可以是操作现代化环境所需的流程或步骤的摘要。

PLC

参见可编程逻辑控制器。

PLM

参见产品生命周期管理。

策略

一个对象，可以在中定义权限（参见基于身份的策略）、指定访问条件（参见基于资源的策略）或定义组织中所有账户的最大权限 AWS Organizations （参见服务控制策略）。

多语言持久性

根据数据访问模式和其他要求，独立选择微服务的数据存储技术。如果您的微服务采用相同的数据存储技术，它们可能会遇到实现难题或性能不佳。如果微服务使用最适合其需求的数据存储，则可以更轻松地实现微服务，并获得更好的性能和可扩展性。有关更多信息，请参阅在微服务中实现数据持久性。

组合评测

一个发现、分析和确定应用程序组合优先级以规划迁移的过程。有关更多信息，请参阅评估迁移准备情况。

谓词

返回`true`或的查询条件`false`，通常位于子`WHERE`句中。

谓词下推

一种数据库查询优化技术，可在传输前筛选查询中的数据。这减少了必须从关系数据库检索和处理的数据量，并提高了查询性能。

预防性控制

一种安全控制，旨在防止事件发生。这些控制是第一道防线，帮助防止未经授权的访问或对网络的意外更改。有关更多信息，请参阅在 AWS上实施安全控制中的预防性控制。

主体

中 AWS 可以执行操作和访问资源的实体。此实体通常是 IAM 角色的根用户或用户。 AWS 账户有关更多信息，请参阅 IAM 文档中角色术语和概念中的主体。

隐私设计

一种贯穿整个工程化过程考虑隐私的系统工程方法。

私有托管区

私有托管区就是一个容器，其中包含的信息说明您希望 Amazon Route 53 如何响应一个或多个 VPC 中的某个域及其子域的 DNS 查询。有关更多信息，请参阅 Route 53 文档中的私有托管区的使用。

主动控制

一种安全控制措施，旨在防止部署不合规的资源。这些控件会在资源置备之前对其进行扫描。如果资源与控件不兼容，则不会对其进行配置。有关更多信息，请参阅 AWS Control Tower 文档中的控制参考指南，并参见在上实施安全控制中的主动控制 AWS。

产品生命周期管理 (PLM)

在产品的整个生命周期中，从设计、开发和上市，到成长和成熟，再到衰落和移除，对产品进行数据和流程的管理。

生产环境

参见环境。

可编程逻辑控制器 (PLC)

在制造业中，一种高度可靠、适应性强的计算机，用于监控机器并实现制造过程自动化。

假名化

用占位符值替换数据集中个人标识符的过程。假名化可以帮助保护个人隐私。假名化数据仍被视为个人数据。

发布/订阅（发布/订阅）

一种支持微服务间异步通信的模式，以提高可扩展性和响应能力。例如，在基于微服务的 MES 中，微服务可以将事件消息发布到其他微服务可以订阅的频道。系统可以在不更改发布服务的情况下添加新的微服务。

Q

查询计划

一系列步骤，例如指令，用于访问 SQL 关系数据库系统中的数据。

查询计划回归

当数据库服务优化程序选择的最佳计划不如数据库环境发生特定变化之前时。这可能是由统计数据、约束、环境设置、查询参数绑定更改和数据库引擎更新造成的。

R

RACI 矩阵

参见 “负责任、负责、咨询、知情”（RACI）。

勒索软件

一种恶意软件，旨在阻止对计算机系统或数据的访问，直到付款为止。

RASCI 矩阵

参见 “负责任、负责、咨询、知情”（RACI）。

RCAC

请参阅行和列访问控制。

只读副本

用于只读目的的数据库副本。您可以将查询路由到只读副本，以减轻主数据库的负载。

重新架构师

见 7 R。

恢复点目标 (RPO)

自上一个数据恢复点以来可接受的最长时间。这决定了从上一个恢复点到服务中断之间可接受的数据丢失情况。

恢复时间目标 (RTO)

服务中断和服务恢复之间可接受的最大延迟。

重构

见 7 R。

区域

地理区域内的 AWS 资源集合。每一个 AWS 区域 都相互隔离，彼此独立，以提供容错、稳定性和弹性。有关更多信息，请参阅指定 AWS 区域 您的账户可以使用的账户。

回归

一种预测数值的 ML 技术。例如，要解决“这套房子的售价是多少？”的问题 ML 模型可以使用线性回归模型，根据房屋的已知事实（如建筑面积）来预测房屋的销售价格。

重新托管

见 7 R。

版本

在部署过程中，推动生产环境变更的行为。

搬迁

见 7 R。

更换平台

见 7 R。

回购

见 7 R。

故障恢复能力

应用程序抵御中断或从中断中恢复的能力。在中规划弹性时，高可用性和灾难恢复是常见的考虑因素。 AWS Cloud有关更多信息，请参阅AWS Cloud 弹性。

基于资源的策略

一种附加到资源的策略，例如 AmazonS3 存储桶、端点或加密密钥。此类策略指定了允许哪些主体访问、支持的操作以及必须满足的任何其他条件。

责任、问责、咨询和知情（RACI）矩阵

定义参与迁移活动和云运营的所有各方的角色和责任的矩阵。矩阵名称源自矩阵中定义的责任类型：负责 (R)、问责 (A)、咨询 (C) 和知情 (I)。支持 (S) 类型是可选的。如果包括支持，则该矩阵称为 RASCI 矩阵，如果将其排除在外，则称为 RACI 矩阵。

响应性控制

一种安全控制，旨在推动对不良事件或偏离安全基线的情况进行修复。有关更多信息，请参阅在AWS上实施安全控制中的响应性控制。

保留

见 7 R。

退休

见 7 R。

旋转

定期更新密钥以使攻击者更难访问凭据的过程。

行列访问控制 (RCAC)

使用已定义访问规则的基本、灵活的 SQL 表达式。RCAC 由行权限和列掩码组成。

RPO

参见恢复点目标。

RTO

参见恢复时间目标。

运行手册

执行特定任务所需的一套手动或自动程序。它们通常是为了简化重复性操作或高错误率的程序而设计的。

S

SAML 2.0

许多身份提供商 (IdPs) 使用的开放标准。此功能支持联合单点登录 (SSO)，因此用户无需在 IAM 中为组织中的所有人创建用户即可登录 AWS Management Console 或调用 AWS API 操作。有关基于 SAML 2.0 的联合身份验证的更多信息，请参阅 IAM 文档中的关于基于 SAML 2.0 的联合身份验证。

SCADA

参见监督控制和数据采集。

SCP

参见服务控制政策。

secret

在中 AWS Secrets Manager，您以加密形式存储的机密或受限信息，例如密码或用户凭证。它由密钥值及其元数据组成。密钥值可以是二进制、单个字符串或多个字符串。有关更多信息，请参阅 Secret s Manager 密钥中有什么？ 在 Secrets Manager 文档中。

安全控制

一种技术或管理防护机制，可防止、检测或降低威胁行为体利用安全漏洞的能力。安全控制主要有四种类型：预防性、侦测、响应式和主动式。

安全加固

缩小攻击面，使其更能抵御攻击的过程。这可能包括删除不再需要的资源、实施授予最低权限的最佳安全实践或停用配置文件中不必要的功能等操作。

安全信息和事件管理（SIEM）系统

结合了安全信息管理（SIM）和安全事件管理（SEM）系统的工具和服务。SIEM 系统会收集、监控和分析来自服务器、网络、设备和其他来源的数据，以检测威胁和安全漏洞，并生成警报。

安全响应自动化

一种预定义和编程的操作，旨在自动响应或修复安全事件。这些自动化可作为侦探或响应式安全控制措施，帮助您实施 AWS 安全最佳实践。自动响应操作的示例包括修改 VPC 安全组、修补 Amazon EC2 实例或轮换证书。

服务器端加密

在目的地对数据进行加密，由接收方 AWS 服务 进行加密。

服务控制策略（SCP）

一种策略，用于集中控制 AWS Organizations的组织中所有账户的权限。SCP 为管理员可以委托给用户或角色的操作定义了防护机制或设定了限制。您可以将 SCP 用作允许列表或拒绝列表，指定允许或禁止哪些服务或操作。有关更多信息，请参阅 AWS Organizations 文档中的服务控制策略。

服务端点

的入口点的 URL AWS 服务。您可以使用端点，通过编程方式连接到目标服务。有关更多信息，请参阅 AWS 一般参考 中的 AWS 服务 端点。

服务水平协议（SLA）

一份协议，阐明了 IT 团队承诺向客户交付的内容，比如服务正常运行时间和性能。

服务级别指示器 (SLI)

对服务性能方面的衡量，例如其错误率、可用性或吞吐量。

服务级别目标 (SLO)

代表服务运行状况的目标指标，由服务级别指标衡量。

责任共担模式

描述您在云安全与合规方面共同承担 AWS 的责任的模型。 AWS 负责云的安全，而您则负责云中的安全。有关更多信息，请参阅责任共担模式。

暹粒

参见安全信息和事件管理系统。

单点故障 (SPOF)

应用程序的单个关键组件出现故障，可能会中断系统。

SLA

参见服务级别协议。

SLI

参见服务级别指标。

SLO

参见服务级别目标。

split-and-seed 模型

一种扩展和加速现代化项目的模式。随着新功能和产品发布的定义，核心团队会拆分以创建新的产品团队。这有助于扩展组织的能力和服务，提高开发人员的工作效率，支持快速创新。有关更多信息，请参阅中的分阶段实现应用程序现代化的方法。 AWS Cloud

恶作剧

参见单点故障。

星型架构

一种数据库组织结构，它使用一个大型事实表来存储交易数据或测量数据，并使用一个或多个较小的维度表来存储数据属性。此结构专为在数据仓库中使用或用于商业智能目的而设计。

strangler fig 模式

一种通过逐步重写和替换系统功能直至可以停用原有的系统来实现单体系统现代化的方法。这种模式用无花果藤作为类比，这种藤蔓成长为一棵树，最终战胜并取代了宿主。该模式是由 Martin Fowler 提出的，作为重写单体系统时管理风险的一种方法。有关如何应用此模式的示例，请参阅使用容器和 Amazon API Gateway 逐步将原有的 Microsoft ASP.NET（ASMX）Web 服务现代化。

子网

您的 VPC 内的一个 IP 地址范围。子网必须位于单个可用区中。

监控和数据采集 (SCADA)

在制造业中，一种使用硬件和软件来监控有形资产和生产操作的系统。

对称加密

一种加密算法，它使用相同的密钥来加密和解密数据。

综合测试

以模拟用户交互的方式测试系统，以检测潜在问题或监控性能。你可以使用 Amazon S CloudWatch ynthetics 来创建这些测试。

T

标签

键值对，充当用于组织资源的元数据。 AWS 标签可帮助您管理、识别、组织、搜索和筛选资源。有关更多信息，请参阅标记您的 AWS 资源。

目标变量

您在监督式 ML 中尝试预测的值。这也被称为结果变量。例如，在制造环境中，目标变量可能是产品缺陷。

任务列表

一种通过运行手册用于跟踪进度的工具。任务列表包含运行手册的概述和要完成的常规任务列表。对于每项常规任务，它包括预计所需时间、所有者和进度。

测试环境

参见环境。

训练

为您的 ML 模型提供学习数据。训练数据必须包含正确答案。学习算法在训练数据中查找将输入数据属性映射到目标（您希望预测的答案）的模式。然后输出捕获这些模式的 ML 模型。然后，您可以使用 ML 模型对不知道目标的新数据进行预测。

中转网关

中转网关是网络中转中心，您可用它来互连 VPC 和本地网络。有关更多信息，请参阅 AWS Transit Gateway 文档中的什么是公交网关。

基于中继的工作流程

一种方法，开发人员在功能分支中本地构建和测试功能，然后将这些更改合并到主分支中。然后，按顺序将主分支构建到开发、预生产和生产环境。

可信访问权限

向您指定的服务授予权限，该服务可代表您在其账户中执行任务。 AWS Organizations 当需要服务相关的角色时，受信任的服务会在每个账户中创建一个角色，为您执行管理任务。有关更多信息，请参阅 AWS Organizations 文档中的AWS Organizations 与其他 AWS 服务一起使用。

优化

更改训练过程的各个方面，以提高 ML 模型的准确性。例如，您可以通过生成标签集、添加标签，并在不同的设置下多次重复这些步骤来优化模型，从而训练 ML 模型。

双披萨团队

一个小 DevOps 团队，你可以用两个披萨来喂食。双披萨团队的规模可确保在软件开发过程中充分协作。

U

不确定性

这一概念指的是不精确、不完整或未知的信息，这些信息可能会破坏预测式 ML 模型的可靠性。不确定性有两种类型：认知不确定性是由有限的、不完整的数据造成的，而偶然不确定性是由数据中固有的噪声和随机性导致的。有关更多信息，请参阅量化深度学习系统中的不确定性指南。

无差别任务

也称为繁重工作，即创建和运行应用程序所必需的工作，但不能为最终用户提供直接价值或竞争优势。无差别任务的示例包括采购、维护和容量规划。

上层环境

参见环境。

V

vacuum 操作

一种数据库维护操作，包括在增量更新后进行清理，以回收存储空间并提高性能。

版本控制

跟踪更改的过程和工具，例如存储库中源代码的更改。

VPC 对等连接

两个 VPC 之间的连接，允许您使用私有 IP 地址路由流量。有关更多信息，请参阅 Amazon VPC 文档中的什么是 VPC 对等连接。

漏洞

损害系统安全的软件缺陷或硬件缺陷。

W

热缓存

一种包含经常访问的当前相关数据的缓冲区缓存。数据库实例可以从缓冲区缓存读取，这比从主内存或磁盘读取要快。

暖数据

不常访问的数据。查询此类数据时，通常可以接受中速查询。

窗口函数

一个 SQL 函数，用于对一组以某种方式与当前记录相关的行进行计算。窗口函数对于处理任务很有用，例如计算移动平均线或根据当前行的相对位置访问行的值。

工作负载

一系列资源和代码，它们可以提供商业价值，如面向客户的应用程序或后端过程。

工作流

迁移项目中负责一组特定任务的职能小组。每个工作流都是独立的，但支持项目中的其他工作流。例如，组合工作流负责确定应用程序的优先级、波次规划和收集迁移元数据。组合工作流将这些资产交付给迁移工作流，然后迁移服务器和应用程序。

蠕虫

参见一次写入，多读。

WQF

请参阅 AWS 工作负载资格框架。

一次写入，多次读取 (WORM)

一种存储模型，它可以一次写入数据并防止数据被删除或修改。授权用户可以根据需要多次读取数据，但他们无法对其进行更改。这种数据存储基础架构被认为是不可变的。

Z

零日漏洞利用

一种利用未修补漏洞的攻击，通常是恶意软件。

零日漏洞

生产系统中不可避免的缺陷或漏洞。威胁主体可能利用这种类型的漏洞攻击系统。开发人员经常因攻击而意识到该漏洞。

僵尸应用程序

平均 CPU 和内存使用率低于 5% 的应用程序。在迁移项目中，通常会停用这些应用程序。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。