為多帳戶 DevOps 環境選擇 Git 分支策略

AWS 方案指引

AWS 方案指引: 為多帳戶 DevOps 環境選擇 Git 分支策略

Table of Contents

為多帳戶 DevOps環境選擇 Git 分支策略

Amazon Web Services （ 貢獻者)

2024 年 2 月 (文件歷史記錄)

遷移至雲端型方法並在 上提供軟體解決方案 AWS 可能是變革性的。這可能需要變更您的軟體開發生命週期程序。一般而言，在 中的開發過程中 AWS 帳戶 會使用多個 AWS 雲端。選擇相容的 Git 分支策略與您的 DevOps 程序配對對於成功至關重要。為您的組織選擇正確的 Git 分支策略，可協助您在開發團隊之間簡潔地傳達 DevOps 標準和最佳實務。Git 分支在單一環境中可能很簡單，但在跨多個環境套用時可能會變得令人困惑，例如沙盒、開發、測試、預備和生產環境。擁有多個環境會增加實作的DevOps複雜性。

本指南提供 Git 分支策略的視覺化圖表，示範組織如何實作多帳戶 DevOps 程序。視覺化指南可協助團隊了解如何將 Git 分支策略與其 DevOps 實務合併。使用 Gitflow、 GitHub Flow 或 Trunk 等標準分支模型來管理原始程式碼儲存庫，有助於開發團隊調整工作。這些團隊也可以在網際網路上使用標準 Git 訓練資源，來了解和實作這些模型和策略。

如需 DevOps 最佳實務 AWS，請參閱 AWS Well-Architected 中的DevOps指南。當您檢閱本指南時，請使用盡職調查來為您的組織選擇正確的分支策略。有些策略可能比其他策略更符合您的使用案例。

目標

本指南是文件系列的一部分，內容是為具有多個的組織選擇和實作 DevOps分支策略 AWS 帳戶。此系列旨在協助您從一開始就採用最符合您的需求、目標和最佳實務的策略，以簡化 中的體驗 AWS 雲端。本指南不包含可執行指令碼，因為它們會根據組織使用的持續整合和持續交付 DevOps (CI/CD) 引擎和技術架構而有所不同。

本指南說明三種常見 Git 分支策略之間的差異： GitHub Flow、Gitflow 和 Trunk。本指南中的建議可協助團隊識別與其組織目標一致的分支策略。檢閱本指南後，您應該能夠為您的組織選擇分支策略。選擇策略後，您可以使用下列其中一種模式，協助您與開發團隊實作該策略：

- 實作多帳戶 DevOps 環境的中繼線分支策略
- 針對多帳戶 DevOps 環境實作 GitHub 流程分支策略
- 針對多帳戶 DevOps 環境實作 Gitflow 分支策略

請務必注意，適用於某個組織、團隊或專案的事項可能不適合其他組織、團隊或專案。Git 分支策略之間的選擇取決於各種因素，例如團隊規模、專案需求，以及協作、整合頻率和發行管理之間的所需平衡。

使用 CI/CD 實務

AWS 建議您實作持續整合和持續交付 (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管道也為開發團隊提供控管和護欄。它們會強制執行一致性、標準、最佳實務和最低接受度，以接受和部署功能。如需詳細資訊，請參閱在 上實作持續整合和持續交付 AWS。

本指南中討論的所有分支策略都非常適合為開發團隊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建立護欄和機制，以防止開發人員有意或無意地規避定義的程序。

AWS 提供一套開發人員服務，旨在協助您建置 CI/CD 管道。例如， AWS CodePipeline 是一種全受管持續交付服務，可協助您自動化發行管道，以取得快速可靠的應用程式和基礎設施更新。 AWS CodeBuild會編譯原始程式碼、執行測試，並產生 ready-to-deploy軟體套件。如需詳細資訊，請參閱上的開發人員工具 AWS。

了解環 DevOps境

若要瞭解分支策略，您必須瞭解每個環境中發生的目的和活動。建立多個環境可協助您將開發活動分成各個階段、監視這些活動，並防止意外發行未核准的功能。您可以在每個環境 AWS 帳戶 中有一個或多個。

大多數組織都有幾個概述可供使用的環境。但是，環境的數量可能會因組織和軟體開發原則而有所不同。本文件系列假設您有下列五個跨開發管線的一般環境，但這些環境可能會以不同的名稱呼叫：

- 沙盒 — 開發人員撰寫程式碼、犯錯及執行概念驗證工作的環境。
- 開發 — 一種環境，開發人員整合他們的程式碼，以確認它們都可以作為一個單一的、有凝聚力的應用程式運作。
- 測試 — 進行 QA 團隊或驗收測試的環境。團隊通常會在此環境中進行效能或整合測試。
- 測試 — 一種生產前環境，您可以在其中驗證程式碼和基礎結構是否在相當於生產的情況下如預期般執行。此環境已設定為與生產環境盡可能相似。
- 生產 — 處理最終使用者和客戶流量的環境。

本節詳細說明每個環境。它還描述了每個環境的構建步驟，部署步驟和退出準則，以便您可以繼續進行下一個操作。下列影像會依序展示這些環境。

本節主題：

- 沙盒環境
- 開發環境
- 測試環境
- 暫存環境
- 生產環境

沙盒環境

沙盒環境是開發人員撰寫程式碼、犯錯以及執行概念驗證工作的地方。您可以從本機工作站或透過本機工作站上的指令碼部署至沙箱環境。

存取

開發人員應該擁有沙箱環境的完全訪問權限。

建置步驟

開發人員準備好將變更部署到沙箱環境時，在其本機工作站上手動執行組建。

1. 使用 git 秘密（GitHub）掃描敏感信息
2. 林特源代碼
3. 建置並編譯原始程式碼 (如果適用)
4. 執行單元測試
5. 執行代碼覆蓋率分析
6. 執行靜態程式碼分析
7. 建置基礎架構即程式碼 (IaC)
8. 執行 IaC 安全分析
9. 擷取開放原始碼授權
10. 發佈組建成品

部署步驟

如果您使用的是 Gitflow 或 Trunk 模型，則在沙箱環境中成功構建`feature`分支時，部署步驟會自動啟動。如果您使用的是 GitHub Flow 模型，則手動執行下列部署步驟。以下是沙箱環境中的部署步驟：

1. 下載發佈的成品
2. 執行資料庫版本
3. 執行 IAC 部署
4. 執行整合測試

轉移到開發環境之前的期望

- 在沙箱環境中成功構建`feature`分支
- 開發人員已在沙箱環境中手動部署並測試了該功能

開發環境

開發環境是開發人員將他們的代碼集成在一起，以確保它們都可以作為一個有凝聚力的應用程序工作。在 Gitflow 中，開發環境包含合併請求包含的最新功能，並準備好發布。在 GitHub Flow 和 Trunk 策略中，開發環境被認為是測試環境，而且程式碼庫可能不穩定且不適合部署到生產環境。

存取

根據最小權限原則分配權限。最低權限是授予執行任務所需的最低許可的安全最佳實務。與沙箱環境相比，開發人員對開發環境的存取權限應該較少。

建置步驟

創建對`develop`分支（Gitflow）或分`main`支（幹線或 GitHub 流程）的合併請求會自動啟動構建。

1. 使用 git 秘密（GitHub）掃描敏感信息
2. 林特源代碼
3. 建置並編譯原始程式碼 (如果適用)
4. 執行單元測試
5. 執行代碼覆蓋率分析
6. 執行靜態程式碼分析
7. 建立合家歡
8. 執行 IaC 安全分析
9. 擷取開放原始碼授權

部署步驟

如果您使用的是 Gitflow 模型，則在開發環境中成功構建`develop`分支時，部署步驟會自動啟動。如果您使用的是 GitHub Flow 模型或 Trunk 模型，則在針對`main`分支建立合併請求時，部署步驟會自動啟動。以下是開發環境中的部署步驟：

1. 從建置步驟下載已發佈的成品
2. 執行資料庫版本
3. 執行 IAC 部署
4. 執行整合測試

移至測試環境之前的期望

- 在開發環境中成功構建和部署`develop`分支（Gitflow）或`main`分支（幹線或 GitHub 流程）
- 單元測試通過 100%
- 成功的 IaC 構建
- 已成功建立部署人工因素
- 開發人員已執行手動驗證，以確認功能如預期般運作

測試環境

品質保證 (QA) 人員使用測試環境來驗證功能。他們在完成測試後批准更改。當他們核准時，分支會移至下一個環境，即暫存。在 Gitflow 中，此環境及其上面的其他環境僅適用於從`release`分支機構部署。分`release`支是以包含計劃功能的`develop`分支為基礎。

存取

根據最小權限原則分配權限。開發人員對測試環境的訪問應該比他們對開發環境的訪問更少。QA 人員需要足夠的權限來測試該功能。

建置步驟

此環境中的構建過程僅適用於使用 Gitflow 策略時的錯誤修復。建立`bugfix`分支的合併要求會自動啟動建置。

1. 使用 git 秘密（GitHub）掃描敏感信息
2. 林特源代碼
3. 建置並編譯原始程式碼 (如果適用)
4. 執行單元測試
5. 執行代碼覆蓋率分析

6. 執行靜態程式碼分析
7. 建立合家歡
8. 執行 IaC 安全分析
9. 擷取開放原始碼授權

部署步驟

在開發環境中`release`部署後，在測試環境中自動啟動`main`分支（Git GitHub flow）或分支（幹線或 Flow）的部署。以下是測試環境中的部署步驟：

1. 在測試環境中部署`release`分支（Gitflow）或`main`分支（幹線或 GitHub 流程）
2. 暫停由指定人員進行手動核准
3. 下載發佈的成品
4. 執行資料庫版本
5. 執行 IAC 部署
6. 執行整合測試
7. 執行效能測試
8. 質量保證批准

移至暫存環境之前的期望

- 開發和 QA 團隊已經執行了足夠的測試，以滿足您組織的需求。
- 開發團隊已經通過`bugfix`分支解決了任何發現的錯誤。

暫存環境

暫存環境設定為與生產環境相同。例如，資料設定的範圍和大小應與生產工作負載相似。使用預備環境驗證程式碼和基礎結構是否如預期般運作。此環境也是業務使用案例 (例如預覽或客戶示範) 的偏好選擇。

存取

根據最小權限原則分配權限。開發人員應該擁有與執行生產環境相同的暫存環境存取權。

建置步驟

無。測試環境中使用的相同成品會在測試環境中重複使用。

部署步驟

在測試環境中核准和部署之後，自動在測試環境中啟動`main`分支（Git GitHub flow）或分支（幹線或 Flow）的部署。`release`以下是測試環境中的部署步驟：

1. 在臨時環境中部署`release`分支（Gitflow）或`main`分支（幹線或 GitHub 流程）
2. 暫停由指定人員進行手動核准
3. 下載發佈的成品
4. 執行資料庫版本
5. 執行 IAC 部署
6. (選擇性) 執行整合測試
7. (選擇性) 執行負載測試
8. 取得所需開發、品質保證、產品或業務核准者的核准

轉移到生產環境之前的期望

- 生產等效版本已成功部署到測試環境
- (選擇性) 整合與負載測試成功

生產環境

生產環境支持發布的產品，由真實客戶處理實際數據。這是以最低權限指派存取權的受保護環境，而且只能在有限的時間內透過稽核的例外處理程序允許提升的存取權。

存取

在生產環境中，開發人員在 AWS 管理主控台中應具有有限的唯讀存取權限。例如，開發人員應該能夠訪問日誌數據以進 day-to-day 行操作。所有發行到生產環境的版本都應在部署之前由核准步驟來控制。

建置步驟

無。在測試和測試環境中使用的相同成品會在生產環境中重複使用。

部署步驟

在測試環境中核准和部署之後，自動啟動生產環境中的`main`分支（Git GitHub flow）或分支（幹線或 Flow）的部署。`release`以下是生產環境中的部署步驟：

1. 在生產環境中部署`release`分支（Gitflow）或`main`分支（幹線或 GitHub 流程）
2. 暫停由指定人員進行手動核准
3. 下載發佈的成品
4. 執行資料庫版本
5. 執行 IAC 部署

基於 Git 的開發的最佳實踐

要成功採用基於 Git 的開發，請務必遵循一組促進協作，維護代碼質量以及支持持續集成和持續交付（CI/CD）的最佳實踐非常重要。除了本指南中的最佳做法外，還請參閱 AWS Well-Architected DevOps 的指引。以下是基於 Git 的開發的一些關鍵最佳實踐： AWS

- 保持較小且頻繁的變更 — 鼓勵開發人員進行微小的漸進變更或功能。這樣可以減少合併衝突的風險，並使其更容易快速識別和修復問題。
- 使用功能切換 — 若要管理不完整或實驗性功能的發行，請使用功能切換或功能旗標。這有助於您隱藏，啟用或禁用生產中的特定功能，而不會影響主分支的穩定性。
- 維護強大的測試套件 — 全面，維護良好的測試套件對於及早檢測問題並驗證代碼庫是否保持穩定至關重要。投資於測試自動化，並優先修復任何失敗的測試。
- 擁抱持續集成 — 使用持續集成工具和實踐自動構建，測試代碼更改並集成到`develop`分支（Gitflow）或`main`分支（Trunk 或 GitHub Flow）中。這有助於您及早發現 catch 題並簡化開發流程。
- 執行代碼審查 — 鼓勵對代碼進行同行審查，以保持質量，共享知識並在將潛在 catch 題集成到`main`分支之前發現它們。使用提取請求或其他程式碼檢閱工具來加速此程序。
- 監視和修復損壞的構建 — 當構建中斷或測試失敗時，請盡快優先修復問題。這使`develop`分支（Gitflow）或分`main`支（幹線或 GitHub Flow）處於可釋放的狀態，並將對其他開發人員的影響降到最低。
- 溝通和協作 — 促進團隊成員之間的開放式溝通和協作。確保開發人員知道正在進行的工作以及對代碼庫進行的更改。
- 持續重構 — 定期重構程式碼庫，以提高其可維護性並減少技術負債。鼓勵開發人員將代碼保留在比他們發現的更好的狀態。
- 對於複雜的任務使用短暫的分支-對於較大或更複雜的任務，請使用短期分支（也稱為任務分支）來處理更改。但是，請確保保持分支壽命短，通常不到一天。盡快將更改合併回`develop`分支（Gitflow）或`main`分支（主幹或 GitHub 流程）中。與一個大型合併請求相比，團隊更容易使用和處理更小且頻繁的合併和審核。
- 培訓和支持團隊 — 為剛接觸 Git 開發的開發人員或需要採用其最佳實踐指導的開發人員提供培訓和支持。

Git 分支策略

本指南詳細介紹了以下基於 Git 的分支策略，從最小到最複雜的順序：

- T@@ run k — 基於 Trunk 的開發是一種軟件開發實踐，其中所有開發人員都在單個分支上工作，通常稱為`trunk`或`main`分支。這種方法背後的想法是通過頻繁整合代碼更改並依賴自動化測試和持續集成來保持代碼庫持續釋放的狀態。
- GitHub 流程 — GitHub Flow 是輕量型、以分支為基礎的工作流程，由開發。 GitHub它是基於短命`feature`分支的想法。當功能完成並準備好部署時，該功能會合併到`main`分支中。
- Gitflow — 使用 Gitflow 方法，可以在各個功能分支中完成開發。核准之後，您可以將`feature`分支合併到通常名為的整合分支中`develop`。當`develop`分支中累積了足夠的功能時，將創建一個`release`分支以將功能部署到較高的環境中。

每個分支策略都有優點和缺點。儘管它們都使用相同的環境，但並非所有使用相同的分支或手動批准步驟。在本指南的這一部分中，詳細檢閱每個分支策略，以便您熟悉其細微差別，並評估它是否符合您組織的使用案例。

本節主題：

- 幹線分支策略
- GitHub 流程分支策略
- 潮流分支策略

幹線分支策略

基於 Trunk 的開發是一種軟件開發實踐，其中所有開發人員都在單個分支上工作，通常稱為`trunk`或`main`分支。這種方法背後的想法是通過頻繁整合代碼更改並依賴自動化測試和持續集成來保持代碼庫持續釋放的狀態。

在基於主幹的開發中，開發人員每天多次將其更改提交到`main`分支，旨在進行小型的增量更新。這可讓您快速回饋迴圈、減少合併衝突的風險，並促進團隊成員之間的協同合作。這種做法強調維護良好的測試套件的重要性，因為它依賴於自動化測試來及早發現潛在問題，並確保代碼庫保持穩定和可釋放。

基於 Trunk 的開發通常與基於功能的開發（也稱為功能分支或功能驅動開發）形成鮮明對比，其中每個新功能或錯誤修復都是在自己的專用分支中開發的，與主分支分開。主幹型開發與功能型開發之間的選擇取決於各種因素，例如團隊規模、專案需求，以及協同作業、整合頻率和發行管理之間所需的平衡。

如需 Trunk 分支策略的詳細資訊，請參閱下列資源：

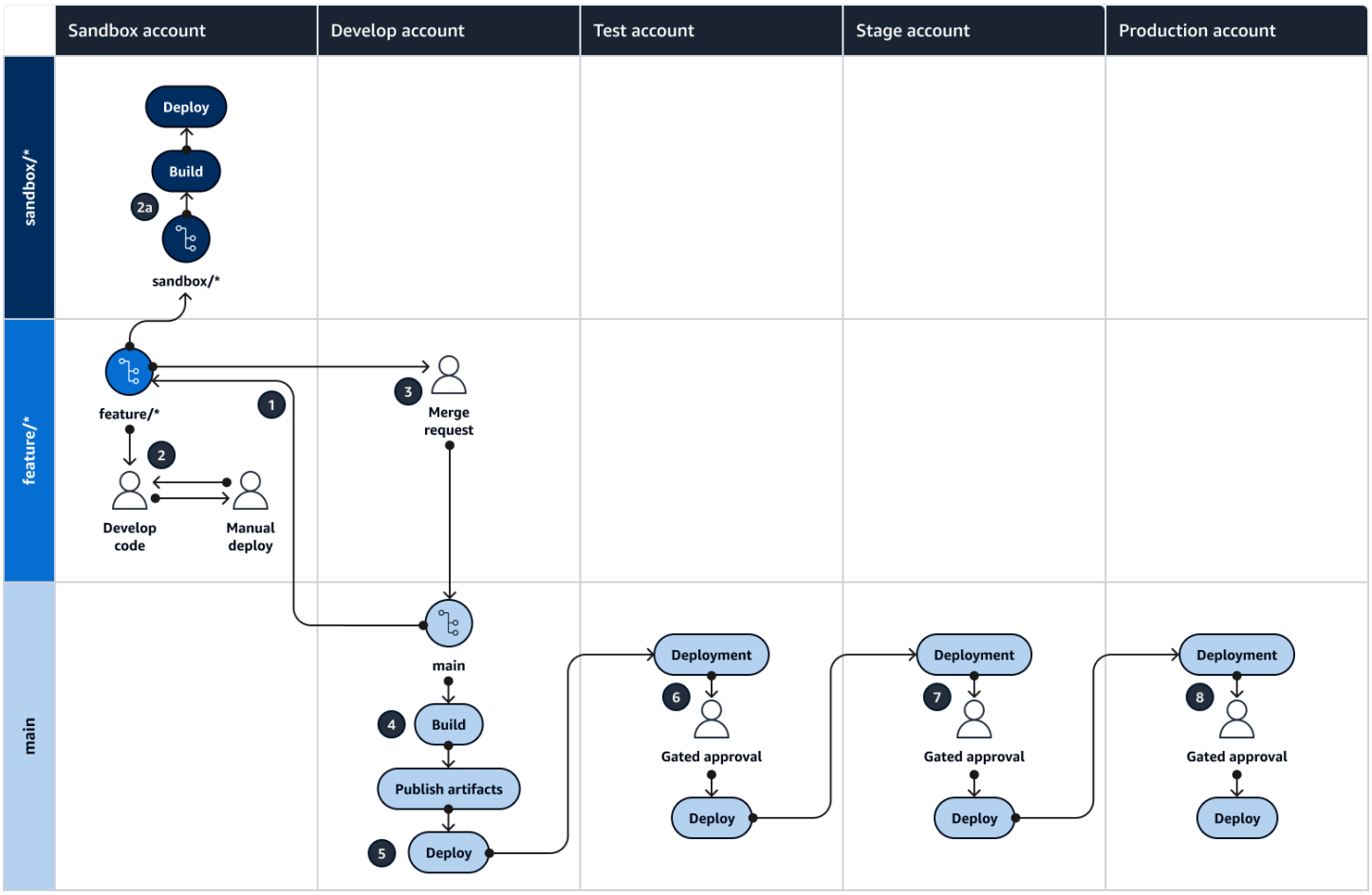
- 實作多帳戶 DevOps 環境的幹線分支策略 (AWS 規範指引)
- 基於主幹的開發簡介（基於主幹的開發網站）

本節主題：

- 幹線策略的視覺化概觀
- 幹線策略中的分支
- 幹線策略的優點和缺點

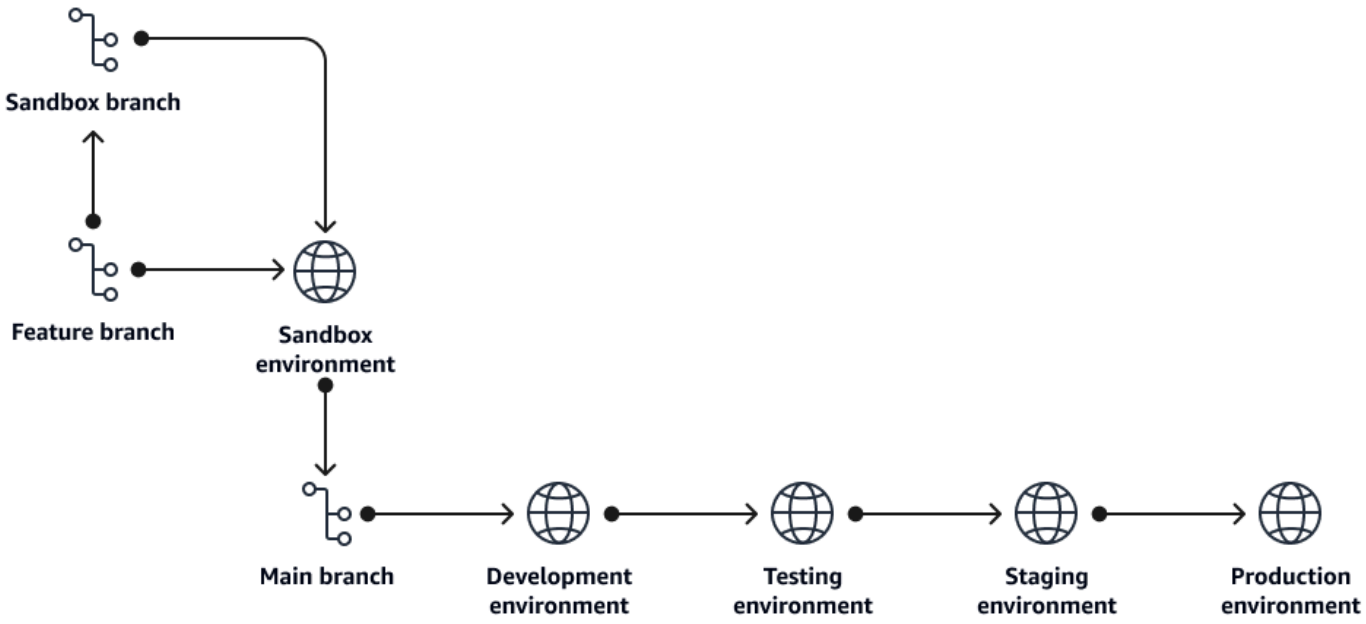
幹線策略的視覺化概觀

下圖可以像 Punnett 廣場（維基百科）一樣使用，以了解樹幹分支策略。將垂直軸上的分支與水平軸上的 AWS 環境對齊，以決定在每個案例中要執行的動作。圈出的數字會引導您完成圖表中表示的動作順序。此圖表顯示 Trunk 分支策略的開發工作流程，從沙箱環境中的`feature`分支到`main`分支的生產版本。如需有關每個環境中發生之活動的詳細資訊，請參閱本指南中的環DevOps 境。



幹線策略中的分支

幹線分支策略通常具有以下分支。



功能分支

您可以開發功能或在feature分支中建立 hotfix。要創建一個feature分支，你分支離分main支。開發人員在feature分支中迭代，提交和測試代碼。功能完成後，開發人員會提升該功能。從feature分支前進只有兩條路徑：

- 合併到分sandbox支
- 創建一個合併請求到main分支

命名慣例：	`feature/<story number>_<developer initials>_<descriptor>`
命名慣例示例：	`feature/123456_MS_Implement_Feature_A`

沙箱分支

這個分支是一個非標準的幹線分支，但它對於 CI/CD 管道開發很有用。該sandbox分支主要用於以下目的：

- 使用 CI/CD 管線執行沙箱環境的完整部署
- 在提交合併請求之前，先開發和測試管道，以便在較低的環境中進行完整測試，例如開發或測試。

Sandbox分支在本質上是暫時的，旨在是短暫的。它們應該在特定測試完成後刪除。

命名慣例：	sandbox/<story number>_<developer initials>_<descriptor>
命名慣例示例：	sandbox/123456_MS_Test_Pipeline_Deploy

主要分支

該main分支始終表示正在生產中運行的代碼。代碼從分支main，開發，然後合併回main。部署可main以針對任何環境。若要防止刪除，請為分支啟用main分支保護。

命名慣例：	main

補丁分支

主幹型工作流程中沒有專用hotfix分支。修補程式使用feature分支。

幹線策略的優點和缺點

Trunk 分支策略非常適合具有較強溝通能力的規模、成熟、開發團隊。如果您有應用程序的連續滾動功能版本，它也可以很好地工作。如果您擁有龐大或分散的開發團隊，或者您擁有廣泛的計劃功能版本，則此功能並不適合。合併衝突會在此模型中發生，因此請注意解決合併衝突是一項關鍵技能。所有團隊成員必須接受相應的培訓。

優點

基於 Trunk 的開發提供了幾個優勢，可以改善開發流程，簡化協作並提高軟件的整體質量。以下是一些主要優點：

- 更快的回饋迴圈 — 透過以中繼為基礎的開發，開發人員可以頻繁地整合程式碼變更，通常每天多次。這樣可以更快速地提供潛在問題的回饋，並協助開發人員比以功能為基礎的開發模型更快地識別和修復問題。
- 減少合併衝突 — 在基於主幹的開發中，由於不斷整合更改，因此可以最小化大而複雜的合併衝突的風險。這有助於維護更簡潔的程式碼基底，並減少解決衝突所花費的時間。在基於功能的開發中，解決衝突既耗時又容易出錯。

- 改善協同作業 — 以 Trunk 為基礎的開發可鼓勵開發人員在同一個分支機構上共同合作，促進團隊內部更好的溝通與協作。這可以導致更快的問題解決和更具凝聚力的團隊動態。
- 更輕鬆的程式碼檢閱 — 因為程式碼變更在中繼式開發中較小且頻繁，因此進行徹底的程式碼檢閱可能會比較容易。較小的變更通常更容易理解和檢閱，從而更有效地識別潛在問題和改進。
- 持續整合與交付 — 以中繼為基礎的開發支援持續整合與持續交付 (CI/CD) 的原則。透過將程式碼庫保持在可釋放的狀態並頻繁整合變更，團隊可以更輕鬆地採用 CI/CD 實務，進而加快部署週期並改善軟體品質。
- 增強的代碼質量-通過頻繁的集成，測試和代碼審查，基於主幹的開發可以有助於提高整體代碼質量。開發人員可以更快地發現和修復問題，從而減少技術債務隨著時間累積的可能性。
- 簡化的分支策略 — 以中繼為基礎的開發透過減少長期分支機構的數量來簡化分支策略。這可以更容易地管理和維護代碼庫，特別是對於大型項目或團隊。

缺點

基於 Trunk 的開發確實有一些缺點，這可能會影響開發過程和團隊動態。以下是一些顯著的缺點：

- 有限的隔離 — 由於所有開發人員都在同一個分支上工作，所以團隊中的每個人都可以立即看到他們的變更。這可能會導致干擾或衝突，導致意外的副作用或破壞構建。相比之下，基於功能的開發可以更好地隔離更改，以便開發人員可以更獨立地工作。
- 測試壓力增加 — 基於 Trunk 的開發依靠持續集成和自動化測試來快速發現 catch 題。但是，這種方法可能會給測試基礎設施帶來很大的壓力，並且需要維護良好的測試套件。如果測試不全面或可靠，則可能會導致主分支中未檢測到的問題。
- 減少對發布的控制-基於 Trunk 的開發旨在使代碼庫處於連續可釋放的狀態。儘管這可能很有利，但它可能並不總是適合具有嚴格發布時間表的項目或需要一起發布特定功能的項目。基於功能的開發可以更好地控制功能的發布時間和方式。
- 代碼流失 — 隨著開發人員不斷將更改集成到主要分支中，基於中繼的開發可能會導致代碼流失增加。這可能會讓開發人員難以追蹤程式碼基底的目前狀態，而且在嘗試瞭解最近變更的影響時可能會造成混淆。
- 需要強大的團隊文化-基於 Trunk 的開發需要團隊成員之間的高水平紀律，溝通和協作。維護這可能具有挑戰性，尤其是在較大的團隊中，或者與這種方法經驗不足的開發人員合作時。
- 可擴充性挑戰 — 隨著開發團隊的規模成長，整合到主要分支中的程式碼變更數量可能會迅速增加。這可能會導致更頻繁的建置中斷和測試失敗，因此很難將程式碼基底保持在可釋放的狀態。

GitHub 流程分支策略

GitHub Flow 是一種輕量級的，基於分支的工作流程由. GitHub GitHub流程基於短期功能分支的想法，這些分支會在功能完成並準備部署時合併到主分支中。 GitHub 流量的主要原則是：

- 分支是輕量級的 — 開發人員只需單擊幾下即可為其工作創建功能分支，從而提高協作和實驗的能力，而不會影響主分支。
- 持續部署 — 只要將變更合併到主分支中，就會立即部署，以便快速回饋和反覆運算。
- 合併請求 — 開發人員使用合併請求來啟動討論和審查過程，然後再將其更改合併到主分支中。

如需 GitHub Flow 的詳細資訊，請參閱下列資源：

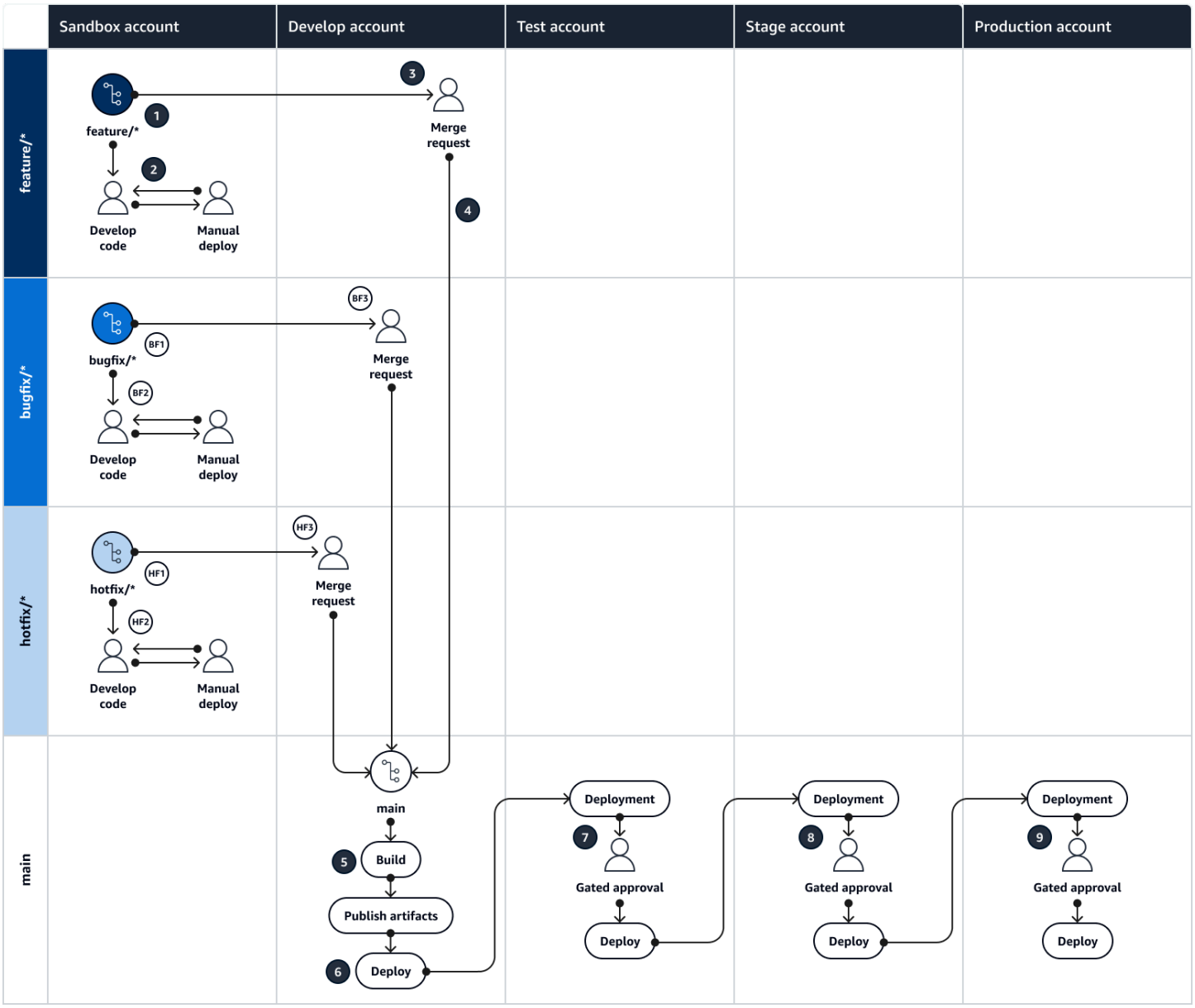
- 為多帳戶 DevOps 環境實作 GitHub 流程分支策略 (AWS 規範指引)
- GitHub 流快速入門（GitHub 文檔）
- 為什麼要 GitHub 流動？ （GitHub流量網站）

本節主題：

- GitHub 流程策略的視覺化概觀
- GitHub 流程策略中的分支
- GitHub 流量策略的優缺點

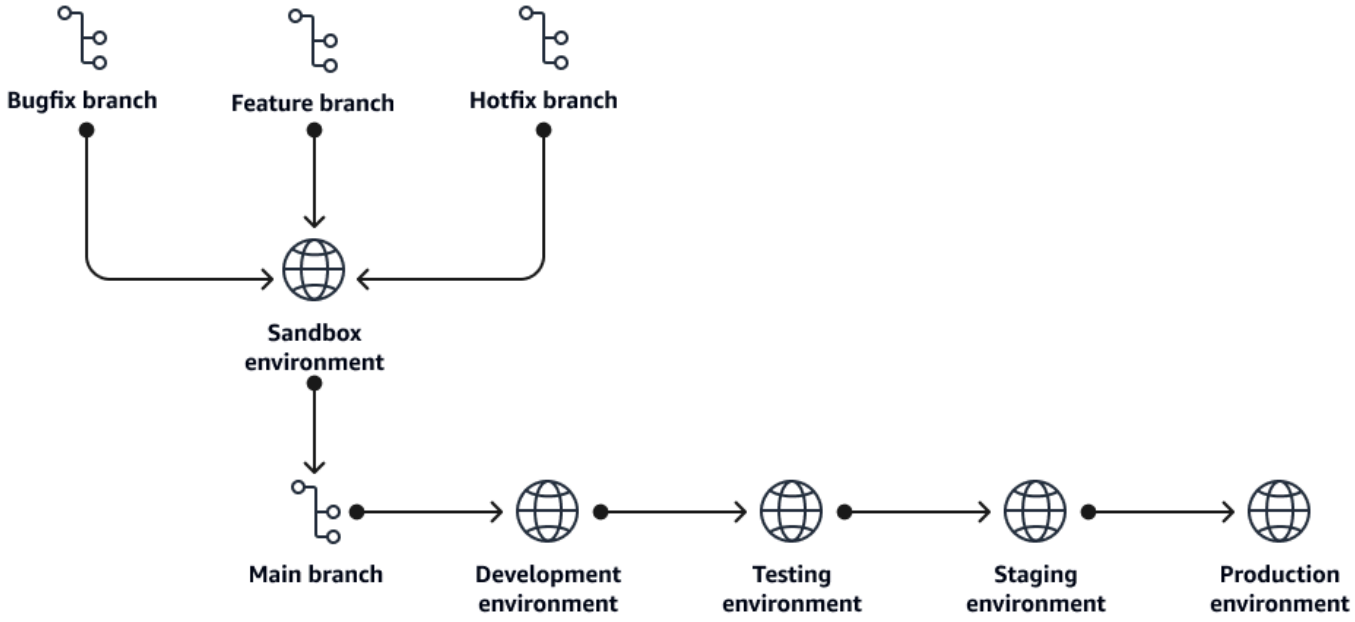
GitHub 流程策略的視覺化概觀

下圖可以像 Punnett 廣場一樣使用，以了解 GitHub 流量分支策略。將垂直軸上的分支與水平軸上的 AWS 環境對齊，以決定在每個案例中要執行的動作。圈出的數字會引導您完成圖表中表示的動作順序。此圖表顯示 GitHub Flow 分支策略的開發工作流程，從沙箱環境中的功能分支到主分支的生產版本。如需有關每個環境中發生之活動的詳細資訊，請參閱本指南中的環DevOps 境。



GitHub 流程策略中的分支

GitHub 流程分支策略通常具有以下分支。



功能分支

您可以在feature分支中開發特徵。要創建一個feature分支，你分支離分main支。開發人員在feature分支中迭代，提交和測試代碼。功能完成後，開發人員會透過建立合併要求來提升該功能main。

命名慣例：	`feature/<story number>_<developer initials>_<descriptor>`
命名慣例示例：	`feature/123456_MS_Implement_Feature_A`

錯誤修正分支

該bugfix分支用於解決問題。這些分支是從分main支出來的。在沙箱或任何較低環境中測試錯誤修正之後，可以main透過合併要求將其合併到更高的環境中進行升級。這是組織和跟踪的建議命名約定，也可以使用功能分支來管理此過程。

命名慣例：	`bugfix/<ticket number>_<developer initials>_<descriptor>`
命名慣例示例：	`bugfix/123456_MS_Fix_Problem_A`

補丁分支

該hotfix分支用於解決高影響的關鍵問題，並在開發人員與生產環境中部署的代碼之間的最小延遲。這些分支是從分main支出來的。在沙箱或任何較低的環境中測試 hotfix 之後，可以main透過合併要求將其合併到提升到較高的環境。這是組織和跟踪的建議命名約定，也可以使用功能分支來管理此過程。

命名慣例：	`hotfix/<ticket number>_<developer initials>_<descriptor>`
命名慣例示例：	`hotfix/123456_MS_Fix_Problem_A`

主要分支

該main分支始終表示正在生產中運行的代碼。通過使用合併請求將代碼從mainfeature分支合併到分支中。為了防止刪除並防止開發人員將代碼直接推送到main，請為分支啟用main分支保護。

命名慣例：	`main`

GitHub 流量策略的優缺點

Github Flow 分支策略非常適合具有較強溝通能力的規模、成熟、開發團隊。此策略非常適合想要實作持續交付的團隊，並且得到一般 CI/CD 引擎的良好支援。 GitHub Flow 是輕量級的，沒有太多規則，並且能夠支持快速發展的團隊。如果您的團隊有嚴格的合規性或發布流程要遵循，則不太適合。合併衝突在此模型中很常見，並且可能經常發生。解決合併衝突是一項關鍵技能，您必須相應地訓練所有團隊成員。

優點

GitHub Flow 提供多項優勢，可改善開發流程、簡化協同合作，並提升軟體的整體品質。以下是一些主要優點：

- 靈活且輕巧 — GitHub Flow 是輕量且靈活的工作流程，可協助開發人員在軟體開發專案上進行協作。它允許以最小的複雜性快速迭代和實驗。
- 簡化協同合作 — GitHub Flow 為管理功能開發提供清晰且簡化的流程。它鼓勵小的，集中的變化，可以快速審查和合併，從而提高效率。
- 清除版本控制 — 使用 GitHub Flow，每項變更都會在單獨的分支中進行。這會建立清晰且可追蹤的版本控制歷程記錄。這有助於開發人員跟踪和了解更改，必要時還原並維護可靠的代碼庫。

- 無縫持續整合 — GitHub Flow 與持續整合工具整合。建立提取要求可啟動自動化測試和部署程序。CI 工具可以幫助您在將更改合併到`main`分支之前徹底測試更改，從而降低將錯誤引入代碼庫的風險。
- 快速反饋和持續改進 — GitHub Flow 通過提取請求促進頻繁的代碼審查和討論來鼓勵快速反饋循環。這有助於早期發現問題，促進團隊成員之間的知識共享，並最終導致更高的代碼質量和開發團隊內部更好的協作。
- 簡化的回復和還原 — 如果程式碼變更導致未預期的錯誤或問題， GitHub Flow 會簡化復原或還原變更的程序。通過擁有清晰的提交和分支歷史記錄，可以更容易地識別和恢復有問題的更改，從而有助於維護穩定且功能強大的代碼庫。
- 輕量級學習曲線 — GitHub Flow 比 Gitflow 更容易學習和採用，特別是對於已經熟悉 Git 和版本控制概念的團隊而言。它的簡單性和直觀的分支模型使不同經驗水平的開發人員可以訪問它，從而減少了採用新開發工作流程相關的學習曲線。
- 持續開發 — GitHub Flow 讓團隊能夠採用持續部署方法，只要變更合併到`main`分支，就能立即部署。這個簡化的流程可消除不必要的延遲，並確保使用者能夠快速獲得最新的更新和改進功能。這會導致更加敏捷和響應迅速的開發週期。

缺點

雖然 GitHub Flow 提供了幾個優點，但重要的是要考慮其潛在缺點：

- 適用於大型專案的適用性有限 — GitHub Flow 可能不適合具有複雜程式碼庫和多個長期功能分支的大型專案。在這種情況下，更結構化的工作流程（例如 Gitflow）可能會對並發開發和發布管理提供更好的控制。
- 缺少正式發行結構 — GitHub Flow 未明確定義發行程序或支援版本控制、Hotfix 或維護分支等功能。對於需要嚴格發布管理或需要長期支持和維護的項目來說，這可能是一個限制。
- 對長期發行規劃的支援有限 — GitHub Flow 著重於短期功能分支，這些分支可能與需要長期發行規劃的專案 (例如具有嚴格藍圖或廣泛功能相依性的專案) 不符。在 GitHub Flow 的限制下，管理複雜的發行排程可能具有挑戰性。
- 經常發生合併衝突的可能性 — 由於 GitHub Flow 鼓勵頻繁地進行分支和合併，因此可能會遇到合併衝突，尤其是在具有大量開發活動的項目中。解決這些衝突可能很耗時，並且可能需要開發團隊的額外努力。
- 缺乏正式化的工作流程階段 — GitHub Flow 未定義明確的開發階段，例如 Alpha、beta 或發行候選階段。這可能會使傳達項目的當前狀態或不同分支或發行版本的穩定性級別變得更加困難。
- 突破性變更的影響 — 由於 GitHub Flow 鼓勵經常將變更合併到`main`分支中，因此導入影響程式碼基底穩定性的重大變更的風險較高。嚴格的代碼審查和測試實踐對於有效降低這種風險至關重要。

潮流分支策略

Gitflow 是一種分支模型，涉及使用多個分支將代碼從開發移到生產環境。Gitflow 非常適合具有計劃發布週期並且需要將功能集合定義為發布的團隊。開發是在個別功能分支中完成，這些分支已經核准合併到用於整合的開發分支中。這個分支中的功能被認為是準備生產。當所有計劃的功能都累積在開發分支中時，會為部署到較高環境建立發行分支。此分隔可改善控制要將哪些變更移至已定義明細表上的具名環境。如有必要，您可以將此程序加速成更快的部署模式。

如需 Gitflow 分支策略的詳細資訊，請參閱下列資源：

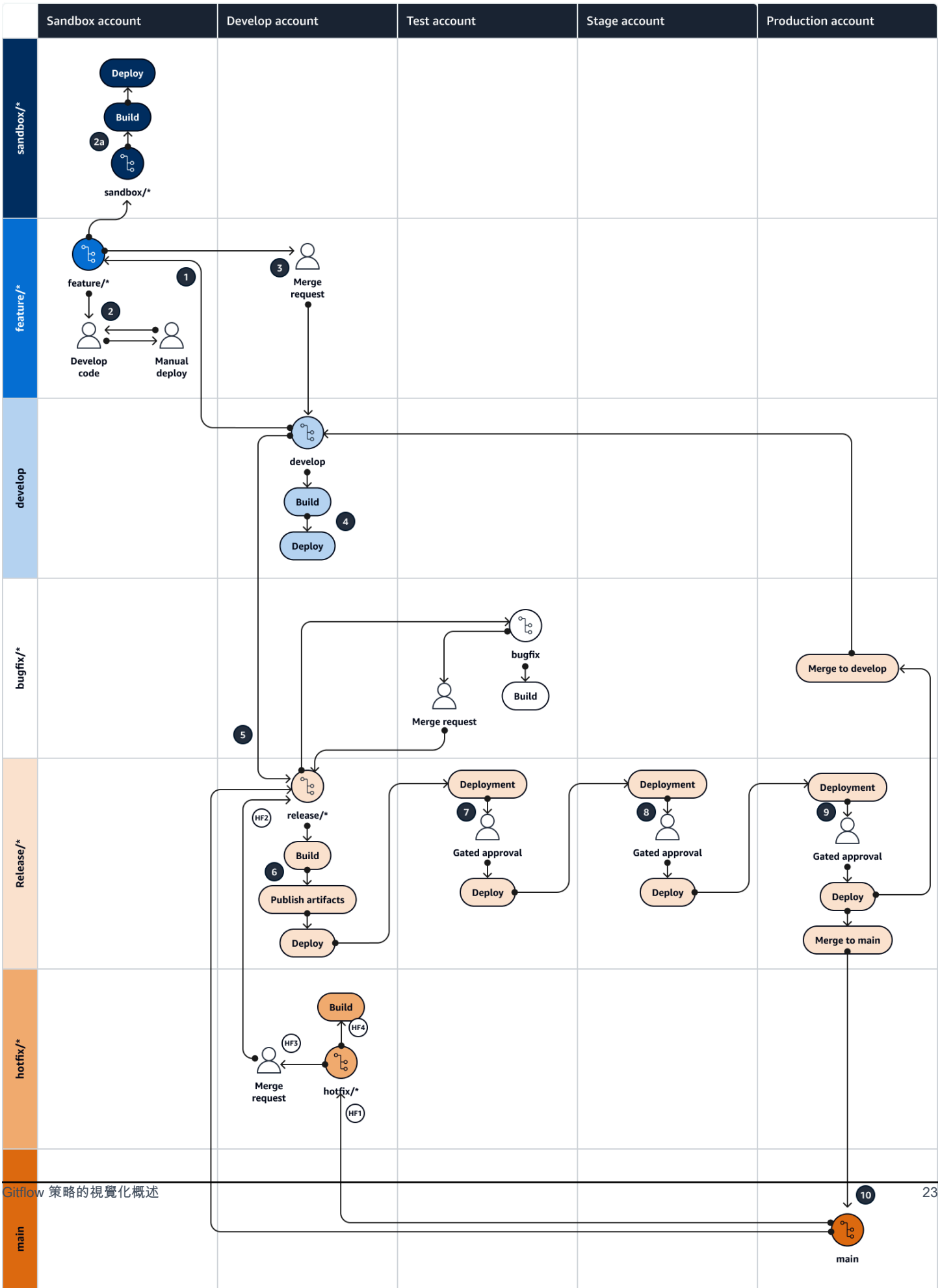
- 為多帳戶 DevOps 環境實施 Gitflow 分支策略（規範指南）AWS
- 原來的 Gitflow 博客（文森特·德森博客文章）
- Gitflow 工作流程 (大地圖)

本節主題：

- Gitflow 策略的視覺化概述
- Gitflow 策略中的分支
- 優點和吉特流策略的缺點

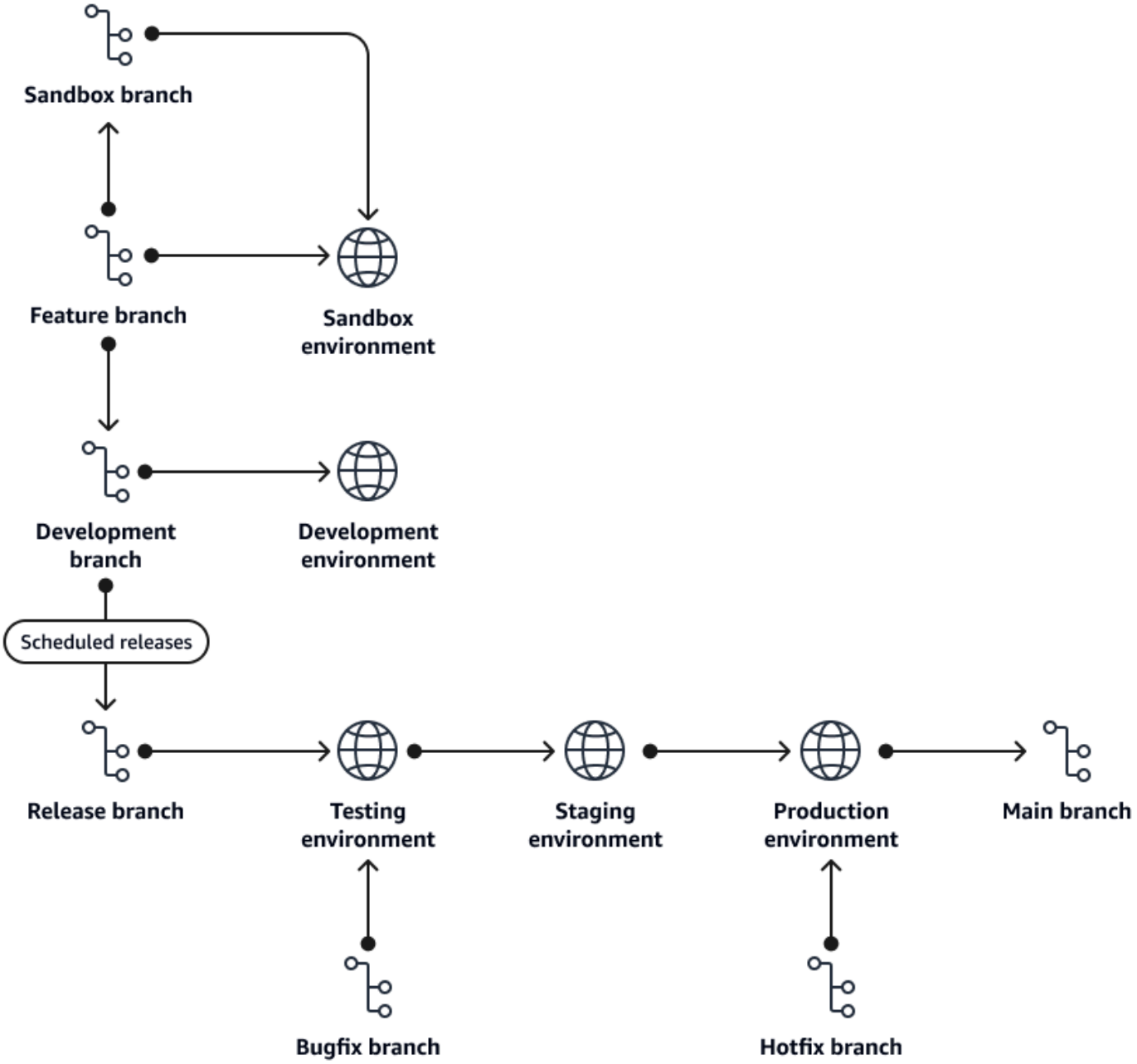
Gitflow 策略的視覺化概述

下圖可以像普內特廣場一樣使用，以了解 Gitflow 分支策略。將垂直軸上的分支與水平軸上的 AWS 環境對齊，以決定在每個案例中要執行的動作。圈出的數字會引導您完成圖表中表示的動作順序。如需有關每個環境中發生之活動的詳細資訊，請參閱本指南中的環DevOps 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分支是您在其中開發特徵的短期分支。分feature支是通過分支離開develop分支來創建的。開發人員在feature分支中迭代，提交和測試代碼。功能完成後，開發人員會提升該功能。從特徵分支向前只有兩條路徑：

- 合併到分sandbox支
- 創建一個合併請求到develop分支

命名慣例：	`feature/<story number>_<developer initials>_<descriptor>`
命名慣例示例：	`feature/123456_MS_Implement_Feature_A`

沙箱分支

該sandbox分支是 Gitflow 的非標準短期分支。但是，它對於 CI/CD 管道開發很有用。該sandbox分支主要用於以下目的：

- 使用 CI/CD 管線而非手動部署，對沙箱環境執行完整部署。
- 在提交合併請求之前，先開發和測試管道，以便在較低的環境中進行完整測試，例如開發或測試。

Sandbox分支在本質上是暫時的，並不意味著長壽。它們應該在特定測試完成後刪除。

命名慣例：	`sandbox/<story number>_<developer initials>_<descriptor>`
命名慣例示例：	`sandbox/123456_MS_Test_Pipeline_Deploy`

開發分公司

該develop分支是一個長期使用的分支，其中的功能已集成，構建，驗證和部署到開發環境。所有feature分支都合併到分develop支中。合併到develop分支是透過需要成功建置和兩次開發人員核准的合併要求來完成。為了防止刪除，請在分支上啟用develop分支保護。

命名慣例：	`develop`

發行分支

在 Gitflow 中，release分支機構是短期分支機構。這些分支很特別，因為您可以將它們部署到多個環境中，並採用構建一次，多個部署方法。 Release分支可以針對測試，測試或生產環境。在開發團隊決定將功能推廣到更高的環境之後，他們會建立新的release分支，並使用增加舊版本的版本號碼。在每個環境的閘口處，部署都需要手動核准才能繼續。 Release分支應該要求更改合併請求。

分release支部署到生產環境之後，應該合併回develop和main分支，以確保任何錯誤修正或 hotfix 合併回 future 的開發工作。

命名慣例：	release/v{major}.{minor}
命名慣例示例：	release/v1.0

主要分支

該main分支是一個長壽命的分支，它始終代表在生產環境中運行的代碼。從發main行管道部署成功後，程式碼會自動從發行分支合併到分支中。為了防止刪除，請在分支上啟用main分支保護。

命名慣例：	main

錯誤修正分支

該分bugfix支是一個短期分支，用於修復尚未發布到生產環境的發布分支中的問題。分bugfix支應該只用於將release分支中的修正提升到測試、測試或生產環境。一個bugfix分支總是從release分支上分支。

在測試錯誤修正之後，可以透過合併要求將其提升至release分支。然後，您可以按照標準發布過程將release分支向前推進。

命名慣例：	bugfix/<ticket>_<developer initials>_<descriptor>
命名慣例示例：	bugfix/123456_MS_Fix_Problem_A

補丁分支

該hotfix分支是一個短期分支，用於修復生產中的問題。它僅用於促進必須加速到達生產環境的修復程序。一個hotfix分支總是從main分支。

在測試 Hotfix 之後，您可以透過合併要求將其升級至從中建立的release分支main。對於測試，您可以按照標準發布過程將release分支向前推動。

命名慣例：	hotfix/<ticket>_<developer initials>_<descriptor>
命名慣例示例：	hotfix/123456_MS_Fix_Problem_A

優點和吉特流策略的缺點

Gitflow 分支策略非常適合具有嚴格發布和合規性要求的大型，分散式更多的團隊。Gitflow 為組織提供了可預測的發布週期，這通常是較大的組織的首選。Gitflow 也非常適合需要護欄正確完成軟件開發生命週期的團隊。這是因為該策略中內置了多種評論和質量保證的機會。Gitflow 也非常適合必須同時維護多個生產版本版本的團隊。GitFlow 的一些缺點是它比其他分支模型更複雜，並且需要嚴格遵守該模式才能成功完成。由於管理發布分支的嚴格性質，Gitflow 對於努力持續交付的組織而言不能很好地工作。Gitflow 發布分支可以是長壽命分支，如果未及時正確解決，則可以累積技術債務。

優點

基於 GITFlow 的開發提供了幾個優勢，可以改善開發過程，簡化協作並提高軟件的整體質量。以下是一些主要優點：

- 可預測的發布過程 — Gitflow 遵循常規且可預測的發布過程。它非常適合具有定期開發和發布節奏的團隊。
- 改善合作 — Gitflow 鼓勵使用feature和release分支機構。這兩個分支可以幫助團隊在彼此之間的依賴性最小的情況下 parallel 工作。
- 非常適合多種環境-Gitflow 使用release分支機構，這些分支可以是壽命更長的分支機構。這些分支使團隊能夠在較長的時間內定位單個發行版本。
- 生產中的多個版本 — 如果您的團隊在生產中支持多個版本的軟件，則 Gitflow release 分支機構支持此要求。
- 內置代碼質量評論 — Gitflow 需要並鼓勵在代碼升級到另一個環境之前使用代碼審查和批准。此過程通過對所有代碼促銷都要求此步驟消除了開發人員之間的摩擦。

- 組織利益 — Gitflow 在組織層面也具有優勢。Gitflow 鼓勵使用標準發布週期，這有助於組織了解和預測發布時間表。由於企業現在了解何時可以交付新功能，因此由於有設定的交付日期，因此可以減少時間軸的摩擦。

缺點

基於 Gitflow 的開發確實有一些缺點，可能會影響開發過程和團隊動態。以下是一些顯著的缺點：

- 複雜性 — Gitflow 對於新團隊來說是一種複雜的學習模式，您必須遵守 Gitflow 的規則才能成功使用它。
- 持續部署 — Gitflow 不適合以快速方式將許多部署發佈到生產環境的模型。這是因為 Gitflow 需要使用多個分支和嚴格的工作流程來管理分支。`release`
- 分支管理 — Gitflow 使用許多分支機構，這可能會變得負擔。跟踪各個分支並合併發布的代碼，以使分支彼此正確對齊可能會很具挑戰性。
- 技術債務 — 由於 Gitflow 發行版本通常比其他分支模型慢，因此可以累積更多的功能以進行釋放，這可能會導致技術債務累積。

在決定基於 GitFlow 的開發是否是其項目的正確方法時，團隊應仔細考慮這些缺點。

後續步驟

本指南解釋了三種常見的 Git 分支策略之間的差異： GitHub流量，Gitflow 和幹線。它詳細描述了他們的工作流程，並提供了每個工作流程的優點和缺點。接下來的步驟是為您的組織選擇其中一個標準工作流程。若要實作這些分支策略之一，請參閱下列內容：

- 為多帳戶環境實作幹線分支 DevOps 策略
- 為多帳戶環境實作 GitHub Flow 分支 DevOps 策略
- 為多帳戶環境實施 Gitflow 分支策略 DevOps

如果您不確定團隊要從何處開始使用 Git 和 DevOps 程序，我們建議您選擇標準解決方案並進行測試。使用標準分支慣例有助於團隊建立在現有文件之上，並了解哪些方法最適合他們。

如果策略不適用於您的組織或開發團隊，請不要害怕改變策略。開發團隊的需求和需求可能會隨著時間而改變，並且沒有單一的完美解決方案。

資源

本指南不包含 Git 的訓練；不過，如果您需要這項訓練，網際網路上有許多高品質的資源可供使用。我們建議您從 Git 文件網站開始。

下列資源可協助您在 AWS 雲端.

AWS 規定指引

- 為多帳戶環境實作幹線分支 DevOps 策略
- 為多帳戶環境實作 GitHub Flow 分支 DevOps 策略
- 為多帳戶環境實施 Gitflow 分支策略 DevOps

其他 AWS 指引

- AWS DevOps 指引
- AWS 部署管線參考架構
- 什麼是 DevOps？
- DevOps資源

其他資源

- 十二因素應用程序方法（12 因素 .net）
- Git 的秘密（ ）GitHub
- Gitflow
  - 原來的 Gitflow 博客（文森特·德森博客文章）
  - Gitflow 工作流程 (大地圖)
  - 啟用 Gitflow GitHub：如何使用 Git 流程工作流程搭配 GitHub 基礎存放庫 (影片) YouTube
  - Git 流程初始化示例（YouTube 視頻）
  - 從開始到結束的 Gitflow 發布分支（YouTube 視頻）
- GitHub 流程
  - GitHub 流快速入門（GitHub 文檔）

  - 為什麼要 GitHub 流動？（GitHub 流量網站）
- 行李箱
  - 基於主幹的開發簡介（基於後備箱的開發網站）

貢獻者

撰寫

- 邁克·斯蒂芬斯，高級雲端應用程式架構師 AWS
- Rayjan 威爾遜，高級雲應用程序架構師， AWS
- 阿布拉什·維諾德，團隊負責人，高級雲端應用程式架構師， AWS

檢閱

- 史蒂芬 DiCato, 高級安全顧問, AWS
- 雲端應用程式架構師 AWS
- 史蒂芬·古根海默，團隊負責人，高級雲應用程序架構師， AWS

技術寫作

- 莉莉 AbouHarb，高級技術作家， AWS

文件歷史紀錄

下表描述了本指南的重大變更。如果您想收到有關未來更新的通知，可以訂閱 RSS 摘要。

變更	描述	日期
初次出版	—	2024年2月15日

AWS 規範性指引詞彙表

以下是 AWS Prescriptive Guidance 提供的策略、指南和模式中常用的術語。若要建議項目，請使用詞彙表末尾的提供意見回饋連結。

數字

7 R

將應用程式移至雲端的七種常見遷移策略。這些策略以 Gartner 在 2011 年確定的 5 R 為基礎，包括以下內容：

- 重構/重新架構 – 充分利用雲端原生功能來移動應用程式並修改其架構，以提高敏捷性、效能和可擴展性。這通常涉及移植作業系統和資料庫。範例：將您的現場部署 Oracle 資料庫遷移至 Amazon Aurora PostgreSQL 相容版本。
- 平台轉換 (隨即重塑) – 將應用程式移至雲端，並引入一定程度的優化以利用雲端功能。範例：將您的現場部署 Oracle 資料庫遷移至 中的 Amazon Relational Database Service (Amazon RDS) for Oracle AWS 雲端。
- 重新購買 (捨棄再購買) – 切換至不同的產品，通常從傳統授權移至 SaaS 模型。範例：將您的客戶關係管理 (CRM) 系統遷移至 Salesforce.com。
- 主機轉換 (隨即轉移) – 將應用程式移至雲端，而不進行任何變更以利用雲端功能。範例：將您的現場部署 Oracle 資料庫遷移至 中 EC2 執行個體上的 Oracle AWS 雲端。
- 重新放置 (虛擬機器監視器等級隨即轉移) – 將基礎設施移至雲端，無需購買新硬體、重寫應用程式或修改現有操作。您可以將伺服器從內部部署平台遷移到相同平台的雲端服務。範例：將 Microsoft Hyper-V應用程式遷移至 AWS。
- 保留 (重新檢視) – 將應用程式保留在來源環境中。其中可能包括需要重要重構的應用程式，且您希望將該工作延遲到以後，以及您想要保留的舊版應用程式，因為沒有業務理由來進行遷移。
- 淘汰 – 解除委任或移除來源環境中不再需要的應用程式。

A

ABAC

請參閱屬性型存取控制。

抽象服務

請參閱 受管服務。

ACID

請參閱原子性、一致性、隔離性、耐久性。

主動-主動式遷移

一種資料庫遷移方法，其中來源和目標資料庫保持同步 (透過使用雙向複寫工具或雙重寫入操作)，且兩個資料庫都在遷移期間處理來自連接應用程式的交易。此方法支援小型、受控制批次的遷移，而不需要一次性切換。它更靈活，但比主動-被動遷移需要更多的工作。

主動-被動式遷移

一種資料庫遷移方法，其中來源和目標資料庫保持同步，但只有來源資料庫處理來自連接應用程式的交易，同時將資料複寫至目標資料庫。目標資料庫在遷移期間不接受任何交易。

彙總函數

在一組資料列上運作的 SQL 函數，會計算群組的單一傳回值。彙總函數的範例包括 SUM和 MAX。

AI

請參閱人工智慧。

AIOps

請參閱人工智慧操作。

匿名化

在資料集中永久刪除個人資訊的程序。匿名化有助於保護個人隱私權。匿名資料不再被視為個人資料。

反模式

經常用於重複性問題的解決方案，其中解決方案具有反生產力、無效或比替代解決方案更有效。

應用程式控制

一種安全方法，僅允許使用核准的應用程式，以協助保護系統免受惡意軟體攻擊。

應用程式組合

有關組織使用的每個應用程式的詳細資訊的集合，包括建置和維護應用程式的成本及其商業價值。此資訊是產品組合探索和分析程序的關鍵，有助於識別要遷移、現代化和優化的應用程式並排定其優先順序。

人工智慧 (AI)

電腦科學領域，致力於使用運算技術來執行通常與人類相關的認知功能，例如學習、解決問題和識別模式。如需詳細資訊，請參閱什麼是人工智慧？

人工智慧操作 (AIOps)

使用機器學習技術解決操作問題、減少操作事件和人工干預以及提高服務品質的程序。如需有關如何在 AWS 遷移策略中使用 AIOps 的詳細資訊，請參閱操作整合指南。

非對稱加密

一種加密演算法，它使用一對金鑰：一個用於加密的公有金鑰和一個用於解密的私有金鑰。您可以共用公有金鑰，因為它不用於解密，但對私有金鑰存取應受到高度限制。

原子性、一致性、隔離性、耐久性 (ACID)

一組軟體屬性，即使在出現錯誤、電源故障或其他問題的情況下，也能確保資料庫的資料有效性和操作可靠性。

屬性型存取控制 (ABAC)

根據使用者屬性 (例如部門、工作職責和團隊名稱) 建立精細許可的實務。如需詳細資訊，請參閱《 AWS Identity and Access Management (IAM) 文件》中的 ABAC for AWS。

授權資料來源

您存放主要版本資料的位置，被視為最可靠的資訊來源。您可以將授權資料來源中的資料複製到其他位置，以處理或修改資料，例如匿名、修訂或假名化資料。

可用區域

中的不同位置 AWS 區域 ，可隔離其他可用區域中的故障，並提供相同區域中其他可用區域的低成本、低延遲網路連線。

AWS 雲端採用架構 (AWS CAF)

的指導方針和最佳實務架構 AWS ，可協助組織制定高效且有效的計劃，以成功地移至雲端。 AWS CAF 將指導方針組織到六個重點領域：業務、人員、治理、平台、安全和營運。業務、人員和控管層面著重於業務技能和程序；平台、安全和操作層面著重於技術技能和程序。例如，人員層面針對處理人力資源 (HR)、人員配備功能和人員管理的利害關係人。因此， AWS CAF 為人員開發、訓練和通訊提供指引，協助組織做好成功採用雲端的準備。如需詳細資訊，請參閱 AWS CAF 網站和 AWS CAF 白皮書。

AWS 工作負載資格架構 (AWS WQF)

一種工具，可評估資料庫遷移工作負載、建議遷移策略，並提供工作預估值。 AWS WQF 隨附於 AWS Schema Conversion Tool (AWS SCT)。它會分析資料庫結構描述和程式碼物件、應用程式程式碼、相依性和效能特性，並提供評估報告。

B

錯誤的機器人

旨在中斷或傷害個人或組織的機器人。

BCP

請參閱業務持續性規劃。

行為圖

資源行為的統一互動式檢視，以及一段時間後的互動。您可以將行為圖與 Amazon Detective 搭配使用來檢查失敗的登入嘗試、可疑的 API 呼叫和類似動作。如需詳細資訊，請參閱偵測文件中的行為圖中的資料。

大端序系統

首先儲存最高有效位元組的系統。另請參閱 Endianness。

二進制分類

預測二進制結果的過程 (兩個可能的類別之一)。例如，ML 模型可能需要預測諸如「此電子郵件是否是垃圾郵件？」等問題 或「產品是書還是汽車？」

Bloom 篩選條件

一種機率性、記憶體高效的資料結構，用於測試元素是否為集的成員。

藍/綠部署

一種部署策略，您可以在其中建立兩個不同但相同的環境。您可以在一個環境 （藍色） 中執行目前的應用程式版本，並在另一個環境 （綠色） 中執行新的應用程式版本。此策略可協助您快速復原，並將影響降至最低。

機器人

透過網際網路執行自動化任務並模擬人類活動或互動的軟體應用程式。有些機器人有用或有益，例如在網際網路上為資訊編製索引的 Web 爬蟲程式。有些其他機器人稱為惡意機器人，旨在中斷或傷害個人或組織。

殭屍網路

受到惡意軟體感染且受單一方控制之機器人的網路，稱為機器人繼承器或機器人運算子。殭屍網路是擴展機器人及其影響的最佳已知機制。

分支

程式碼儲存庫包含的區域。儲存庫中建立的第一個分支是主要分支。您可以從現有分支建立新分支，然後在新分支中開發功能或修正錯誤。您建立用來建立功能的分支通常稱為功能分支。當準備好發佈功能時，可以將功能分支合併回主要分支。如需詳細資訊，請參閱關於分支 (GitHub 文件)。

碎片存取

在特殊情況下，以及透過核准的程序，讓使用者能夠快速存取他們通常無權存取 AWS 帳戶 的 。如需詳細資訊，請參閱 Well-Architected 指南中的 AWS 實作打破玻璃程序指標。

棕地策略

環境中的現有基礎設施。對系統架構採用棕地策略時，可以根據目前系統和基礎設施的限制來設計架構。如果正在擴展現有基礎設施，則可能會混合棕地和綠地策略。

緩衝快取

儲存最常存取資料的記憶體區域。

業務能力

業務如何創造價值 (例如，銷售、客戶服務或營銷)。業務能力可驅動微服務架構和開發決策。如需詳細資訊，請參閱在 AWS上執行容器化微服務白皮書的圍繞業務能力進行組織部分。

業務連續性規劃 (BCP)

一種解決破壞性事件 (如大規模遷移) 對營運的潛在影響並使業務能夠快速恢復營運的計畫。

C

CAF

請參閱AWS 雲端採用架構。

Canary 部署

版本對最終使用者的緩慢和增量版本。當您有信心時，您可以部署新版本並完全取代目前的版本。

CCoE

請參閱 Cloud Center of Excellence。

CDC

請參閱變更資料擷取。

變更資料擷取 (CDC)

追蹤對資料來源 (例如資料庫表格) 的變更並記錄有關變更的中繼資料的程序。您可以將 CDC 用於各種用途，例如稽核或複寫目標系統中的變更以保持同步。

混沌工程

故意引入故障或破壞性事件，以測試系統的彈性。您可以使用 AWS Fault Injection Service (AWS FIS) 執行實驗，為您的 AWS 工作負載帶來壓力，並評估其回應。

CI/CD

請參閱持續整合和持續交付。

分類

有助於產生預測的分類程序。用於分類問題的 ML 模型可預測離散值。離散值永遠彼此不同。例如，模型可能需要評估影像中是否有汽車。

用戶端加密

在目標 AWS 服務 接收資料之前，在本機加密資料。

雲端卓越中心 (CCoE)

一個多學科團隊，可推動整個組織的雲端採用工作，包括開發雲端最佳實務、調動資源、制定遷移時間表以及領導組織進行大規模轉型。如需詳細資訊，請參閱 AWS 雲端 企業策略部落格上的 CCoE 文章。

雲端運算

通常用於遠端資料儲存和 IoT 裝置管理的雲端技術。雲端運算通常連接到邊緣運算技術。

雲端操作模型

在 IT 組織中，用於建置、成熟和最佳化一或多個雲端環境的操作模型。如需詳細資訊，請參閱建置您的雲端操作模型。

採用雲端階段

組織在遷移至 時通常會經歷的四個階段 AWS 雲端：

- 專案 – 執行一些與雲端相關的專案以進行概念驗證和學習用途
- 基礎 – 進行基礎投資以擴展雲端採用 (例如，建立登陸區域、定義 CCoE、建立營運模型)

- 遷移 – 遷移個別應用程式
- 重塑 – 優化產品和服務，並在雲端中創新

這些階段由 Stephen Orban 於部落格文章 The Journey Toward Cloud-First 和 Enterprise Strategy 部落格上的採用階段中定義。 AWS 雲端 如需有關它們如何與 AWS 遷移策略相關的詳細資訊，請參閱遷移整備指南。

CMDB

請參閱組態管理資料庫。

程式碼儲存庫

透過版本控制程序來儲存及更新原始程式碼和其他資產 (例如文件、範例和指令碼) 的位置。常見的雲端儲存庫包括 GitHub或 Bitbucket Cloud。程式碼的每個版本都稱為分支。在微服務結構中，每個儲存庫都專用於單個功能。單一 CI/CD 管道可以使用多個儲存庫。

冷快取

一種緩衝快取，它是空的、未填充的，或者包含過時或不相關的資料。這會影響效能，因為資料庫執行個體必須從主記憶體或磁碟讀取，這比從緩衝快取讀取更慢。

冷資料

很少存取且通常是歷史資料的資料。查詢這類資料時，通常可接受慢查詢。將此資料移至效能較低且成本較低的儲存層或類別，可以降低成本。

電腦視覺 (CV)

使用機器學習從數位影像和影片等視覺化格式分析和擷取資訊的 AI 欄位。例如，Amazon SageMaker AI 提供 CV 的影像處理演算法。

組態偏離

對於工作負載，組態會從預期狀態變更。這可能會導致工作負載變得不合規，而且通常是漸進和無意的。

組態管理資料庫 (CMDB)

儲存和管理有關資料庫及其 IT 環境的資訊的儲存庫，同時包括硬體和軟體元件及其組態。您通常在遷移的產品組合探索和分析階段使用 CMDB 中的資料。

一致性套件

您可以組合的 AWS Config 規則和修補動作集合，以自訂您的合規和安全檢查。您可以使用 YAML 範本，將一致性套件部署為 AWS 帳戶 和 區域中或整個組織的單一實體。如需詳細資訊，請參閱 AWS Config 文件中的一致性套件。

持續整合和持續交付 (CI/CD)

自動化軟體發行程序的來源、建置、測試、暫存和生產階段的程序。CI/CD 通常被描述為管道。CI/CD 可協助您將程序自動化、提升生產力、改善程式碼品質以及加快交付速度。如需詳細資訊，請參閱持續交付的優點。CD 也可表示持續部署。如需詳細資訊，請參閱持續交付與持續部署。

CV

請參閱電腦視覺。

D

靜態資料

網路中靜止的資料，例如儲存中的資料。

資料分類

根據重要性和敏感性來識別和分類網路資料的程序。它是所有網路安全風險管理策略的關鍵組成部分，因為它可以協助您確定適當的資料保護和保留控制。資料分類是 AWS Well-Architected Framework 中安全支柱的元件。如需詳細資訊，請參閱資料分類。

資料偏離

生產資料與用於訓練 ML 模型的資料之間有意義的變化，或輸入資料隨時間有意義的變更。資料偏離可以降低 ML 模型預測的整體品質、準確性和公平性。

傳輸中的資料

在您的網路中主動移動的資料，例如在網路資源之間移動。

資料網格

架構架構，提供分散式、分散式資料擁有權與集中式管理。

資料最小化

僅收集和處理嚴格必要資料的原則。在 中實作資料最小化 AWS 雲端 可以降低隱私權風險、成本和分析碳足跡。

資料周邊

AWS 環境中的一組預防性防護機制，可協助確保只有信任的身分才能從預期的網路存取信任的資源。如需詳細資訊，請參閱在 上建置資料周邊 AWS。

資料預先處理

將原始資料轉換成 ML 模型可輕鬆剖析的格式。預處理資料可能意味著移除某些欄或列，並解決遺失、不一致或重複的值。

資料來源

在整個生命週期中追蹤資料的原始伺服器和歷史記錄的程序，例如資料的產生、傳輸和儲存方式。

資料主體

正在收集和處理其資料的個人。

資料倉儲

支援商業智慧的資料管理系統，例如 分析。資料倉儲通常包含大量歷史資料，通常用於查詢和分析。

資料庫定義語言 (DDL)

用於建立或修改資料庫中資料表和物件之結構的陳述式或命令。

資料庫處理語言 (DML)

用於修改 (插入、更新和刪除) 資料庫中資訊的陳述式或命令。

DDL

請參閱資料庫定義語言。

深度整體

結合多個深度學習模型進行預測。可以使用深度整體來獲得更準確的預測或估計預測中的不確定性。

深度學習

一個機器學習子領域，它使用多層人工神經網路來識別感興趣的輸入資料與目標變數之間的對應關係。

深度防禦

這是一種資訊安全方法，其中一系列的安全機制和控制項會在整個電腦網路中精心分層，以保護網路和其中資料的機密性、完整性和可用性。當您在 上採用此策略時 AWS，您可以在 AWS Organizations 結構的不同層新增多個控制項，以協助保護資源。例如，defense-in-depth方法可能會結合多重要素驗證、網路分割和加密。

委派的管理員

在 中 AWS Organizations，相容的服務可以註冊 AWS 成員帳戶，以管理組織的帳戶和管理該服務的許可。此帳戶稱為該服務的委派管理員。如需詳細資訊和相容服務清單，請參閱 AWS Organizations 文件中的可搭配 AWS Organizations運作的服務。

部署

在目標環境中提供應用程式、新功能或程式碼修正的程序。部署涉及在程式碼庫中實作變更，然後在應用程式環境中建置和執行該程式碼庫。

開發環境

請參閱 環境。

偵測性控制

一種安全控制，用於在事件發生後偵測、記錄和提醒。這些控制是第二道防線，提醒您注意繞過現有預防性控制的安全事件。如需詳細資訊，請參閱在 AWS上實作安全控制中的偵測性控制。

開發值串流映射 (DVSM)

一種程序，用於識別對軟體開發生命週期中的速度和品質造成負面影響的限制並排定優先順序。DVSM 延伸了原本專為精簡製造實務設計的價值串流映射程序。它著重於透過軟體開發程序建立和移動價值所需的步驟和團隊。

數位分身

真實世界系統的虛擬呈現，例如建築物、工廠、工業設備或生產線。數位分身支援預測性維護、遠端監控和生產最佳化。

維度資料表

在星星結構描述中，較小的資料表包含有關事實資料表中量化資料的資料屬性。維度資料表屬性通常是文字欄位或離散數字，其行為類似於文字。這些屬性通常用於查詢限制、篩選和結果集標記。

災難

防止工作負載或系統在其主要部署位置中實現其業務目標的事件。這些事件可能是自然災難、技術故障或人為動作的結果，例如意外設定錯誤或惡意軟體攻擊。

災難復原 (DR)

您用來將災難造成的停機時間和資料遺失降至最低的策略和程序。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的 上工作負載災難復原 AWS：雲端中的復原。

DML

請參閱資料庫處理語言。

領域驅動的設計

一種開發複雜軟體系統的方法，它會將其元件與每個元件所服務的不斷發展的領域或核心業務目標相關聯。Eric Evans 在其著作 Domain-Driven Design: Tackling Complexity in the Heart of Software (Boston: Addison-Wesley Professional, 2003) 中介紹了這一概念。如需有關如何將領域驅動的設計與 strangler fig 模式搭配使用的資訊，請參閱使用容器和 Amazon API Gateway 逐步現代化舊版 Microsoft ASP.NET (ASMX) Web 服務。

DR

請參閱災難復原。

偏離偵測

追蹤與基準組態的偏差。例如，您可以使用 AWS CloudFormation 來偵測系統資源中的偏離，也可以使用 AWS Control Tower 來偵測登陸區域中可能影響控管要求合規性的變更。 https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-stack-drift.html

DVSM

請參閱開發值串流映射。

E

EDA

請參閱探索性資料分析。

EDI

請參閱電子資料交換。

邊緣運算

提升 IoT 網路邊緣智慧型裝置運算能力的技術。與雲端運算相比，邊緣運算可以減少通訊延遲並改善回應時間。

電子資料交換 (EDI)

在組織之間自動交換商業文件。如需詳細資訊，請參閱什麼是電子資料交換。

加密

一種運算程序，可將人類可讀取的純文字資料轉換為加密文字。

加密金鑰

由加密演算法產生的隨機位元的加密字串。金鑰長度可能有所不同，每個金鑰的設計都是不可預測且唯一的。

端序

位元組在電腦記憶體中的儲存順序。大端序系統首先儲存最高有效位元組。小端序系統首先儲存最低有效位元組。

端點

請參閱 服務端點。

端點服務

您可以在虛擬私有雲端 (VPC) 中託管以與其他使用者共用的服務。您可以使用 建立端點服務，AWS PrivateLink 並將許可授予其他 AWS 帳戶 或 AWS Identity and Access Management (IAM) 委託人。這些帳戶或主體可以透過建立介面 VPC 端點私下連接至您的端點服務。如需詳細資訊，請參閱 Amazon Virtual Private Cloud (Amazon VPC) 文件中的建立端點服務。

企業資源規劃 (ERP)

一種系統，可自動化和管理企業的關鍵業務流程 （例如會計、MES 和專案管理）。

信封加密

使用另一個加密金鑰對某個加密金鑰進行加密的程序。如需詳細資訊，請參閱 AWS Key Management Service (AWS KMS) 文件中的信封加密。

環境

執行中應用程式的執行個體。以下是雲端運算中常見的環境類型：

- 開發環境 – 執行中應用程式的執行個體，只有負責維護應用程式的核心團隊才能使用。開發環境用來測試變更，然後再將開發環境提升到較高的環境。此類型的環境有時稱為測試環境。
- 較低的環境 – 應用程式的所有開發環境，例如用於初始建置和測試的開發環境。
- 生產環境 – 最終使用者可以存取的執行中應用程式的執行個體。在 CI/CD 管道中，生產環境是最後一個部署環境。
- 較高的環境 – 核心開發團隊以外的使用者可存取的所有環境。這可能包括生產環境、生產前環境以及用於使用者接受度測試的環境。

epic

在敏捷方法中，有助於組織工作並排定工作優先順序的功能類別。epic 提供要求和實作任務的高層級描述。例如， AWS CAF 安全概念包括身分和存取管理、偵測控制、基礎設施安全、資料保護和事件回應。如需有關 AWS 遷移策略中的 Epic 的詳細資訊，請參閱計畫實作指南。

ERP

請參閱企業資源規劃。

探索性資料分析 (EDA)

分析資料集以了解其主要特性的過程。您收集或彙總資料，然後執行初步調查以尋找模式、偵測異常並檢查假設。透過計算摘要統計並建立資料可視化來執行 EDA。

F

事實資料表

星狀結構描述中的中央資料表。它存放有關業務操作的量化資料。一般而言，事實資料表包含兩種類型的資料欄：包含度量的資料，以及包含維度資料表外部索引鍵的資料欄。

快速失敗

一種使用頻繁和增量測試來縮短開發生命週期的理念。這是敏捷方法的關鍵部分。

故障隔離界限

在 中 AWS 雲端，像是可用區域 AWS 區域、控制平面或資料平面等邊界會限制故障的影響，並有助於改善工作負載的彈性。如需詳細資訊，請參閱AWS 故障隔離界限。

功能分支

請參閱分支。

特徵

用來進行預測的輸入資料。例如，在製造環境中，特徵可能是定期從製造生產線擷取的影像。

功能重要性

特徵對於模型的預測有多重要。這通常表示為可以透過各種技術來計算的數值得分，例如 Shapley Additive Explanations (SHAP) 和積分梯度。如需詳細資訊，請參閱 的機器學習模型可解譯性 AWS。

特徵轉換

優化 ML 程序的資料，包括使用其他來源豐富資料、調整值、或從單一資料欄位擷取多組資訊。這可讓 ML 模型從資料中受益。例如，如果將「2021-05-27 00:15:37」日期劃分為「2021」、「五月」、「週四」和「15」，則可以協助學習演算法學習與不同資料元件相關聯的細微模式。

少量擷取提示

在要求 LLM 執行類似的任務之前，提供少量示範任務和所需輸出的範例。此技術是內容內學習的應用程式，其中模型會從內嵌在提示中的範例 (快照) 中學習。對於需要特定格式、推理或網域知識的任務，少量的提示非常有效。另請參閱零鏡頭提示。

FGAC

請參閱精細存取控制。

精細存取控制 (FGAC)

使用多個條件來允許或拒絕存取請求。

閃切遷移

一種資料庫遷移方法，透過變更資料擷取使用連續資料複寫，以盡可能在最短的時間內遷移資料，而不是使用分階段方法。目標是將停機時間降至最低。

FM

請參閱基礎模型。

基礎模型 (FM)

大型深度學習神經網路，已針對廣義和未標記資料的大量資料集進行訓練。FMs 能夠執行各種一般任務，例如了解語言、產生文字和影像，以及以自然語言交談。如需詳細資訊，請參閱什麼是基礎模型。

G

生成式 AI

已針對大量資料進行訓練的 AI 模型子集，可使用簡單的文字提示建立新的內容和成品，例如影像、影片、文字和音訊。如需詳細資訊，請參閱什麼是生成式 AI。

地理封鎖

請參閱地理限制。

地理限制 (地理封鎖)

Amazon CloudFront 中的選項，可防止特定國家/地區的使用者存取內容分發。您可以使用允許清單或封鎖清單來指定核准和禁止的國家/地區。如需詳細資訊，請參閱 CloudFront 文件中的限制內容的地理分佈。

Gitflow 工作流程

這是一種方法，其中較低和較高環境在原始碼儲存庫中使用不同分支。Gitflow 工作流程會被視為舊版，而以幹線為基礎的工作流程是現代、偏好的方法。

黃金影像

系統或軟體的快照，做為部署該系統或軟體新執行個體的範本。例如，在製造中，黃金映像可用於在多個裝置上佈建軟體，並有助於提高裝置製造操作的速度、可擴展性和生產力。

綠地策略

新環境中缺乏現有基礎設施。對系統架構採用綠地策略時，可以選擇所有新技術，而不會限制與現有基礎設施的相容性，也稱為棕地。如果正在擴展現有基礎設施，則可能會混合棕地和綠地策略。

防護機制

有助於跨組織單位 (OU) 來管控資源、政策和合規的高層級規則。預防性防護機制會強制執行政策，以確保符合合規標準。透過使用服務控制政策和 IAM 許可界限來將其實作。偵測性防護機制可偵測政策違規和合規問題，並產生提醒以便修正。它們是透過使用 AWS Config AWS Security Hub、Amazon GuardDuty、、Amazon Inspector AWS Trusted Advisor和自訂 AWS Lambda 檢查來實作。

H

HA

請參閱高可用性。

異質資料庫遷移

將來源資料庫遷移至使用不同資料庫引擎的目標資料庫 (例如，Oracle 至 Amazon Aurora)。異質遷移通常是重新架構工作的一部分，而轉換結構描述可能是一項複雜任務。AWS 提供有助於結構描述轉換的 AWS SCT。

高可用性 (HA)

在遇到挑戰或災難時，工作負載能夠在不介入的情況下持續運作。HA 系統的設計目的是自動容錯移轉、持續提供高品質的效能，以及處理不同的負載和故障，並將效能影響降至最低。

歷史現代化

一種方法，用於現代化和升級操作技術 (OT) 系統，以更好地滿足製造業的需求。歷史資料是一種資料庫，用於從工廠中的各種來源收集和存放資料。

保留資料

從用於訓練機器學習模型的資料集中保留的部分歷史標記資料。您可以使用保留資料，透過比較模型預測與保留資料來評估模型效能。

異質資料庫遷移

將您的來源資料庫遷移至共用相同資料庫引擎的目標資料庫 (例如，Microsoft SQL Server 至 Amazon RDS for SQL Server)。同質遷移通常是主機轉換或平台轉換工作的一部分。您可以使用原生資料庫公用程式來遷移結構描述。

熱資料

經常存取的資料，例如即時資料或最近的轉譯資料。此資料通常需要高效能儲存層或類別，才能提供快速的查詢回應。

修補程序

緊急修正生產環境中的關鍵問題。由於其緊迫性，通常會在典型 DevOps 發行工作流程之外執行修補程式。

超級護理期間

在切換後，遷移團隊在雲端管理和監控遷移的應用程式以解決任何問題的時段。通常，此期間的長度為 1-4 天。在超級護理期間結束時，遷移團隊通常會將應用程式的責任轉移給雲端營運團隊。

I

IaC

將基礎設施視為程式碼。

身分型政策

連接至一或多個 IAM 主體的政策，可定義其在 AWS 雲端 環境中的許可。

閒置應用程式

90 天期間 CPU 和記憶體平均使用率在 5% 至 20% 之間的應用程式。在遷移專案中，通常會淘汰這些應用程式或將其保留在內部部署。

IIoT

請參閱工業物聯網。

不可變的基礎設施

為生產工作負載部署新基礎設施的模型，而不是更新、修補或修改現有基礎設施。不可變基礎設施本質上比可變基礎設施更一致、可靠且可預測。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的使用不可變基礎設施部署最佳實務。

傳入 (輸入) VPC

在 AWS 多帳戶架構中，接受、檢查和路由來自應用程式外部之網路連線的 VPC。AWS 安全參考架構建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

增量遷移

一種切換策略，您可以在其中將應用程式分成小部分遷移，而不是執行單一、完整的切換。例如，您最初可能只將一些微服務或使用者移至新系統。確認所有項目都正常運作之後，您可以逐步移動其他微服務或使用者，直到可以解除委任舊式系統。此策略可降低與大型遷移關聯的風險。

工業 4.0

Klaus 於 2016 年引進的術語，透過連線能力、即時資料、自動化、分析和 AI/ML 的進展，指製造程序的現代化。

基礎設施

應用程式環境中包含的所有資源和資產。

基礎設施即程式碼 (IaC)

透過一組組態檔案來佈建和管理應用程式基礎設施的程序。IaC 旨在協助您集中管理基礎設施，標準化資源並快速擴展，以便新環境可重複、可靠且一致。

工業物聯網 (IIoT)

在製造業、能源、汽車、醫療保健、生命科學和農業等產業領域使用網際網路連線的感測器和裝置。如需詳細資訊，請參閱建立工業物聯網 (IIoT) 數位轉型策略。

檢查 VPC

在 AWS 多帳戶架構中，集中式 VPC 可管理 VPCs 之間 （在相同或不同的 中 AWS 區域)、網際網路和內部部署網路之間的網路流量檢查。AWS 安全參考架構建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

物聯網 (IoT)

具有內嵌式感測器或處理器的相連實體物體網路，其透過網際網路或本地通訊網路與其他裝置和系統進行通訊。如需詳細資訊，請參閱什麼是 IoT？

可解釋性

機器學習模型的一個特徵，描述了人類能夠理解模型的預測如何依賴於其輸入的程度。如需詳細資訊，請參閱 的機器學習模型可解譯性 AWS。

IoT

請參閱物聯網。

IT 資訊庫 (ITIL)

一組用於交付 IT 服務並使這些服務與業務需求保持一致的最佳實務。ITIL 為 ITSM 提供了基礎。

IT 服務管理 (ITSM)

與組織的設計、實作、管理和支援 IT 服務關聯的活動。如需有關將雲端操作與 ITSM 工具整合的資訊，請參閱操作整合指南。

ITIL

請參閱 IT 資訊庫。

ITSM

請參閱 IT 服務管理。

L

標籤型存取控制 (LBAC)

強制存取控制 (MAC) 的實作，其中使用者和資料本身都會獲得明確指派的安全標籤值。使用者安全標籤和資料安全標籤之間的交集會決定使用者可以看到哪些資料列和資料欄。

登陸區域

登陸區域是架構良好的多帳戶 AWS 環境，可擴展且安全。這是一個起點，您的組織可以從此起點快速啟動和部署工作負載與應用程式，並對其安全和基礎設施環境充滿信心。如需有關登陸區域的詳細資訊，請參閱設定安全且可擴展的多帳戶 AWS 環境。

大型語言模型 (LLM)

預先訓練大量資料的深度學習 AI 模型。LLM 可以執行多個任務，例如回答問題、摘要文件、將文字翻譯成其他語言，以及完成句子。如需詳細資訊，請參閱什麼是 LLMs。

大型遷移

遷移 300 部或更多伺服器。

LBAC

請參閱標籤型存取控制。

最低權限

授予執行任務所需之最低許可的安全最佳實務。如需詳細資訊，請參閱 IAM 文件中的套用最低權限許可。

隨即轉移

請參閱 7 個 R。

小端序系統

首先儲存最低有效位元組的系統。另請參閱 Endianness。

LLM

請參閱大型語言模型。

較低的環境

請參閱 環境。

M

機器學習 (ML)

一種使用演算法和技術進行模式識別和學習的人工智慧。機器學習會進行分析並從記錄的資料 (例如物聯網 (IoT) 資料) 中學習，以根據模式產生統計模型。如需詳細資訊，請參閱機器學習。

主要分支

請參閱分支。

惡意軟體

旨在危及電腦安全或隱私權的軟體。惡意軟體可能會中斷電腦系統、洩露敏感資訊，或取得未經授權的存取。惡意軟體的範例包括病毒、蠕蟲、勒索軟體、特洛伊木馬程式、間諜軟體和鍵盤記錄器。

受管服務

AWS 服務 會 AWS 操作基礎設施層、作業系統和平台，而您會存取端點來存放和擷取資料。Amazon Simple Storage Service (Amazon S3) 和 Amazon DynamoDB 是受管服務的範例。這些也稱為抽象服務。

製造執行系統 (MES)

一種軟體系統，用於追蹤、監控、記錄和控制生產程序，將原物料轉換為現場成品。

MAP

請參閱遷移加速計劃。

機制

建立工具、推動工具採用，然後檢查結果以進行調整的完整程序。機制是在操作時強化和改善自身的循環。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的建置機制。

成員帳戶

除了屬於組織一部分的管理帳戶 AWS 帳戶 之外的所有 AWS Organizations。一個帳戶一次只能是一個組織的成員。

製造執行系統

請參閱製造執行系統。

訊息佇列遙測傳輸 (MQTT)

根據發佈/訂閱模式的輕量型machine-to-machine(M2M) 通訊協定，適用於資源受限的 IoT 裝置。

微服務

一種小型的獨立服務，它可透過定義明確的 API 進行通訊，通常由小型獨立團隊擁有。例如，保險系統可能包含對應至業務能力 (例如銷售或行銷) 或子領域 (例如購買、索賠或分析) 的微服務。微服務的優點包括靈活性、彈性擴展、輕鬆部署、可重複使用的程式碼和適應力。如需詳細資訊，請參閱使用無 AWS 伺服器服務整合微服務。

微服務架構

一種使用獨立元件來建置應用程式的方法，這些元件會以微服務形式執行每個應用程式程序。這些微服務會使用輕量型 API，透過明確定義的介面進行通訊。此架構中的每個微服務都可以進行更新、部署和擴展，以滿足應用程式特定功能的需求。如需詳細資訊，請參閱在 上實作微服務 AWS。

Migration Acceleration Program (MAP)

一種 AWS 計畫，提供諮詢支援、訓練和服務，協助組織建立強大的營運基礎，以移至雲端，並協助抵銷遷移的初始成本。MAP 包括用於有條不紊地執行舊式遷移的遷移方法以及一組用於自動化和加速常見遷移案例的工具。

大規模遷移

將大部分應用程式組合依波次移至雲端的程序，在每個波次中，都會以更快的速度移動更多應用程式。此階段使用從早期階段學到的最佳實務和經驗教訓來實作團隊、工具和流程的遷移工廠，以透過自動化和敏捷交付簡化工作負載的遷移。這是 AWS 遷移策略的第三階段。

遷移工廠

可透過自動化、敏捷的方法簡化工作負載遷移的跨職能團隊。遷移工廠團隊通常包括營運、業務分析師和擁有者、遷移工程師、開發人員以及從事 Sprint 工作的 DevOps 專業人員。20% 至 50% 之間的企業應用程式組合包含可透過工廠方法優化的重複模式。如需詳細資訊，請參閱此內容集中的遷移工廠的討論和雲端遷移工廠指南。

遷移中繼資料

有關完成遷移所需的應用程式和伺服器的資訊。每種遷移模式都需要一組不同的遷移中繼資料。遷移中繼資料的範例包括目標子網路、安全群組和 AWS 帳戶。

遷移模式

可重複的遷移任務，詳細描述遷移策略、遷移目的地以及所使用的遷移應用程式或服務。範例：使用 AWS Application Migration Service 重新託管遷移至 Amazon EC2。

遷移組合評定 (MPA)

線上工具，提供驗證商業案例以遷移至 的資訊 AWS 雲端。MPA 提供詳細的組合評定 (伺服器適當規模、定價、總體擁有成本比較、遷移成本分析) 以及遷移規劃 (應用程式資料分析和資料收集、應用程式分組、遷移優先順序，以及波次規劃)。MPA 工具 （需要登入） 可供所有 AWS 顧問和 APN 合作夥伴顧問免費使用。

遷移準備程度評定 (MRA)

使用 AWS CAF 取得組織雲端整備狀態的洞見、識別優缺點，以及建立行動計劃以消除已識別差距的程序。如需詳細資訊，請參閱遷移準備程度指南。MRA 是 AWS 遷移策略的第一階段。

遷移策略

用來將工作負載遷移至 的方法 AWS 雲端。如需詳細資訊，請參閱本詞彙表中的 7 個 Rs 項目，並請參閱動員您的組織以加速大規模遷移。

機器學習 (ML)

請參閱機器學習。

現代化

將過時的 (舊版或單一) 應用程式及其基礎架構轉換為雲端中靈活、富有彈性且高度可用的系統，以降低成本、提高效率並充分利用創新。如需詳細資訊，請參閱《》中的現代化應用程式的策略 AWS 雲端。

現代化準備程度評定

這項評估可協助判斷組織應用程式的現代化準備程度；識別優點、風險和相依性；並確定組織能夠在多大程度上支援這些應用程式的未來狀態。評定的結果就是目標架構的藍圖、詳細說明現代化程序的開發階段和里程碑的路線圖、以及解決已發現的差距之行動計畫。如需詳細資訊，請參閱《》中的評估應用程式的現代化準備 AWS 雲端程度。

單一應用程式 (單一)

透過緊密結合的程序作為單一服務執行的應用程式。單一應用程式有幾個缺點。如果一個應用程式功能遇到需求激增，則必須擴展整個架構。當程式碼庫增長時，新增或改進單一應用程式的功能也會變得更加複雜。若要解決這些問題，可以使用微服務架構。如需詳細資訊，請參閱將單一體系分解為微服務。

MPA

請參閱遷移產品組合評估。

MQTT

請參閱訊息佇列遙測傳輸。

多類別分類

一個有助於產生多類別預測的過程 (預測兩個以上的結果之一)。例如，機器學習模型可能會詢問「此產品是書籍、汽車還是電話？」 或者「這個客戶對哪種產品類別最感興趣？」

可變基礎設施

更新和修改生產工作負載現有基礎設施的模型。為了提高一致性、可靠性和可預測性， AWS Well-Architected Framework 建議使用不可變的基礎設施作為最佳實務。

O

OAC

請參閱原始存取控制。

OAI

請參閱原始存取身分。

OCM

請參閱組織變更管理。

離線遷移

一種遷移方法，可在遷移過程中刪除來源工作負載。此方法涉及延長停機時間，通常用於小型非關鍵工作負載。

OI

請參閱 操作整合。

OLA

請參閱操作層級協議。

線上遷移

一種遷移方法，無需離線即可將來源工作負載複製到目標系統。連接至工作負載的應用程式可在遷移期間繼續運作。此方法涉及零至最短停機時間，通常用於關鍵的生產工作負載。

OPC-UA

請參閱開啟程序通訊 - 統一架構。

開放程序通訊 - 統一架構 (OPC-UA)

用於工業自動化machine-to-machine(M2M) 通訊協定。OPC-UA 提供資料加密、身分驗證和授權機制的互通性標準。

操作水準協議 (OLA)

一份協議，闡明 IT 職能群組承諾向彼此提供的內容，以支援服務水準協議 (SLA)。

操作整備審查 (ORR)

問題及相關最佳實務的檢查清單，可協助您了解、評估、預防或減少事件和可能失敗的範圍。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的操作準備度審查 (ORR)。

操作技術 (OT)

使用實體環境控制工業操作、設備和基礎設施的硬體和軟體系統。在製造業中，整合 OT 和資訊技術 (IT) 系統是工業 4.0 轉型的關鍵重點。

操作整合 (OI)

在雲端中將操作現代化的程序，其中包括準備程度規劃、自動化和整合。如需詳細資訊，請參閱操作整合指南。

組織追蹤

由 建立的線索 AWS CloudTrail 會記錄 AWS 帳戶 組織中所有 的所有事件 AWS Organizations。在屬於組織的每個 AWS 帳戶 中建立此追蹤，它會跟蹤每個帳戶中的活動。如需詳細資訊，請參閱 CloudTrail 文件中的建立組織追蹤。

組織變更管理 (OCM)

用於從人員、文化和領導力層面管理重大、顛覆性業務轉型的架構。OCM 透過加速變更採用、解決過渡問題，以及推動文化和組織變更，協助組織為新系統和策略做好準備，並轉移至新系統和策略。在 AWS 遷移策略中，此架構稱為人員加速，因為雲端採用專案所需的變更速度。如需詳細資訊，請參閱 OCM 指南。

原始存取控制 (OAC)

CloudFront 中的增強型選項，用於限制存取以保護 Amazon Simple Storage Service (Amazon S3) 內容。OAC 支援所有 S3 儲存貯體中的所有伺服器端加密 AWS KMS (SSE-KMS) AWS 區域，以及對 S3 儲存貯體的動態PUT和DELETE請求。

原始存取身分 (OAI)

CloudFront 中的一個選項，用於限制存取以保護 Amazon S3 內容。當您使用 OAI 時，CloudFront 會建立一個可供 Amazon S3 進行驗證的主體。經驗證的主體只能透過特定 CloudFront 分發來存取 S3 儲存貯體中的內容。另請參閱 OAC，它可提供更精細且增強的存取控制。

ORR

請參閱操作整備審核。

OT

請參閱操作技術。

傳出 (輸出) VPC

在 AWS 多帳戶架構中，處理從應用程式內啟動之網路連線的 VPC。AWS 安全參考架構建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

P

許可界限

附接至 IAM 主體的 IAM 管理政策，可設定使用者或角色擁有的最大許可。如需詳細資訊，請參閱 IAM 文件中的許可界限。

個人身分識別資訊 (PII)

直接檢視或與其他相關資料配對時，可用來合理推斷個人身分的資訊。PII 的範例包括名稱、地址和聯絡資訊。

PII

請參閱個人身分識別資訊。

手冊

一組預先定義的步驟，可擷取與遷移關聯的工作，例如在雲端中提供核心操作功能。手冊可以採用指令碼、自動化執行手冊或操作現代化環境所需的程序或步驟摘要的形式。

PLC

請參閱可程式設計邏輯控制器。

PLM

請參閱產品生命週期管理。

政策

可定義許可的物件（請參閱身分型政策）、指定存取條件（請參閱資源型政策），或定義組織中所有帳戶的最大許可 AWS Organizations（請參閱服務控制政策）。

混合持久性

根據資料存取模式和其他需求，獨立選擇微服務的資料儲存技術。如果您的微服務具有相同的資料儲存技術，則其可能會遇到實作挑戰或效能不佳。如果微服務使用最適合其需求的資料儲存，則可以更輕鬆地實作並達到更好的效能和可擴展性。如需詳細資訊，請參閱在微服務中啟用資料持久性。

組合評定

探索、分析應用程式組合並排定其優先順序以規劃遷移的程序。如需詳細資訊，請參閱評估遷移準備程度。

述詞

傳回 `true`或 的查詢條件`false`，通常位於 `WHERE`子句中。

述詞下推

一種資料庫查詢最佳化技術，可在傳輸前篩選查詢中的資料。這可減少必須從關聯式資料庫擷取和處理的資料量，並改善查詢效能。

預防性控制

旨在防止事件發生的安全控制。這些控制是第一道防線，可協助防止對網路的未經授權存取或不必要變更。如需詳細資訊，請參閱在 AWS上實作安全控制中的預防性控制。

委託人

中可執行動作和存取資源 AWS 的實體。此實體通常是 AWS 帳戶、IAM 角色或使用者的根使用者。如需詳細資訊，請參閱 IAM 文件中角色術語和概念中的主體。

設計隱私權

透過整個開發程序將隱私權納入考量的系統工程方法。

私有託管區域

一種容器，它包含有關您希望 Amazon Route 53 如何回應一個或多個 VPC 內的域及其子域之 DNS 查詢的資訊。如需詳細資訊，請參閱 Route 53 文件中的使用私有託管區域。

主動控制

旨在防止部署不合規資源的安全控制。這些控制項會在佈建資源之前對其進行掃描。如果資源不符合控制項，則不會佈建。如需詳細資訊，請參閱 AWS Control Tower 文件中的控制項參考指南，並參閱實作安全控制項中的主動控制項。 AWS

產品生命週期管理 (PLM)

管理產品整個生命週期的資料和程序，從設計、開發和啟動，到成長和成熟，再到拒絕和移除。

生產環境

請參閱 環境。

可程式邏輯控制器 (PLC)

在製造中，高度可靠、可調整的電腦，可監控機器並自動化製造程序。

提示鏈結

使用一個 LLM 提示的輸出做為下一個提示的輸入，以產生更好的回應。此技術用於將複雜任務分解為子任務，或反覆精簡或展開初步回應。它有助於提高模型回應的準確性和相關性，並允許更精細、個人化的結果。

擬匿名化

將資料集中的個人識別符取代為預留位置值的程序。假名化有助於保護個人隱私權。假名化資料仍被視為個人資料。

發佈/訂閱 (pub/sub)

一種模式，可啟用微服務之間的非同步通訊，以提高可擴展性和回應能力。例如，在微服務型 MES 中，微服務可以將事件訊息發佈到其他微服務可訂閱的頻道。系統可以新增新的微服務，而無需變更發佈服務。

Q

查詢計劃

一系列步驟，如指示，用於存取 SQL 關聯式資料庫系統中的資料。

查詢計劃迴歸

在資料庫服務優化工具選擇的計畫比對資料庫環境進行指定的變更之前的計畫不太理想時。這可能因為對統計資料、限制條件、環境設定、查詢參數繫結的變更以及資料庫引擎的更新所導致。

R

RACI 矩陣

請參閱負責、負責、諮詢、告知 (RACI)。

RAG

請參閱擷取增強生成。

勒索軟體

一種惡意軟體，旨在阻止對計算機系統或資料的存取，直到付款為止。

RASCI 矩陣

請參閱負責、負責、諮詢、告知 (RACI)。

RCAC

請參閱資料列和資料欄存取控制。

僅供讀取複本

用於唯讀用途的資料庫複本。您可以將查詢路由至僅供讀取複本以減少主資料庫的負載。

重新架構師

請參閱 7 Rs。

復原點目標 (RPO)

自上次資料復原點以來可接受的時間上限。這會決定最後一個復原點與服務中斷之間可接受的資料遺失。

復原時間目標 (RTO)

服務中斷和服務還原之間的可接受延遲上限。

重構

請參閱 7 個 R。

區域

地理區域中的 AWS 資源集合。每個 AWS 區域 都獨立於其他 ，以提供容錯能力、穩定性和彈性。如需詳細資訊，請參閱指定 AWS 區域 您的帳戶可以使用哪些。

迴歸

預測數值的 ML 技術。例如，為了解決「這房子會賣什麼價格？」的問題 ML 模型可以使用線性迴歸模型，根據已知的房屋事實 (例如，平方英尺) 來預測房屋的銷售價格。

重新託管

請參閱 7 個 R。

版本

在部署程序中，它是將變更提升至生產環境的動作。

重新放置

請參閱 7 Rs。

Replatform

請參閱 7 Rs。

回購

請參閱 7 Rs。

彈性

應用程式抵禦中斷或從中斷中復原的能力。在 中規劃彈性時，高可用性和災難復原是常見的考量 AWS 雲端。如需詳細資訊，請參閱AWS 雲端 彈性。

資源型政策

附接至資源的政策，例如 Amazon S3 儲存貯體、端點或加密金鑰。這種類型的政策會指定允許存取哪些主體、支援的動作以及必須滿足的任何其他條件。

負責者、當責者、事先諮詢者和事後告知者 (RACI) 矩陣

定義所有涉及遷移活動和雲端操作之各方的角色和責任的矩陣。矩陣名稱衍生自矩陣中定義的責任類型：負責人 (R)、責任 (A)、已諮詢 (C) 和知情 (I)。支援 (S) 類型為選用。如果您包含支援，則矩陣稱為 RASCI 矩陣，如果您排除它，則稱為 RACI 矩陣。

回應性控制

一種安全控制，旨在驅動不良事件或偏離安全基準的補救措施。如需詳細資訊，請參閱在 AWS上實作安全控制中的回應性控制。

保留

請參閱 7 Rs。

淘汰

請參閱 7 個 R。

檢索增強生成 (RAG)

一種生成式 AI 技術，其中 LLM 會在產生回應之前參考訓練資料來源以外的授權資料來源。例如，RAG 模型可能會對組織的知識庫或自訂資料執行語意搜尋。如需詳細資訊，請參閱什麼是 RAG。

輪換

定期更新秘密的程序，讓攻擊者更難存取登入資料。

資料列和資料欄存取控制 (RCAC)

使用已定義存取規則的基本彈性 SQL 表達式。RCAC 包含資料列許可和資料欄遮罩。

RPO

請參閱復原點目標。

RTO

請參閱復原時間目標。

執行手冊

執行特定任務所需的一組手動或自動程序。這些通常是為了簡化重複性操作或錯誤率較高的程序而建置。

S

SAML 2.0

許多身分提供者 (IdP) 使用的開放標準。此功能可啟用聯合單一登入 (SSO)，讓使用者可以登入 AWS Management Console 或呼叫 AWS API 操作，而無需為您組織中的每個人在 IAM 中建立使用者。如需有關以 SAML 2.0 為基礎的聯合詳細資訊，請參閱 IAM 文件中的關於以 SAML 2.0 為基礎的聯合。

SCADA

請參閱監督控制和資料擷取。

SCP

請參閱服務控制政策。

秘密

您以加密形式存放的 AWS Secrets Manager機密或限制資訊，例如密碼或使用者登入資料。它由秘密值及其中繼資料組成。秘密值可以是二進位、單一字串或多個字串。如需詳細資訊，請參閱 Secrets Manager 文件中的 Secrets Manager 秘密中的什麼內容？。

依設計的安全性

透過整個開發程序將安全性納入考量的系統工程方法。

安全控制

一種技術或管理防護機制，它可預防、偵測或降低威脅行為者利用安全漏洞的能力。安全控制有四種主要類型：預防性、偵測性、回應性和主動性。

安全強化

減少受攻擊面以使其更能抵抗攻擊的過程。這可能包括一些動作，例如移除不再需要的資源、實作授予最低權限的安全最佳實務、或停用組態檔案中不必要的功能。

安全資訊與事件管理 (SIEM) 系統

結合安全資訊管理 (SIM) 和安全事件管理 (SEM) 系統的工具與服務。SIEM 系統會收集、監控和分析來自伺服器、網路、裝置和其他來源的資料，以偵測威脅和安全漏洞，並產生提醒。

安全回應自動化

預先定義和程式設計的動作，旨在自動回應或修復安全事件。這些自動化可做為偵測或回應式安全控制，協助您實作 AWS 安全最佳實務。自動化回應動作的範例包括修改 VPC 安全群組、修補 Amazon EC2 執行個體或輪換登入資料。

伺服器端加密

由 AWS 服務 接收資料的 在其目的地加密資料。

服務控制政策 (SCP)

為 AWS Organizations中的組織的所有帳戶提供集中控制許可的政策。SCP 會定義防護機制或設定管理員可委派給使用者或角色的動作限制。您可以使用 SCP 作為允許清單或拒絕清單，以指定允許或禁止哪些服務或動作。如需詳細資訊，請參閱 AWS Organizations 文件中的服務控制政策。

服務端點

的進入點 URL AWS 服務。您可以使用端點，透過程式設計方式連接至目標服務。如需詳細資訊，請參閱 AWS 一般參考 中的 AWS 服務 端點。

服務水準協議 (SLA)

一份協議，闡明 IT 團隊承諾向客戶提供的服務，例如服務正常執行時間和效能。

服務層級指標 (SLI)

服務效能方面的測量，例如其錯誤率、可用性或輸送量。

服務層級目標 (SLO)

代表服務運作狀態的目標指標，由服務層級指標測量。

共同責任模式

描述您與 共同 AWS 承擔雲端安全與合規責任的模型。 AWS 負責雲端的安全，而 負責雲端的安全。如需詳細資訊，請參閱共同責任模式。

SIEM

請參閱安全資訊和事件管理系統。

單點故障 (SPOF)

應用程式的單一關鍵元件故障，可能會中斷系統。

SLA

請參閱服務層級協議。

SLI

請參閱服務層級指標。

SLO

請參閱服務層級目標。

先拆分後播種模型

擴展和加速現代化專案的模式。定義新功能和產品版本時，核心團隊會進行拆分以建立新的產品團隊。這有助於擴展組織的能力和服務，提高開發人員生產力，並支援快速創新。如需詳細資訊，請參閱 中的階段式應用程式現代化方法 AWS 雲端。

SPOF

請參閱單一故障點。

星狀結構描述

使用一個大型事實資料表來存放交易或測量資料的資料庫組織結構，並使用一或多個較小的維度資料表來存放資料屬性。此結構旨在用於資料倉儲或商業智慧用途。

Strangler Fig 模式

一種現代化單一系統的方法，它會逐步重寫和取代系統功能，直到舊式系統停止使用為止。此模式源自無花果藤，它長成一棵馴化樹並最終戰勝且取代了其宿主。該模式由 Martin Fowler 引入，作為重寫單一系統時管理風險的方式。如需有關如何套用此模式的範例，請參閱使用容器和 Amazon API Gateway 逐步現代化舊版 Microsoft ASP.NET (ASMX) Web 服務。

子網

您 VPC 中的 IP 地址範圍。子網必須位於單一可用區域。

監控控制和資料擷取 (SCADA)

在製造中，使用硬體和軟體來監控實體資產和生產操作的系統。

對稱加密

使用相同金鑰來加密及解密資料的加密演算法。

合成測試

以模擬使用者互動的方式測試系統，以偵測潛在問題或監控效能。您可以使用 Amazon CloudWatch Synthetics 來建立這些測試。

系統提示

一種向 LLM 提供內容、指示或指導方針以指示其行為的技術。系統提示有助於設定內容，並建立與使用者互動的規則。

T

標籤

做為中繼資料以組織 AWS 資源的鍵值對。標籤可協助您管理、識別、組織、搜尋及篩選資源。如需詳細資訊，請參閱標記您的 AWS 資源。

目標變數

您嘗試在受監督的 ML 中預測的值。這也被稱為結果變數。例如，在製造設定中，目標變數可能是產品瑕疵。

任務清單

用於透過執行手冊追蹤進度的工具。任務清單包含執行手冊的概觀以及要完成的一般任務清單。對於每個一般任務，它包括所需的預估時間量、擁有者和進度。

測試環境

請參閱 環境。

訓練

為 ML 模型提供資料以供學習。訓練資料必須包含正確答案。學習演算法會在訓練資料中尋找將輸入資料屬性映射至目標的模式 (您想要預測的答案)。它會輸出擷取這些模式的 ML 模型。可以使用 ML 模型，來預測您不知道的目標新資料。

傳輸閘道

可以用於互連 VPC 和內部部署網路的網路傳輸中樞。如需詳細資訊，請參閱 AWS Transit Gateway 文件中的什麼是傳輸閘道。

主幹型工作流程

這是一種方法，開發人員可在功能分支中本地建置和測試功能，然後將這些變更合併到主要分支中。然後，主要分支會依序建置到開發環境、生產前環境和生產環境中。

受信任的存取權

將許可授予您指定的服務，以代表您在組織中 AWS Organizations 及其帳戶中執行任務。受信任的服務會在需要該角色時，在每個帳戶中建立服務連結角色，以便為您執行管理工作。如需詳細資訊，請參閱 文件中的 AWS Organizations 搭配使用 AWS Organizations 與其他 AWS 服務。

調校

變更訓練程序的各個層面，以提高 ML 模型的準確性。例如，可以透過產生標籤集、新增標籤、然後在不同的設定下多次重複這些步驟來訓練 ML 模型，以優化模型。

雙比薩團隊

兩個比薩就能吃飽的小型 DevOps 團隊。雙披薩團隊規模可確保軟體開發中的最佳協作。

U

不確定性

這是一個概念，指的是不精確、不完整或未知的資訊，其可能會破壞預測性 ML 模型的可靠性。有兩種類型的不確定性：認知不確定性是由有限的、不完整的資料引起的，而隨機不確定性是由資料中固有的噪聲和隨機性引起的。如需詳細資訊，請參閱量化深度學習系統的不確定性指南。

未區分的任務

也稱為繁重工作，是建立和操作應用程式的必要工作，但不為最終使用者提供直接價值或提供競爭優勢。未區分任務的範例包括採購、維護和容量規劃。

較高的環境

請參閱 環境。

V

清空

一種資料庫維護操作，涉及增量更新後的清理工作，以回收儲存並提升效能。

版本控制

追蹤變更的程序和工具，例如儲存庫中原始程式碼的變更。

VPC 對等互連

兩個 VPC 之間的連線，可讓您使用私有 IP 地址路由流量。如需詳細資訊，請參閱 Amazon VPC 文件中的什麼是 VPC 對等互連。

漏洞

危及系統安全性的軟體或硬體瑕疵。

W

暖快取

包含經常存取的目前相關資料的緩衝快取。資料庫執行個體可以從緩衝快取讀取，這比從主記憶體或磁碟讀取更快。

暖資料

不常存取的資料。查詢這類資料時，通常可接受中等緩慢的查詢。

視窗函數

SQL 函數，對與目前記錄在某種程度上相關的資料列群組執行計算。視窗函數適用於處理任務，例如根據目前資料列的相對位置計算移動平均值或存取資料列的值。

工作負載

提供商業價值的資源和程式碼集合，例如面向客戶的應用程式或後端流程。

工作串流

遷移專案中負責一組特定任務的功能群組。每個工作串流都是獨立的，但支援專案中的其他工作串流。例如，組合工作串流負責排定應用程式、波次規劃和收集遷移中繼資料的優先順序。組合工作串流將這些資產交付至遷移工作串流，然後再遷移伺服器和應用程式。

WORM

請參閱寫入一次，讀取許多。

WQF

請參閱AWS 工作負載資格架構。

寫入一次，讀取許多 (WORM)

儲存模型，可一次性寫入資料，並防止資料遭到刪除或修改。授權使用者可以視需要多次讀取資料，但無法變更資料。此資料儲存基礎設施被視為不可變。

Z

零時差入侵

利用零時差漏洞的攻擊，通常是惡意軟體。

零時差漏洞

生產系統中未緩解的瑕疵或漏洞。威脅行為者可以使用這種類型的漏洞來攻擊系統。開發人員經常因為攻擊而意識到漏洞。

零鏡頭提示

提供 LLM 執行任務的指示，但沒有可協助引導任務的範例 (快照)。LLM 必須使用其預先訓練的知識來處理任務。零鏡頭提示的有效性取決於任務的複雜性和提示的品質。另請參閱少量擷取提示。

殭屍應用程式

CPU 和記憶體平均使用率低於 5% 的應用程式。在遷移專案中，通常會淘汰這些應用程式。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。