



维护 Amazon RDS 和 Amazon Aurora 中的 PostgreSQL 数据库活动，以避免性能问题

AWS 规范性指导



AWS 规范性指导: 维护 Amazon RDS 和 Amazon Aurora 中的 PostgreSQL 数据库活动，以避免性能问题

Table of Contents

简介	1
目标业务成果	1
多版本并发控制 (MVCC)	1
自动清理和分析表格	3
Autovacuum 与内存相关的参数	4
调整 autovacuum 参数	5
集群或实例级别	5
表级别	5
在表格级别使用激进的 autovacuum 设置	5
优点和限制	6
手动清理和分析表格	8
并行运行 vacuum 和清理操作	8
使用 VACUUM FULL 重写整个表	12
使用 pg_repack 移除膨胀	14
重建索引	16
创建新索引	19
重建索引	20
示例	20
示例：使用 autovacuum 和 VACUUM FULL 功能来回收空间	22
资源	27
文档历史记录	28
术语表	29
#	29
A	29
B	32
C	34
D	36
E	40
F	41
G	43
H	44
我	45
L	47
M	48

O	52
P	54
Q	56
R	56
S	59
T	62
U	63
V	64
W	64
Z	65
.....	lxvi

维护 Amazon RDS 和 Amazon Aurora 中的 PostgreSQL 数据库活动，以避免性能问题

Anuradha Chintha、Rajesh Madiwale 和 Srinivas Potlachervoo，Amazon Web Services (AWS)

2025 年 8 月 ([文档历史记录](#))

Amazon Aurora PostgreSQL 兼容版数据库服务或 Amazon Relational Database Service (Amazon RDS) for PostgreSQL 数据库服务是 PostgreSQL 数据库的完全托管型关系数据库服务。这些托管服务使数据库管理员能够从许多维护和管理任务中解放出来。但是，某些维护任务 (例如 VACUUM) 需要根据您的数据库使用情况进行密切监控和配置。本指南介绍了 Amazon RDS 和 Aurora 中的 PostgreSQL 维护活动。

目标业务成果

数据库性能是衡量企业成功与否的关键指标。在兼容 Aurora PostgreSQL 的数据库和 Amazon RDS for PostgreSQL 数据库上执行维护活动可带来以下益处：

- 帮助实现最佳查询性能
- 腾出膨胀的空间，以便未来的事务重复使用
- 防止事务循环操作
- 帮助优化程序生成良好的计划
- 确保适当使用索引

多版本并发控制 (MVCC)

PostgreSQL 数据库维护需要了解 PostgreSQL 的机制：多版本并发控制 (MVCC)。当数据库中同时处理多个事务时，MVCC 能确保保持原子性和隔离性，这是原子性、一致性、隔离性、持久性 (ACID) 事务的两个特性。在 MVCC 中，每次写入操作都会生成数据的新版本，并存储之前的版本。读取器和写入器不会互相妨碍。当事务读取数据时，系统会选择其中一个版本来提供事务隔离功能。PostgreSQL 和一些关系数据库使用 MVCC 的改编版，称为快照隔离 (SI)。例如，Oracle 通过使用回滚段来实施 SI。在写入操作过程中，Oracle 将数据的旧版本写入回滚段，并用新版本的数据覆盖原有的数据区域。PostgreSQL 数据库通过使用可见性检查规则评估版本来实施 SI。将新数据放入表页面时，PostgreSQL 会使用这些规则为读取操作选择适当的数据版本。

修改表行中的数据时，PostgreSQL 使用 MVCC 来维护该行的多个版本。在对表进行 UPDATE 和 DELETE 操作期间，数据库会保留旧版本的行，供其他正在运行的事务使用，这些事务可能需要一致的数据视图。这些旧版本被称为死行（元组）。死元组的集合会产生膨胀。数据库中出现大量膨胀可能会引发一系列问题，比如查询计划生成效率低下、查询速度变慢以及存储较旧版本数据所使用的磁盘空间增加。

消除膨胀并保持数据库的正常运行需要定期进行维护，这些维护活动包括以下内容，将在后续部分中进行详细阐述：

- [自动清理和分析表格](#)
- [手动清理和分析表格](#)
- [使用 pg_repack 移除膨胀](#)
- [重建索引](#)

自动清理和分析表格

Autovacuum 是一个进程守护程序（也就是说，它在后台运行），它会自动 vacuum（清理）死元组，回收存储空间并收集统计信息。它会检查数据库中是否有膨胀的表，并清除膨胀以重新利用这些空间。它会监控数据库表和索引，并在这些表或索引的更新或删除操作达到特定阈值后，将它们添加到 vacuum 作业中。

Autovacuum 通过自动执行 PostgreSQL VACUUM 和 ANALYZE 命令来管理 vacuum 操作。VACUUM 从表中移除膨胀并回收空间，同时 ANALYZE 更新统计信息，使优化程序能够制定有效的计划。VACUUM 还执行一项名为 vacuum 冻结的主要任务，以防止数据库中出现事务 ID 循环问题。数据库中每一个被更新的行都会从 PostgreSQL 事务控制机制中获得一个事务 ID。它们 IDs 控制该行对其他并发事务的可见性。事务 ID 是一个 32 位的数字。20 亿 IDs 总是留在可见的过去。剩余 IDs 的（约 22 亿）用于将来将要进行的交易，并且隐藏在当前交易之外。PostgreSQL 需要偶尔清理和冻结旧行，以防止事务回绕并在创建新事务时使旧的现有行不可见。有关更多信息，请参阅 PostgreSQL 文档中的[防止事务 ID 重现故障](#)。

建议使用 Autovacuum 且它在默认情况下处于启用状态。其参数包括以下选项：

参数	描述	Amazon RDS 的默认设置	Aurora 的默认设置
autovacuum_vacuum_threshold	在 autovacuum 对一个表进行清理操作之前，该表上必须发生的最小的元组更新或删除操作次数。	50 次操作	50 次操作
autovacuum_analyze_threshold	在 autovacuum 分析一个表之前，该表上必须发生的最小的元组插入、更新或删除操作次数。	50 次操作	50 次操作
autovacuum_vacuum_scale_factor	在 autovacuum 对表进行清理之前，该表中必须修改的元组所占的百分比。	0.1	0.1

<code>autovacuum_analyze_scale_factor</code>	在 <code>autovacuum</code> 对表进行分析之前，该表中必须修改的元组所占的比例。	0.05	0.05
<code>autovacuum_freeze_max_age</code>	清空表 IDs 之前的最大冻结期限，以防止事务 ID 环绕问题。	2 亿次事务	2 亿次事务

Autovacuum 会根据特定的阈值公式生成需要处理的表的列表，具体如下。

- 在表格上运行 VACUUM 的阈值：

```
vacuum threshold = autovacuum_vacuum_threshold + (autovacuum_vacuum_scale_factor *
Total row count of table)
```

- 在表格上运行 ANALYZE 的阈值：

```
analyze threshold = autovacuum_analyze_threshold + (autovacuum_analyze_scale_factor *
Total row count of table)
```

对于中小型表，默认值可能就足够用了。但是，经常进行数据修改的大型表中会存在更多的死元组。在这种情况下，`autovacuum` 可能会频繁地对此表进行处理以进行维护操作，而其他表的维护工作可能会因此被推迟或忽略，直到这个大型表的维护工作完成为止。为避免这种情况，您可以调整下一部分中所述的 `autovacuum` 参数。

Autovacuum 与内存相关的参数

`autovacuum_max_workers`

指定可以同时运行的 `autovacuum` 进程（不包括 `autovacuum` 启动程序）的最大数量。只有在启动服务器时，才能设置此参数。如果 `autovacuum` 进程正忙于处理大型表，则此参数有助于对其他表进行清理操作。

`maintenance_work_mem`

指定维护操作（如 VACUUM、CREATE INDEX 和 ALTER）要使用的最大内存量。在 Amazon RDS 和 Aurora 中，使用公式 `GREATEST({DBInstanceClassMemory}/63963136*1024}, 65536)`

根据实例类分配内存。autovacuum 运行时，所分配的内存量可能会达到该计算值的最多 `autovacuum_max_workers` 倍，因此请注意不要把该值定得过高。要对此进行控制，可以单独设置 `autovacuum_work_mem`。

autovacuum_work_mem

指定要由每个 autovacuum Worker 进程使用的最大内存量。此参数默认为 -1，这表示您应改用 `maintenance_work_mem` 的值。

有关 autovacuum 内存参数的更多信息，请参阅 Amazon RDS 文档中的[为 autovacuum 分配内存](#)。

调整 autovacuum 参数

用户可能需要根据其更新和删除操作来调整 autovacuum 参数。以下参数的设置可以在表级别、实例级别或集群级别进行设置。

集群或实例级别

举个例子，让我们来看一个银行数据库的实例，在这个数据库中，预期会进行持续的数据操纵语言（DML）操作。为了保持数据库的运行状况，您需要针对 Aurora 数据库在集群级别调整 autovacuum 参数，针对 Amazon RDS 数据库在实例级别调整该参数，并且将相同的参数组应用于读取器。在失效转移情况下，应将相同的参数应用于新的写入程序。

表级别

例如，在一个用于食品配送的数据库中，如果需要对名为 `orders` 的单个表进行持续的 DML 操作，则应该通过使用以下命令在表级别调整 `autovacuum_analyze_threshold` 参数：

```
ALTER TABLE <table_name> SET (autovacuum_analyze_threshold = <threshold rows>)
```

在表格级别使用激进的 autovacuum 设置

由于默认的 autovacuum 设置，具有持续更新和删除操作的示例 `orders` 表成为了进行清理操作的候选对象。这会导致计划生成不准确以及查询速度变慢。清除膨胀并更新统计信息需要设置表级别的激进式 autovacuum 参数。

要确定设置，请跟踪在此表上运行的查询的持续时间，并确定导致计划变更的 DML 操作所占的百分比。`pg_stat_user_tables` 视图可帮助您跟踪插入、更新和删除操作。

示例：

假设每当 5% 的 orders 表发生变化时，优化程序就会生成错误的计划。在这种情况下，应将扩展因子阈值更改为 2%，具体如下：

```
ALTER TABLE orders SET (autovacuum_vacuum_scale_factor = 0.02)
```

Tip

请谨慎选择激进式 autovacuum 设置，以免造成资源的大量消耗。

有关更多信息，请参阅下列内容：

- [了解 Amazon RDS for PostgreSQL AWS 环境中的自动真空](#) (博客文章)
- [自动清理](#) (PostgreSQL 文档)
- [调整亚马逊 RDS 和亚马逊 Aurora 中的 PostgreSQL 参数](#) (规范性指南AWS)

为了确保 autovacuum 有效运行，请定期监控死行、磁盘使用情况以及 autovacuum 或 ANALYZE 上一次定期运行的时间。pg_stat_all_tables 视图提供有关每个表 (relname) 以及表中有多少死元组 (n_dead_tup) 的信息。

监控每个表 (尤其是那些频繁更新的表) 中的死元组数量，有助于您判断 autovacuum 进程是否定期移除这些死元组，以便重新利用其占用的磁盘空间，从而提高系统性能。您可以使用以下查询来检查死元组的数量以及上一次对表运行 autovacuum 的时间：

```
SELECT
relname AS TableName,n_live_tup AS LiveTuples,n_dead_tup AS DeadTuples,
last_autovacuum AS Autovacuum,last_autoanalyze AS Autoanalyze_FROM
pg_stat_user_tables;
```

优点和限制

Autovacuum 具备下述优点：

- 它会自动移除表格中的膨胀。
- 它可以防止事务 ID 循环。

- 它使数据库统计信息保持最新。

限制：

- 如果查询采用并行处理方式，则 autovacuum 所需的 Worker 进程数量可能会不够。
- 如果 autovacuum 在高峰时段运行，则资源利用率可能会提高。您应该调整参数来处理这个问题。
- 如果表格页面被另一个会话占用，autovacuum 可能会跳过这些页面。
- Autovacuum 无法访问临时表。

手动清理和分析表格

如果您的数据库通过 autovacuum 过程进行清理的，则最佳实践是避免过于频繁地对整个数据库进行手动清理操作。手动清理可能会导致不必要的 I/O 负载或 CPU 峰值，也可能无法删除任何死元组。只有在真正必要时才运行手动吸尘器，例如当活元组与死元组的比例较低时，或者当自动吸尘器之间存在长距离时。table-by-table 此外，在用户活动极少的情况下，您应当进行手动清理操作。

Autovacuum 还能确保表格的统计信息始终保持最新状态。当您手动运行 ANALYZE 命令时，它会重建这些统计信息，而不是更新它们。在常规的 autovacuum 过程已更新统计信息的情况下，再次重建统计信息可能会导致系统资源占用。

我们建议您在以下情况下手动运行 [VACUUM](#) 和 [ANALYZE](#) 命令：

- 在繁忙表格的低峰时段，autovacuum 操作可能还不够的时候。
- 在将数据批量加载到目标表之后。在这种情况下，手动运行 ANALYZE 可以完全重建统计信息，这比等待 autovacuum 操作开始更好。
- 要清理临时表（autovacuum 无法访问这些表）。

要减少运行 VACUUM 和 ANALYZE 命令时 I/O 对并发数据库活动的影响，可以使用 vacuum_cost_delay 参数。在许多情况下，诸如 VACUUM 和 ANALYZE 之类的维护命令不必很快完成。但是，这些命令不应影响系统执行其他数据库操作的能力。为防止出现这种情况，您可以使用 vacuum_cost_delay 参数启用基于成本的 vacuum 延迟。对于手动发出的 VACUUM 命令，此参数默认处于禁用状态。要启用它，请将其设置为非零值。

并行运行 vacuum 和清理操作

VACUUM 命令 [PARALLEL](#) 选项在索引 vacuum 和索引清理阶段使用并行 Worker，其默认情况下处于禁用状态。并行 Worker 的数量（即并行度）由表中的索引数量决定，并且可以由用户指定。如果您在运行并行 VACUUM 操作时没有整数参数，则并行度根据表中的索引数量计算。

以下参数可帮助您在 Amazon RDS for PostgreSQL 和 Aurora PostgreSQL 版本中配置并行 vacuum 操作：

- [max_worker_processes](#) 用于设置最大并行 Worker 进程数。
- [min_parallel_index_scan_size](#) 用于设定在考虑进行并行扫描时，必须扫描的最小索引数据量。
- [max_parallel_maintenance_workers](#) 用于设定单个实用程序命令能够启动的并行 Worker 的最大数量。

Note

PARALLEL 选项仅用于 vacuum 操作目的。它不影响 ANALYZE 命令。

以下示例说明了对数据库使用手动 VACUUM 和 ANALYZE 时的数据库行为。

以下是禁用 autovacuum 的示例表（仅用于说明目的；不建议禁用 autovacuum）：

```
create table t1 ( a int, b int, c int );
alter table t1 set (autovacuum_enabled=false);
```

```
apgl=> \d+ t1
Table "public.t1"
Column | Type | Collation | Nullable | Default | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----+-----+-----
+-----+
a | integer | | | plain | |
b | integer | | | plain | |
c | integer | | | plain | |
Access method: heap
Options: autovacuum_enabled=false
```

在表 t1 中添加 100 万行：

```
apgl=> select count(*) from t1;
count
1000000
(1 row)
```

表 t1 的统计信息：

```
select * from pg_stat_all_tables where relname='t1';
-[ RECORD 1 ]-----+-----
relid          | 914744
schemaname     | public
relname        | t1
seq_scan       | 0
seq_tup_read   | 0
idx_scan       |
idx_tup_fetch  |
```

```

n_tup_ins      | 1000000
n_tup_upd      | 0
n_tup_del      | 0
n_tup_hot_upd  | 0
n_live_tup     | 1000000
n_dead_tup     | 0
n_mod_since_analyze | 1000000
last_vacuum    |
last_autovacuum |
last_analyze   |
last_autoanalyze |
vacuum_count   | 0
autovacuum_count | 0
analyze_count  | 0
autoanalyze_count | 0

```

添加索引：

```
create index i2 on t1 (b,a);
```

运行 EXPLAIN 命令 (计划 1)：

```

Bitmap Heap Scan on t1 (cost=10521.17..14072.67 rows=5000 width=4)
Recheck Cond: (a = 5)
# Bitmap Index Scan on i2 (cost=0.00..10519.92 rows=5000 width=0)
Index Cond: (a = 5)
(4 rows)

```

运行 EXPLAIN ANALYZE 命令 (计划 2)：

```

explain (analyze,buffers,costs off) select a from t1 where b = 5;
QUERY PLAN
Bitmap Heap Scan on t1 (actual time=0.023..0.024 rows=1 loops=1)
Recheck Cond: (b = 5)
Heap Blocks: exact=1
Buffers: shared hit=4
# Bitmap Index Scan on i2 (actual time=0.016..0.016 rows=1 loops=1)
Index Cond: (b = 5)
Buffers: shared hit=3
Planning Time: 0.054 ms
Execution Time: 0.076 ms
(9 rows)

```

EXPLAIN 和 EXPLAIN ANALYZE 命令显示不同的计划，因为对表格的 autovacuum 操作被禁用，并且 ANALYZE 命令不是手动执行的。现在让我们更新表中的一个值并重新生成 EXPLAIN ANALYZE 计划：

```
update t1 set a=8 where b=5;
explain (analyze,buffers, costs off) select a from t1 where b = 5;
```

EXPLAIN ANALYZE 命令 (计划 3) 现在显示：

```
apgl=> explain (analyze,buffers, costs off) select a from t1 where b = 5;
QUERY PLAN
Bitmap Heap Scan on t1 (actual time=0.075..0.076 rows=1 loops=1)
Recheck Cond: (b = 5)
Heap Blocks: exact=1
Buffers: shared hit=5
# Bitmap Index Scan on i2 (actual time=0.017..0.017 rows=2 loops=1)
Index Cond: (b = 5)
Buffers: shared hit=3
Planning Time: 0.053 ms
Execution Time: 0.125 ms
```

如果您将计划 2 和计划 3 的成本进行比较，您就会发现两者在计划和执行时间上的差异，因为我们尚未收集统计信息。

现在让我们对表格运行手动 ANALYZE 操作，然后检查统计信息并重新生成计划：

```
apgl=> analyze t1
apgl# ;
ANALYZE
Time: 212.223 ms

apgl=> select * from pg_stat_all_tables where relname='t1';
-[ RECORD 1 ]-----+-----
reloid          | 914744
schemaname      | public
relname         | t1
seq_scan        | 3
seq_tup_read    | 1000000
idx_scan        | 3
idx_tup_fetch   | 3
n_tup_ins       | 1000000
n_tup_upd       | 1
```

```

n_tup_del      | 0
n_tup_hot_upd  | 0
n_live_tup     | 1000000
n_dead_tup     | 1
n_mod_since_analyze | 0
last_vacuum    |
last_autovacuum |
last_analyze   | 2023-04-15 11:39:02.075089+00
last_autoanalyze |
vacuum_count   | 0
autovacuum_count | 0
analyze_count  | 1
autoanalyze_count | 0

```

Time: 148.347 ms

运行 EXPLAIN ANALYZE 命令 (计划 4) :

```

apgl=> explain (analyze,buffers, costs off) select a from t1 where b = 5;
QUERY PLAN
Index Only Scan using i2 on t1 (actual time=0.022..0.023 rows=1 loops=1)
Index Cond: (b = 5)
Heap Fetches: 1
Buffers: shared hit=4
Planning Time: 0.056 ms
Execution Time: 0.068 ms
(6 rows)

```

Time: 138.462 ms

如果您在手动分析表格并收集统计信息后对所有计划结果进行比较，您就会发现优化程序的“计划 4”比其他计划更好，并且还能缩短查询执行时间。这个示例说明，对数据库进行维护活动是多么重要。

使用 VACUUM FULL 重写整个表

使用 FULL 参数运行 VACUUM 命令会将表格的全部内容重写到一个新的磁盘文件中，且不会占用额外空间，同时还会将未使用的空间归还给操作系统。此操作要慢得多，需要对每个表进行 ACCESS EXCLUSIVE 锁定。此外，它还需要额外的磁盘空间，因为其会创建表格的新副本，并且在操作完成之前不会释放旧副本。

VACUUM FULL 在以下情况下很有用：

- 当您想从表格中回收大量空间时。
- 当您想要回收非主键表中的膨胀空间时。

我们建议，当您有非主键表且您的数据库能够承受停机时间时，请使用 VACUUM FULL。

由于 VACUUM FULL 所需的锁定资源比其他操作更多，所以在关键数据库上运行该操作的成本会更高。要替换此方法，您可以使用 `pg_repack` 扩展程序，[下一部分](#)将对此进行介绍。此选项与 VACUUM FULL 相似但需要的锁定最少，且受 Amazon RDS for PostgreSQL 和 Aurora PostgreSQL 版本的支持。

使用 pg_repack 移除膨胀

您可以使用 `pg_repack` 扩展程序以最少的数据库锁定来移除表和索引膨胀。您可以在数据库实例中创建此扩展程序，然后从 Amazon Elastic Compute Cloud (Amazon EC2) 或可以连接到您的数据库的计算机上运行 `pg_repack` 客户端 (客户端版本与扩展版本相匹配的位置)。

与 `VACUUM FULL` 不同的是，`pg_repack` 不需要停机时间或维护窗口，也不会阻止其他会话。

`pg_repack` 在 `VACUUM FULL`、`CLUSTER`、或 `REINDEX` 可能不起作用的情况下起到帮助作用。它会创建一个包含膨胀表数据的新表，跟踪原始表中的更改，然后用新表替换原始表。在构建新表时，它不会锁定原始表来防止读取或写入操作。

您可以将 `pg_repack` 用于完整表或索引。要查看任务列表，请参阅 [pg_repack 文档](#)。

限制：

- 要运行 `pg_repack`，您的表必须具有主键或唯一索引。
- `pg_repack` 不适用于临时表。
- `pg_repack` 不适用于具有全局索引的表。
- 当 `pg_repack` 正在进行时，您不能对表执行 DDL 操作。

下表介绍了 `pg_repack` 与 `VACUUM FULL` 之间的区别。

VACUUM FULL	pg_repack
内置命令	您从 Amazon EC2 或本地计算机中运行的扩展程序
对表格执行操作时需要 ACCESS EXCLUSIVE 锁	仅在短时间内需要 ACCESS EXCLUSIVE 锁
适用于所有表	适用于仅具有主键和唯一键的表
所需存储空间是表和索引所消耗空间的两倍	所需存储空间是表和索引所消耗空间的两倍

要在表上运行 `pg_repack`，请使用以下命令：

```
pg_repack -h <host> -d <dbname> --table <tablename> -k
```

要在索引上运行 `pg_repack`，请使用以下命令：

```
pg_repack -h <host> -d <dbname> --index <index name>
```

有关更多信息，请参阅 AWS 博客文章[使用 pg_repack 移除 Amazon Aurora 和 RDS for PostgreSQL 中的膨胀](#)。

注意事项

`pg_repack` 中的 `error-on-invalid-index` 错误通常意味着表中的一个或多个索引已损坏或无效。`pg_repack` 无法安全地对具有无效索引的表进行操作，因为它在重新打包过程中依赖索引来保持数据一致性。

此错误发生在以下情况下：

- 索引被标记为无效（例如，由于 `CREATE INDEX CONCURRENTLY` 语句失败）。
- 索引损坏（可能是由于硬件问题或突然关闭所致）。

使用以下查询来识别无效的索引，并在找到无效索引时先将其删除。

```
SELECT indexrelid::regclass, indisvalid FROM pg_index WHERE indrelid =  
'orders'::regclass AND NOT indisvalid; Drop the invalid index: DROP INDEX  
index_name;
```

重建索引

PostgreSQL [REINDEX](#) 命令通过使用存储在索引表中的数据并替换索引的旧副本来重建索引。我们建议在以下情况下使用 REINDEX：

- 当索引出现损坏且不再包含有效数据时。这可能是由于软件或硬件故障导致的。
- 当之前使用该索引的查询不再使用它时。
- 当索引中充斥着大量空白页面或几乎为空的页面时。您应该在膨胀百分比 (bloat_pct) 大于 20 时运行 REINDEX。

以下查询可帮助您查找 bloat_pct：

```
SELECT current_database(), nspname AS schemaname, tblname, idxname,
bs*(relpages)::bigint AS real_size,
bs*(relpages-est_pages)::bigint AS extra_size,
100 * (relpages-est_pages)::float / relpages AS extra_pct,
fillfactor,
CASE WHEN relpages > est_pages_ff
  THEN bs*(relpages-est_pages_ff)
  ELSE 0
END AS bloat_size,
100 * (relpages-est_pages_ff)::float / relpages AS bloat_pct,
is_na
-- , 100-(pst).avg_leaf_density AS pst_avg_bloat, est_pages, index_tuple_hdr_bm,
maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, reltuples, relpages -- (DEBUG
INFO)
FROM (
  SELECT coalesce(1 +
    ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0
  -- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
  ) AS est_pages,
  coalesce(1 +
    ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/
(100*(4+nulldatahdrwidth)::float))), 0
  ) AS est_pages_ff,
  bs, nspname, tblname, idxname, relpages, fillfactor, is_na
  -- , pgstatindex(idxoid) AS pst, index_tuple_hdr_bm, maxalign, pagehdr,
nulldatawidth, nulldatahdrwidth, reltuples -- (DEBUG INFO)
FROM (
```

```

SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, idxoid,
fillfactor,
    ( index_tuple_hdr_bm +
      maxalign - CASE -- Add padding to the index tuple header to align on
MAXALIGN
          WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
          ELSE index_tuple_hdr_bm%maxalign
        END
      + nulldatawidth + maxalign - CASE -- Add padding to the data to align on
MAXALIGN
          WHEN nulldatawidth = 0 THEN 0
          WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
          ELSE nulldatawidth::integer%maxalign
        END
    )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
-- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
FROM (
  SELECT n.nspname, i.tblname, i.idxname, i.reltuples, i.relpages,
    i.idxoid, i.fillfactor, current_setting('block_size')::numeric AS bs,
    CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
      WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|
amd64' THEN 8
      ELSE 4
    END AS maxalign,
    /* per page header, fixed size: 20 for 7.X, 24 for others */
    24 AS pagehdr,
    /* per page btree opaque data */
    16 AS pageopqdata,
    /* per tuple header: add IndexAttributeBitMapData if some cols are null-
able */
    CASE WHEN max(coalesce(s.null_frac,0)) = 0
      THEN 8 -- IndexTupleData size
      ELSE 8 + (( 32 + 8 - 1 ) / 8) -- IndexTupleData size +
IndexAttributeBitMapData size ( max num filed per index + 8 - 1 /8)
    END AS index_tuple_hdr_bm,
    /* data len: we remove null values save space using it fractionnal part
from stats */
    sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS
nulldatawidth,
    max( CASE WHEN i.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0
END ) > 0 AS is_na
  FROM (

```

```

SELECT ct.relname AS tblname, ct.relnamespace, ic.idxname, ic.attpos,
ic.indkey, ic.indkey[ic.attpos], ic.reltuples, ic.relpages, ic.tbloid, ic.idxoid,
ic.fillfactor,
        coalesce(a1.attnum, a2.attnum) AS attnum, coalesce(a1.attname,
a2.attname) AS attname, coalesce(a1.atttypid, a2.atttypid) AS atttypid,
        CASE WHEN a1.attnum IS NULL
        THEN ic.idxname
        ELSE ct.relname
        END AS attrelname
FROM (
        SELECT idxname, reltuples, relpages, tbloid, idxoid, fillfactor,
indkey,
                pg_catalog.generate_series(1,indnatts) AS attpos
        FROM (
                SELECT ci.relname AS idxname, ci.reltuples, ci.relpages,
i.indrelid AS tbloid,
                        i.indexrelid AS idxoid,
                        coalesce(substring(
                                array_to_string(ci.reloptions, ' ')
                                from 'fillfactor=(\d+)'::smallint, 90) AS fillfactor,
                        i.indnatts,
                        pg_catalog.string_to_array(pg_catalog.textin(
                                pg_catalog.int2vectorout(i.indkey)), ' '))::int[] AS indkey
                FROM pg_catalog.pg_index i
                JOIN pg_catalog.pg_class ci ON ci.oid = i.indexrelid
                WHERE ci.relam=(SELECT oid FROM pg_am WHERE amname = 'btree')
                AND ci.relpages > 0
        ) AS idx_data
        ) AS ic
JOIN pg_catalog.pg_class ct ON ct.oid = ic.tbloid
LEFT JOIN pg_catalog.pg_attribute a1 ON
        ic.indkey[ic.attpos] <> 0
        AND a1.attrelid = ic.tbloid
        AND a1.attnum = ic.indkey[ic.attpos]
LEFT JOIN pg_catalog.pg_attribute a2 ON
        ic.indkey[ic.attpos] = 0
        AND a2.attrelid = ic.idxoid
        AND a2.attnum = ic.attpos
) i
JOIN pg_catalog.pg_namespace n ON n.oid = i.relnamespace
JOIN pg_catalog.pg_stats s ON s.schemaname = n.nsname
        AND s.tablename = i.attrelname
        AND s.attname = i.attname
GROUP BY 1,2,3,4,5,6,7,8,9,10,11

```

```

) AS rows_data_stats
) AS rows_hdr_pdg_stats
) AS relation_stats
ORDER BY nspname, tblname, idxname;

```

完全空白的索引页面会被回收供重新利用。但是，如果页面上的索引键已被删除但仍有空间被分配，我们建议定期重新索引。

重新创建索引有助于提高查询效率。您可以通过以下三种方式来重新创建索引，相关说明见下表。

方法	描述	限制
带有 CONCURRENTLY 选项的 CREATE INDEX 和 DROP INDEX	构建新索引并移除旧索引。优化程序通过使用新创建的索引来生成计划，而非使用旧索引。在低峰时段，您可以放弃使用旧索引。	如果使用 CONCURRENTLY 选项，创建索引需要的时间会更长，因为它必须跟踪所有传入的更改。更改被冻结后，过程将标记为完成。
具有 CONCURRENTLY 选项的 REINDEX	在重建过程中锁定写入操作。PostgreSQL 版本 12 及更高版本提供了避免这些锁定的 CONCURRENTLY 选项。	使用 CONCURRENTLY 需要更长的时间来重建索引。
pg_repack 扩展程序	清除表中的膨胀并重建索引。	您必须从连接到数据库的 EC2 实例或本地计算机中运行此扩展程序。

创建新索引

DROP INDEX 和 CREATE INDEX 命令一起使用时，会重建索引：

```

DROP INDEX <index_name>
CREATE INDEX <index_name> ON TABLE <table_name> (<column1>[,<column2>])

```

这种方法的缺点在于它对表的锁定是独占的，这会在这个操作过程中影响性能。DROP INDEX 命令将获取独占锁，从而阻止对表的读取和写入操作。CREATE INDEX 命令会阻止对表的写入操作。它允许读取操作，但是在创建索引期间，读取操作的成本很高。

重建索引

REINDEX 命令可帮助您保持一致的数据库性能。对表执行大量 DML 操作时，会导致表和索引膨胀。索引用于加快对表的查找速度，从而提高查询效率。索引膨胀会影响查找和查询性能。因此，我们建议您对那些执行大量 DML 操作的表进行重建索引，以确保查询的性能一致性。

REINDEX 命令通过锁定底层表的写入操作从头开始重建索引，但它允许对表进行读取操作。可这确实会阻止对索引的读取操作。使用相应索引的查询会被阻止，但其他查询则不会。

PostgreSQL 版本 12 引入了一个新的可选参数 CONCURRENTLY，该参数从头开始重建索引，但不会锁定表的写入或读取操作，也不会锁定使用该索引的查询操作。但是，如果使用这个选项，完成整个过程所需的时间会更长。

示例

创建和删除索引

使用 CONCURRENTLY 选项创建新索引：

```
create index CONCURRENTLY on table(columns) ;
```

使用 CONCURRENTLY 选项删除旧索引：

```
drop index CONCURRENTLY <index name> ;
```

重建索引

要重建单个索引，请执行以下操作：

```
reindex index <index name> ;
```

要重建表中的所有索引，请执行以下操作：

```
reindex table <table name> ;
```

要重建架构中的所有索引，请执行以下操作：

```
reindex schema <schema name> ;
```

同时重建索引

要重建单个索引，请执行以下操作：

```
reindex index CONCURRENTLY <indexname> ;
```

要重建表中的所有索引，请执行以下操作：

```
reindex table CONCURRENTLY <tablename> ;
```

要重建架构中的所有索引，请执行以下操作：

```
reindex schema CONCURRENTLY <schemaname> ;
```

仅重建或重新定位索引

要重建单个索引，请执行以下操作：

```
pg_repack -h <hostname> -d <dbname> -i <indexname> -k
```

要重建所有索引，请执行以下操作：

```
pg_repack -h <hostname> -d <dbname> -x <indexname> -t <tablename> -k
```

示例：使用 autovacuum 和 VACUUM FULL 功能来回回收空间

例如，让我们创建一个包含 500000 行的 emp 表格，然后用新值更新这些行。Autovacuum 已启用，因此它将在此表上同时运行 VACUUM 和 ANALYZE 命令来移除膨胀并回收空间。回收的空间可以再次使用，但不会被归还给操作系统。

以下查询用于确定表中的膨胀情况：

```
-- WARNING: When run with a non-superuser role, the query inspects only indexes on
tables you are granted to read.
-- WARNING: Rows with is_na = 't' are known to have bad statistics ("name" type is not
supported).
-- This query is compatible with PostgreSQL 8.2 and later.
SELECT current_database(), nspname AS schemaname, tblname, idxname,
bs*(relpages)::bigint AS real_size,
bs*(relpages-est_pages)::bigint AS extra_size,
100 * (relpages-est_pages)::float / relpages AS extra_pct,
fillfactor,
CASE WHEN relpages > est_pages_ff
THEN bs*(relpages-est_pages_ff)
ELSE 0
END AS bloat_size,
100 * (relpages-est_pages_ff)::float / relpages AS bloat_pct,
is_na
-- , 100-(pst).avg_leaf_density AS pst_avg_bloat, est_pages, index_tuple_hdr_bm,
maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, reltuples, relpages -- (DEBUG
INFO)
FROM (
SELECT coalesce(1 +
ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0
-- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
) AS est_pages,
coalesce(1 +
ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/
(100*(4+nulldatahdrwidth)::float))), 0
) AS est_pages_ff,
bs, nspname, tblname, idxname, relpages, fillfactor, is_na
-- , pgstatindex(idxoid) AS pst, index_tuple_hdr_bm, maxalign, pagehdr,
nulldatawidth, nulldatahdrwidth, reltuples -- (DEBUG INFO)
FROM (
SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, idxoid,
fillfactor,
```

```

        ( index_tuple_hdr_bm +
          maxalign - CASE -- Add padding to the index tuple header to align on
MAXALIGN
            WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
            ELSE index_tuple_hdr_bm%maxalign
          END
        + nulldatawidth + maxalign - CASE -- Add padding to the data to align on
MAXALIGN
            WHEN nulldatawidth = 0 THEN 0
            WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
            ELSE nulldatawidth::integer%maxalign
          END
        )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
        -- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
FROM (
  SELECT n.nspname, i.tblname, i.idxname, i.reltuples, i.relpages,
         i.idxoid, i.fillfactor, current_setting('block_size')::numeric AS bs,
         CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
           WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|
amd64' THEN 8
           ELSE 4
         END AS maxalign,
         /* per page header, fixed size: 20 for 7.X, 24 for others */
         24 AS pagehdr,
         /* per page btree opaque data */
         16 AS pageopqdata,
         /* per tuple header: add IndexAttributeBitMapData if some cols are null-
able */
         CASE WHEN max(coalesce(s.null_frac,0)) = 0
           THEN 8 -- IndexTupleData size
           ELSE 8 + (( 32 + 8 - 1 ) / 8) -- IndexTupleData size +
IndexAttributeBitMapData size ( max num filed per index + 8 - 1 /8)
         END AS index_tuple_hdr_bm,
         /* data len: we remove null values save space using it fractionnal part
from stats */
         sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS
nulldatawidth,
         max( CASE WHEN i.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0
END ) > 0 AS is_na
  FROM (
    SELECT ct.relname AS tblname, ct.relnamespace, ic.idxname, ic.attpos,
           ic.indkey, ic.indkey[ic.attpos], ic.reltuples, ic.relpages, ic.tbloid, ic.idxoid,
           ic.fillfactor,

```

```

        coalesce(a1.attnum, a2.attnum) AS attnum, coalesce(a1.attname,
a2.attname) AS attname, coalesce(a1.atttypid, a2.atttypid) AS atttypid,
        CASE WHEN a1.attnum IS NULL
        THEN ic.idxname
        ELSE ct.relname
        END AS attrelname
    FROM (
        SELECT idxname, reltuples, relpages, tbloid, idxoid, fillfactor,
indkey,
        pg_catalog.generate_series(1,indnatts) AS attpos
    FROM (
        SELECT ci.relname AS idxname, ci.reltuples, ci.relpages,
i.indrelid AS tbloid,
        i.indexrelid AS idxoid,
        coalesce(substring(
            array_to_string(ci.reloptions, ' ')
            from 'fillfactor=([0-9]+)')::smallint, 90) AS fillfactor,
        i.indnatts,
        pg_catalog.string_to_array(pg_catalog.textin(
            pg_catalog.int2vectorout(i.indkey)), ' ')::int[] AS indkey
    FROM pg_catalog.pg_index i
    JOIN pg_catalog.pg_class ci ON ci.oid = i.indexrelid
    WHERE ci.relam=(SELECT oid FROM pg_am WHERE amname = 'btree')
    AND ci.relpages > 0
    ) AS idx_data
    ) AS ic
    JOIN pg_catalog.pg_class ct ON ct.oid = ic.tbloid
    LEFT JOIN pg_catalog.pg_attribute a1 ON
        ic.indkey[ic.attpos] <> 0
        AND a1.attrelid = ic.tbloid
        AND a1.attnum = ic.indkey[ic.attpos]
    LEFT JOIN pg_catalog.pg_attribute a2 ON
        ic.indkey[ic.attpos] = 0
        AND a2.attrelid = ic.idxoid
        AND a2.attnum = ic.attpos
    ) i
    JOIN pg_catalog.pg_namespace n ON n.oid = i.relnamespace
    JOIN pg_catalog.pg_stats s ON s.schemaname = n.nspname
        AND s.tablename = i.attrelname
        AND s.attname = i.attname
    GROUP BY 1,2,3,4,5,6,7,8,9,10,11
    ) AS rows_data_stats
    ) AS rows_hdr_pdg_stats
) AS relation_stats

```

```
ORDER BY nspname, tblname, idxname;
```

查询结果显示，该表的膨胀率约为 51%：

```
current_database | schemaname | tblname | real_size | extra_size | extra_pct |
fillfactor | bloat_size | bloat_pct | is_na
-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
apgl | public | emp | 60383232 | 30744576 | 50.91575091575091 | 100 | 30744576 |
50.91575091575091 | f
```

以下是 pg_stat_all_tables 视图中的统计数据：

```
relid          | 914748
schemaname     | public
relname        | emp
seq_scan       | 5
seq_tup_read   | 1500000
idx_scan       | 0
idx_tup_fetch  | 0
n_tup_ins      | 600000
n_tup_upd      | 500000
n_tup_del      | 0
n_tup_hot_upd  | 0
n_live_tup     | 500000
n_dead_tup     | 0
n_mod_since_analyze | 0
last_vacuum    |
last_autovacuum | 2023-04-15 11:59:54.957449+00
last_analyze   |
last_autoanalyze | 2023-04-15 11:59:55.016352+00
vacuum_count   | 0
autovacuum_count | 2
analyze_count  | 0
autoanalyze_count | 3
```

请注意，autovacuum 在运行后更新了 last_autovacuum 和 last_autoanalyze 列。

现在，让我们在表格中插入一些行并检查 extra_size(bloat_size)，因为空白空间也被视为膨胀。

```
apgl=> select count(*) from emp;
```

```

count | 900000

current_database | schemaname | tblname | real_size | extra_size | extra_pct |
fillfactor | bloat_size | bloat_pct | is_na
-----+-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----
apgl | public | emp | 61349888 | 327680 | 0.5341167044999332 | 100 | 327680 |
0.5341167044999332 | f
(1 row)

```

输出中的 `bloat_pct` 列表示清理后的空间已被新插入的行占用。让我们来运行 `VACUUM FULL`：

```

apgl=> vacuum full emp ;
VACUUM

current_database | schemaname | tblname | real_size | extra_size | extra_pct |
fillfactor | bloat_size | bloat_pct | is_na
-----+-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----
apgl | public | emp | 60792832 | -229376 | 0 | 100 | 0 | 0 | f
(1 row)

```

从这个输出中，您可以看到空白空间和膨胀部分已被清除，空间已归还给操作系统。

Note

您可以运行 `pg_repack` 以获取同样的结果，而不是 `VACUUM FULL`。

资源

- [了解 Amazon RDS for PostgreSQL AWS 环境中的自动真空](#) (博客文章)
- [自动清理](#) (PostgreSQL 文档)
- [为 autovacuum 分配内存](#) (Amazon RDS 文档)
- [防止交易 ID 环绕失败](#) ; (PostgreSQL 文档)
- [使用 pg_repack 移除 Amazon Aurora 和 RDS for PostgreSQL 中的膨胀](#) (AWS 博客文章)

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	更正了错误，并在 自动清理和分析表 和 使用 pg_repack 移除膨胀 部分中增加了信息。	2025 年 8 月 22 日
更正了 reindex 语法	在 同时重建索引 部分中，更正了 reindex 示例。	2025 年 6 月 30 日
初次发布	—	2023 年 12 月 22 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：

- Refactor/re-architect — 充分利用云原生功能来提高敏捷性、性能和可扩展性，从而移动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 数据库迁移到亚马逊 Aurora PostgreSQL-Compatible 版。
- 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：将本地 Oracle 数据库迁移到 AWS Cloud 中的 Amazon Relational Database Service (Amazon RDS) for Oracle。
- 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。
- 重新托管 (直接迁移)：将应用程序迁移到云，无需进行任何更改即可利用云功能。示例：将本地 Oracle 数据库迁移到 AWS Cloud 中 EC2 实例上的 Oracle。
- 重新放置 (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。您将服务器从本地平台迁移到同一平台的云服务中。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

A2A () Agent-to-Agent

一种支持任务委托和状态转移的代理到代理协作的状态协议。

ABAC

请参阅[基于属性的访问控制](#)。

抽象服务

请参阅[托管服务](#)。

ACID

请参阅[原子性、一致性、隔离性、持久性](#)。

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。它比[主动-被动迁移](#)更灵活，但工作量更大。

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。

座席

一种能够使用工具自主推理、计划和采取行动来实现目标的人工智能系统。

特工行动

在生产环境中大规模构建、测试、部署和运行 AI 代理的操作实践。

聚合函数

一种 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

请参阅[人工智能](#)。

AIOps

请参阅[人工智能运营](#)。

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 [IAM](#)) 文档 [AWS 中的 AB AC](#)。

权威数据来源

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来源复制到其他位置，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

恶意机器人

一种旨在扰乱或伤害个人或组织的[机器人](#)。

BCP

请参阅[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。另请参阅[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。

blue/green 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应用程序版本（蓝色），在另一个环境中运行新应用程序版本（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交互的软件应用程序。有些机器人是有用或有益的，例如在互联网上索引信息的 Web 爬网程序。还有一些被称为恶意机器人的机器人，其目的是扰乱或伤害个人或组织。

僵尸网络

被**恶意软件**感染并受单方（称为僵尸网络控制者或僵尸网络操作者）控制的**僵尸网络**。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）。

紧急（break-glass）访问

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，请参阅指南中的[“实施破碎玻璃程序”](#) AWS Well-Architected 指示器。

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

请参阅 [AWS 云采用框架](#)。

金丝雀部署

缓慢而渐进地向最终用户发布版本。当您确信无误后，即可部署新版本，并完全替换当前版本。

CCoE

请参阅 [云卓越中心](#)。

CDC

请参阅 [更改数据捕获](#)。

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韧性。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

请参阅 [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。

公民开发者

使用无code/low代码平台创建 AI 应用程序但没有专业技术技能的企业用户。

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。云计算通常连接到[边缘计算](#)技术。

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 AWS Cloud 中时通常会经历四个阶段：

- 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
- Re-invention — 优化产品和服务，在云端进行创新

Stephen Orban 在 AWS Cloud 企业战略博客的博客文章 [《走向之旅 Cloud-First 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

请参阅[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 Bitbucket Cloud。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)

一种 [AI](#) 领域，它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，Amazon SageMaker AI 为 CV 提供了图像处理算法。

配置偏移

对于工作负载而言，一种偏离预期状态的配置更改。这可能会导致工作负载变得不合规，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、提高生产力、提高代码质量和更快地交付。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

请参阅[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《AWS Well-Architected 框架》中安全支柱的组成部分。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

请参阅[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。

深度防御

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，深度防御方法可能将多因素身份验证、网络分段和加密结合起来。

委派管理员

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。此帐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

请参阅[环境](#)。

侦测性控制

一种安全控制，在事件发生后进行检测、记录日志和发出提醒。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

[星型架构](#)中的一种较小的表，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)

您用来最大程度地减少由[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 [《工作负载灾难恢复 AWS：AWS Well-Architected 框架中的云端恢复》](#)。

DML

请参阅[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。埃里克·埃文斯 (Eric Evans) 在他的《Domain-Driven 设计：解决软件核心的复杂性》(波士顿：Addison-Wesley 专业版，2003年) 一书中介绍了这个概念。有关如何使用带有 strangler fig 模式的域驱动设计的信息，请参阅使用容器和 [Amazon API Gateway 逐步实现传统微软 ASP.NET \(ASMX\) 网络服务的现代化](#)。

DR

请参阅[灾难恢复](#)。

偏差检测

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

请参阅[开发价值流映射](#)。

E

EDA

请参阅[探索性数据分析](#)。

EDI

请参阅[电子数据交换](#)。

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](#)比较时，边缘计算可以减少通信延迟并缩短响应时间。

电子数据交换 (EDI)

组织之间业务文件的自动交换。有关更多信息，请参阅[什么是电子数据交换](#)。

加密

一种将人类可读的纯文本数据转换为加密文字的计算流程。

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。Big-endian 系统首先存储最重要的字节。Little-endian 系统首先存储最低有效字节。

端点

请参阅[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](#)。

环境

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

请参阅 [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

[星型架构](#) 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。

快速失效机制

一种使用频繁且增量式的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

请参阅 [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。

少样本提示

在要求 [LLM](#) 执行类似任务之前，先向其提供少量示例，以演示任务和预期输出。这种技术是情境学习的应用，模型可以从提示中嵌入的示例 (镜头) 中学习。Few-shot 对于需要特定格式、推理或领域知识的任务，提示可能非常有效。另请参阅 [零样本提示](#)。

FGAC

请参阅 [精细访问控制](#)。

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，通过 [更改数据捕获](#) 使用连续数据复制，在极短的时间内迁移数据，而非使用分阶段方法。目标是将停机时间降至最低。

FM

请参阅 [基础模型](#)。

基础模型 (FM)

一个大型深度学习神经网络，它已使用海量的通用和未标注数据集进行训练。FM 能够执行各种常规任务，例如理解语言、生成文本和图像以及使用自然语言进行对话。有关更多信息，请参阅[什么是基础模型](#)。

FM 网关

一种集中式中介，用于控制和规范对[基础模型](#)的访问。也称为 LLM 网关。

G

生成式人工智能

[AI](#) 模型的一个子集，这些模型已经过大量数据训练，可以使用简单的文本提示来创建新的内容和构件，例如图像、视频、文本和音频。有关更多信息，请参阅[什么是生成式人工智能](#)。

地理阻止

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工作流程，而[基于中继的工作流程](#)则是现代的、首选的方法。

黄金映像

系统或软件的快照，用作部署该系统或软件的新实例的模板。例如，在制造业中，黄金映像可用于在多个设备上预调配软件，并有助于提高设备制造操作的速度、可扩展性和生产效率。

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。侦测性护栏会检测策略违规和合规性问题，并生成提醒以进行修复。它们通过使用 AWS Config、Amazon、AWS Security Hub CSPM GuardDuty AWS Trusted Advisor、Amazon Inspector 和自定义 AWS Lambda 支票来实现。

护栏 (AI)

用于过滤、验证和限制[代理](#)输入和输出的安全机制，有助于确保负责任和安全的 AI 行为。

H

HA

请参阅[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。

保留数据

从用于训练[机器学习](#)模型的数据集中保留的一部分标注的历史数据。通过将模型预测与保留数据进行比较，您可以使用保留数据来评估模型性能。

人机在圈 (HitL)

一种工作流程模式，其中[代理](#)执行在关键决策点暂停以供人工审查和批准。

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。

我

laC

请参阅[基础设施即代码](#)。

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

请参阅[工业物联网](#)。

不可变基础设施

一种模型，可为生产工作负载部署新的基础设施，而不是更新、修补或修改现有基础设施。不可变基础设施本质上比[可变基础设施](#)更一致、更可靠、更可预测。有关更多信息，请参阅框架中的[使用不可变基础架构部署](#)最佳实践。AWS Well-Architected

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 \(Klaus Schwab \)](#)在2016年推出，指的是通过连接性、实时数据、自动化、分析和的进步实现制造流程的现代化。AI/ML

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调配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[机器学习模型的可解释性 AWS](#)。

物联网

请参阅[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](#)。

大语言模型 (LLM)

一种基于大量数据进行预训练的深度学习 [AI](#) 模型。LLM 可以执行多项任务，例如回答问题、总结文档、将文本翻译成其他语言以及完成句子。有关更多信息，请参阅[什么是 LLM](#)。

大规模迁移

迁移 300 台或更多服务器。

LBAC

请参阅[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

请参阅 [7 R](#)。

小端序系统

一个先存储最低有效字节的系统。另请参阅[字节顺序](#)。

LLM

请参阅[大型语言模型](#)。

下层环境

请参阅[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

请参阅[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权限。恶意软件的示例包括病毒、蠕虫、勒索软件、木马、间谍软件和键盘记录器。

托管式服务

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。Amazon Simple Storage Service (Amazon S3) 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、记录和控制将原材料转化为成品的生产过程。

MAP

请参阅[迁移加速计划](#)。

MCP

参见[模型上下文协议](#)。

模型上下文协议 (MCP)

一种用于[代理](#)与[工具](#)通信的无状态协议。

MCP 服务器

一种通过[模型上下文协议](#)公开一个或多个[工具](#)的服务。

机制

一个完整的过程，您可以在其中创建工具，推动工具的采用，然后检查结果以进行调整。机制是一种在运作过程中自我强化和改善的循环。有关更多信息，请参阅在 AWS Well-Architected 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

请参阅[制造执行系统](#)。

消息队列遥测传输 (MQTT)

[一种基于publish/subscribe模式的轻量级机器对机器 \(M2M\) 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

Cross-functional 通过自动化、敏捷的方法简化工作负载迁移的团队。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，提供了用于验证迁移到 AWS Cloud 的业务案例的信息。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

将工作负载迁移到 AWS Cloud 的方法。有关更多信息，请参见术语表中的 [7 R](#) 词条，以及[动员您的组织以加快大规模迁移](#)。

ML

请参阅[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 and 利用创新。有关更多信息，请参阅[在 AWS Cloud 中实现应用程序现代化的策略](#)。

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[在 AWS Cloud 中评估应用程序的现代化准备情况](#)。

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

请参阅[迁移组合评测](#)。

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？”

可变基础设施

一种用于更新和修改生产工作负载的现有基础设施的模型。为了提高一致性、可靠性和可预测性，该 AWS Well-Architected 框架建议使用[不可变基础设施](#)作为最佳实践。

O

OAC

请参阅[来源访问控制](#)。

OAI

请参阅[来源访问身份](#)。

OCM

请参阅[组织变革管理](#)。

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

请参阅[运营集成](#)。

OLA

请参阅[运营级别协议](#)。

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

请参阅[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机器对机器 (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)

一份问题核对清单和关联的最佳实践，可帮助您了解、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 AWS Well-Architected 框架中的[运营准备情况审查 \(ORR\)](#)。

运营技术 (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](#) 转型的关键重点。

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。

ORR

请参阅[运营准备情况审查](#)。

OT

请参阅[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

请参阅[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

请参阅[可编程逻辑控制器](#)。

PLM

请参阅[产品生命周期管理](#)。

policy

一个对象，可以定义权限（请参阅[基于身份的策略](#)）、指定访问条件（请参阅[基于资源的策略](#)）或定义 AWS Organizations 的组织中所有账户的最大权限（请参阅[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。

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 false 的查询条件，通常位于 WHERE 子句中。

谓词下推

一种数据库查询优化技术，可在传输之前筛选查询中的数据。这将减少从关系数据库检索和处理的数据量，并提高查询性能。

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

一种在整个开发过程中都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](#)，旨在防止部署不合规资源。这些控制会在资源预置之前对其进行扫描。如果资源与控制不兼容，则不会预置它。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)

对产品在其整个生命周期内的数据和流程的管理，从设计、开发和发布，到增长和成熟，再到衰退和淘汰。

生产环境

请参阅[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。

提示串接

使用一个 [LLM](#) 提示的输出作为下一个提示的输入，以生成更好的响应。该技术用于将复杂的任务分解为子任务，或者迭代地完善或扩展初步响应。它有助于提高模型响应的准确性和相关性，并允许获得更精细的个性化结果。

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。

publish/subscribe (pub/sub)

一种支持微服务间异步通信的模式，可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

一系列用于访问 SQL 关系数据库系统中的数据的步骤，类似于指令。

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

请参阅[责任、问责、咨询和知情 \(RACI \)](#)。

RAG

请参阅[检索增强生成](#)。

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

请参阅[责任、问责、咨询和知情 \(RACI \)](#)。

RCAC

请参阅[行列访问控制](#)。

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

请参阅 [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

请参阅 [7 R](#)。

Region

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[指定您的账户可以使用的 AWS 区域](#)。

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

请参阅 [7 R](#)。

版本

在部署过程中，推动生产环境变更的行为。

重新放置

请参阅 [7 R](#)。

更换平台

请参阅 [7 R](#)。

重新购买

请参阅 [7 R](#)。

韧性

应用程序抵御中断或从中断中恢复的能力。在 AWS Cloud 中规划韧性时，[高可用性](#)和[灾难恢复](#)是常见的考虑因素。有关更多信息，请参阅 [AWS Cloud 韧性](#)。

基于资源的策略

一种附加到资源的策略，例如 Amazon S3 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

请参阅 [7 R](#)。

停用

请参阅 [7 R](#)。

检索增强生成 (RAG)

一种[生成式人工智能](#)技术，其中 [LLM](#) 在生成响应之前引用其训练数据来源之外的权威数据来源。例如，RAG 模型可以对组织的知识库或自定义数据执行语义搜索。有关更多信息，请参阅[什么是 RAG](#)。

轮换

定期更新[密钥](#)以使攻击者更难访问凭证的过程。

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

请参阅[恢复点目标](#)。

RTO

请参阅[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 管理控制台 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

请参阅[监督控制和数据采集](#)。

SCP

请参阅[服务控制策略](#)。

机密密钥

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 Secrets Manager 文档中的[什么是 Amazon Secrets Manager 密钥？](#)。

安全设计

一种在整个开发过程中都考虑安全的系统工程方法。

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。安全控制有以下四种类型：[预防性](#)、[检测性](#)、[响应性](#)和[主动性](#)。

安全固化

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

一种预定义的程序化操作，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 实例或轮换凭证。

服务器端加密

由接收数据的人在目的地对数据 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。

服务水平指示器 (SLI)

对服务性能方面的衡量，例如错误率、可用性或吞吐量。

服务水平目标 (SLO)

代表服务运行状况的目标指标，由[服务水平指示器](#)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](#)。

暗影人工智能

在组织内受管控渠道之外构建或使用的未经授权的 [AI](#) 应用程序。

SIEM

请参阅[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

请参阅[服务水平协议](#)。

SLI

请参阅[服务水平指示器](#)。

SLO

请参阅[服务水平目标](#)。

split-and-seed 模式

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[在 AWS Cloud 中实现应用程序现代化的分阶段方法](#)。

SPOF

请参阅[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事务数据或测量数据，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 逐步实现传统微软 ASP.NET \(ASMX\) 网络服务的现代化](#)。

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。

监督控制和数据采集 (SCADA)

在制造业中，一种使用硬件和软件来监控实物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。您可以使用 [Amazon S CloudWatch ynthetic](#) 来创建这些测试。

系统提示

一种为 [LLM](#) 提供上下文、说明或准则以指导其行为的技术。系统提示有助于设置上下文并制定与用户交互的规则。

T

标签

Key-value 对充当用于组织 AWS 资源的元数据。标签有助于您管理、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

请参阅[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。

工具

[代理](#)可以调用以在外部系统中执行操作的函数或 API。

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，该服务可以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。

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

请参阅[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

一种对与当前记录有某种关联的一组行执行计算的 SQL 函数。窗口函数对于处理任务很有用，例如计算移动平均值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。

WORM

请参阅[一次写入多次读取](#)。

WQF

请参阅[AWS 工作负载资格鉴定框架](#)。

一次写入多次读取 (WORM)

一种存储模型，可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，但无法对其进行更改。此数据存储基础设施被认为[不可变](#)。

Z

零日漏洞利用

一种利用[零日漏洞](#)的攻击，通常为恶意软件。

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。

零样本提示

为[LLM](#)提供执行任务的说明，但没有可以帮助指导的示例（样本）。LLM 必须使用预先训练的知识来处理任务。零样本提示的有效性取决于任务的复杂性和提示的质量。另请参阅[少样本提示](#)。

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。