AWS 白皮書

在 AWS 上實作持續整合與持續交付

在 AWS 上實作持續整合與持續交付: AWS 白皮書

Table of Contents

摘要 1
摘要 1
軟體交付的挑戰 2
什麼是持續整合和持續交付/部署？ 3
持續整合 3
持續交付和部署 3
持續交付不是持續部署 3
持續交付的優點 5
自動化軟體發行程序 5
改善開發人員生產力 5
改善程式碼品質 5
更快速地交付更新 5
實作持續整合與持續交付 6
持續整合/持續交付的路徑 6
持續整合 7
持續交付：建立預備環境 7
持續交付：建立生產環境 8
持續部署 8
成熟及超越 9
團隊 9
應用程式團隊 10
基礎設施團隊 10
工具團隊 10
持續整合和持續交付的測試階段 10
設定原始檔 11
設定及執行建置 12
建置 12
預備 12
生產 13
建置管道 13
從可進行持續整合的最基本可行之管道開始 13
持續交付管道 19
新增 Lambda 動作 19
手動核准 20

在 CI/CD 管道中部署基礎設施程式碼變更 21
無伺服器應用程式的 CI/CD 21
適用於多個團隊、分支和 AWS 區域的管道 21
與 AWS CodeBuild 的管道整合 21
與 Jenkins 的管道整合 22
部署方法 24
一次全部 (就地部署) 25
滾動部署 25
不變式與藍/綠法部署 25
資料庫結構描述變更 27
最佳實務摘要 28
結論 30
深入閱讀 31
作者群 32
文件修訂 33
聲明 34

在 AWS 上實作持續整合與持續交付

出版日期：2021 年 10 月 27 日 (文件修訂)

摘要

本文說明在軟體開發環境中，使用持續整合和持續交付 (CI/CD) 及 Amazon Web Services (AWS) 工具的功能和優點。持續整合和持續交付不僅是最佳實務，也是 DevOps 計劃的重要部分。

軟體交付的挑戰

如今，企業面臨著快速變化的競爭環境、不斷演進的安全需求，以及效能可擴展性所帶來的挑戰。企業必須縮小營運穩定性和快速開發功能間的差距。持續整合和持續交付 (CI/CD) 是相當實務的做法，可讓您進行快速的軟體變更，同時維持系統穩定性和安全。

Amazon 很早就意識到為 Amazon.com 零售客戶、Amazon 子公司和 Amazon Web Services (AWS) 提供功能業務需求，將需要全新且創新的軟體交付方式。在像 Amazon 這樣規模的公司裡，成千上萬的獨立軟體團隊必須能夠平行運作，以快速、安全、可靠地交付軟體，且對於任何中斷零容許。

透過學習如何快速地交付軟體，Amazon 和其他具前瞻思維的組織率先開啟了 DevOps。DevOps 是文化哲學、實務和工具的組合，可提升組織快速交付應用程式及服務的能力。組織使用 DevOps 原則，可以比使用傳統軟體開發和基礎設施管理程序的組織，更快速地演進及改善產品。這樣的速度讓組織可以為客戶提供更佳的服務，並在市場上更有效率地競爭。

其中一些原則，例如雙披薩團隊和微型服務/服務導向架構 (SOA) 不在本白皮書探討的範圍內。本白皮書探討 Amazon 建置及持續改善的 CI/CD 功能。CI/CD 是快速且可靠地交付軟體功能的關鍵。

AWS 現在以一組開發人員服務的形式提供這些 CI/CD 功能：AWS CodeStar、AWS CodeCommit、AWS CodePipeline、AWS CodeBuild、AWS CodeDeploy 及 AWS CodeArtifact。從事 DevOps 的開發人員和 IT 營運專業人員可以使用這些服務，快速、安全地交付軟體。這些服務結合在一起，可協助您安全地存放，並將版本控制套用到您應用程式的原始檔。您可以使用 AWS CodeStar，快速地協調使用這些服務的端對端軟體發行工作流程。針對現有環境，AWS CodePipeline 具備了靈活性，可使用您現有的工具獨立整合每項服務。這些具備高可用性且可輕鬆整合的服務，和任何其他的 AWS 服務相同，可透過 AWS 管理主控台、AWS 應用程式的程式設計界面 (API) 和 AWS 軟體開發工具組 (SDK)，進行存取。

什麼是持續整合和持續交付/部署？

本節探討持續整合和持續交付的實務做法，並說明持續交付和持續部署之間的區別。

持續整合

持續整合 (CI) 是一種軟體開發實務，開發人員會定期將其程式碼變更合併到一個中央儲存庫中，然後執行自動化建置和測試。CI 最常指的是軟體發行程序的建置或整合階段，並且同時需要自動化元件 (例如 CI 或建置服務) 和文化元件 (例如經常學習整合)。CI 的關鍵目標是更快發現和解決錯誤、改善軟體品質，以及減少驗證和發行新軟體更新所需的時間。

持續整合側重於整合較少量的遞交和較少量的程式碼變更。開發人員需定期遞交程式碼，至少每天一次。開發人員會從程式碼儲存庫中提取程式碼，確保在推送到組建伺服器之前先合併本地主機上的程式碼。在這個階段，組建伺服器會執行各種測試，並接受或拒絕程式碼遞交。

實作 CI 的基本挑戰，包括更頻繁地遞交至通用程式碼基底、維護單一原始檔儲存庫、自動化建置和自動化測試。其他挑戰包括在與生產相似的環境中進行測試，讓團隊能看到程序，並讓開發人員可輕鬆取得任何版本的應用程式。

持續交付和部署

持續交付 (CD) 是一項軟體開發實務，在此實務中會自動建置、測試和準備程式碼變更，以進行生產發行。持續交付會在建置階段完成之後，將所有程式碼變更部署到測試環境、生產環境或同時部署到兩者，藉此擴張持續整合。持續交付可以透過工作流程程序完全自動化，也可以在關鍵點透過手動步驟實現部分自動化。在適當地實作持續交付時，開發人員一律都會有已通過標準化測試程序，且準備好部署的建置成品。

透過持續部署，修訂版本會自動部署到生產環境，而不需要開發人員的明確核准，如此可能讓整個軟體的發行程序自動化。這反過來又可以在產品生命週期的早期提供持續的客戶回饋迴圈。

持續交付不是持續部署

關於持續交付的其中一個誤解是，這表示所遞交的每一項變更，都會在通過自動化測試後立即套用到生產環境中。但是，持續交付的重點不是立即將每一項變更都套用到生產，而是確保每一項更改都已準備好可進入生產環境。

在將變更部署到生產環境之前，可以實作一個決策程序，確保已授權該生產部署，且會經過稽核。這項決定可以由一個人決定，然後由工具執行。

使用持續交付，上線的決策會成為一項業務決策，而非技術決策。每次遞交都會進行技術驗證。

將變更推出到生產環境不是一項干擾性事件。部署不需要技術團隊停止進行下一組變更，且不需要專案計劃、交接文件，或是維護時段。部署會成為一項可重複執行的程序，在測試環境中多次執行及進行驗證。

持續交付的優點

CD 可為您的軟體開發團隊提供許多優點，包括自動化程序、改善開發人員的生產力、改善程式碼品質，以及更快速地為您的客戶提供更新。

自動化軟體發行程序

CD 能為您的團隊提供一種方法，可簽入自動建置、測試及備妥可發行到生產環境的程式碼，使您的軟體交付更有效率、更具彈性、更快速且更安全。

改善開發人員生產力

CD 實務可讓開發人員不用進行手動任務，解決複雜的相依性，並將開發人員的重點轉回到在軟體中交付新功能，以改善您團隊的生產力。開發人員可以專注在提供所需功能的程式碼邏輯，而不是將其程式碼與業務的其他部分整合在一起，然後花費時間在如何將程式碼部署到平台。

改善程式碼品質

CD 有助於在交付的程序中提早發現和解決錯誤，避免這些錯誤演變成更大的問題。由於整個程序都已自動化，您的團隊可以輕鬆執行其他類型的程式碼測試。隨著更頻繁進行更多測試的紀律進行，團隊可以更快進行反覆工作，並且可以立即了解變更所造成之影響的回饋。如此可讓團隊在穩定性和安全保證度高的情況下，推動品質更佳的程式碼。開發人員將可透過立即性回饋，了解新的程式碼運作是否正常，以及其是否引進了任何重大變更或錯誤。在開發程序早期發現的錯誤，是最容易修正的錯誤。

更快速地交付更新

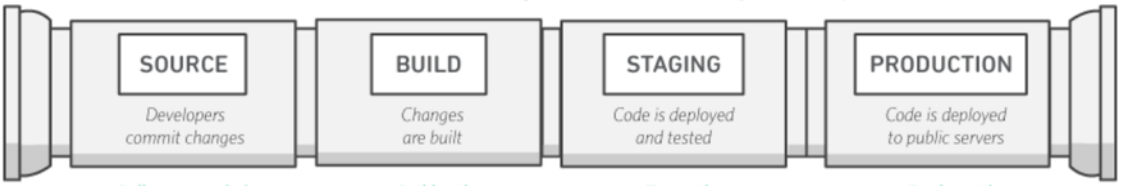
CD 可協助您的團隊快速且頻繁地為客戶提供更新。實作 CI/CD 時，整個團隊的速度 (包括功能發行和錯誤修正) 都會加快。企業可以更快速地因應市場變化、安全挑戰、客戶需求及成本壓力。例如，如果需要新的安全功能，您的團隊可以搭配自動化測試來實作 CI/CD，快速且可靠地將修正引進生產系統，完全不用擔心。過去需要數週和數個月的任務，現在可以在幾天甚至幾個小時內完成。

實作持續整合與持續交付

本節討論您可在組織中採用，以開始實作 CI/CD 模型的方法。本白皮書不討論具備成熟 DevOps 和雲端轉型模型的組織，要如何建置及使用 CI/CD 管道。為協助您的 DevOps 之旅，AWS 擁有為數眾多經認證的 DevOps 合作夥伴，可提供資源和工具。如需準備移往 AWS 雲端的詳細資訊，請參閱建置雲端運作模型。

持續整合/持續交付的路徑

CI/CD 可描繪成一個管道 (請參閱下圖)，其中，新的程式碼會在一端提交，經過一系列階段 (原始檔、建置、預備及生產) 的測試，然後以生產就緒程式碼的形式發佈。若您的組織初次使用 CI/CD，可以透過反覆操作的方式了解此管道。這表示您應從較小的範圍開始，在每個階段反覆進行操作，使您可以了解並透過有助於組織成長的方式來開發程式碼。



CI/CD 管道

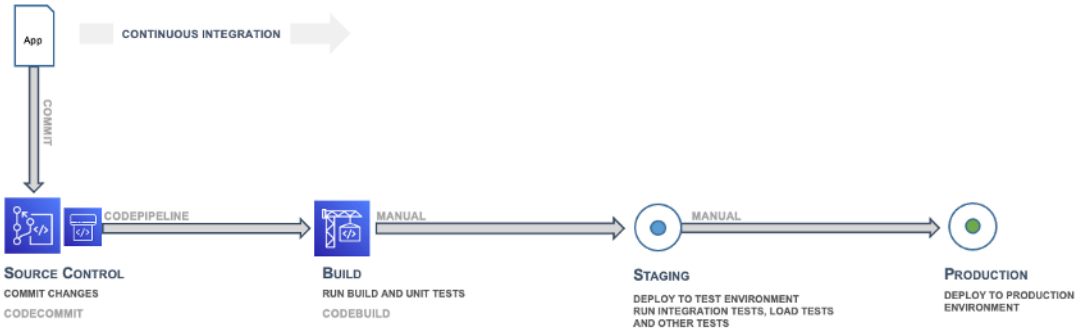
CI/CD 管道的每個階段，都會在交付程序中以邏輯單位的形式結構化。此外，每個階段都可做為一個閘門，檢查程式碼的特定面向。隨著程式碼通過管道，我們會假設程式碼的品質在後期階段較高，因為程式碼的更多面向將持續獲得驗證。早期階段發現的問題會阻擋程式碼通過管道。測試結果會立即傳送給團隊，而且如果軟體未通過該階段，所有進一步的建置和發行都會停止。

這些階段皆屬建議。您可以根據業務需求調整階段。某些階段可以針對多種類型的測試、安全和性能而重複進行。根據專案複雜性和您團隊結構之不同，有些階段在不同層級可能會重複進行數次。例如，一個團隊的最終產品可能會成為下一個團隊專案所需的項目。這表示第一個團隊的最終產品之後會做為下一個團隊專案中的成品。

CI/CD 管道的存在對您組織功能的成熟，會有重大的影響。該組織應先從較小的步驟開始，而不是一開始就試圖建置一個包含多重環境、許多測試階段，同時所有階段均包含自動化的完全成熟之管道。請記住，即使是擁有高成熟度 CI/CD 環境的組織，也仍然需要不斷改善其管道。

建置具備 CI/CD 功能的組織是一個旅程，一路上有許多目標。下一節會討論您組織能採用的可能路徑，從持續整合開始並經歷各種持續交付層級。

持續整合



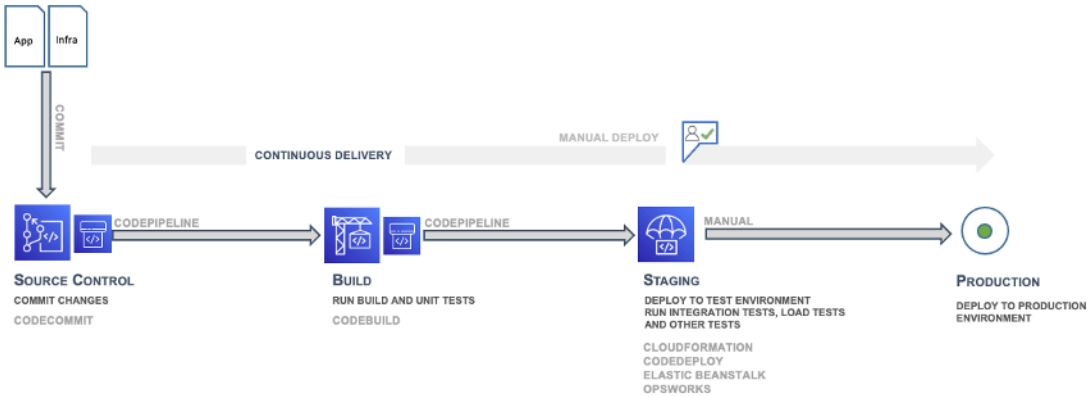
持續整合 – 原始檔和建置

CI/CD 旅程的第一階段，是在持續整合方面發展成熟。您應確保所有開發人員都會定期將其程式碼遞交到中央儲存庫 (例如託管在 CodeCommit 或 GitHub 上的儲存庫)，並會將所有變更合併到應用程式的發行分支。任何開發人員都不應將程式碼變為孤立狀態。如果需要一段時間的功能分支，應透過盡可能經常與上游分支合併的方式，使其保持在最新狀態。建議且鼓勵在過程中，團隊應頻繁遞交以及與完成的工作單位合併，遵循紀律進行。盡早且經常合併程式碼的開發人員，在過程中遇到的整合問題較少。

建議您也鼓勵開發人員先盡早為其應用程式建立單元測試，並執行這些測試後，再將這些程式碼推送到中央儲存庫。在軟體開發程序中早期發現的錯誤，成本最低且最易修正。

將程式碼推送到原始檔儲存庫中的分支後，監控該分支的工作流程引擎，會傳送命令至建置器工具以建置程式碼，並在受控制的環境中執行單元測試。建置程序的大小應進行適當調整，以可處理所有活動 (包括推送及在遞交階段期間可能發生的測試)，以快速取得回饋。此階段也可能進行其他品質檢查，例如單元測試涵蓋範圍、樣式檢查，以及靜態分析。最後，建置器工具會為應用程式建立一或多個二進位組建及其他成品，例如影像、樣式表和文件。

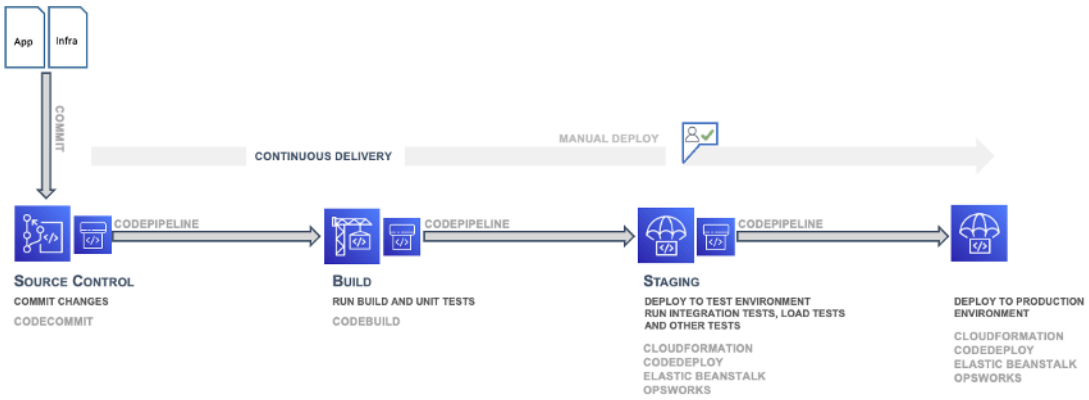
持續交付：建立預備環境



持續交付 – 預備

持續交付 (CD) 是下一個階段，涵蓋了在預備環境中部署應用程式的程式碼，這個環境是生產堆疊的複本，並會執行更多功能測試。預備環境可以是預先為測試製作的靜態環境，或者您可以佈建和設定具備專用基礎設施和組態程式碼的動態環境，以進行測試及部署應用程式程式碼。

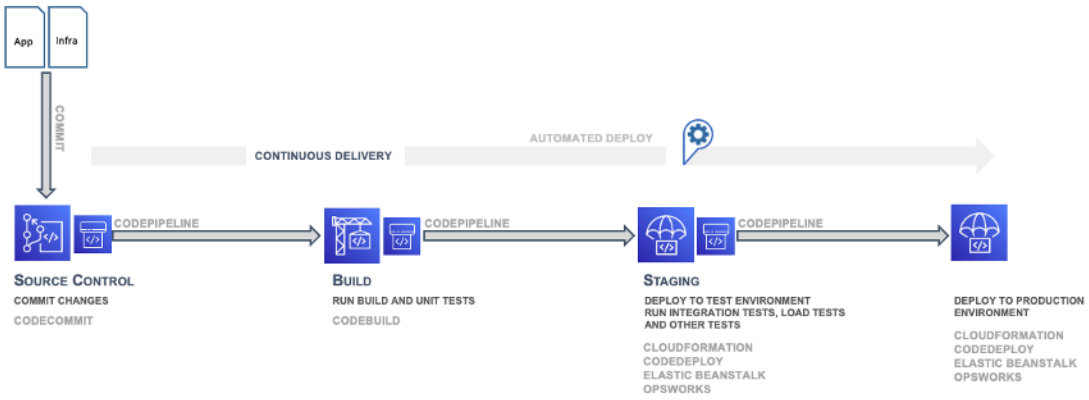
持續交付：建立生產環境



持續交付 – 生產

部署/交付管道序列中，在預備環境之後便是生產環境，而此環境也是使用「基礎設施即程式碼」(IaC) 建置。

持續部署



持續部署

CI/CD 部署管道的最終階段，即是持續部署，其中包括整個軟體發行程序的完整自動化 (包括部署至生產環境)。在完全成熟的 CI/CD 環境中，通往生產環境的路徑會完全自動化，讓程式碼得以在信心度高的情況下進行部署。

成熟及超越

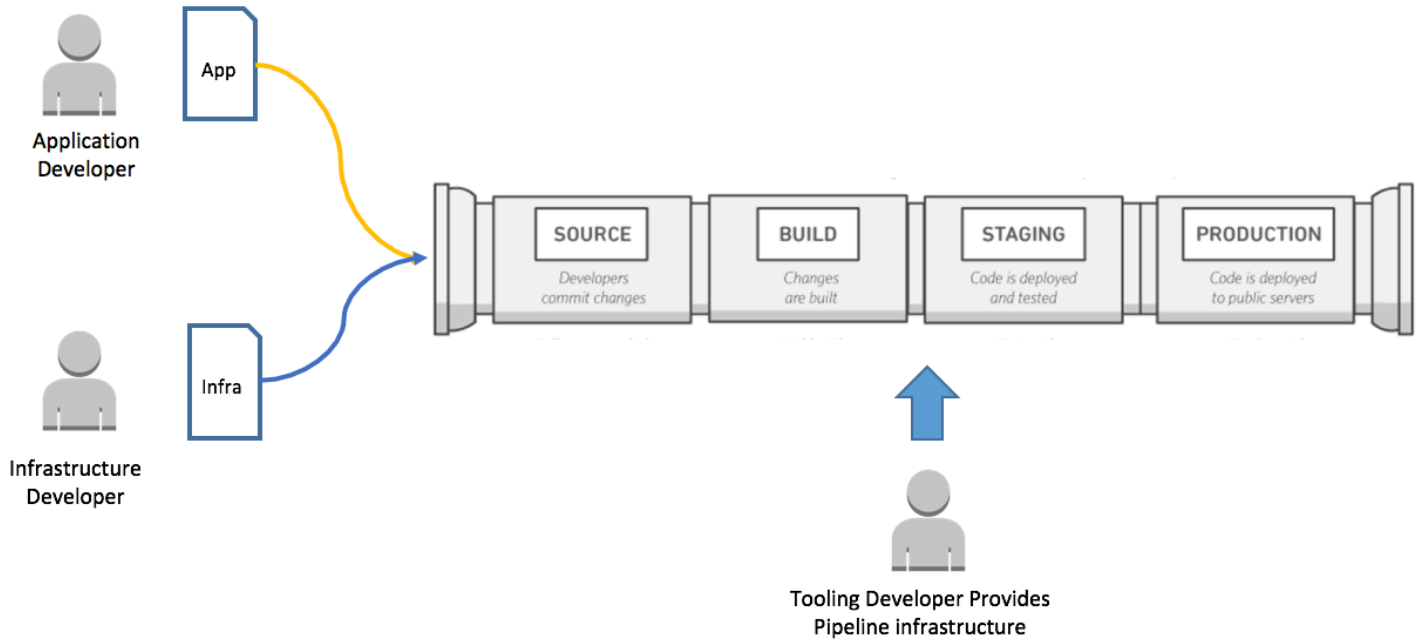
隨著組織的成熟，您的組織也將會持續發展 CI/CD 模型，使其包含更多下列改善項目：

- 更多適用於特定效能、合規、安全和使用者界面 (UI) 測試的預備環境
- 基礎設施及組態程式碼和應用程式程式碼的單元測試
- 與其他系統和程序相整合，例如程式碼檢閱、問題追蹤及事件通知。
- 與資料庫結構描述遷移相整合 (若適用)
- 適用於稽核和業務核准的其他步驟

即使是擁有複雜多重環境 CI/CD 管道的最成熟組織，也仍會持續進行改善。DevOps 是一個旅程，而不是一個目標。開發團隊的不同部門之間，會進行協同合作，持續收集針對管道的意見回饋，並在速度、規模、安全和可靠性等方面進行改善。

團隊

AWS 建議組織三個開發人員團隊來實作 CI/CD 環境：應用程式團隊、基礎設施團隊，以及工具團隊 (請參閱下圖)。此組織呈現一系列最佳實務，這些實務都是由快速成長的新創公司、大型企業組織及 Amazon 本身發展的，並已獲得採用。團隊的大小不應大於能吃完兩個披薩的團體，或大約 10 至 12 人。此遵循溝通的規則，即有意義的對話在團隊的大小增加，且溝通線呈倍數成長時，達到極限。



應用程式、基礎設施，以及工具團隊

應用程式團隊

應用程式團隊會製作應用程式。應用程式開發人員負責後端記錄、案例及單元測試，並且會根據指定的應用程式目標來開發功能。此團隊的組織目標是將開發人員花費在非核心應用程式任務上的時間，降至最低。

除了具備應用程式語言中實用的程式設計技能之外，應用程式團隊也應具備平台技能，並了解系統組態。如此可讓該團隊只需專注在開發功能及強化應用程式上。

基礎設施團隊

基礎設施團隊會撰寫程式碼，建立及設定執行應用程式所需要的基礎設施。此團隊可能會使用原生 AWS 工具 (例如 AWS CloudFormation)，或是一般工具 (例如 Chef、Puppet 或 Ansible)。基礎設施團隊負責指定需要哪些資源，且會和應用程式團隊密切合作。若為小型應用程式，基礎設施團隊可能會只由一或兩名人員組成。

該團隊應具備基礎設施佈建方法的技能，例如 AWS CloudFormation 或 HashiCorp Terraform。該團隊也應使用 Chef、Ansible、Puppet 或 Salt 等工具，開發組態自動化技能。

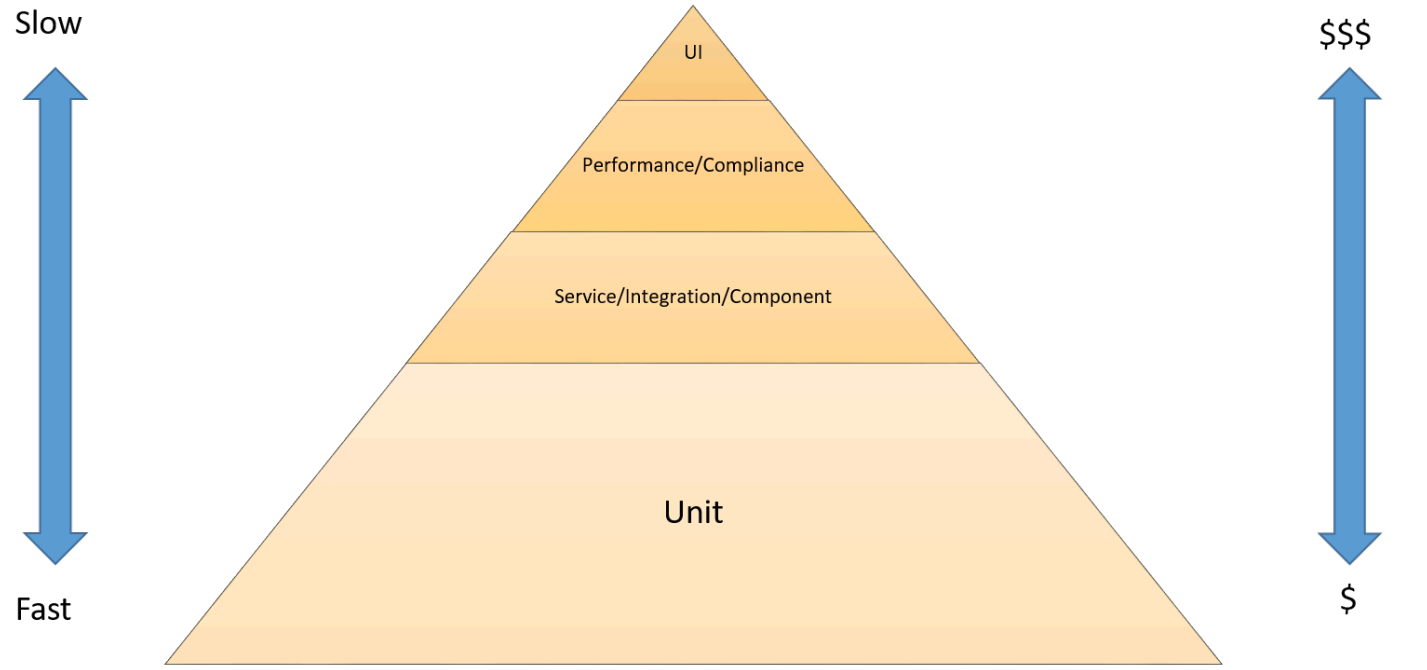
工具團隊

工具團隊會建置及管理 CI/CD 管道。此團隊負責構成管道的基礎設施和工具。此團隊不是可吃完兩個披薩團隊的一部分；此團隊要建立組織中應用程式及基礎設施團隊使用的工具。組織需要持續推動其工具團隊的發展，使工具團隊總是能領先發展中的應用程式及基礎設施團隊一步。

工具團隊必須具備建置及整合 CI/CD 管道所有部分的技能。這包括建置原始檔控制儲存庫、工作流程引擎、建置環境、測試框架，以及成品儲存庫。此團隊可以選擇實作 AWS CodeStar、AWS CodePipeline、AWS CodeCommit、AWS CodeDeploy、AWS CodeBuild，及 AWS CodeArtifact 等軟體，以及 Jenkins、GitHub、Artifactory、TeamCity 和其他類似的工具。有些組織可能會將此團隊稱為 DevOps 團隊，但 AWS 不鼓勵這種說法，而是鼓勵將 DevOps 視為軟體交付中人員、程序和工具的總和。

持續整合和持續交付的測試階段

三個 CI/CD 團隊應將測試整合到 CI/CD 管道中不同階段的軟體開發生命週期內。整體而言，測試應盡早開始。以下的測試金字塔，是由 Mike Cohn 在《敏捷帶來成功》中提供的概念。其顯示了各種軟體測試與其成本和執行速度的關係。



CI/CD 測試金字塔

單元測試位於金字塔的底部。其不僅執行速度最快，也是最便宜的。因此，單元測試應在您的測試策略中佔絕大部分。根據良好的經驗法則大約為 70%。由於此階段發現的錯誤可以快速且修正成本低，所以應為幾近整個程式碼都進行單元測試。

服務、元件和整合測試，在金字塔中位於單元測試之上。由於這些測試需要詳盡的環境，因此在基礎設施需求部分成本較高，且執行速度較慢。效能和合規測試則為下一個層級。由於其需要具備生產品質的環境，因此較為昂貴。UI 和使用者接受度測試則位於金字塔的頂端，也需要生產品質的環境。

所有這些測試都是確保軟體高品質的完整策略之一部分。但針對開發速度而言，應強調位於金字塔下半部的測試數量及涵蓋範圍。

下列各章節會討論 CI/CD 階段。

設定原始檔

在專案開始時，設定可存放原始程式碼及組態和結構描述變更的原始檔，相當重要。請在原始檔階段選擇原始檔儲存庫，例如託管在 GitHub 或 AWS CodeCommit 的儲存庫。

設定及執行建置

建置自動化對 CI 程序而言相當重要。在設定建置自動化時，首要任務就是選擇正確的建置工具。有許多建置工具可供選擇，例如：

- 適用於 Java 的 Ant、Maven 及 Gradle
- 適用於 C/C++ 的 Make
- 適用於 JavaScript 的 Grunt
- 適用於 Ruby 的 Rake

最適合您的建置工具取決於您專案的程式設計語言，以及您團隊的技能程度。選擇建置工具後，需要在建置指令碼中清楚定義所有相依性及建置步驟。對最終的建置成品進行版本控制，以易於部署及追蹤問題，也是一項最佳實務。

建置

在建置階段，建置工具會將所有對原始程式碼儲存庫進行的變更當作輸入、建置軟體，並執行下列類型的測試：

單元測試 – 測試特定程式碼區段，確保該程式碼能如預期般運作。單元測試由軟體開發人員在開發階段執行。在此階段可運用靜態程式碼分析、資料流程分析、程式碼涵蓋範圍，以及其他軟體驗證程序。

靜態程式碼分析 – 在建置及進行單元測試後，會在不實際執行應用程式的情況下執行這項測試。這項分析有助於找出編碼錯誤及安全漏洞，也能確保符合編碼準則的規定。

預備

在預備階段，會建立完整且和最終生產環境完全一樣的環境。此階段會執行下列測試：

整合測試 – 驗證元件與軟體設計之間的界面。整合測試是一項反覆的程序，有助於建置健全的界面及系統完整性。

元件測試 – 測試在不同元件之間傳遞訊息，以及結果如何。這項測試的關鍵目標可能是元件測試中的冪等。測試可包含極為龐大的資料量，或是極端情況及異常輸入。

系統測試 – 測試系統的端對端，並驗證軟體是否滿足業務需求。這可能包含測試使用者界面 (UI)、API、後端邏輯及結束狀態。

效能測試 – 判斷系統在執行特定工作負載時的回應能力及穩定性。效能測試也可用於調查、評量、驗證或確定其他系統的品質屬性，例如可擴展性、可靠性及資源使用量。效能測試的類型也可能包含負載測試、壓力測試和尖峰測試。效能測試可用於針對預先定義的條件進行基準化比對。

合規測試 – 檢查程式碼變更是否符合非功能規格及 (或) 法規的需求。這項測試會判斷您的實作符合定義的標準。

使用者接受度測試 – 驗證端對端業務流程。這項測試由最終使用者在預備環境中執行，並確認系統是否符合需求規格的需求。通常，客戶會在此階段採用 Alpha 和 Beta 測試方法。

生產

最後，在通過先前各項測試後，會在生產環境內重複預備階段。在此階段可先透過只在少部分的伺服器或甚至只在一部伺服器或一個 AWS 區域上，部署新的程式碼來進行 Canary 測試，然後再將程式碼部署到整個生產環境。部署方法一節涵蓋了有關如何安全地部署至生產環境的特定資訊。

下一節會討論建置管道以整合這些階段和測試。

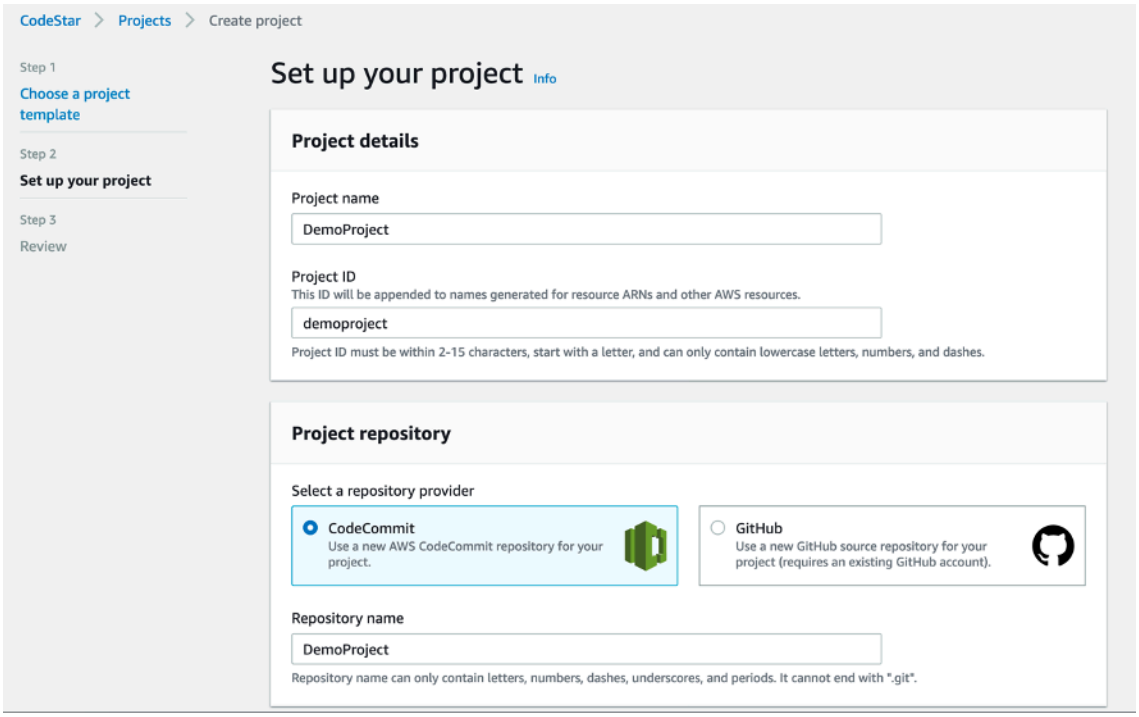
建置管道

本節探討建置管道。首先，建立一個只具備 CI 所需元件的管道，之後再轉換成具備更多元件和階段的持續交付管道。本節也會探討如何考慮針對大型專案使用 AWS Lambda 函數及手動核准，為多個團隊、分支及 AWS 區域進行規劃。

從可進行持續整合的最基本可行之管道開始

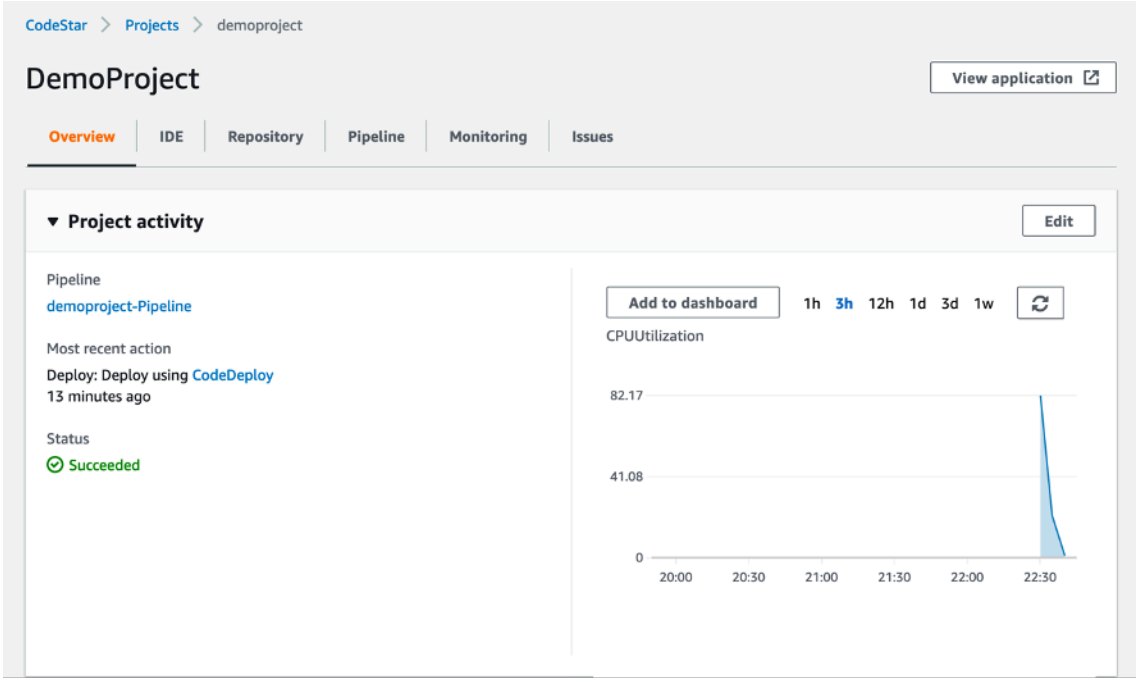
組織邁向持續交付的旅程，從最基本可行的管道 (MVP) 開始。如實作持續整合及持續交付中所探討，團隊可以從非常簡單的程序開始，例如實作執行程式碼樣式檢查或單一單元測試，而不進行部署的管道。

其中一項關鍵元件是持續交付的協同運作工具。為了協助您建置此管道，Amazon 開發了 AWS CodeStar。



AWS CodeStar 設定頁面

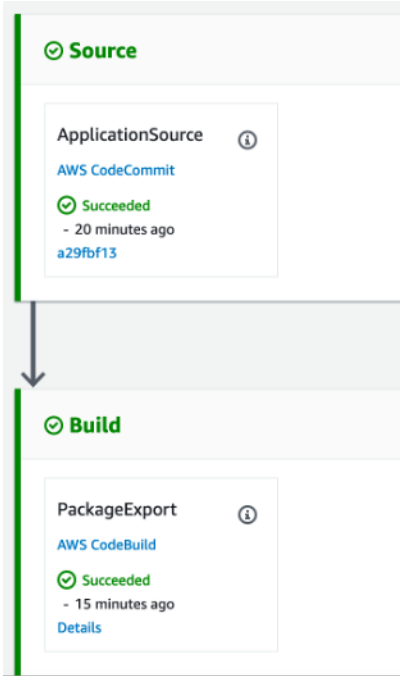
AWS CodeStar 使用 AWS CodePipeline、AWS CodeBuild、AWS CodeCommit 及 AWS CodeDeploy，並包含經整合的設定程序、工具、範本及儀表板。AWS CodeStar 提供在 AWS 上快速開發、建置和部署應用程式所需的一切。其可讓您能更快速地開始發行程式碼。已熟悉 AWS 管理主控台並尋求更高程度控制的客戶，可以手動設定其選擇的開發人員工具，並且可視需要佈建個別的 AWS 服務。



AWS CodeStar 儀表板

AWS CodePipeline 是一項 CI/CD 服務，可透過 AWS CodeStar 或 AWS 管理主控台使用，其可進行快速且可靠的應用程式及基礎設施更新。AWS CodePipeline 會在每次發生程式碼變更時，根據您定義的發行程序模型，建置、測試及部署您的程式碼。您如此即可快速且可靠地交付功能和更新。透過使用熱門的第三方服務 (像是 GitHub) 預先建置之外掛程式，或是將自己自訂的外掛程式整合至發行程序的任何階段，您可以輕鬆建置端對端解決方案。使用 AWS CodePipeline 時，只需按實際用量付費。沒有預付費用，也無需長期承諾。

AWS CodeStar 和 AWS CodePipeline 的步驟，會直接映射到原始檔、建置、預備及生產 CI/CD 階段。雖然持續交付固然重要，但您可以從簡單的兩步驟管道開始，檢查原始檔儲存庫，並執行建置動作：

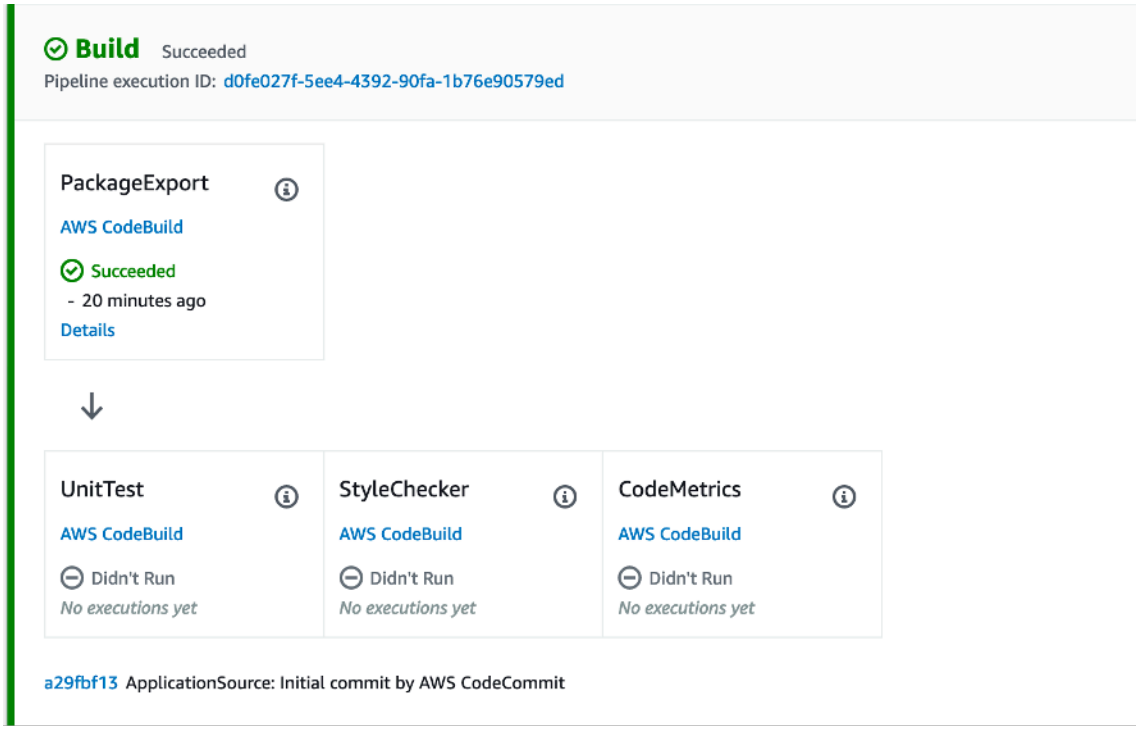


AWS CodePipeline – 原始檔及建置階段

針對 AWS CodePipeline，原始檔階段可接受來自 GitHub、AWS CodeCommit 以及 Amazon Simple Storage Service (Amazon S3) 的輸入。自動化建置程序是實作持續交付及邁向持續部署關鍵的第一個步驟。消除製作建置成品中的人為介入，可降低您團隊的負擔，將因為手動封裝而產生的錯誤降至最少，並可讓您開始更頻繁地封裝可供使用的成品。

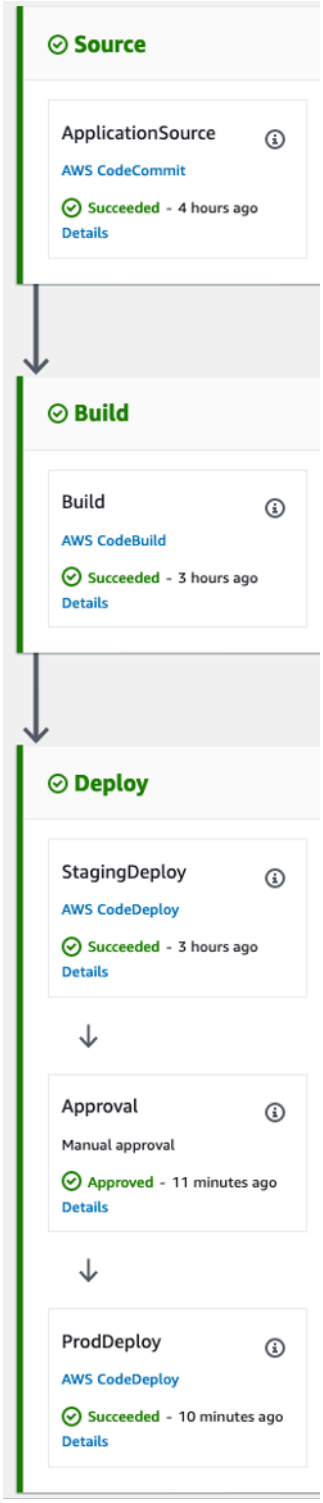
AWS CodePipeline 可和 AWS CodeBuild 一同順暢地運作，這是一種全受管的建置服務，可讓您更輕易地在管道中設定建置步驟，封裝您的程式碼及執行單元測試。有了 AWS CodeBuild 之後，您不需要佈建、管理或擴展您自己的組建伺服器。AWS CodeBuild 可持續擴展，並會同時處理多個建置，使您的建置不會在佇列中等候。AWS CodePipeline 也可以和 Jenkins、Solano CI 和 TeamCity 等組建伺服器整合。

例如，在以下建置階段，三個動作 (單元測試、程式碼樣式檢查，以及收集程式碼指標) 會以平行方式執行。使用 AWS CodeBuild，這些步驟可做為新的專案新增，無需進一步建置或安裝組建伺服器來處理負載。



AWS CodePipeline – 建置功能

AWS CodePipeline — 原始檔和建置階段圖中所顯示的原始檔和建置階段，以及支援程序和自動化，可協助您的團隊轉換至持續整合。到達此成熟度時，開發人員需要定期注意建置和測試結果。他們也需要發展及維持狀況良好的單元測試基礎。而這反過來又會增強整個團隊對 CI/CD 管道的信心，進一步推動其採用。



AWS CodePipeline 階段

持續交付管道

在實作持續整合管道並建立支援程序後，您的團隊便可以開始轉換到持續交付管道。這項轉換需要團隊自動化建置及部署應用程式。

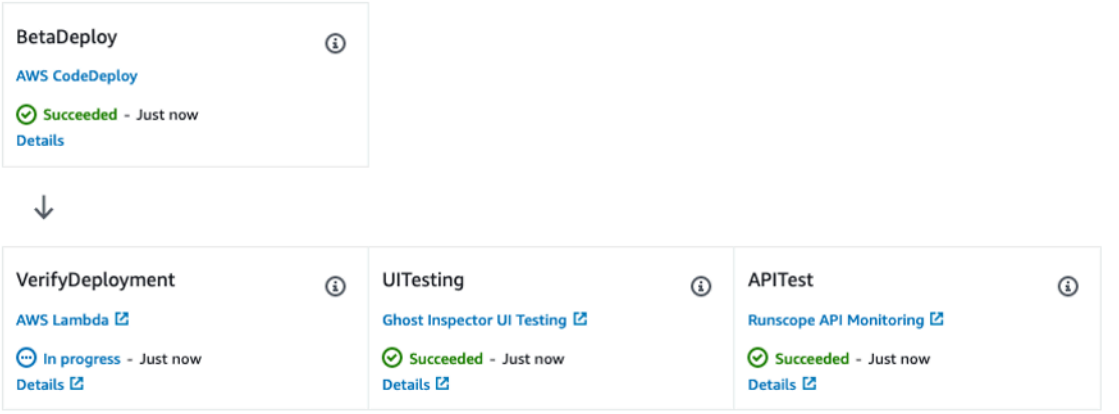
持續交付管道的特性，是存在預備及生產步驟，其中的生產步驟會在手動核准後執行。

與建置持續整合管道的方式相同，您的團隊可以透過撰寫部署指令碼，開始逐步建置持續交付管道。

視應用程式需求的不同，有些部署步驟可透過現有的 AWS 服務抽象化。例如，AWS CodePipeline 可和 AWS CodeDeploy (一種將程式碼自動化部署到 Amazon EC2 執行個體及現場部署執行個體的服務)、AWS OpsWorks (一種組態管理服務，可協助您操作使用 Chef 的應用程式)，以及 AWS Elastic Beanstalk (一種用於部署及擴展 Web 應用程式及服務的服務) 直接整合。

AWS 提供詳細的文件，說明如何實作並將 AWS CodeDeploy 與您的基礎設施及管道整合。

在您的團隊成功自動化應用程式部署後，部署階段便可經由進行各種測試而加以擴張。例如，您可以新增其他立即可用的服務整合，像是 Ghost Inspector、Runscope，以及其他如下圖所示的服務。



AWS CodePipeline – 部署階段中的程式碼測試

新增 Lambda 動作

AWS CodeStar 及 AWS CodePipeline 支援與 AWS Lambda 相整合。這種整合可讓您實作各式各樣的任務，例如在您的環境中建立自訂資源、與第三方系統 (例如 Slack) 相整合，以及在您新部署的環境上執行檢查。

Lambda 函數可在 CI/CD 管道中用來執行下列任務：

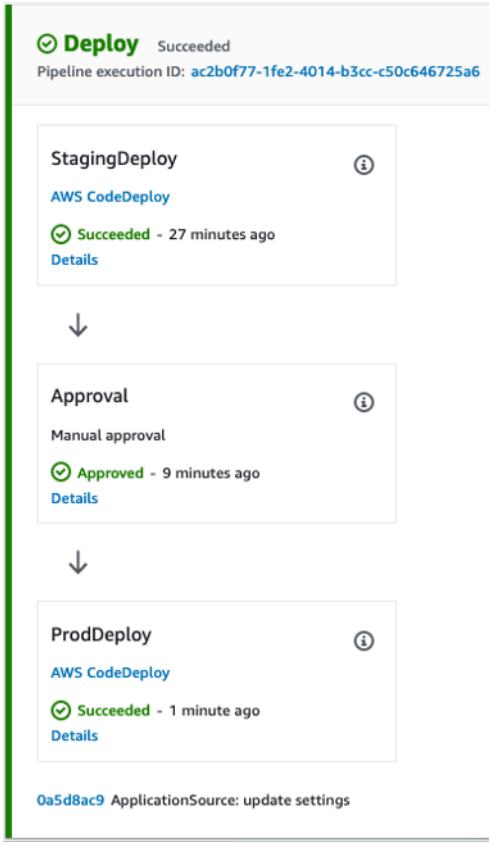
- 套用或更新 AWS CloudFormation 範本，將變更推出到您的環境。

- 使用 AWS CloudFormation 視需要在管道的其中一個階段建立資源，然後在另一個階段刪除該資源。
- 搭配交換正式名稱記錄 (CNAME) 值的 Lambda 函數，在 AWS Elastic Beanstalk 中實現零停機時間部署應用程式版本。
- 部署至 Amazon Elastic Container Service (ECS) Docker 執行個體。
- 建立 AMI 快照，在建置或部署前備份資源。
- 將與第三方產品的整合新增到您的管道，例如張貼訊息至 Internet Relay Chat (IRC) 用戶端。

手動核准

將核准動作新增至管道中您希望管道處理停止的某個階段，讓具備必要 AWS Identity and Access Management (IAM) 許可的人員可以核准或拒絕該動作。

若動作獲得核准，則管道將繼續處理。若動作遭到拒絕，或者在管道抵達該動作並停止後的七天內沒有任何人核准或拒絕該動作，則結果會與動作失敗相同，且管道將不會繼續執行。



AWS CodeDeploy – 手動核准

在 CI/CD 管道中部署基礎設施程式碼變更

AWS CodePipeline 可讓您在管道中的任何階段，選取 AWS CloudFormation 做為部署動作。然後，您可以選擇您希望 AWS CloudFormation 執行的特定動作，例如建立或刪除堆疊，以及建立或執行變更集。堆疊是一種 AWS CloudFormation 概念，代表相關 AWS 資源的群組。儘管有許多方式可佈建「基礎設施即程式碼」，AWS CloudFormation 仍是一種 AWS 推薦的全方位工具，能做為可擴展的完整解決方案，以程式碼的方式描述最全面的 AWS 資源集。AWS 建議在 AWS CodePipeline 專案中使用 AWS CloudFormation 來追蹤基礎設施變更和測試。

無伺服器應用程式的 CI/CD

您還可以使用 AWS CodeStar、AWS CodePipeline、AWS CodeBuild 以及 AWS CloudFormation 來建置無伺服器應用程式的 CI/CD 管道。無伺服器應用程式整合了 Amazon Cognito、Amazon S3 及 Amazon DynamoDB 等受管服務，並具備事件驅動服務及 AWS Lambda 來部署應用程式，且無需管理伺服器。若您是無伺服器應用程式開發人員，可以混合使用 AWS CodePipeline、AWS CodeBuild 及 AWS CloudFormation，自動化建置、測試和部署以 AWS 無伺服器應用模型所建置範本來表達的無伺服器應用程式。如需詳細資訊，請參閱自動化部署 Lambda 型應用程式的 AWS Lambda 文件。

您也可以使用 AWS 無伺服器應用模型管道 (AWS SAM 管道)，建立安全的 CI/CD 管道，遵循組織的最佳實務進行。AWS SAM 管道是 AWS SAM CLI 的新功能，只要數分鐘即可享有 CI/CD 的好處，例如加速部署頻率、縮短變更的前置時間，以及減少部署錯誤。AWS SAM 管道隨附一組適用於 AWS CodeBuild/CodePipelines 的預設管道範本，這些範本皆遵循 AWS 部署最佳實務。如需詳細資訊及檢視教學，請參閱 AWS SAM 管道簡介部落格。

適用於多個團隊、分支和 AWS 區域的管道

針對大型專案，同時有多個專案團隊針對不同元件進行工作的情況並不罕見。若多個團隊使用單一程式碼儲存庫，您可以進行映射，使每個團隊都擁有自己的分支。其中也應具備整合或發行分支，以進行專案的最終合併。若使用服務導向或微型服務架構，每個團隊都可擁有自己的程式碼儲存庫。

在第一種情況下，若僅使用單一管道，則可能會因其中一個團隊封鎖咳管道而影響其他團隊的進度。AWS 建議您為團隊分支建立特定管道，以及建立另一個發行管道，以進行最終產品交付。

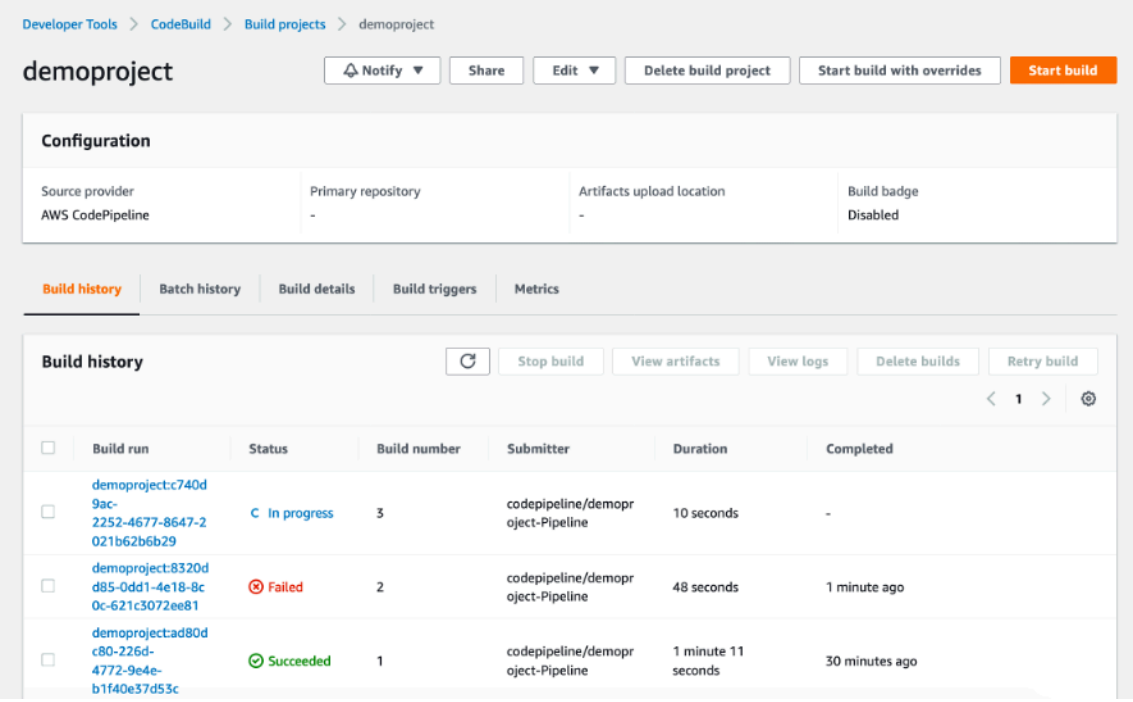
與 AWS CodeBuild 的管道整合

AWS CodeBuild 旨在讓您的組織建置高可用性建置程序，且規模幾乎沒有限制。AWS CodeBuild 為許多熱門的語言提供了快速入門環境，以及執行任何您指定之 Docker 容器的能力。

透過與 AWS CodeCommit、AWS CodePipeline 及 AWS CodeDeploy，以及與 Git 和 CodePipeline Lambda 動作的緊密整合優勢，CodeBuild 工具非常靈活。

您可以透過包含 `buildspec.yml` 檔案來建置軟體，其中指定了每個建置步驟 (包括建置前和建置後的動作)，或是透過 CodeBuild 工具指定動作。

您可以使用 CodeBuild 儀表板，檢視每個建置的詳細歷史記錄。事件會儲存為 Amazon CloudWatch Logs 日誌檔案。



AWS CodeBuild 中的 CloudWatch Logs 日誌檔案

與 Jenkins 的管道整合

您可以使用 Jenkins 建置工具建立交付管道。這些管道使用定義實作持續交付階段步驟的標準作業。但是，這種方法對較大的專案而言，可能並非最佳的做法，因為在 Jenkins 重新啟動後，不會保有管道目前的狀態，實作手動核准並不直覺，且追蹤複雜管道的狀態也可能相當複雜。

AWS 建議您改為使用 AWS Code Pipeline 外掛程式搭配 Jenkins 來實作持續交付。此外掛程式可讓您使用與 Groovy 相似的領域專屬語言，來描述複雜的工作流程，並可用於協調複雜的管道。AWS Code Pipeline 外掛程式的功能可透過使用管線階段檢視外掛程式 (可視覺化管道中所定義之階段的目前進度) 或管道多重分支外掛程式 (可將來自不同分支的組建分組) 增強。

AWS 建議您將管道組態儲存在 Jenkinsfile 中，並將其簽入原始檔儲存庫。這可讓您追蹤對管道程式碼的變更，而這項工作在使用「管道多重分支外掛程式」時會變得更為重要。AWS 也建議您將管道分成

多個階段。這可透過邏輯方式將管道步驟分組，也可以讓「管道階段檢視外掛程式」視覺化管道目前的狀態。

下圖顯示一個 Jenkins 管道範例，其中包括四個並透過「管道階段檢視外掛程式」視覺化的定義階段。

Jenkins 管道的定義階段，透過「管道階段檢視外掛程式」視覺化

部署方法

您可以考慮多種部署策略和變化方式，以在持續交付程序中推出新的軟體版本。本節討論最常見的部署方法：一次全部 (就地部署)、滾動法、不變式，以及藍/綠法。AWS 會指出 AWS CodeDeploy 和 AWS Elastic Beanstalk 支援哪些方法。

下表摘要說明每一種部署方法的特性。

方法	部署失敗的影響	部署時間	零停機時間	無 DNS 變更	回復程序	程式碼部署目標
就地部署	停機時間	🕒	X	✓	重新部署	現有執行個體
滾動式	單一批次停止服務。失敗前成功的批次，會執行新的應用程式版本。	🕒 🕒 †	✓	✓	重新部署	現有執行個體
以額外批次進行滾動 (Beanstalk)	若第一個批次即失敗，則影響極小；否則與滾動式程度相近。	🕒 🕒 🕒 †	✓	✓	重新部署	新執行個體和現有執行個體
不變式	極小	🕒 🕒 🕒 🕒	✓	✓	重新部署	新執行個體
流量分割	極小	🕒 🕒 🕒	✓	✓	重新路由流量並終止新的執行個體	新執行個體

方法	部署失敗的影響	部署時間	零停機時間	無 DNS 變更	回復程序	程式碼部署目標
		🕒				
藍/綠法	極小	🕒 🕒 🕒 🕒	✓	X	切換回舊環境	新執行個體

一次全部 (就地部署)

一次全部 (就地部署) 是一種可用於將新的應用程式程式碼，部署到現有伺服器機群的方法。此方法會在單一部署動作中替換掉所有程式碼。由於會一次更新機群中的所有伺服器，所以其需要停機時間。其無須更新現有 DNS 記錄。若部署失敗，則還原作業唯一的方法是再次對所有伺服器重新部署程式碼。

在 AWS Elastic Beanstalk 中，此部署稱為一次全部，可供單一和負載平衡的應用程式使用。在 AWS CodeDeploy 中，此部署方法稱為就地部署，其部署組態為 `AllAtOnce`。

滾動部署

透過滾動部署，機群會劃分為幾個部分，因此不會同時升級整個機群。在部署程序期間，相同機群上會執行新的和舊的兩個軟體版本。此方法允許零停機時間更新。若該部署失敗，只會影響機群在進行更新的部分。

滾動部署方法的一種變化方式 (稱為 Canary 版本) 涉及先對相當小百分比的伺服器，部署新的軟體版本。透過這種方式，可以觀察軟體在少數伺服器上的生產執行狀況，同時最大限度地降低重大變更的影響。如果 Canary 部署的錯誤率高，則會回復該軟體。否則會逐漸增加使用新版本的伺服器百分比。

AWS Elastic Beanstalk 遵循滾動部署模式，並有兩個部署選項：滾動和以額外批次進行滾動。這些選項可讓應用程式先擴充規模，再停止伺服器的服務，以在部署期間保持完整的功能。AWS CodeDeploy 透過就地部署的變化且搭配 OneAtATime 和 HalfAtATime 等模式，來實現此模式。

不變式與藍/綠法部署

不變式模式會透過使用新組態或新的應用程式程式碼版本，啟動一組全新的伺服器，來指定應用程式程式碼的部署。此模式運用雲端功能，使用簡易的 API 呼叫來建立新的伺服器資源。

藍/綠法部署策略是一種不可變部署類型，也需要建立另一個環境。新環境開始運作且通過所有測試之後，流量便會移動到此新的部署項目。重要的是舊環境 (「藍」環境) 會維持在閒置狀態，以在需要時提供回復。

AWS Elastic Beanstalk 支援不變式和藍/綠法部署模式。AWS CodeDeploy 也支援藍/綠法模式。如需 AWS 服務如何實現這些不變模式的詳細資訊，請參閱《AWS 上的藍/綠法部署》白皮書。

資料庫結構描述變更

現代軟體通常會有一個資料庫層。一般來說會使用關聯式資料庫，其可同時存放資料與資料結構。在持續交付的程序中，經常需要修改資料庫。處理關聯式資料庫中的變更，需要特殊的考量，且會帶來與部署應用程式二進位檔案時不同的其他挑戰。通常，當您升級應用程式二進位檔時，會先停止該應用程式再進行升級，然後再次啟動。您通常不會在意應用程式的狀態，因為這些狀態並非由應用程式處理。

但升級資料庫時，確實需要考慮狀態，因為資料庫包含很多狀態，但相較之下僅包含較少的邏輯和結構。

套用變更之前和之後的資料庫結構描述，應視為不同版本的資料庫。您可以使用 Liquibase 和 Flyway 等工具來管理版本。

一般而言，這些工具採用下列方法的一些變化方式：

- 將資料表新增至存放資料庫版本的資料庫。
- 追蹤資料庫變更命令，並將其整合在版本控制變更集中。Liquibase 會將這些變更存放在 XML 檔案中。Flyway 則採用稍微不同的方法，將變更集做為不同的 SQL 檔案處理，或是有時候針對較為複雜的轉換，將變更集做為不同的 Java 類別處理。
- 當 Liquibase 收到升級資料庫的請求時，其會查看中繼資料資料表，並判斷要執行哪些變更集，以將資料庫升級到最新版本。

最佳實務摘要

下列是一些 CI/CD 的最佳實務。

請：

- 將您的基礎設施視為程式碼
  - 針對您的基礎設施程式碼使用版本控制。
  - 利用錯誤追蹤/票務系統。
  - 在套用變更前讓同儕檢閱變更。
  - 建立基礎設施程式碼模式/設計。
  - 以程式碼變更的相同方式測試基礎設施變更。
- 將開發人員整合到不超過 12 名其作業獨立於其他團隊的開發人員。
- 讓所有開發人員經常將程式碼遞交到主幹，避免長時間存在的功能分支。
- 在您的整個組織中一致採用建置系統 (例如 Maven 或 Gradle) 並標準化建置。
- 讓開發人員建置單元測試，並以測試 100% 的程式碼基底為目標。
- 確保單元測試在持續時間、數量和範圍方面，佔整體測試的 70%。
- 確保所有單元測試都維持在最新狀態，未遭到忽視。應修正單元測試失敗，而非略過。
- 將您的持續交付組態視為程式碼。
- 建立依角色區分的安全控制 (也就是，誰何時可以執行哪些操作)。
  - 盡可能監控/追蹤每個資源。
  - 注意服務、可用性及回應時間。
  - 發現、學習及改善。
  - 與團隊中的每位成員分享存取。
  - 將指標和監控納入生命週期規劃。
- 保持及追蹤標準指標。
  - 建置次數。
  - 部署次數。
  - 將變更送抵生產環境的平均時間。
  - 第一個管道階段到每個階段的平均時間。
  - 送達生產環境的變更數。

  - 平均建置時間。
- 針對每個分支和團隊使用多個獨立管道。

請勿：

- 擁有長時間且大型又複雜相合併的分支。
- 進行手動測試。
- 進行手動核准程序、閘門、程式碼檢閱和安全檢閱。

結論

持續整合和持續交付對您組織的應用程式團隊來說，是理想的搭配。您的開發人員只需將程式碼推送到儲存庫即可。這些代碼將會經過整合、測試、部署、再次測試、與基礎設施合併、通過安全和品質檢閱，並在具備高度信心的情況下準備好進行部署。

使用 CI/CD 可改善程式碼的品質，快速且安心地交付軟體更新，因為您知道其中不包含任何重大變更。任何版本所產生的影響，都可能與生產和營運中的資料彼此有關聯。其也可以用於規劃下一個週期 — 這是組織雲端轉型中一項重要的 DevOps 實務。

深入閱讀

如需本白皮書所討論之主題的詳細資訊，請參閱下列 AWS 白皮書：

- 《AWS 部署選項概觀》
- 《AWS 的藍/綠法部署》
- 《透過將 Jenkins 與 AWS CodeBuild 和 AWS CodeDeploy 整合以設定 CI/CD 管道》
- 《AWS 上的微型服務》
- 《AWS 上的 Docker：在雲端執行容器》

作者群

協力完成本文件的個人與組織如下：

- AWS 首席解決方案架構師 Amrish Thakkar
- AWS 專業服務 DevOps 資深顧問 David Stacy
- AWS 解決方案架構師 Asif Khan
- AWS 資深解決方案架構師 Xiang Shen

文件修訂

若要收到此白皮書更新的通知，請訂閱 RSS 摘要。

update-history-change	update-history-description	update-history-date
初版	白皮書初版	2021 年 10 月 27 日
初版	白皮書初版	2017 年 6 月 1 日

聲明

客戶應負責對本文件中的資訊自行進行獨立評估。本文件：(a) 僅供參考之用，(b) 代表目前的 AWS 產品供應與實務，如有變更恕不另行通知，以及 (c) 不構成 AWS 及其附屬公司、供應商或授權人的任何承諾或保證。AWS 產品或服務以「現況」提供，不提供任何明示或暗示的擔保、主張或條件。AWS 對其客戶之責任與義務，應受 AWS 協議之約束，且本文件並不屬於 AWS 與其客戶間之任何協議的一部分，亦非上述協議之修改。