AWS 白皮書

AWS 上的 DevOps 簡介

AWS 上的 DevOps 簡介: AWS 白皮書

Table of Contents

AWS 上的 DevOps 簡介

出版日期：2020 年 10 月 16 日 (文件修訂)

摘要

如今，企業比以往任何時候都更積極開始數位轉型之旅，與客戶建立更深入的關聯，藉以實現永續和持久的商業價值。Amazon Elastic Container Service。各種型態和規模的組織透過比以往更快的創新打亂競爭對手陣腳並進入新市場。對於這些組織來說，必須關注創新和軟體顛覆，因此簡化軟體交付變得極為重要。將創意到生產的時間縮短的組織優先注重速度和敏捷性，這些組織可能會成為未來的顛覆因素。

雖然在成為下一個數位化顛覆者時需要考慮一些因素，但本白皮書重點介紹 DevOps 以及 AWS 平台中的服務和功能，這些服務和功能將有助於提高組織高速交付應用程式和服務的能力。

簡介

DevOps 是文化、工程實務和模式以及工具的結合，可提高組織以高速和更高品質交付應用程式和服務的能力。隨著時間推移，採用 DevOps 時出現了幾種基本實務：持續整合、持續交付、Infrastructure as Code 以及監控和記錄。

本白皮書重點介紹了可幫助您加快 DevOps 之旅的 AWS 功能，以及 AWS 服務如何幫助消除與 DevOps 適應有關的無差別繁重作業。我們也強調如何在不管理伺服器或建置節點的情況下建置持續整合和交付功能，以及如何利用 Infrastructure as Code 以一致、可重複的方式佈建和管理雲端資源。

- 持續整合：是一項軟體開發實務，指的是開發人員在執行自動化建置與測試之後，定期將他們的程式碼變更合併到中央儲存庫。
- 持續交付：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發行到生產。
- Infrastructure as Code：是使用程式碼和軟體開發技術來佈建與管理基礎設施的實務，例如版本控制和持續整合。
- 監控和記錄：可讓組織查看應用程式和基礎設施效能對產品最終使用者體驗影響的方式。
- 溝通與協作：透過建置工作流程和分配 DevOps 的責任來建立實務，以便團隊更加緊密。
- 安全：應該是一個貫穿各領域的問題。應保護您的持續整合和持續交付 (CI/CD) 管道和相關服務，並設定適當的存取控制許可。

透過對上述每項原則的檢查，都會發現與 Amazon Web Services (AWS) 提供的產品有著密切的關聯。

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程式碼儲存庫中，然後執行自動化建置和測試。CI 有助於更快發現和解決錯誤、改善軟體品質，並減少驗證和釋出新軟體更新所需的時間。

AWS 為持續整合提供以下服務：

主題

- AWS CodeCommit
- AWS CodeBuild
- AWS CodeArtifact

AWS CodeCommit

AWS CodeCommit 是安全且具高可擴展性的受管原始程式碼控制服務，可託管私有的 Git 儲存庫。CodeCommit 不需要您操作自己的來源控制系統，不需要佈建和擴展硬體，也不需要安裝、設定和操作軟體。您可以使用 CodeCommit 儲存從程式碼到二進位檔案的任何項目，且其支援 Git 的標準功能，這讓它能與您現有以 Git 為基礎的工具無縫地運作。您的團隊也可以使用 CodeCommit 的線上程式碼工具來瀏覽、編輯和協作專案。AWS CodeCommit 有一些效益：

協作 - AWS CodeCommit 專為協作軟體開發而設計。您可以輕鬆遞交、分支和合併程式碼，讓您輕鬆保有團隊的專案控制權。CodeCommit 也支援提取請求，提供請求程式碼檢閱和與合作者討論程式碼的機制。

加密 - 您可以選擇透過 HTTPS 或 SSH 從 AWS CodeCommit 傳出和傳入檔案。您的儲存庫也可使用客戶特定金鑰，透過 AWS Key Management Service (AWS KMS) 自動靜態加密。

存取控制 - AWS CodeCommit使用 AWS Identity and Access Management (IAM) 控制和監控哪些人可以存取您的資料，以及存取資料的方式、時間和位置。CodeCommit 也能協助您透過 AWS CloudTrail 和 Amazon CloudWatch 監控儲存庫。

高可用性和耐久性 - AWS CodeCommit 將您的儲存庫存放在Amazon Simple Storage Service (Amazon S3) 和 Amazon DynamoDB中。您的加密資料會以冗餘方式存放在多個設施中。這個架構可提高儲存庫資料的可用性和耐久性。

通知和自訂指令碼 - 您現在可以接收會影響您儲存器事件的通知。通知將以 Amazon Simple Notification Service (Amazon SNS) 管道通知的形式發出。每則通知都會包括狀態訊息，以及產生該通

知的事件資源連結。此外，使用 AWS CodeCommit 儲存器觸發程序，您可以透過 Amazon SNS 傳送通知和建立 HTTP Webhook，或叫用 AWS Lambda 函數以回應您選擇的儲存器事件。

AWS CodeBuild

AWS CodeBuild 是全受管的持續整合服務，可編譯原始碼、執行測試，並產生可立即部署的軟體套件。您不必佈建、管理、擴展自己的組建伺服器。CodeBuild 可以使用 GitHub、GitHub Enterprise、BitBucket、AWS CodeCommit 或 Amazon S3 作為來源提供者。

CodeBuild 會持續擴展並且可同時處理多個組建。CodeBuild 為各種版本的 Microsoft Windows 和 Linux 提供各種預先設定的環境。客戶也可以使用自訂的建置環境作為 Docker 容器。CodeBuild 也整合開放原始碼工具，例如 Jenkins 和 Spinnaker。

CodeBuild 也可以為單位、功能測試或整合測試建立報告。這些報告對於已執行的測試案例數量以及通過或失敗的測試案例提供視覺檢視。建置程序也可以在 Amazon Virtual Private Cloud (Amazon VPC) 內執行，如果您的整合服務或資料庫部署在 VPC 內，這將非常有用。

AWS CodeArtifact

AWS CodeArtifact是全受管成品儲存庫服務，組織可以用來安全存放、發佈和共享其軟體開發程序中使用的軟體套件。可將 CodeArtifact 組態為自動擷取軟體套件和公有成品儲存庫的相依性，讓開發人員存取最新版本。

軟體開發團隊越來越依賴開放原始碼套件來執行應用程式套件中的常見任務。因此，現今成為軟體開發團隊非常重要的一環，就是必須確保開放原始碼軟體的特定版本沒有任何漏洞。使用 CodeArtifact，您可以設定強制執行上述操作的控制項。

CodeArtifact 搭配常用的套件管理程式運作，並建立類似 Maven、Gradle、npm、yarn、twine 和 pip 之類的工具，可輕鬆整合至現有的開發工作流程中。

持續交付

持續交付是一項軟體開發實務，這項實務會自動準備程式碼變更以發行到生產環境。持續交付是現代化應用程式開發的支柱，它透過在建置階段之後將所有程式碼部署到測試環境和/或生產環境，結合持續整合來進一步延伸。適當實作時，開發人員永遠都會有已經部署好的建置成品，且已透過標準化測試程序。

持續交付讓開發人員不只是自動化單位測試之類的測試，所以他們將應用程式更新部署到客戶之前可以從多方面來驗證更新。這些測試可能包含 UI 測試、負載測試、整合測試、API 可靠性測試等。這可協助開發人員更徹底地驗證更新並提前發現問題。使用雲端，要自動建立和複寫多個測試環境不但輕鬆而且經濟實惠，這點之前在內部部署環境是很難做到的。

AWS 為持續交付提供以下服務：

- AWS CodeBuild
- AWS CodeDeploy
- AWS CodePipeline

主題

- AWS CodeDeploy
- AWS CodePipeline

AWS CodeDeploy

AWS CodeDeploy是一項全受管的部署服務，可自動將軟體部署到各種運算服務，例如Amazon Elastic Compute Cloud (Amazon EC2)、AWS Fargate、AWS Lambda 和您的內部部署伺服器。AWS CodeDeploy 使您能夠更輕鬆地快速發佈新功能，幫助您避免在應用程式部署期間停機，並處理更新應用程式的複雜性。您可以使用 CodeDeploy 自動進行軟體部署作業，避免容易出錯的人為操作。此服務可根據您的部署需求進行擴展。

CodeDeploy 具有幾個與持續部署的 DevOps 原則保持一致的效益：

自動化部署：CodeDeploy 可全面自動化您的軟體部署，讓您以可靠且快速的方式進行部署。

集中控制：CodeDeploy 可讓您透過 AWS 管理主控台或 AWS CLI 輕鬆啟動和追蹤您的應用程式部署狀態。CodeDeploy 提供詳細的報告，可讓您查看每個應用程式修訂版的部署時間和位置。您也可以建立推送通知，來接收有關部署的即時更新。

減少停機時間：CodeDeploy 可在軟體部署過程中協助將應用程式可用性提升到最高。它會慢慢增加變動量，並根據可設定的規則來追蹤應用程式運作狀態。如果發生錯誤，可以輕鬆停止並復原軟體部署。

易於採用：CodeDeploy 可與任何應用程式配合使用，並對於不同的平台和語言提供相同的體驗。您可以輕鬆重複使用現有的安裝程式碼。CodeDeploy 也可以與您現有的軟體發佈程序或持續交付工具鏈 (例如，AWS CodePipeline、GitHub、Jenkins) 整合。

AWS CodeDeploy 支援多個部署選項。如需詳細資訊，請參閱部署策略。

AWS CodePipeline

AWS CodePipeline 是一種持續交付的服務，讓您能夠將發行軟體所需的步驟模型化、視覺化和自動化。使用 AWS CodePipeline 時，您要針對建置程式碼、部署到進入生產階段前的環境、測試應用程式與發行到生產環境，製作完整的發行程序模型。每次發生程式碼變更時，AWS CodePipeline 會根據定義的工作流程建置、測試和部署應用程式。您可以將 APN 合作夥伴工具和您自己的自訂工具整合到發行程序的任一階段中，形成端對端持續交付解決方案。

AWS CodePipeline 具有幾個與持續部署的 DevOps 原則保持一致的效益：

快速交付：AWS CodePipeline 可自動化您的軟體發行程序，讓您將新功能快速釋出給使用者。您可以使用 CodePipeline 快速逐一查看意見回饋，並更快速地將新功能提供給使用者。

提升品質：AWS CodePipeline 藉由將您的建置、測試和發行程序自動化，透過一組一致的品質檢查執行所有新變更，讓您能夠提高軟體更新的速度和品質。

輕鬆整合：AWS CodePipeline 可輕鬆擴展以符合您的特定需求。您可以在發行程序的任何步驟使用我們預先建置的外掛程式或自己的自訂外掛程式。例如，您可以從 GitHub 提取原始程式碼、使用內部部署 Jenkins 組建伺服器、使用第三方服務執行負載測試，或者將部署資訊傳送至您的自訂操作儀表板。

可設定的工作流程：AWS CodePipeline 可讓您使用主控台介面、AWS CLI、AWS CloudFormation 或 AWS 開發套件，製作軟體發行程序不同階段的模型。您可以輕鬆指定要執行的測試，以及自訂用來部署應用程式和其相依項的步驟。

部署策略

部署策略定義了您希望如何交付軟體。組織根據其業務模式遵循不同的部署策略。有些組織可能會選擇提供經過全面測試的軟體，而其他組織則可能希望使用者提供意見反映，並讓使用者評估正在開發的功能 (例如測試版)。在以下的部分中，我們將討論各種部署策略。

主題

就地部署

在此策略中，部署的方法是停止部署群組中每個執行個體上的應用程式、安裝最新的應用程式修訂版，然後啟動並驗證新版本的應用程式。您可以使用負載平衡器，讓每個執行個體在其部署期間取消註冊，於部署完成後回復服務。就地部署可以是一次全部，假定服務中斷，也可以作為滾動更新完成。AWS CodeDeploy 和 AWS Elastic Beanstalk 提供一次一個、一次一半和一次全部的部署組態。這些相同的就地部署策略在藍/綠部署中可用。

藍/綠部署

藍/綠 (有時稱為紅黑) 部署是一種應用程式發佈技術，採用的方法是在兩個執行不同應用程式版本的相同環境之間轉換流量。藍/綠部署有助於盡可能減少應用程式更新期間的停機時間，從而降低與停機和回復功能相關的風險。藍/綠部署可讓您與舊版本 (藍) 一起啟動應用程式的新版本 (綠)，並在將流量重新路由到新版本之前監控和測試新版本，並在問題偵測時回復。

Canary 部署

流量以兩個增量轉移。Canary 部署是更加規避風險的藍/綠策略，其中採用分階段辦法。這可以是兩步驟或線性過程，在這種情況下，新的應用程式原始碼會被部署並公開以供試用，並在驗收後推廣到環境的其他部分或以線性方式進行。

線性部署

線性部署表示流量以每個增量之間的相等分鐘數以同等增量轉移。您可從預先指定的線性選項中指定每次增量的流量轉移百分比例，以及在每個增量之間的分鐘數。

一次全部部署

一次全部部署表示所有流量一次全部從原始環境轉移到替換環境。

部署策略矩陣

以下矩陣列出對於 Amazon Elastic Container Service (Amazon ECS)、AWS Lambda 和 Amazon EC2/內部部署支援的部署策略。

- Amazon ECS 是全受管的協同運作服務。
- AWS Lambda 可讓您直接執行程式碼，無需佈建或管理伺服器。
- Amazon EC2 可讓您執行安全、可調整大小的雲端運算容量。

	A	B	C	D
1	部署策略矩陣	Amazon ECS	AWS Lambda	Amazon EC2/內部部署
2	就地	✓	✓	✓
3	藍/綠	✓	✓	✓*
4	Canary	✓	✓	X
5	線性	✓	✓	X
6	一次全部	✓	✓	X

> **Note**
>
> 使用 EC2/內部部署的藍/綠部署僅適用於 EC2 執行個體。

AWS Elastic Beanstalk 部署策略

AWS Elastic Beanstalk 支援以下類型的部署策略：

- 一次全部：在所有執行個體上執行就地部署。
- 滾動：將執行個體拆分為批次，並一次部署到一個批次。

- 使用其他批次進行滾動：將部署拆分為多個批次，但對於第一個批次，則建立新的 EC2 執行個體，而不是在現有 EC2 執行個體上部署。
- 不可變：如果您需要使用新執行個體而不是使用現有執行個體進行部署。
- 流量分割：執行不可變部署，然後在預先確定的持續時間內將流量百分比轉發到新執行個體。如果執行個體保持運作狀態，則將所有流量轉發到新執行個體並關閉舊執行個體。

Infrastructure as Code

DevOps 的一個基本原則是以開發人員對待程式碼的管道對待基礎設施。應用程式原始碼具有定義的格式和語法。如果程式碼不是按照程式設計語言的規則編寫，則無法建立應用程式。程式碼儲存在版本管理或來源控制系統中，其中會記錄程式碼開發歷史記錄、變更和錯誤修復。程式碼被編譯或內建到應用程式中時，我們期望建置一致的應用程式，而且建置可重複而且可靠。

採取 Infrastructure as Code 表示將同樣嚴格的應用程式原始碼開發套用於基礎設施佈建。所有組態都應以宣告方式定義，並儲存在來源控制系統中，例如AWS CodeCommit，與應用程式原始碼相同。基礎設施佈建、協同運作和部署也應支援 Infrastructure as Code 的使用。

傳統上，基礎設施是使用指令碼和手動流程的組合進行佈建。有時，這些指令碼儲存在版本控制系統中，或者逐步記錄在文字檔案或執行手冊中。編寫執行手冊的人通常不是執行這些指令碼或按照執行手冊進行的人。如果這些指令碼或執行手冊不經常更新，則可能不利於部署。這會導致新環境的建立並不總是可重複、可靠或一致。

與上述情況不同，AWS 提供一種以 DevOps 為中心來建立和維護基礎設施的方式。與軟體開發人員編寫應用程式碼的方式類似，AWS 以程式設計、描述性的和聲明性的方式提供支援基礎設施的建立、部署和維護的服務。這些服務提供的嚴謹性、清晰度和可靠性。本白皮書中討論的 AWS 服務是 DevOps 方法的核心，構成了許多更高等級的 AWS DevOps 原則和實務所依據的基礎。

AWS 提供以下服務將基礎設施定義為程式碼。

- AWS CloudFormation
- AWS Cloud Development Kit (AWS CDK)
- AWS Cloud Development Kit for Kubernetes

AWS CloudFormation

AWS CloudFormation 是有助於開發人員能夠以有序、可預測的方式建立 AWS 資源的服務。資源是使用 JavaScript Object Notation (JSON) 或 Yet Another Markup Language (YAML) 格式編寫的文字檔案。這些範本需要特定的語法和結構，而需要的語法和結構取決於要建立和管理的資源類型。您可以使用任何程式碼編輯器 (例如AWS Cloud9) 在 JSON 或 YAML 中建立資源，並簽入到版本控制系統中，然後 CloudFormation 會以安全、可重複的方式建置指定的服務。

CloudFormation 範本會部署到 AWS 環境中成為堆疊。您可以透過 AWS 管理主控台、AWS 命令列介面或 AWS CloudFormation API 管理堆疊。如果您需要變更堆疊內的執行中資源，請更新堆疊。在變

更資源之前，您可以產生變更集以列出請求變更的摘要。藉由變更集，您可以先掌握該變更對執行中資源 (特別是重要資源) 造成的影響，再實行變更作業。

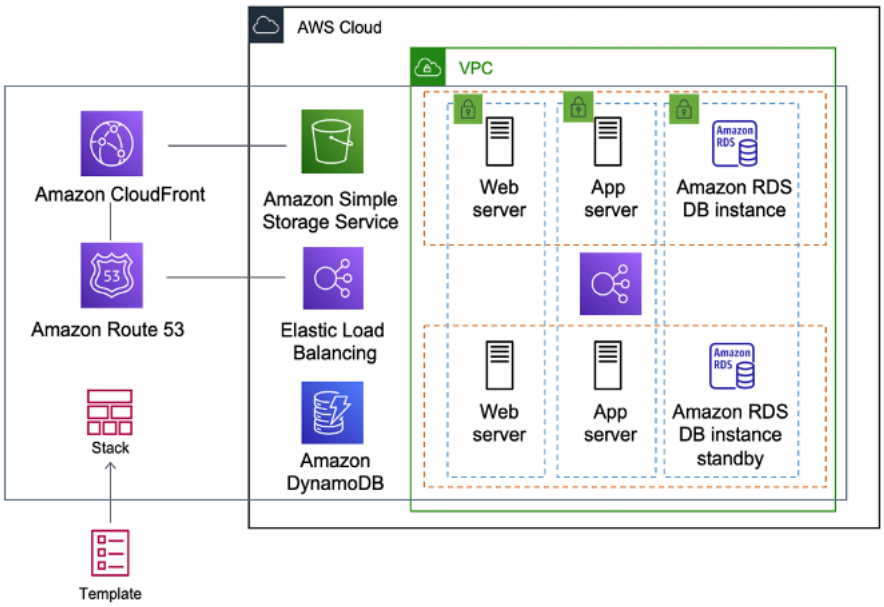

圖 1 - AWS CloudFormation 從一個範本工作流程建立整個環境 (堆疊)

您可以使用單一範本建立和更新整個環境，也可以使用個別的範本管理環境中的多個圖層。如此一來，範本即可模組化，同時也提供對許多組織都非常重要的管控層。

在主控台中建立或更新堆疊時，顯示組態狀態的事件將顯示。如果發生錯誤，堆疊預設將回復到以前的狀態。Amazon Simple Notification Service (Amazon SNS) 提供事件的通知。例如，您可以使用 Amazon SNS 透過電子郵件追蹤建立和刪除堆疊的進度，並以程式設計方式和其他程序整合。

AWS CloudFormation 讓組織和部署 AWS 資源集變得更輕鬆，並讓您在設定堆疊時能夠描述任何相依項或傳入特殊參數。

藉由 CloudFormation 範本，您可以使用多種 AWS 服務，例如 Amazon S3、Auto Scaling、Amazon CloudFront、Amazon DynamoDB、Amazon EC2、Amazon ElastiCache、AWS Elastic Beanstalk、Elastic Load Balancing、IAM、AWS OpsWorks 和 Amazon VPC。關於受支援的資源，如需資源的最新清單，請參閱 AWS 資源和屬性類型參考。

AWS Cloud Development Kit

AWS Cloud Development Kit (AWS CDK) 屬於開放原始碼軟體開發架構，可使用您熟悉的程式設計語言建立您的雲端應用程式資源並且加以佈建。AWS CDK 讓您能夠使用 TypeScript、Python、Java 和 .NET 為應用程式基礎設施建模。開發人員可以利用本身現有的整合開發環境 (IDE)，藉由自動完成和內嵌文件等工具加快開發基礎設施。

AWS CDK 在背景利用 AWS CloudFormation 以安全、可重複的方式佈建資源。建構是 CDK 程式碼的基本建置區塊。建構表示一個雲端組件，並封裝了 AWS CloudFormation 建立組件所需的所有內容。AWS CDK 包括AWS 建構程式庫，其中包含代表許多 AWS 服務的建構。透過將建構結合在一起，就可以快速輕鬆地建立用於在 AWS 中部署的複雜架構。

AWS Cloud Development Kit for Kubernetes

AWS Cloud Development Kit for Kubernetes (cdk8s) 是開放原始碼軟體開發架構，用於使用通用程式設計語言定義 Kubernetes 應用程式。

一旦您用程式設計語言定義應用程式 (截至發佈日期僅支援 Python 和 TypeScript)，cdk8s 會將您的應用程式描述轉換為 Kubernetes 之前的 YML。然後，在任何位置執行的任何 Kubernetes 叢集都可以使用此 YML 檔案。由於結構是用程式設計語言定義的，因此可以使用程式設計語言提供的豐富功能。您可以使用程式設計語言的抽象功能建立您自己的樣板程式碼，並在所有部署中重複使用。

自動化

DevOps 的另一個核心理念和實務是自動化。自動化側重於基礎設施及其中執行的應用程式有關的設定、組態、部署和支援。透過使用自動化，您能夠以標準化和可重複的方式加速設定環境。移除手動流程是成功實施 DevOps 策略的關鍵。過去，伺服器組態和應用程式部署主要是手動過程。環境變得非標準化，在出現問題時複製環境並不容易。

自動化的使用對於實現雲端的全部優勢極為重要。在內部，AWS 非常依賴自動化來提供彈性和可擴展性的核心功能。手動流程容易出錯，不可靠，不足以支援敏捷業務。通常，一個組織可能會佔用高技能的資源來提供手動組態，當時可以更妥善花費時間來支援業務內其他更重要和更高價值的活動。

現代操作環境通常依靠全自動化來消除手動干預或對生產環境的存取。這包括所有軟體發佈、機器組態、作業系統修補、故障診斷或錯誤修復。許多等級的自動化實務可以一起使用，以提供更高等級的端對端自動化過程。

自動化具有以下關鍵優勢：

- 快速變化
- 提升生產力
- 可重複的組態
- 可重現的環境
- 運用彈性
- 運用自動擴展
- 自動測試

自動化是 AWS 服務的基石，並且在所有服務、功能和產品中都受到內部支援。

主題

- AWS OpsWorks
- AWS Elastic Beanstalk

AWS OpsWorks

AWS OpsWorks更進一步採用了 DevOps 的原則AWS Elastic Beanstalk。它可以被視為應用程式管理服務，而不僅僅是一個應用程式容器。AWS OpsWorks透過與組態管理軟體 (Chef) 整合和應用程式生

命週期管理等附加功能，提供更高等級的自動化。您可以使用應用程式生命週期管理來定義何時設定、組態、部署、取消部署或關閉資源。

為AWS OpsWorks了增加靈活性，您可以在可設定的堆疊中定義您的應用程式。您也可以選擇預定義的應用程式堆疊。應用程式堆疊包含應用程式所需的所有 AWS 資源佈建，包括應用程式伺服器、Web 伺服器、資料庫和負載平衡器。

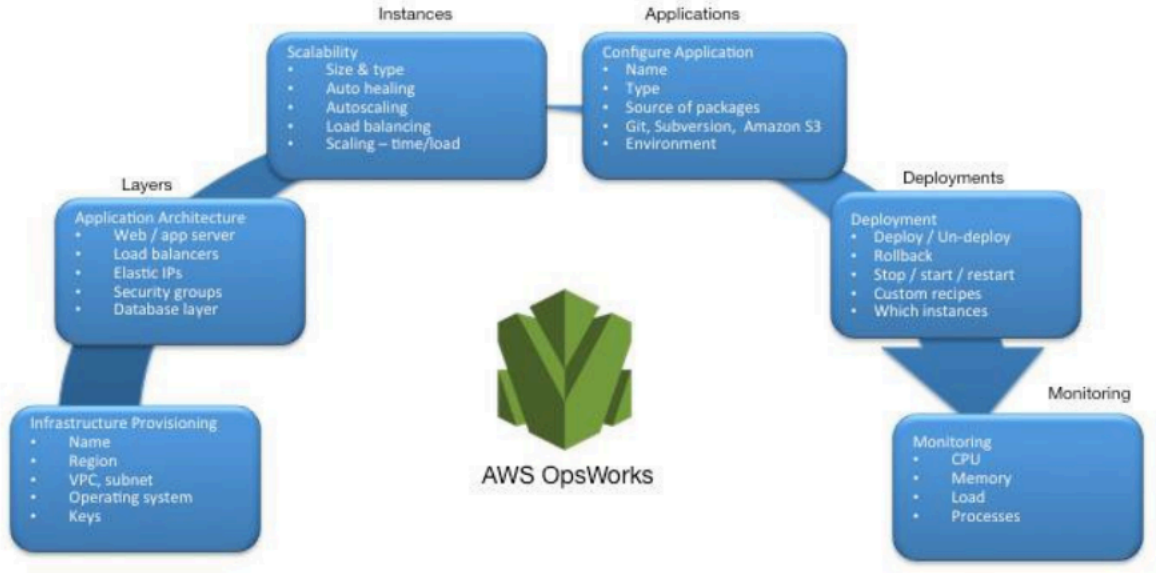

圖 2 - 顯示開發 DevOps 功能和架構的 AWS OpsWorks

應用程式堆疊被組織成架構層，便於個別維護堆疊。範例圖層可以包括 Web 層、應用程式層和資料庫層。開箱即用的 AWS OpsWorks 也簡化了 Auto Scaling 群組和 Elastic Load Balancing 負載平衡器的設定，進一步體現 DevOps 的自動化原則。與 AWS Elastic Beanstalk 一樣，AWS OpsWorks 支援應用程式版本控制、持續部署和基礎設施組態管理。

AWS OpsWorks 也支援 DevOps 監控和記錄的做法 (在下一節中介紹)。監控支援由 Amazon CloudWatch 提供。所有生命週期事件都會予以記錄，而且個別的 Chef 日誌會記錄正在執行的任何 Chef 配方以及任何例外情況。

AWS Elastic Beanstalk

AWS Elastic Beanstalk 是一項服務，用於在熟悉的伺服器 (例如 Apache、NGINX、Passenger 和 IIS) 上部署和擴展以 Java、.NET、PHP、Node.js、Python、Ruby、Go 和 Docker 開發的 Web 應用程式和服務。

Elastic Beanstalk 是 Amazon EC2 之上的抽象 (Auto Scaling)，並透過提供其他功能 (例如，複製、藍/綠部署、Elastic Beanstalk 命令列介面 (eb cli) 以及與 AWS Toolkit for Visual Studio、Visual Studio 程式碼、Eclipse 和 IntelliJ) 提高開發人員的工作效率，藉以簡化部署。

監控與記錄

溝通和協作是 DevOps 理念的基礎。為了促進這一點，回饋極為重要。在 AWS 中，意見反映由兩個核心服務提供：Amazon CloudWatch 和 AWS CloudTrail。它們共同提供了強大的監控、提醒和稽核基礎設施，以便開發人員和營運團隊能夠密切、透明地協作。

AWS 提供以下用於監控和記錄的服務：

主題

- Amazon CloudWatch
- Amazon CloudWatch 警示
- Amazon CloudWatch Logs
- Amazon CloudWatch Logs Insights
- Amazon CloudWatch Events
- Amazon EventBridge
- AWS CloudTrail

Amazon CloudWatch

Amazon CloudWatch 指標會自動從 AWS 服務中收集資料，例如 Amazon EC2 執行個體、Amazon EBS 磁碟區和 Amazon RDS 資料庫執行個體。然後，可以將這些指標組織為儀表板，並可建立警示或事件以觸發事件或執行 Auto Scaling 動作。

Amazon CloudWatch 警示

您可以根據 Amazon CloudWatch 指標收集的指標設定警示。然後，警示可以向 Amazon Simple Notification Service (Amazon SNS) 主題傳送通知或啟動 Auto Scaling 動作。警示需要一段時間 (評估指標的時間長度)、評估週期 (最近資料點的數量) 和警示的資料點 (評估週期內的資料點數)。

Amazon CloudWatch Logs

Amazon CloudWatch Logs 是一項日誌彙總和監控服務。AWS CodeBuild、CodeCommit、CodeDeploy 和 CodePipeline 提供與 CloudWatch Logs 的整合，以便集中監控所有日誌。此外，前面提到的其他各種 AWS 服務提供與 CloudWatch 的直接整合。

藉由 CloudWatch Logs，您可以：

- 查詢您的日誌資料
- 監控來自 Amazon EC2 執行個體的日誌
- 監控 AWS CloudTrail 記錄的事件
- 定義日誌保留政策

Amazon CloudWatch Logs Insights

Amazon CloudWatch Logs Insights 可掃描您的日誌，並且可使您執行互動式查詢和視覺化。它理解各種日誌格式，並自動探索 JSON 日誌中的欄位。

Amazon CloudWatch Events

Amazon CloudWatch Events 會提供近乎即時的系統事件串流，說明 AWS 資源的變動情形。使用您可以快速設定的簡單規則，您可以比對事件並將它們路由到一個或多個目標函數或串流。CloudWatch Events 在營運變更時會查覺到。CloudWatch Events 會回應這些操作變更並視需要進行修正動作，透過傳送訊息來回應環境、啟用功能、執行變更和擷取狀態資訊。

您可以在 CloudWatch Events 中設定規則，以提醒您 AWS 服務中的變更，並使用 Amazon EventBridge 將這些事件與其他第三方系統整合。以下是與 CloudWatch Events 整合的 AWS DevOps 相關服務。

- Application Auto Scaling Events
- CodeBuild 事件
- CodeCommit 事件
- CodeDeploy 事件
- CodePipeline 事件

Amazon EventBridge

Amazon CloudWatch Events 和 EventBridge 是相同的基礎服務和 API，不過 EventBridge 提供了更多功能。

Amazon EventBridge 是一個無伺服器事件樞紐，可實現 AWS 服務、軟體即服務 (SaaS) 和您的應用程式之間的整合。除了建置事件驅動的應用程式之外，EventBridge 也可用於通知 CodeBuild、CodeDeploy、CodePipeline 和 CodeCommit 等服務中的事件。

AWS CloudTrail

為了接受 DevOps 協作、溝通和透明度的原則，瞭解哪些人對您的基礎設施進行修改非常重要。在 AWS 中，這種透明度由 AWS CloudTrail 服務提供。所有 AWS 互動都透過由監控和記錄的 AWS API 呼叫進行處理 AWS CloudTrail。所有產生的日誌檔案都儲存在您定義的 Amazon S3 儲存貯體中。日誌檔是使用 Amazon S3 伺服器端加密 (SSE) 進行加密。無論是直接來自使用者還是 AWS 服務代表使用者發出，所有 API 呼叫都會予以記錄。許多團隊都可以從 CloudTrail 日誌中受益，包括負責支援的營運團隊、負責管控的安全團隊和負責帳務的財務團隊。

通訊與協作

無論您是在組織中採用 DevOps 文化，還是進行 DevOps 文化轉型溝通，協作都是您的方法之中的重要組成部分。Amazon 已經意識到需要改變團隊的思維方式，因此採用了雙披薩團隊的概念。

主題

雙披薩團隊

Bezos 說：「我們試圖建立不超過兩個披薩可以餵養的團隊。我們稱之為雙披薩團隊規則。」

團隊越小，協作就越好。協作也非常重要，因為軟體版本的發展速度比以往任何時候都更快。而且，團隊交付軟體的能力可能是貴組織與競爭對手之間的差異化因素。描述需要發佈新產品功能或需要修復錯誤的情況，您希望儘快進行，以便您可以縮短上市時間。這一點也很重要，因為您不會希望轉型的過程緩慢，而會希望過程明快，過程中的變革都能夠產生影響。

隨著我們走向共同責任模式，並開始擺脫孤立的開發方法，團隊之間的溝通也很重要。這對於團隊帶來了所有權的概念，並轉變了他們認為轉型是端對端的觀點。您的團隊不應將生產環境視為沒有可見性的黑箱。

文化轉型也很重要，因為您可能正在建立共同的 DevOps 團隊，另一種方法是您的團隊中有一個或多個以 DevOps 為中心的成員。這兩種方法確實給團隊帶來了共同責任。

安全性

無論是要進行 DevOps 轉型，還是第一次實施 DevOps 原則，都應該考慮將安全性整合到 DevOps 流程中。這應該是在整個建置、測試部署階段貫穿各領域的關注重點。

在談論 AWS DevOps 中的安全性之前，我們先看一下 AWS 共同責任模式。

主題

- AWS 共同責任模式
- Identity and Access Management

AWS 共同責任模式

安全是 AWS 和客戶的共同責任。共同責任模式的不同部分解釋如下：

- AWS 負責「雲端本身的安全」– AWS 負責保護執行 AWS 雲端提供的所有服務的基礎設施。此基礎設施由硬體、軟體、聯網以及執行 AWS 雲端服務的設施所組成。
- 客戶負責「雲端內部的安全」– 客戶的責任由自行選擇的 AWS 雲端服務決定。這決定客戶在履行其安全性責任過程中必須執行的設定工作量。

這種共同模式有助減輕客戶的運作負擔，因為 AWS 會深入服務運作所在設施的實體安全性，操作、管理並控制主機作業系統及虛擬化層的元件。在客戶希望瞭解建置環境安全性的情況下，這一點極為重要。

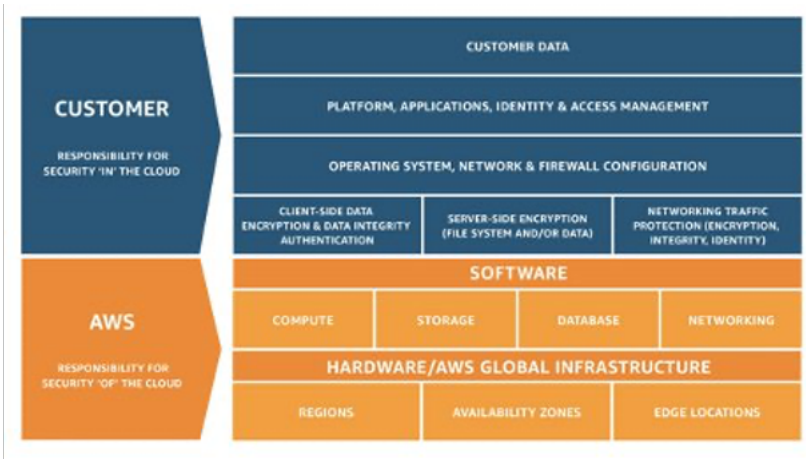

圖 3：AWS 共同責任模式

Identity and Access Management

AWS Identity and Access Management (IAM) 定義了用於管理對 AWS 資源進行存取的控制和政策。使用 IAM，您可以建立使用者和群組，並定義對各種 DevOps 服務的許可。

除了使用者之外，各種服務可能也需要存取 AWS 資源。例如，您的 CodeBuild 專案可能需要存取許可才能在Amazon Elastic Container Registry (Amazon ECR)中儲存 Docker 影像，而且需要許可才能寫入 Amazon ECR。這些類型的許可由稱為服務角色的特殊類型角色定義。

IAM 是 AWS 安全基礎設施的一個組成部分。藉由 IAM，即可集中管理群組、使用者、服務角色和安全憑證 (例如密碼、存取金鑰和許可政策)，藉以控制可供使用者存取的 AWS 服務和資源。IAM 政策可讓您定義許可集。然後，可將此政策連接到角色、使用者或服務以定義各自的許可。您也可以使用 IAM 建立在所需 DevOps 策略中廣泛使用的角色。在某些情況下，以程式設計方式進行 AssumeRole 而不是直接獲得許可可能非常有意義。服務或使用者使用角色時，會獲得臨時憑證來存取您通常無法存取的服務。

結論

為了使雲端之旅順利、有效率而且有成效，科技公司應採用 DevOps 原則和實務。這些原則嵌入在 AWS 平台中。事實上，這些原則構成了許多 AWS 服務的基石，尤其是部署和監控產品中的服務。

首先，使用服務AWS CloudFormation或AWS Cloud Development Kit (AWS CDK)定義 Infrastructure as Code。接下來，定義應用程式在 AWS CodeBuild、AWS CodeDeploy、AWS CodePipeline 和 AWS CodeCommit 等服務的幫助下使用持續部署的方式。在應用程式等級，使用 AWS Elastic Beanstalk、Amazon Elastic Container Service (Amazon ECS) 或 Amazon Elastic Kubernetes Service (Amazon EKS) 和 AWS OpsWorks 等容器，並簡化通用架構的組態。使用這些服務也可以輕鬆包含其他重要服務，例如 Auto Scaling 和 Elastic Load Balancing。最後，使用 DevOps 的監控策略 (例如 Amazon CloudWatch) 以及可靠的安全實務 (例如 AWS IAM)。

藉由 AWS 作為您的合作夥伴，您的 DevOps 原則可為您的業務和 IT 組織帶來敏捷性，並加快您的雲端之旅。

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
已還原缺少的「貢獻者」區段	已還原缺少的「貢獻者」區段並進行小幅度的文字變更	2020 年 11 月 21 日
已更新一些區段而納入新服務	已更新一些區段而納入新服務	2020 年 10 月 16 日
初始出版	白皮書初始出版	2014 年 12 月 1 日

作者群

此文件的作者包括：

- Muhammad Mansoor，解決方案架構師
- Ajit Zadgaonkar，現代化全球技術主管
- Juan Lamadrid - 解決方案架構師
- Darren Ball - 解決方案架構師
- Rajeswari Malladi - 解決方案架構師
- Pallavi Nargund - 解決方案架構師
- Bert Zahniser - 解決方案架構師
- Abdullahi Olaoye – 雲端解決方案架構師
- Mohamed Kiswani – 軟體開發經理
- Tara McCann – 經理解決方案架構師

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。