为多 DevOps 账户环境选择 Git 分支策略

AWS 规范性指导

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS 规范性指导: 为多 DevOps 账户环境选择 Git 分支策略

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

简介 1
目标 1
使用 CI/CD 实践 2
了解 DevOps环境 3
沙盒环境 3
访问 4
生成步骤 4
部署步骤 4
迁移到开发环境之前的期望 4
开发环境 5
访问 4
生成步骤 4
部署步骤 4
迁移到测试环境之前的期望 6
测试环境 6
访问 4
生成步骤 4
部署步骤 4
迁移到暂存环境之前的期望 7
暂存环境 7
访问 4
生成步骤 4
部署步骤 4
迁移到生产环境之前的期望 8
生产环境 8
访问 4
生成步骤 4
部署步骤 4
基于 Git 的开发的最佳实践 10
Git 分支策略 11
树干分支策略 11
Trunk 策略的直观概述 12
主干中的分支策略 13
中继策略的优缺点 15

GitHub 流量分支策略 17
GitHub Flow 策略的可视化概述 17
GitHub Flow 策略中的分支 18
GitHub Flow 策略的优缺点 20
Gitflow 分支策略 22
Gitflow 策略的可视化概述 22
Gitflow 策略中的分支 24
Gitflow 策略的优缺点 27
后续步骤 29
资源 30
AWS 规范性指导 30
其他 AWS 指导 30
其他资源 30
贡献者 32
编写 32
正在审阅 32
技术写作 32
文档历史记录 33
术语表 34
..... 34
A 34
B 37
C 38
D 41
E 44
F 46
G 47
H 48
我 49
L 51
M 52
O 56
P 58
Q 60
R 61
S 63

T 66
U 67
V 68
W 68
Z 69
..... lxx

为多 DevOps账户环境选择 Git 分支策略

亚马逊 Web Services（贡献者）

2024 年 2 月（文档历史记录）

转向基于云的方法并提供软件解决方案 AWS 可能具有变革性。这可能需要更改您的软件开发生命周期流程。通常， AWS 账户 在开发过程中会使用多个 AWS Cloud。选择兼容的 Git 分支策略与您的 DevOps 流程配对是成功的关键。为您的组织选择正确的 Git 分支策略可以帮助您在开发团队之间简洁地传达 DevOps 标准和最佳实践。Git 分支在单个环境中可能很简单，但是当应用于多个环境（例如沙盒、开发、测试、暂存和生产环境）时，它可能会变得混乱。拥有多个环境会增加 DevOps实施的复杂性。

本指南提供了 Git 分支策略的可视化图表，展示了组织如何实现多账户流程 DevOps 。可视化指南可帮助团队了解如何将 Git 分支策略与 DevOps 实践相结合。使用标准分支模型（例如 Gitflow、Fl GitHub ow 或 Trunk）来管理源代码存储库可以帮助开发团队协调工作。这些团队还可以使用互联网上的标准 Git 培训资源来理解和实施这些模型和策略。

有关 DevOps 的最佳实践 AWS，请查看《Well-Architect AWS ed》中的DevOps指南。在阅读本指南时，请使用尽职调查为您的组织选择正确的分支策略。有些策略可能比其他策略更适合您的用例。

目标

本指南是关于为拥有多个 AWS 账户分支机构的组织选择和实施 DevOps分支策略的文档系列的一部分。本系列旨在帮助您从一开始就应用最符合您的要求、目标和最佳实践的策略，以简化您的体验 AWS Cloud。本指南不包含 DevOps 可执行脚本，因为它们因您的组织使用的持续集成和持续交付 (CI/CD) 引擎和技术框架而异。

本指南解释了三种常见的 Git 分支策略之间的区别： GitHub Flow、Gitflow 和 Trunk。本指南中的建议可帮助团队确定与其组织目标相一致的分支策略。阅读本指南后，您应该能够为组织选择分支策略。选择策略后，您可以使用以下模式之一来帮助您与开发团队一起实施该策略：

- 为多 DevOps 账户环境实施中继分支策略
- 为多 DevOps 账户环境实施 GitHub Flow 分支策略
- 为多账户环境实施 Gitflow 分支策略 DevOps

值得注意的是，对一个组织、团队或项目有效的方法可能不适合其他组织、团队或项目。Git 分支策略之间的选择取决于各种因素，例如团队规模、项目要求以及协作、集成频率和发布管理之间的理想平衡。

使用 CI/CD 实践

AWS 建议您实施持续集成和持续交付（CI/CD), which is the process of automating the software release lifecycle. It automates much or all of the manual DevOps processes that are traditionally required to get new code from development into production. A CI/CD pipeline encompasses the sandbox, development, testing, staging, and production environments. In each environment, the CI/CD pipeline provisions any infrastructure that is needed to deploy or test the code. By using CI/CD, development teams can make changes to code that are then automatically tested and deployed. CI/CD管道还为开发团队提供治理和护栏。他们强制执行功能接受和部署的一致性、标准、最佳实践和最低接受水平。有关更多信息，请参阅上的 “练习持续集成和持续交付” AWS。

本指南中讨论的所有分支策略都非常适合为开发团队CI/CD practices. The complexity of the CI/CD pipeline increases with the complexity of the branching strategy. For example, Gitflow is the most complex branching strategy discussed in this guide. CI/CD pipelines for this strategy require more steps (such as for compliance reasons), and they must support multiple, simultaneous production releases. Using CI/CD also becomes more important as the complexity of the branching strategy increases. This is because CI/CD建立护栏和机制，以防止开发人员有意或无意地绕过定义的流程。

AWS 提供了一套开发人员服务，旨在帮助您构建 CI/CD 管道。例如，AWS CodePipeline是一项完全托管的持续交付服务，可帮助您实现发布管道的自动化，从而实现快速可靠的应用程序和基础设施更新。 AWS CodeBuild编译源代码、运行测试和生成 ready-to-deploy软件包。有关更多信息，请参阅上的开发者工具 AWS。

了解 DevOps环境

要了解分支策略，您必须了解每种环境中发生的目的和活动。建立多个环境可以帮助您将开发活动分成几个阶段，监控这些活动，并防止无意中发布未经批准的功能。每个环境 AWS 账户 中都可以有一个或多个。

大多数组织都概述了几种可供使用的环境。但是，环境的数量可能因组织和软件开发策略而异。本文档系列假设您有以下五个跨越开发管道的常见环境，尽管它们的名称可能不同：

- 沙箱 — 开发人员编写代码、犯错误和执行概念验证工作的环境。
- 开发 — 开发人员可以在其中集成其代码以确认所有代码均可作为一个统一的应用程序运行的环境。
- 测试 — 进行 QA 团队或验收测试的环境。团队经常在这种环境中进行性能或集成测试。
- 暂存 — 一种预生产环境，您可以在其中验证代码和基础架构在生产等效环境下是否按预期运行。此环境被配置为尽可能与生产环境相似。
- 生产 — 一种处理来自最终用户和客户的流量的环境。

本节详细描述了每种环境。它还描述了每个环境的构建步骤、部署步骤和退出标准，以便您可以继续下一个环境。下图按顺序显示了这些环境。

本节中的主题：

- 沙盒环境
- 开发环境
- 测试环境
- 暂存环境
- 生产环境

沙盒环境

沙盒环境是开发人员编写代码、犯错误和执行概念验证工作的地方。您可以从本地工作站或通过本地工作站上的脚本部署到沙盒环境。

访问

开发人员应该拥有对沙盒环境的完全访问权限。

生成步骤

当开发人员准备好将更改部署到沙盒环境时，他们会在本地工作站上手动运行构建。

1. 使用 git-sec rets (GitHub) 扫描敏感信息
2. 整理源代码
3. 生成并编译源代码（如果适用）
4. 执行单元测试
5. 执行代码覆盖率分析
6. 执行静态代码分析
7. 以代码形式构建基础架构 (IaC)
8. 执行 IaC 安全分析
9. 提取开源许可证
10. 发布构建工件

部署步骤

如果您使用的是 Gitflow 或 Trunk 模型，则在沙盒环境中成功构建`feature`分支后，部署步骤会自动启动。如果您使用的是 GitHub Flow 模型，则需要手动执行以下部署步骤。以下是沙盒环境中的部署步骤：

1. 下载已发布的文物
2. 执行数据库版本控制
3. 执行 IaC 部署
4. 执行集成测试

迁移到开发环境之前的期望

- 在沙盒环境中成功构建`feature`分支
- 开发人员已在沙盒环境中手动部署并测试了该功能

开发环境

开发环境是开发人员将他们的代码集成在一起的地方，以确保所有代码都作为一个有凝聚力的应用程序运行。在 Gitflow 中，开发环境包含合并请求中包含的最新功能，并且已准备好发布。在 GitHub Flow 和 Trunk 策略中，开发环境被视为测试环境，代码库可能不稳定，不适合部署到生产环境。

访问

根据最小权限原则分配权限。最低权限是授予执行任务所需的最低权限的安全最佳实践。开发人员对开发环境的访问权限应少于对沙盒环境的访问权限。

生成步骤

向`develop`分支（Gitflow）或分支（Trunk 或 Fl GitHub ow）创建合并请求会自动开始构建。`main`

1. 使用 git-sec rets (GitHub) 扫描敏感信息
2. 整理源代码
3. 生成并编译源代码（如果适用）
4. 执行单元测试
5. 执行代码覆盖率分析
6. 执行静态代码分析
7. 构建 IaC
8. 执行 IaC 安全分析
9. 提取开源许可证

部署步骤

如果您使用的是 Gitflow 模型，则在开发环境中成功构建`develop`分支后，部署步骤会自动启动。如果您使用的是 GitHub Flow 模型或 Trunk 模型，则在针对`main`分支创建合并请求时，部署步骤会自动启动。以下是开发环境中的部署步骤：

1. 从构建步骤中下载已发布的工件
2. 执行数据库版本控制
3. 执行 IaC 部署

4. 执行集成测试

迁移到测试环境之前的期望

- 在开发环境中成功构建和部署`develop`分支 (Gitflow) 或`main`分支（Trunk 或 GitHub Flow）
- 单元测试以 100% 的比率通过
- 成功构建 IaC
- 部署对象已成功创建
- 开发人员已执行手动验证，以确认该功能按预期运行

测试环境

质量保证 (QA) 人员使用测试环境来验证功能。他们在完成测试后批准更改。当他们批准后，分支机构就会进入下一个环境，即暂存环境。在 Gitflow 中，此环境及其以上的其他环境只能从`release`分支部署。分`release`支基于包含计划功能的`develop`分支。

访问

根据最小权限原则分配权限。开发人员对测试环境的访问权限应少于对开发环境的访问权限。QA 人员需要足够的权限才能测试该功能。

生成步骤

此环境中的构建过程仅适用于使用 Gitflow 策略时的错误修复。向`bugfix`分支创建合并请求会自动开始构建。

1. 使用 git-sec rets (GitHub) 扫描敏感信息
2. 整理源代码
3. 生成并编译源代码（如果适用）
4. 执行单元测试
5. 执行代码覆盖率分析
6. 执行静态代码分析
7. 构建 IaC
8. 执行 IaC 安全分析
9. 提取开源许可证

部署步骤

在开发环境中部署后，在测试环境中自动启动release分main支 (Gitfl GitHub ow) 或分支（Trunk 或 Flow）的部署。以下是测试环境中的部署步骤：

1. 在测试环境中部署release分支（Gitflow）或main分支（Trunk 或 GitHub Flow）
2. 暂停以供指定人员手动批准
3. 下载已发布的文物
4. 执行数据库版本控制
5. 执行 IaC 部署
6. 执行集成测试
7. 执行性能测试
8. 质量保证批准

迁移到暂存环境之前的期望

- 开发和 QA 团队已经进行了充分的测试，足以满足贵组织的要求。
- 开发团队已通过bugfix分支解决了所有发现的错误。

暂存环境

暂存环境配置为与生产环境相同。例如，数据设置的范围和大小应与生产工作负载相似。使用暂存环境来验证代码和基础架构是否按预期运行。此环境也是业务用例（例如预览或客户演示）的首选。

访问

根据最小权限原则分配权限。开发人员对暂存环境的访问权限应与对生产环境的访问权限相同。

生成步骤

无。在测试环境中使用的相同工件将在暂存环境中重复使用。

部署步骤

在测试环境中获得批准和部署后，自动在暂存环境中启动main分release支 (Git GitHub flow) 或分支（Trunk 或 Flow）的部署。以下是暂存环境中的部署步骤：

1. 在暂存环境中部署release分支（Gitflow）或main分支（Trunk 或 GitHub Flow）
2. 暂停以供指定人员手动批准
3. 下载已发布的文物
4. 执行数据库版本控制
5. 执行 IaC 部署
6. （可选）执行集成测试
7. （可选）执行负载测试
8. 获得所需的开发、QA、产品或业务审批者的批准

迁移到生产环境之前的期望

- 已成功将生产等效版本部署到暂存环境中
- （可选）集成和负载测试成功

生产环境

生产环境支持发布的产品，处理真实客户的真实数据。这是一个受保护的环境，通过最低权限分配访问权限，只有在有限的时间内通过审核的例外流程才能允许使用更高的访问权限。

访问

在生产环境中，开发人员应在 AWS 管理控制台中拥有有限的只读访问权限。例如，开发人员应该能够访问 day-to-day 操作的日志数据。所有发布到生产环境的版本都应在部署之前通过批准步骤进行限制。

生成步骤

无。在测试和暂存环境中使用的相同工件可在生产环境中重复使用。

部署步骤

在暂存环境中获得批准和部署后，自动启动在生产环境中部署main分release支 (Git GitHub flow) 或分支（Trunk 或 Flow）。以下是生产环境中的部署步骤：

1. 在生产环境中部署release分支（Gitflow）或main分支（Trunk 或 GitHub Flow）
2. 暂停以供指定人员手动批准

3. 下载已发布的文物
4. 执行数据库版本控制
5. 执行 IaC 部署

基于 Git 的开发的最佳实践

要成功采用基于 Git 的开发，必须遵循一套最佳实践，以促进协作、维护代码质量并支持持续集成和持续交付 (CI/CD)。除了本指南中的最佳实践外，还请查看《Well-Architect AWS ed 指南》DevOps 。以下是基于 Git 的开发的一些关键最佳实践： AWS

- 保持少量和频繁的更改 — 鼓励开发人员提交较小的增量更改或功能。这样可以降低合并冲突的风险，并且可以更轻松地快速识别和修复问题。
- 使用功能切换-要管理不完整或实验性功能的发布，请使用功能切换或功能标志。这可以帮助您在生产环境中隐藏、启用或禁用特定功能，而不会影响主分支的稳定性。
- 维护强大的测试套件 — 全面、维护良好的测试套件对于及早发现问题并验证代码库是否保持稳定至关重要。投资于测试自动化，并优先修复任何失败的测试。
- 采用持续集成 — 使用持续集成工具和实践自动构建、测试代码变更并将其集成到develop分支（Gitflow）或分支（Trunk 或 main FI GitHub ow）中。这可以帮助您尽早发现问题并简化开发过程。
- 执行代码审查 — 鼓励对代码进行同行评审，以保持质量、共享知识并捕捉潜在问题，然后再将其整合到分main支中。使用拉取请求或其他代码审查工具来简化此过程。
- 监控并修复损坏的构建-当构建中断或测试失败时，请优先尽快修复问题。这样可以使develop分支（Gitflow）或main分支（Trunk 或 GitHub Flow）保持可释放状态，并最大限度地减少对其他开发者的影响。
- 沟通和协作-促进团队成员之间的开放式沟通和协作。确保开发人员知道正在进行的工作和对代码库所做的更改。
- 持续重构 — 定期重构代码库以提高其可维护性并减少技术债务。鼓励开发人员让代码保持比他们发现的更好的状态。
- 使用短期分支执行复杂任务-对于较大或更复杂的任务，请使用短期分支（也称为任务分支）来处理更改。但是，请务必缩短分支寿命，通常少于一天。尽快将更改合并回分develop支（Gitflow）或main分支（Trunk 或 GitHub Flow）。对于团队来说，规模更小、更频繁的合并和审阅比一个大型合并请求更容易使用和处理。
- 培训和支持团队 — 为刚接触基于 Git 的开发或在采用基于 Git 的最佳实践方面需要指导的开发人员提供培训和支持。

Git 分支策略

本指南按从最少到最复杂的顺序详细描述了以下基于 Git 的分支策略：

- Trun@@ k — 基于 Trunk 的开发是一种软件开发实践，其中所有开发人员都在单个分支上工作，通常称为`trunk`或`main`分支。这种方法背后的想法是，通过频繁集成代码更改并依靠自动化测试和持续集成，使代码库保持持续可发布的状态。
- GitHub Flo GitHub w — Flow 是一个基于分支的轻量级工作流程，由开发。 GitHub它基于短寿命`feature`分支的概念。当某项功能完成并准备部署时，该功能将合并到分`main`支中。
- Gitflow — 使用 Gitflow 方法，开发是在各个功能分支中完成的。批准后，您可以将`feature`分支合并到通常命名的集成分支中`develop`。当`develop`分支中积累了足够多的功能时，就会创建一个`release`分支来将这些功能部署到上层环境。

每种分支策略都有优点和缺点。尽管它们都使用相同的环境，但它们并不都使用相同的分支或手动批准步骤。在本指南的这一部分中，详细查看每种分支策略，以便您熟悉其细微差别，并可以评估它是否适合您组织的用例。

本节中的主题：

- 树干分支策略
- GitHub 流量分支策略
- Gitflow 分支策略

树干分支策略

基于主干的开发是一种软件开发实践，其中所有开发人员都在单个分支上工作，通常称为`trunk`或`main`分支。这种方法背后的想法是，通过频繁集成代码更改并依靠自动化测试和持续集成，使代码库保持持续可发布的状态。

在基于主干的开发中，开发人员每天多次向`main`分支提交更改，目标是进行小规模的增量更新。这可以实现快速反馈循环，降低合并冲突的风险，并促进团队成员之间的协作。该实践强调维护良好的测试套件的重要性，因为它依赖于自动测试来尽早发现潜在问题，并确保代码库保持稳定和可发布。

基于主干的开发通常与基于功能的开发（也称为功能分支或功能驱动开发）形成鲜明对比，在后者中，每个新功能或错误修复都是在自己的专用分支中开发的，与主分支是分开的。在基于主干的开发和基于功能的开发之间做出选择取决于团队规模、项目要求以及协作、集成频率和发布管理之间的理想平衡等因素。

有关 Trunk 分支策略的更多信息，请参阅以下资源：

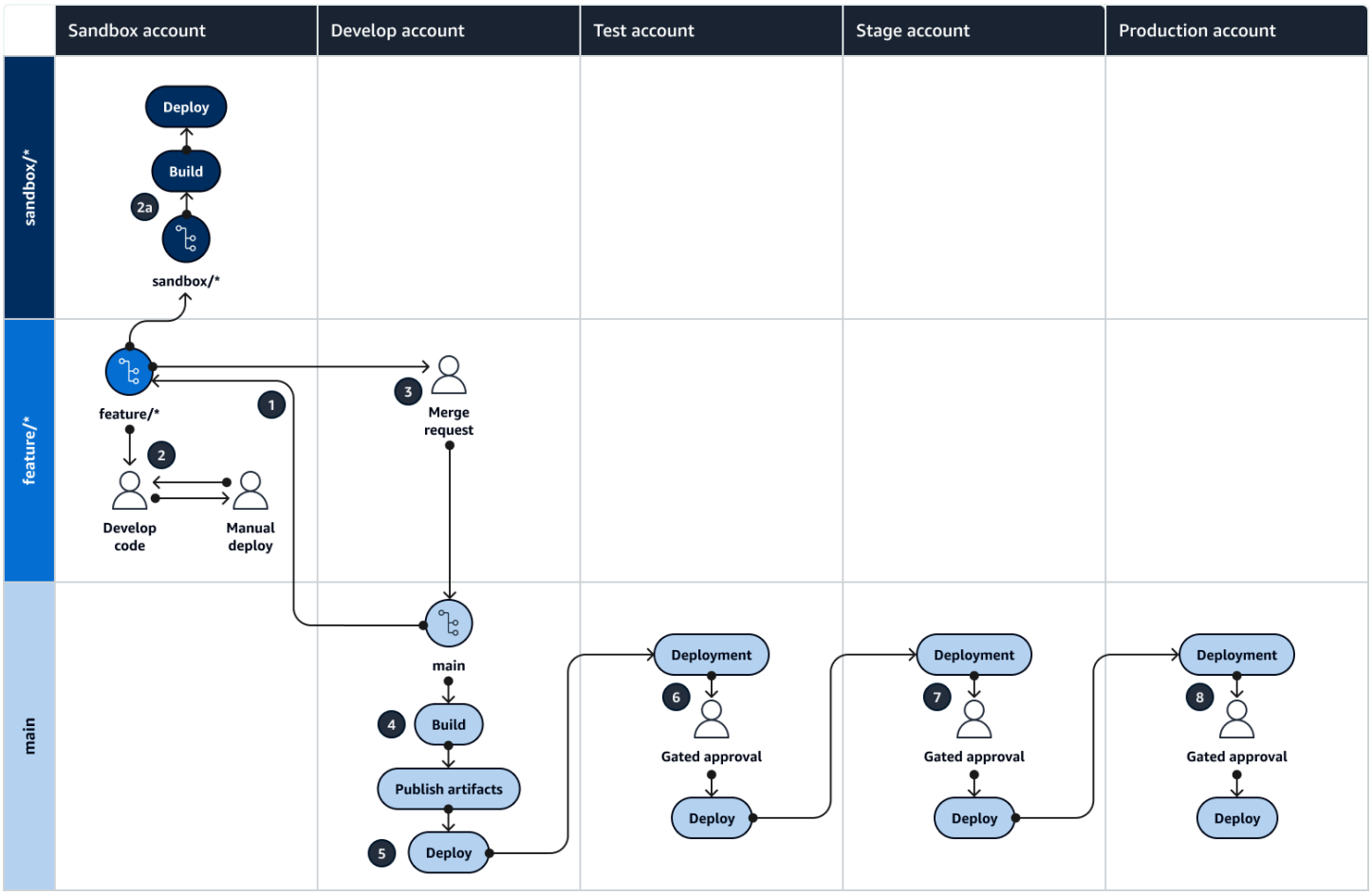
- 为多账户 DevOps 环境实施中继分支策略（AWS 规范性指导）
- 基于@@ 主干的开发简介（基于主干的开发网站）

本节中的主题：

- Trunk 策略的直观概述
- 主干中的分支策略
- 中继策略的优缺点

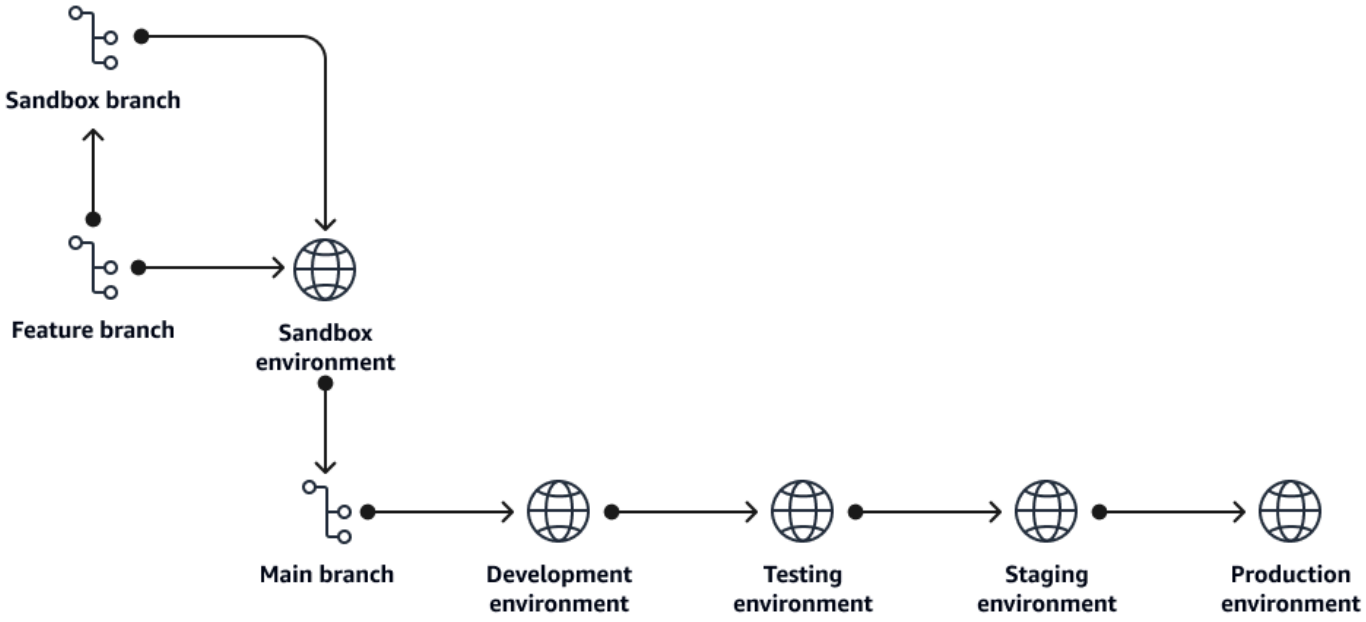
Trunk 策略的直观概述

下图可以像 Punnett 方块（维基百科）一样用来理解主干分支策略。将垂直轴上的分支与水平轴上的 AWS 环境对齐，以确定在每个场景中要执行的操作。带圆圈的数字将引导您完成图中所示的操作顺序。此图显示了 Trunk 分支策略的开发工作流程，从沙盒环境中的`feature`分支到分支的`main`生产版本。有关每种环境中发生的活动的更多信息，请参阅本指南中的DevOps 环境。



主干中的分支策略

中继分支策略通常具有以下分支。



功能分支

您可以在feature分支中开发功能或创建修补程序。要创建feature分支，您需要从该分支中main分支。开发人员在feature分支中迭代、提交和测试代码。某项功能完成后，开发者将对该功能进行推广。从feature分支向前只有两条路径：

- 合并到分sandbox支中
- 向main分支创建合并请求

命名惯例：	`feature/<story number>_<developer initials>_<descriptor>`
命名约定示例：	`feature/123456_MS_Implement_Feature_A`

沙盒分支

这个分支是一个非标准的主干分支，但它对于 CI/CD 管道开发很有用。该sandbox分支主要用于以下目的：

- 使用 CI/CD 管道向沙盒环境执行全面部署
- 在提交合并请求以在较低的环境（例如开发或测试）中进行全面测试之前，先开发和测试管道。

Sandbox分支本质上是临时性的，其寿命是短暂的。应在特定测试完成后将其删除。

命名惯例：	sandbox/<story number>_<developer initials>_<descriptor>
命名约定示例：	sandbox/123456_MS_Test_Pipeline_Deploy

主分支

分main支始终代表在生产环境中运行的代码。代码从中分支main、开发，然后合并回到。main来自的部署main可以针对任何环境。要防止删除，请为分支启用main分支保护。

命名惯例：	main

修补程序分支

基于主干的工作流程中没有专门的hotfix分支。修补程序使用feature分支。

中继策略的优缺点

Trunk 分支策略非常适合规模较小、成熟、具有较强沟通能力的开发团队。如果您为应用程序发布了连续、滚动的功能，它也能很好地运行。如果你有庞大或分散的开发团队，或者你有大量的预定功能发布，那么它就不太适合了。在此模型中会发生合并冲突，因此请注意，解决合并冲突是一项关键技能。所有团队成员都必须接受相应的培训。

优点

基于主干的开发具有多种优势，可以改善开发流程、简化协作并提高软件的整体质量。以下是一些主要好处：

- 更快的反馈循环 — 通过基于主干的开发，开发人员可以频繁地整合代码更改，通常每天多次。与基于功能的开发模型相比，这可以更快地提供有关潜在问题的反馈，并帮助开发人员更快地发现和修复问题。
- 减少合并冲突 — 在基于主干的开发中，由于更改是持续集成的，因此可以最大限度地降低大型、复杂合并冲突的风险。这有助于维护更简洁的代码库，并减少解决冲突所花费的时间。在基于功能的开发中，解决冲突既耗时又容易出错。

- 改善协作 — 基于 Trunk 的开发鼓励开发人员在同一个分支上协作，从而促进团队内部更好的沟通和协作。这可以加快解决问题的速度和更具凝聚力的团队活力。
- 更轻松的代码审查 — 由于在基于主干的开发中，代码更改更小、更频繁，因此可以更轻松地进行全面的代码审查。较小的更改通常更容易理解和审查，从而更有效地识别潜在问题并进行改进。
- 持续集成和交付 — 基于 Trunk 的开发支持持续集成和持续交付 (CI/CD) 的原则。通过将代码库保持在可发布状态并经常集成更改，团队可以更轻松地采用 CI/CD 实践，从而缩短部署周期并提高软件质量。
- 提高代码质量 — 通过频繁的集成、测试和代码审查，基于主干的开发可以提高整体代码质量。开发人员可以更快地发现和修复问题，从而降低技术债务随着时间的推移而积累的可能性。
- 简化的分支策略 — 基于主干的开发通过减少长寿命分支的数量来简化分支策略。这可以使管理和维护代码库变得更加容易，对于大型项目或团队来说尤其如此。

劣势

基于主干的开发确实有一些缺点，这可能会影响开发过程和团队动态。以下是一些明显的缺点：

- 有限隔离 — 由于所有开发人员都在同一个分支上工作，因此团队中的每个人都可以立即看到他们的更改。这可能会导致干扰或冲突，导致意想不到的副作用或破坏构建。相比之下，基于功能的开发可以更好地隔离更改，以便开发人员可以更加独立地工作。
- 测试压力增加 — 基于 Trunk 的开发依赖于持续集成和自动测试来快速发现问题。但是，这种方法可能会给测试基础架构带来很大的压力，并且需要维护良好的测试套件。如果测试不全面或不可靠，则可能导致主分支中出现未被发现的问题。
- 减少对发布的控制 — 基于 Trunk 的开发旨在使代码库保持持续可发布状态。虽然这可能具有优势，但它可能并不总是适合发布时间表严格的项目或需要同时发布特定功能的项目。基于功能的开发可以更好地控制发布功能的时间和方式。
- 代码流失 — 随着开发人员不断将更改集成到主分支中，基于主干的开发可能会导致代码流失增加。这可能会使开发人员难以跟踪代码库的当前状态，并且在尝试了解最近更改的效果时可能会引起混乱。
- 需要强大的团队文化 — 基于主干的开发需要团队成员之间高度的纪律、沟通和协作。这可能很难维护，尤其是在规模较大的团队中，或者与使用这种方法经验较少的开发人员合作时。
- 可扩展性挑战 — 随着开发团队规模的扩大，集成到主分支中的代码更改数量可能会迅速增加。这可能会导致更频繁的构建中断和测试失败，从而使代码库难以保持可发布状态。

GitHub 流量分支策略

GitHub Flow 是一个基于分支的轻量级工作流程，由开发。 GitHub GitHubFlow 基于短期功能分支的概念，当功能完成并准备部署时，这些分支会合并到主分支中。 GitHub Flow 的关键原理是：

- 分支是轻量级的 — 开发人员只需单击几下即可为其工作创建功能分支，从而在不影响主分支的情况下提高协作和实验能力。
- 持续部署 — 更改在合并到主分支后立即进行部署，这样可以快速进行反馈和迭代。
- 合并请求 — 开发人员使用合并请求来启动讨论和审查流程，然后再将其更改合并到主分支。

有关 GitHub Flow 的更多信息，请参阅以下资源：

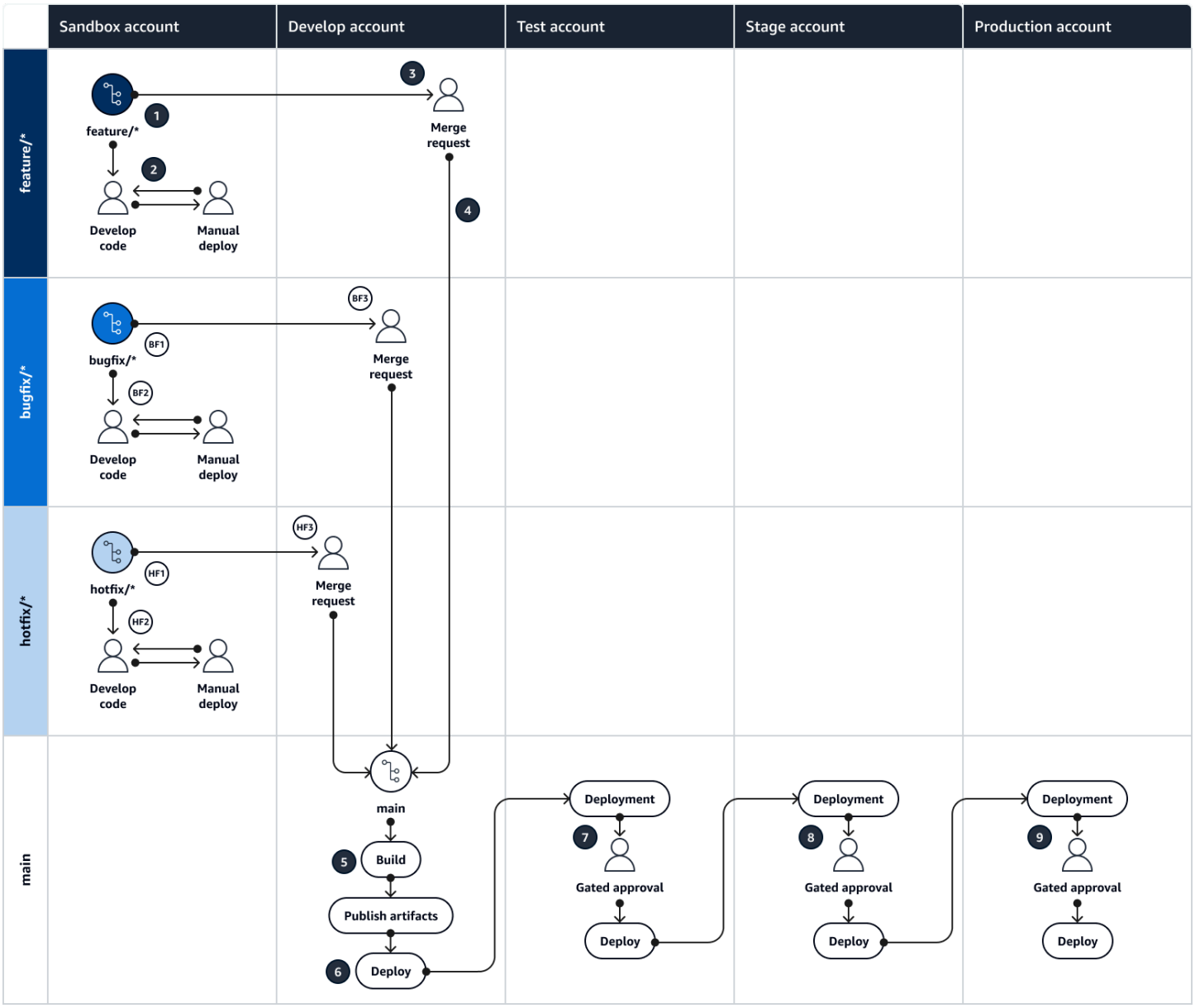
- 为多账户 DevOps 环境实施 GitHub Flow 分支策略（AWS 规范性指导）
- GitHub Flow 快速入门（GitHub 文档）
- 为什么 GitHub Flow？（GitHubFlow 网站）

本节中的主题：

- GitHub Flow 策略的可视化概述
- GitHub Flow 策略中的分支
- GitHub Flow 策略的优缺点

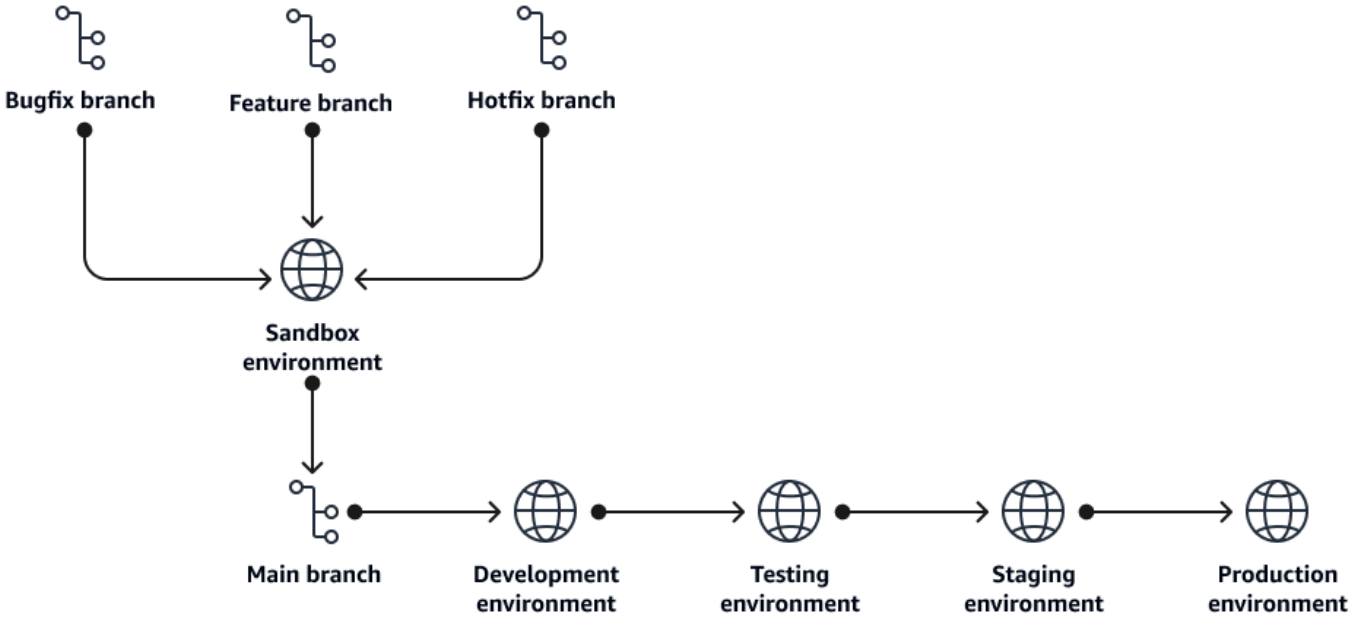
GitHub Flow 策略的可视化概述

下图可以像 Punnett 方块一样用来理解 GitHub Flow 分支策略。将垂直轴上的分支与水平轴上的 AWS 环境对齐，以确定在每个场景中要执行的操作。带圆圈的数字将引导您完成图中所示的操作顺序。此图显示了 GitHub Flow 分支策略的开发工作流程，从沙盒环境中的功能分支到主分支的生产版本。有关每种环境中发生的活动的更多信息，请参阅本指南中的DevOps 环境。



GitHub Flow 策略中的分支

GitHub Flow 分支策略通常具有以下分支。



功能分支

你在feature分支中开发功能。要创建feature分支，您需要从该分支中main分支。开发人员在feature分支中迭代、提交和测试代码。功能完成后，开发者通过向创建合并请求来推广该功能main。

命名惯例：	feature/<story number>_<developer initials>_<descriptor>
命名约定示例：	feature/123456_MS_Implement_Feature_A

错误修复分支

该bugfix分支用于修复问题。这些分支是从分支上分main支的。在沙盒或任何较低的环境中测试错误修复后，可以通过合并请求将其合并到更高的环境中，从而将其提升到main更高的环境。这是组织和跟踪的建议命名惯例，也可以使用功能分支来管理此过程。

命名惯例：	bugfix/<ticket number>_<developer initials>_<descriptor>
命名约定示例：	bugfix/123456_MS_Fix_Problem_A

修补程序分支

该hotfix分支用于解决影响力大的关键问题，最大限度地减少开发人员与部署在生产环境中的代码之间的延迟。这些分支是从分支上分main支的。在沙盒或任何较低的环境中测试此修补程序后，可以通过合并请求将其合并到更高的环境中，将其升级到main更高的环境。这是组织和跟踪的建议命名惯例，也可以使用功能分支来管理此过程。

命名惯例：	`hotfix/<ticket number>_<developer initials>_<descriptor>`
命名约定示例：	`hotfix/123456_MS_Fix_Problem_A`

主分支

分main支始终代表在生产环境中运行的代码。使用合并请求将代码从main分feature支合并到分支中。为了防止删除并防止开发人员将代码直接推送到main，请为分支启用main分支保护。

命名惯例：	`main`

GitHub Flow 策略的优缺点

Github Flow 分支策略非常适合具有较强沟通能力的小型、成熟的开发团队。这种策略非常适合想要实现持续交付的团队，并且得到了常见 CI/CD 引擎的大力支持。 GitHub Flow 是轻量级的，没有太多的规则，并且能够支持快速移动的团队。如果您的团队有严格的合规或发布流程需要遵守，则它不太合适。合并冲突在此模型中很常见，而且可能经常发生。解决合并冲突是一项关键技能，你必须相应地训练所有团队成员。

优点

GitHub Flow 提供了多种优势，可以改善开发流程、简化协作并提高软件的整体质量。以下是一些主要好处：

- 灵活轻便 — GitHub Flow 是一种轻巧灵活的工作流程，可帮助开发人员在软件开发项目上进行协作。它允许以最小的复杂性进行快速迭代和实验。
- 简化协作 — GitHub Flow 为管理功能开发提供了清晰而简化的流程。它鼓励可以快速审查和合并的小型、有针对性的更改，从而提高效率。

- 清晰的版本控制 — 使用 GitHub Flow，每项更改都是在单独的分支中进行的。这样可以建立清晰且可追溯的版本控制历史记录。这可以帮助开发人员跟踪和了解更改，在必要时进行恢复，并维护可靠的代码库。
- 无缝持续集成 — GitHub Flow 与持续集成工具集成。创建拉取请求可以启动自动测试和部署流程。CI 工具可帮助您在更改合并到`main`分支之前对其进行全面测试，从而降低在代码库中引入错误的风险。
- 快速反馈和持续改进 — GitHub Flow 通过拉取请求促进频繁的代码审查和讨论，从而鼓励快速反馈循环。这有助于及早发现问题，促进团队成员之间的知识共享，最终提高代码质量并改善开发团队内部的协作。
- 简化了回滚和还原 — 如果代码更改引入了意想不到的错误或问题， GitHub Flow 可以简化回滚或还原更改的过程。通过清晰的提交和分支历史记录，可以更轻松地识别和还原有问题的更改，从而有助于维护稳定且功能齐全的代码库。
- 轻量级学习曲线 — GitHub Flow 比 Gitflow 更容易学习和采用，特别是对于已经熟悉 Git 和版本控制概念的团队而言。它的简单性和直观的分支模型使不同经验水平的开发人员都可以使用，从而缩短了与采用新开发工作流程相关的学习曲线。
- 持续开发 — GitHub Flow 使团队能够采用持续部署方法，因为每项更改都可以在合并到分`main`支后立即部署。这种简化的流程消除了不必要的延迟，并确保用户可以快速获得最新的更新和改进。这使得开发周期更加敏捷，响应速度更快。

劣势

虽然 GitHub Flow 有几个优点，但也必须考虑其潜在的缺点：

- 对大型项目的适用性有限 — GitHub Flow 可能不太适合具有复杂代码库和多个长期功能分支的大型项目。在这种情况下，结构化程度更高的工作流程（例如 Gitflow）可能会更好地控制并发开发和发布管理。
- 缺乏正式的发布结构 — GitHub Flow 没有明确定义发布流程或支持功能，例如版本控制、修补程序或维护分支。对于需要严格发布管理或需要长期支持和维护的项目来说，这可能是一个限制。
- 对长期发布计划的支持有限 — GitHub Flow 专注于短期功能分支，这些分支可能与需要长期发布计划的项目（例如具有严格路线图或广泛功能依赖关系的项目）不太一致。在 Flo GitHub w 的限制下，管理复杂的发布时间表可能具有挑战性。
- 可能出现频繁的合并冲突 — 由于 GitHub Flow 鼓励频繁进行分支和合并，因此有可能遇到合并冲突，尤其是在有大量开发活动的项目中。解决这些冲突可能很耗时，可能需要开发团队付出额外的努力。

- 缺乏正式的工作流程阶段 — GitHub Flow 未定义明确的开发阶段，例如 alpha、beta 或候选发布阶段。这可能会使传达项目的当前状态或不同分支或版本的稳定性级别变得更加困难。
- 重大更改的影响 — 由于 GitHub Flow 鼓励经常将更改合并到`main`分支中，因此引入影响代码库稳定性的重大更改的风险更高。严格的代码审查和测试实践对于有效降低这种风险至关重要。

Gitflow 分支策略

Gitflow 是一种分支模型，它涉及使用多个分支将代码从开发转移到生产。Gitflow 非常适合那些有计划发布周期并且需要将一系列功能定义为发布的团队。开发是在各个功能分支中完成的，这些分支经批准后合并到用于集成的开发分支中。该分支中的功能被认为已准备就绪，可以投入生产。当所有计划中的功能都积累到开发分支中后，将创建一个发布分支，用于部署到上层环境。这种分离可以更好地控制哪些更改将按定义的时间表移动到哪个命名环境。如有必要，您可以将此过程加快到更快的部署模式。

有关 Gitflow 分支策略的更多信息，请参阅以下资源：

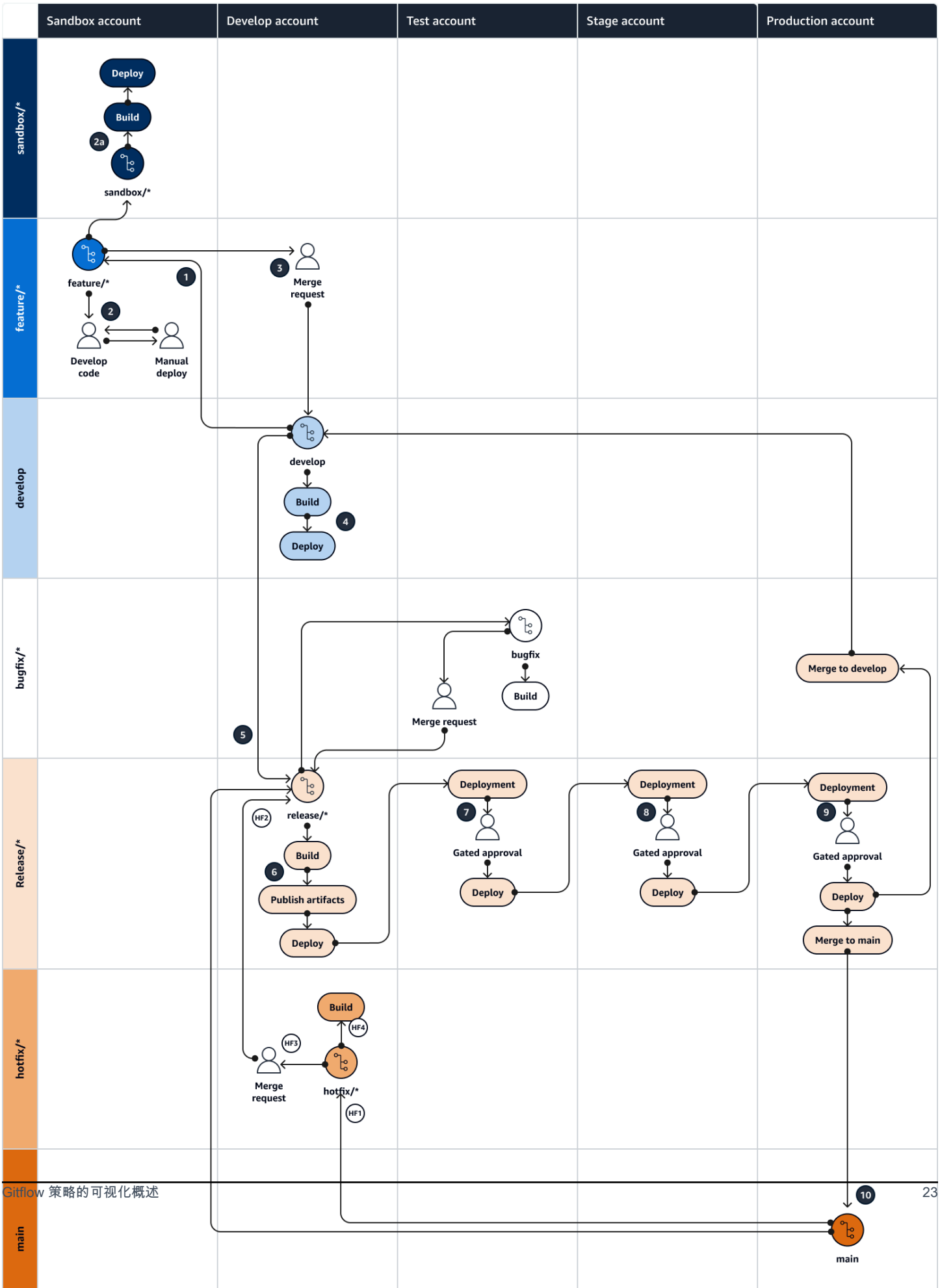
- 为多账户 DevOps 环境实施 Gitflow 分支策略（规范性指南）AWS
- 最初的 Gitflow 博客（Vincent Driessen 博客文章）
- Gitflow 工作流程 (Atlassian)

本节中的主题：

- Gitflow 策略的可视化概述
- Gitflow 策略中的分支
- Gitflow 策略的优缺点

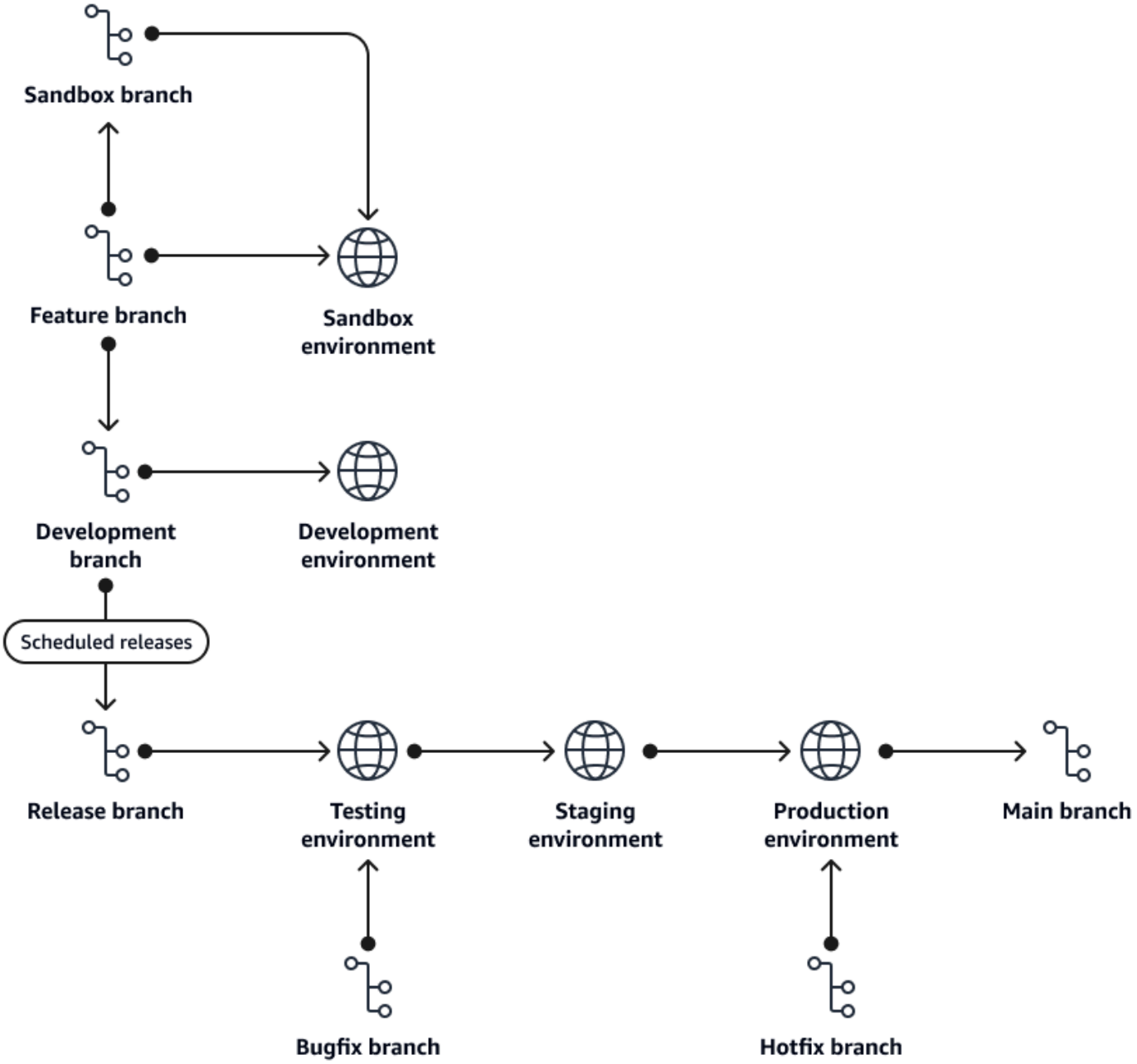
Gitflow 策略的可视化概述

下图可以像 Punnett 方块一样用来理解 Gitflow 的分支策略。将垂直轴上的分支与水平轴上的 AWS 环境对齐，以确定在每个场景中要执行的操作。带圆圈的数字将引导您完成图中所示的操作顺序。有关每种环境中发生的活动的更多信息，请参阅本指南中的DevOps 环境。


Sandbox account
Develop account
Test account
Stage account
Production account
sandbox/*
feature/*
develop
bugfix/*
Release/*
hotfix/*
main
Deploy
Build
2a
sandbox/*
feature/*
1
2
Develop code
Manual deploy
3
Merge request
develop
Build
4
Deploy
bugfix
Build
Merge request
5
Merge to develop
HF2
release/*
Build
6
Publish artifacts
Deploy
Deployment
7
Gated approval
Deploy
Deployment
8
Gated approval
Deploy
Deployment
9
Gated approval
Deploy
Merge to main
Build
HF4
HF3
Merge request
hotfix/*
HF1
10
main

Gitflow 策略中的分支

Gitflow 分支策略通常具有以下分支。



功能分支

Feature分支是你开发功能的短期分支。分feature支是通过从分支中分develop支来创建的。开发人员在feature分支中迭代、提交和测试代码。该功能完成后，开发者将推广该功能。从功能分支出发，只有两条前进的道路：

- 合并到分sandbox支中
- 向develop分支创建合并请求

命名惯例：	`feature/<story number>_<developer initials>_<descriptor>`
命名约定示例：	`feature/123456_MS_Implement_Feature_A`

沙盒分支

该sandbox分支是 Gitflow 的非标准短期分支。但是，它对于 CI/CD 管道开发很有用。该sandbox分支主要用于以下目的：

- 使用 CI/CD 管道而不是手动部署，对沙盒环境进行全面部署。
- 在提交合并请求以在较低的环境（例如开发或测试）中进行全面测试之前，先开发和测试管道。

Sandbox分支本质上是暂时的，并不意味着寿命长。应在特定测试完成后将其删除。

命名惯例：	`sandbox/<story number>_<developer initials>_<descriptor>`
命名约定示例：	`sandbox/123456_MS_Test_Pipeline_Deploy`

开发分支

该develop分支是一个长期存在的分支，其中的功能被集成、构建、验证并部署到开发环境中。所有feature分支都合并到该develop分支中。合并到develop分支是通过合并请求完成的，合并请求需要成功构建并获得两位开发人员的批准。为防止删除，请在分支上启用develop分支保护。

命名惯例：	`develop`

发布分支

在 Gitflow 中，release分支是短期分支。这些分支之所以特别，是因为你可以将它们部署到多个环境中，采用一次构建、多次部署的方法。 Release分支可以针对测试、暂存或生产环境。在开发团队决定将功能提升到更高的环境后，他们会创建一个新release分支，并使用与先前版本相比的增量版本号。在每个环境的门口，部署都需要手动批准才能继续。 Release分支应该要求更改合并请求。

在release分支部署到生产环境后，应将其合并回develop和main分支，以确保将任何错误修复或修补程序合并回未来的开发工作中。

命名惯例：	release/v{major}.{minor}
命名约定示例：	release/v1.0

主分支

该main分支是一个长期存在的分支，它始终代表生产中运行的代码。从发布管道成功部署后，代码将自动从发布分支合并到分支中。main为防止删除，请在分支上启用main分支保护。

命名惯例：	main

错误修复分支

该bugfix分支是一个短期分支，用于修复尚未发布到生产环境的发布分支中的问题。分bugfix支只能用于将分release支中的修复提升到测试、暂存或生产环境。分bugfix支总是从分支上分release支。

测试错误修正后，可以通过合并请求将其提升到release分支。然后，您可以按照标准发布流程推动release分支向前发展。

命名惯例：	bugfix/<ticket>_<developer initials>_<descriptor>
命名约定示例：	bugfix/123456_MS_Fix_Problem_A

修补程序分支

该hotfix分支是一个短期分支，用于修复生产中的问题。它仅用于推广必须加快才能进入生产环境的修复程序。分hotfix支总是从中分支。main

测试完此修补程序后，您可以通过向从main中创建的release分支发出合并请求，将其提升到生产环境。为了进行测试，您可以按照标准发布流程将release分支向前推进。

命名惯例：	hotfix/<ticket>_<developer initials>_<descriptor>
命名约定示例：	hotfix/123456_MS_Fix_Problem_A

Gitflow 策略的优缺点

Gitflow 分支策略非常适合规模更大、分布更分散、有严格发布和合规要求的团队。Gitflow为组织提供了可预测的发布周期，而大型组织通常更喜欢这样做。Gitflow 也非常适合需要护栏才能正确完成软件开发生命周期的团队。这是因为该策略中有多种机会进行审查和质量保证。Gitflow 也非常适合必须同时维护多个生产版本的团队。GitFlow 的一些缺点是，它比其他分支模型更复杂，需要严格遵守模式才能成功完成。由于管理发布分支的严格性质，Gitflow不适合追求持续交付的组织。Gitflow发行分支机构可能是长期存在的分支机构，如果不及时解决，可能会积累技术债务。

优点

基于 GitFlow 的开发具有多种优势，可以改善开发流程、简化协作并提高软件的整体质量。以下是一些主要好处：

- 可预测的发布流程 — Gitflow 遵循定期且可预测的发布流程。它非常适合有规律开发和发布节奏的团队。
- 改善协作 — Gitflow 鼓励使用feature和分支。release这两个分支可以帮助团队并行工作，同时最大限度地减少相互依赖。
- 非常适合多种环境 — Gitflow 使用release分支，分支可以是寿命更长的分支。这些分支使团队能够在更长的时间内定位单个版本。
- 生产中的多个版本 — 如果您的团队支持生产中的软件的多个版本，则 Gitflow release 分支支持此要求。
- 内置代码质量审查 — Gitflow 要求并鼓励在将代码提升到其他环境之前使用代码审查和批准。此过程要求所有代码促销都必须执行此步骤，从而消除了开发者之间的摩擦。

- 组织优势 — Gitflow在组织层面也有优势。Gitflow 鼓励使用标准发布周期，这有助于组织了解和预测发布时间表。由于企业现在知道何时可以交付新功能，因此由于有固定的交付日期，因此减少了时间表上的摩擦。

劣势

基于 GitFlow 的开发确实有一些缺点，可能会影响开发过程和团队动态。以下是一些明显的缺点：

- 复杂性 — Gitflow 是新团队需要学习的复杂模式，你必须遵守 Gitflow 的规则才能成功使用它。
- 持续部署 — Gitflow 不适合将许多部署快速发布到生产环境的模式。这是因为 Gitflow 需要使用多个分支和严格的工作流程来管理分支。`release`
- 分支管理 — Gitflow 使用许多分支，维护起来可能会变得繁重。为了使分支彼此正确对齐，跟踪各个分支并合并已发布的代码可能很困难。
- 技术债务 — 由于 Gitflow 的发布速度通常比其他分支模型慢，因此可以积累更多的功能来发布，这可能会导致技术债务积累。

团队在决定基于 GitFlow 的开发是否是其项目的正确方法时，应仔细考虑这些缺点。

后续步骤

本指南解释了三种常见的 Git 分支策略之间的区别： GitHubFlow、Gitflow 和 Trunk。它详细描述了他们的工作流程，还提供了每种工作流程的优缺点。接下来的步骤是为您的组织选择其中一个标准工作流程。要实现其中一种分支策略，请参阅以下内容：

- 为多 DevOps 账户环境实施中继分支策略
- 为多 DevOps 账户环境实施 GitHub Flow 分支策略
- 为多账户环境实施 Gitflow 分支策略 DevOps

如果您不确定从哪里开始团队使用 Git 和 DevOps 流程，我们建议您选择一个标准解决方案并对其进行测试。使用标准的分支约定可以帮助团队在现有文档的基础上再接再厉，了解最适合他们的方法。

如果你的策略对你的组织或开发团队不起作用，不要害怕改变策略。开发团队的需求和要求会随着时间的推移而发生变化，因此没有单一的完美解决方案。

资源

本指南不包括针对 Git 的培训；但是，如果您需要此培训，互联网上有许多高质量的资源可供选择。我们建议您从 Git 文档网站开始。

以下资源可以帮助您完成 Git 分支之 AWS Cloud旅。

AWS 规范性指导

- 为多 DevOps 账户环境实施中继分支策略
- 为多 DevOps 账户环境实施 GitHub Flow 分支策略
- 为多账户环境实施 Gitflow 分支策略 DevOps

其他 AWS 指导

- AWS DevOps 指导
- AWS 部署管道参考架构
- 什么是 DevOps ？
- DevOps资源

其他资源

- 十二因子应用程序方法论 (12factor.net)
- Git-Sec rets () GitHub
- Gitflow
  - 最初的 Gitflow 博客（Vincent Driessen 博客文章）
  - Gitflow 工作流程 (Atlassian)
  - Gitflow 开启 GitHub：如何将 Git Flow 工作流程与 GitHub 基于 Repos 的存储库一起使用（视频）YouTube
  - Git Flow 初始化示例（YouTube 视频）
  - Gitflow 发布分支从头到尾（视频）YouTube
- GitHub 流量

  - GitHub Flow 快速入门（GitHub 文档）
  - 为什么 GitHub Flow？（GitHub Flow 网站）
- 后备箱
  - 基于@@ 主干的开发简介（基于主干的开发网站）

贡献者

编写

- 迈克·斯蒂芬斯，高级云应用程序架构师， AWS
- Rayjan Wilson，高级云应用程序架构师， AWS
- Abhilash Vinod，团队负责人，高级云应用程序架构师， AWS

正在审阅

- 斯蒂芬 DiCato，高级安全顾问， AWS
- Gaurav Samudra，云应用程序架构师， AWS
- Steven Guggenheimer，团队负责人，高级云应用程序架构师， AWS

技术写作

- Lilly AbouHarb，高级技术撰稿人， AWS

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
初次发布	—	2024 年 2 月 15 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。

ACID

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？

人工智能操作 (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 AIOps AWS 迁移策略中使用的更多信息，请参阅操作集成指南。

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

请参阅变更数据捕获。

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，创建着陆区、定义 CCo E、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)

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的信息。例如，Amazon SageMaker AI 为 CV 提供了图像处理算法。

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 is commonly described as a pipeline. 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

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。

后

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，其中包含事实表中有关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。

EDI

参见电子数据交换。

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。与云计算相比，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)

组织之间自动交换业务文档。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。

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

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，请参阅使用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。

少量提示

在要求法学硕士执行类似任务之前，向其提供少量示例，以演示该任务和所需的输出。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。对于需要特定格式、推理或领域知识的任务，Few-shot 提示可能非常有效。另请参见零镜头提示。

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。

FM

参见基础模型。

基础模型 (FM)

一个大型深度学习神经网络，一直在广义和未标记数据的大量数据集上进行训练。 FMs 能够执行各种各样的一般任务，例如理解语言、生成文本和图像以及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。

G

生成式人工智能

人工智能模型的子集，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工件，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 AI。

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。

金色影像

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配置软件，并有助于提高设备制造运营的速度、可扩展性和生产力。

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

一项高级规则，可帮助管理各组织单位的资源、策略和合规性 (OUs)。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。

抵制数据

从用于训练机器学习模型的数据集中扣留的一部分带有标签的历史数据。通过将模型预测与抵制数据进行比较，您可以使用抵制数据来评估模型性能。

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

IaC

参见基础设施即代码。

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

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 安全参考架构建议设置您的网络帐户，包括入站、出站和检查， VPCs 以保护您的应用程序与更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) 数字化转型战略。

检查 VPC

在 AWS 多账户架构中，一种集中式 VPC，用于管理对 VPCs（相同或不同 AWS 区域）、互联网和本地网络之间的网络流量的检查。AWS 安全参考架构建议设置您的网络帐户，包括入站、出站和检查， VPCs 以保护您的应用程序与更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，请参阅使用机器学习模型的可解释性 AWS。

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。

大型语言模型 (LLM)

一种基于大量数据进行预训练的深度学习 AI 模型。法学硕士可以执行多项任务，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 LLMs。

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

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。

LLM

参见大型语言模型。

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

一种小型的独立服务，通过明确的定义进行通信 APIs ，通常由小型的独立团队拥有。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通过定义明确的接口进行通信。 APIs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： EC2 使用 AWS 应用程序迁移服务重新托管向 Amazon 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

由此创建的跟踪 AWS CloudTrail ，用于记录组织 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

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 安全参考架构建议设置您的网络帐户，包括入站、出站和检查， VPCs 以保护您的应用程序与更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。

policy

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。

通过设计保护隐私

一种在整个开发过程中考虑隐私的系统工程方法。

私有托管区

一个容器，其中包含有关您希望 Amazon Route 53 如何响应针对一个或多个 VPCs域名及其子域名的 DNS 查询的信息。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。这些控件会在资源配置之前对其进行扫描。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。

提示链接

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，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）。

RAG

请参见检索增强生成。

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

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。

检索增强生成（RAG）

一种生成式人工智能技术，其中法学硕士在生成响应之前引用其训练数据源之外的权威数据源。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。

安全性源于设计

一种在整个开发过程中考虑安全性的系统工程方法。

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，用于集中控制组织中所有账户的权限 AWS Organizations。 SCPs 定义防护措施或限制管理员可以委托给用户或角色的操作。您可以使用 SCPs 允许列表或拒绝列表来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。

SIEM

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。

系统提示符

一种向法学硕士提供上下文、说明或指导方针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

tags

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

一个网络传输中心，可用于将您的网络 VPCs 和本地网络互连。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

两者之间的连接 VPCs ，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

参见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。

零镜头提示

向法学硕士提供执行任务的说明，但没有示例（镜头）可以帮助指导任务。法学硕士必须使用其预先训练的知识来处理任务。零镜头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 few-shot 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。