AWS Well-Architected Framework

可靠性支柱

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

可靠性支柱: AWS Well-Architected Framework

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

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能隸屬於 Amazon，或與 Amazon 有合作關係，或由 Amazon 贊助。

Table of Contents

摘要和介紹 ... 1
簡介 ... 1
可靠性 ... 2
彈性的共同責任模型 ... 2
設計原則 ... 4
定義 ... 5
彈性和可靠性的組成部分 ... 6
可用性 ... 6
災難復原 (DR) 目標 ... 9
了解可用性需求 ... 10
基礎 ... 12
管理服務配額和限制 ... 12
REL01-BP01 了解服務配額和限制 ... 12
REL01-BP02 管理跨帳戶和區域的服務配額 ... 17
REL01-BP03 透過架構適應固定服務配額和限制 ... 20
REL01-BP04 監控和管理配額 ... 24
REL01-BP05 自動化配額管理 ... 27
REL01-BP06 確保目前配額與最大用量之間存在足夠的間隙，以適應容錯移轉 ... 29
規劃您的網路拓撲 ... 33
REL02-BP01 針對工作負載公有端點使用高可用性網路連線 ... 33
REL02-BP02 佈建雲端和內部部署環境中私有網路之間的備援連線 ... 37
REL02-BP03 確保 IP 子網路配置考量擴充性和可用性 ... 39
REL02-BP04 偏好軸輻式拓撲而非多對多網狀拓撲 ... 42
REL02-BP05 在所有連線的私有地址空間中強制執行不重疊的私有 IP 地址範圍 ... 44
工作負載架構 ... 47
設計您的工作負載服務架構 ... 47
REL03-BP01 選擇如何分割工作負載 ... 48
REL03-BP02 建置專注於特定業務網域和功能的服務 ... 50
REL03-BP03 提供每個 的服務合約 API ... 53
在分散式系統中設計防止失敗的互動 ... 56
REL04-BP01 識別您依賴的分散式系統種類 ... 56
REL04-BP02 實作鬆散耦合相依性 ... 61
REL04-BP03 持續工作 ... 64
REL04-BP04 讓變異操作冪等 ... 65

設計分散式系統中的互動以緩解或承受失敗 69
REL05-BP01 實作適度降級，以將適用的硬相依性轉換為軟相依性 70
REL05-BP02 限流請求 72
REL05-BP03 控制和限制重試呼叫 76
REL05-BP04 快速失敗並限制佇列 78
REL05-BP05 設定用戶端逾時 81
REL05-BP06 盡可能讓系統處於無狀態 84
REL05-BP07 實作緊急槓桿 85
變更管理 88
監控工作負載資源 88
REL06-BP01 監控工作負載的所有元件 (產生) 89
REL06-BP02 定義和計算指標 (彙總) 92
REL06-BP03 傳送通知 (即時處理和警示) 95
REL06-BP04 自動化回應 (即時處理和警示) 98
REL06-BP05 分析日誌 101
REL06-BP06 定期審查監控範圍和指標 102
REL06-BP07 監控 end-to-end透過您的系統追蹤請求 104
設計工作負載以適應需求變更 107
REL07-BP01 取得或擴展資源時使用自動化 107
REL07-BP02 在偵測到工作負載受損時取得資源 110
REL07-BP03 偵測到工作負載需要更多資源時取得資源 111
REL07-BP04 Load 測試您的工作負載 114
實作變更 115
REL08-BP01 將執行手冊用於部署等標準活動 116
REL08-BP02 將功能測試整合為部署的一部分 117
REL08-BP03 整合彈性測試作為部署的一部分 120
REL08-BP04 使用不可變基礎設施進行部署 121
REL08-BP05 使用自動化部署變更 125
故障管理 128
備份資料 128
REL09-BP01 識別並備份所有需要備份的資料，或從來源複製資料 129
REL09-BP02 保護和加密備份 132
REL09-BP03 自動執行資料備份 133
REL09-BP04 定期執行資料復原以驗證備份的完整性和程序 135
使用故障隔離來保護您的工作負載 138
REL10-BP01 將工作負載部署至多個位置 139

REL10-BP02 針對限制在單一位置的元件將復原自動化 145
REL10-BP03 使用隔板架構限制影響範圍 147
設計工作負載以承受元件失敗 150
REL11-BP01 監控工作負載的所有元件以偵測故障 151
REL11-BP02 容錯移轉至運作良好的資源 153
REL11-BP03 將所有分層的修復自動化 157
REL11-BP04 在復原期間依賴資料平面而非控制平面 160
REL11-BP05 使用靜態穩定性來防止雙模式行為 163
REL11-BP06 當事件影響可用性時傳送通知 166
REL11-BP07 建立您的產品架構以符合可用性目標和運行時間服務水準協議 (SLA) 168
測試可靠性 171
REL12-BP01 使用程序手冊調查失敗 171
REL12-BP02 執行事件後分析 173
REL12-BP03 測試可擴展性和效能需求 175
REL12-BP04 使用混沌工程測試彈性 178
REL12-BP05 定期進行演練日 186
災難復原 (DR) 計畫 189
REL13-BP01 定義停機時間和資料遺失的復原目標 189
REL13-BP02 使用定義的復原策略來滿足復原目標 193
REL13-BP03 測試災難復原實作以驗證實作 204
REL13-BP04 管理 DR 站點或區域的組態偏移 206
REL13-BP05 自動化回復 208
結論 212
貢獻者 213
深入閱讀 214
文件修訂 215
注意 220
AWS 詞彙表 221

可靠性支柱 - AWS Well-Architected Framework

發布日期：2024 年 11 月 6 日 (文件修訂)

本白皮書的重點是 AWS Well-Architected Framework 的可靠性支柱。本文提供了相關指引，可協助客戶在設計、交付和維護 Amazon Web Services (AWS) 環境時應用最佳實務。

簡介

AWS Well-Architected Framework 可協助您了解在 AWS 上建置工作負載時所做決策的優缺點。透過使用該架構，您將了解關於在雲端設計和操作可靠、安全、有效率、經濟實惠且永續的工作負載的架構最佳實務。藉助該架構，可根據最佳實務一致地量測架構，並識別需要改進的方面。我們相信，擁有 Well-Architected 工作負載可大幅提高企業成功的可能性。

AWS Well-Architected Framework 以六個支柱為基礎：

- 操作效能
- 安全
- 可靠性
- 效能效率
- 成本最佳化
- 永續性

本白皮書重點介紹了可靠性支柱，以及如何將其套用於您的解決方案。由於單點故障、缺乏自動化和缺乏彈性，在傳統的內部部署環境中，要實現可靠性極具挑戰性。透過採用本白皮書中的實務，您可以建置具有堅實基礎、彈性的架構、一致的變更管理和經驗證的失敗復原程序的架構。

本白皮書適用於擔任技術職務的人員，例如技術長 (CTO)、架構師、開發人員和營運團隊成員。閱讀本白皮書之後，您將了解在設計可靠性雲端架構時要使用的 AWS 最佳實務和策略。本白皮書包括高階實作詳細資訊和架構模式，以及其他資源的參考資料。

可靠性

可靠性支柱包括工作負載如預期般正確、一致地執行其預期功能的能力。包括在整個生命週期中執行及測試工作負載。本白皮書深入說明在 AWS 上實作可靠工作負載的相關事項，提供最佳實務指引。

主題

- 彈性的共同責任模型
- 設計原則
- 定義
- 了解可用性需求

彈性的共同責任模型

彈性是 AWS 與您之間的共同責任。您了解在此共用模型之下，做為彈性一部分的災難復原 (DR) 和可用性如何操作相當重要。

AWS 責任 - 雲端的彈性

AWS 會負責基礎設施的彈性，以執行 AWS 雲端 提供的所有服務。此基礎設施包含執行 AWS 雲端 服務的硬體、軟體、聯網以及設施。AWS 會採取商業上合理的努力來讓這些 AWS 雲端服務可供使用，以確保服務可用性符合或超過 AWS 服務水準協議 (SLA)。

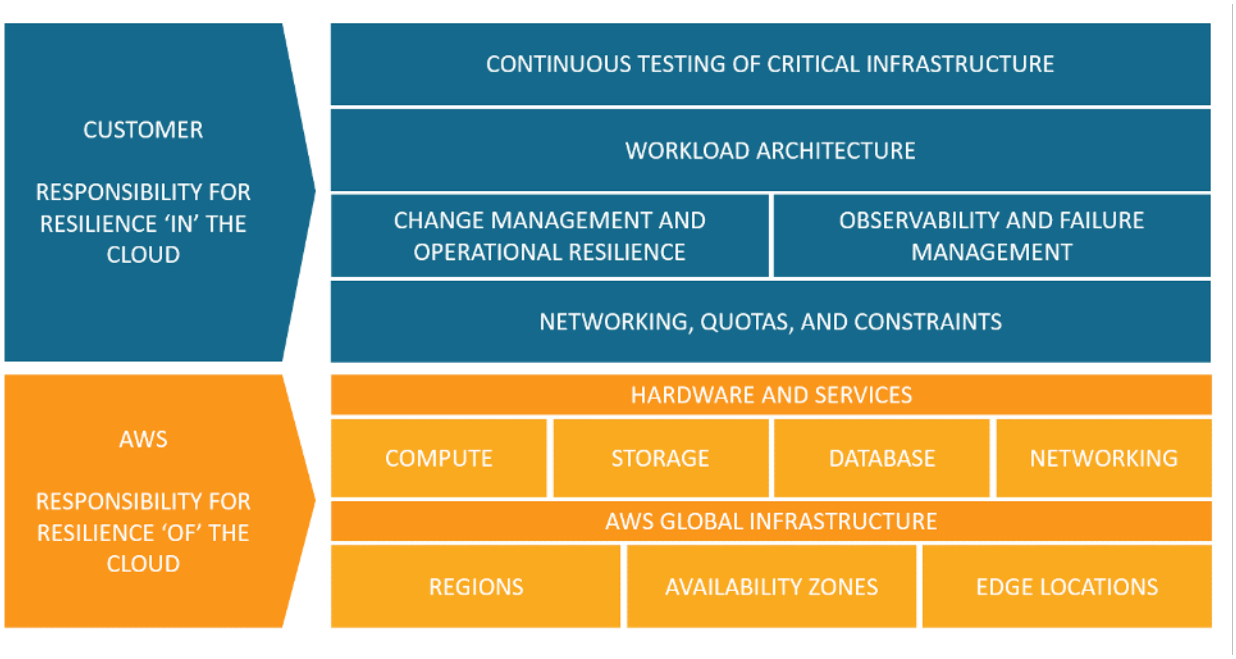
AWS 全球雲端基礎設施旨在可讓客戶建置高彈性的工作負載架構。每個 AWS 區域 區域都是完全隔離的，由多個可用區域組成，這些區域是基礎設施的實體隔離分區。可用區域會隔離可能影響工作負載彈性的故障，防止這些故障影響區域中的其他區域。但是於此同時 AWS 區域中的所有區域都是使用完全備援的專用都會光纖 (在區域之間提供高輸送量、低延遲網路)，搭配高頻寬、低延遲聯網來互連。區域之間的所有流量都會加密。網路效能足以完成區域之間的同步複寫。應用程式跨 AZ 分割時，公司可以獲得更好的隔離和保護，讓您免於停電、雷擊、龍捲風、颶風等問題。

客戶責任 - 雲端中的彈性

您的責任是由您選取的 AWS 雲端服務來決定的。這決定您在履行彈性責任過程中必須執行的設定工作量。例如，Amazon Elastic Compute Cloud (Amazon EC2) 之類的服務需要客戶執行所有必要的彈性組態和管理任務。部署 Amazon EC2 執行個體的客戶要負責在多個位置部署 Amazon EC2 執行個體 (例如 AWS 可用區域)、實作自我修復，使用例如 Auto Scaling 的服務，並且針對安裝在執行個體上的應用程式使用彈性工作負載架構最佳實務。對於 Amazon S3 和 Amazon DynamoDB 等受管服

務，AWS 會操作基礎設施層、作業系統和平台，而且客戶會存取端點以儲存和擷取資料。您負責管理您的資料的彈性，包括備份、版本控制和複寫策略。

在 AWS 區域中的多個可用區域之間部署您的工作負載，是高可用性策略的一部分，該策略的設計目的是藉由將問題隔離到其中一個可用區域來保護工作負載，使用其他可用區域的備援持續為請求提供服務。多可用區域架構也是 DR 策略的一部分，其設計目的是讓工作負載更好地隔離，並且防範例如停電、雷擊、龍捲風、地震等問題。DR 策略也會使用多個 AWS 區域。例如，在主動/被動組態中，如果主動區域再也無法為請求提供服務，則工作負載的服務會從其主動區域容錯移轉到其 DR 區域。



客戶和 AWS 對於雲端中彈性的責任。

您可以使用 AWS 服務來達成您的彈性目標。身為客戶，您負責管理系統的下列層面，達成雲端中的彈性。如需具體各個服務的詳細資訊，請參閱 AWS 文件。

聯網、配額和限制

- 此區域的共同責任模式最佳實務在基礎底下詳細說明。
- 根據適用情況下的預期負載請求增加，使用足夠的空間規劃您的架構，以便擴展和了解您所包含服務的服務配額和限制。
- 將您的網路拓撲設計成高度可用、備援和可擴展。

變更管理和營運彈性

- 變更管理包括如何在您的環境中引入和管理變更。實作變更需要建置執行手冊並且保持最新狀態，以及您的應用程式和基礎設施的部署策略。
- 監控工作負載資源的彈性策略會考慮所有元件，包括技術和商業指標、通知、自動化和分析。
- 雲端中的工作負載必須適應需求變更擴展，以因應損害或用量波動。

可觀測性和失敗管理

- 需要透過監控觀察失敗才能自動化修復，讓您的工作負載可以承受元件失敗。
- 失敗管理需要備份資料、套用最佳實務以便讓您的工作負載承受元件失敗，以及規劃災難復原。

工作負載架構

- 您的工作負載架構包括您如何設計商業網域的服務、套用 SOA 和分散式系統設計來防止失敗，以及建置限流、重試、佇列管理、逾時和緊急控制桿之類的功能。
- 仰賴經實證的 AWS 解決方案，Amazon 建置者資料中心和無伺服器模式可以與最佳實務保持一致，並且立即開始實作。
- 使用持續改善將您的系統分解成分散式服務，更快擴展和創新。使用 AWS 微型服務指引和受管服務選項，簡化及加速您引入變更和創新的能力。

持續測試關鍵基礎設施

- 測試可靠性是指測試功能、效能和混沌層級，以及採用事件分析和演練日實務來建置專業知識，解決尚未充分了解的問題。
- 對於全面雲端和混合應用程式，了解發生問題或元件停機時的應用程式行為方式，可讓您快速且可靠地從中斷復原。
- 建立和記載可重複的試驗，了解事情未如預期般運作時，您的系統的行為方式。這些測試會證明您的整體彈性的有效性，並且在您的操作程序面臨實際失敗情境時，提供意見回饋循環。

設計原則

在雲端，有很多原則可協助您提高可靠性。討論最佳實務時，請謹記以下幾點：

- 自動從故障中復原：透過監控工作負載的關鍵績效指標 (KPI)，您可在達到臨界值時執行自動化。這些 KPI 應為業務價值的衡量指標，而非服務營運的技術方面。如此一來，即可自動通知和追蹤失

敗，以及自動化可解決或修復失敗的復原程序。藉助更複雜的自動化功能，您可以在發生失敗前進行預測和修補。

- 測試復原程序：在內部部署環境中，經常執行測試以證明工作負載可在特定情況下正常工作。測試通常不可用於驗證復原策略。在雲端，您可測試工作負載會發生哪些失敗情境，同時可驗證復原程序。您可使用自動化來模擬不同的失敗情境或重新建立會導致之前失敗的情境。此方法會在實際的失敗情境發生前公開您可以測試和修復的失敗路徑，從而降低風險。
- 水平擴展，以增加彙總工作負載的可用性：使用多個小資源取代一個大資源，以降低整體工作負載上發生單一失敗時造成的影響。將請求分散到多個較小的資源，以確保它們不會有共同的失敗點。
- 停止猜測容量：內部部署工作負載失敗的一個常見原因是資源飽和，即當對工作負載的需求超出該工作負載的容量時發生的情況 (這通常為阻斷服務攻擊的目標)。在雲端，您可以監控需求和工作負載利用率，並自動新增或刪除資源，以保持可滿足需求的最佳水平，而不會過度佈建或佈建不足。仍然存在限制，但是某些配額可以控制，而其他限制則可管理 (請參閱管理服務配額和限制)。
- 透過自動化管理變更：應透過自動化來執行對基礎架構的變更。需要管理的變更包括之後可以追蹤和審查的自動化變更。

定義

本白皮書涵蓋雲端可靠性，並介紹以下四大領域的最佳實務：

- 基礎
- 工作負載架構
- 變更管理
- 失敗管理

若要實現可靠性，您必須先從基礎開始，即服務配額和網路拓撲能適應工作負載的環境。分散式系統的工作負載架構在設計上必須能防止失敗並減輕失敗的影響。工作負載必須處理需求或要求的變更，且在設計上須能偵測失敗並自動進行自我修復。

主題

- 彈性和可靠性的組成部分
- 可用性
- 災難復原 (DR) 目標

彈性和可靠性的組成部分

雲端中的工作負載可靠性取決於數項因素，其中主要的因素是彈性：

- 彈性是工作負載在以下方面的能力：從基礎架構或服務中斷恢復、動態取得運算資源以符合需求，以及緩解中斷狀況 (例如，設定錯誤或暫時性網路問題)。

影響工作負載可靠性的其他因素如下所述：

- 卓越營運，包括變更自動化、使用程序手冊回應失敗，以及營運準備度審查 (ORR)，以確認應用程式已準備好用於生產營運。
- 安全性，包括防止惡意動作項目損毀資料或基礎架構，進而影響可用性。例如，加密備份以確保資料安全無虞。
- 效能達成效率，包括實現工作負載最大請求率和最小延遲的設計。
- 成本最佳化，包括是否在 EC2 執行個體上投入更多資源，以實現靜態穩定性，或是否在需要更多容量時仰賴自動調整規模等方面的權衡。

彈性是本白皮書的重點。

其他四個方面也很重要，並都有各自的 AWS Well-Architected Framework 支柱白皮書說明。這裡的許多最佳實務也會處理可靠性的這些層面，但重點是在彈性上。

可用性

可用性 (也稱為服務可用性) 既是定量衡量彈性的常用指標，也是目標彈性目標。

- 可用性是工作負載可供使用的時間百分比。

可供使用是指工作負載在需要時成功執行其議定的功能。

該百分比根據一段時間 (例如，一個月、一年或過去三年) 計算得出。套用最嚴格的解釋，只要應用程式無法正常運行 (包括計劃和非計劃中的中斷)，可用性就會降低。我們將可用性定義如下：

$$Availability = \frac{Available\ for\ Use\ Time}{Total\ Time}$$

- 可用性是一段時間 (通常為一個月或一年) 內運行時間的百分比 (例如 99.9%)

- 常見的簡寫僅以「九的數量」表示；例如，「五個九」可轉化為 99.999% 的可用率
- 一些客戶選擇從公式中的總時間中排除計劃的服務停機時間 (例如，計劃的維護)。不過，不建議這樣做，因為您的使用者可能想要在這些時間使用您的服務。

下表為通用應用程式可用性設計目標及在保證仍可達到目標的情況下，一年內發生的最大中斷時長。該表格包含我們通常會在每個可用性層看到的應用程式類型範例。在本文件中，我們將引用這些值。

可用性	最高的無法使用程度 (每年)	應用程式類別
99%	3 天 15 小時	批次處理、資料擷取、傳輸和載入任務
99.9%	8 小時 45 分鐘	知識管理、專案追蹤等內部工具
99.95%	4 小時 22 分鐘	線上商務、銷售點
99.99%	52 分鐘	影片交付、廣播工作負載
99.999%	5 分鐘	ATM 交易、電信工作負載

根據請求衡量可用性。對於您的服務，計算成功和失敗的請求數可能比計算「可用時間」更容易。在此情況下，可以使用下列計算：

$$Availability = \frac{Successful\ Responses}{Valid\ Requests}$$

這通常以一分鐘或五分鐘期間進行測量。然後可以根據這些期間的平均值計算每月運行時間百分比 (時間型可用性測量)。如果在給定期間未收到任何請求，則該時間的可用率為 100%。

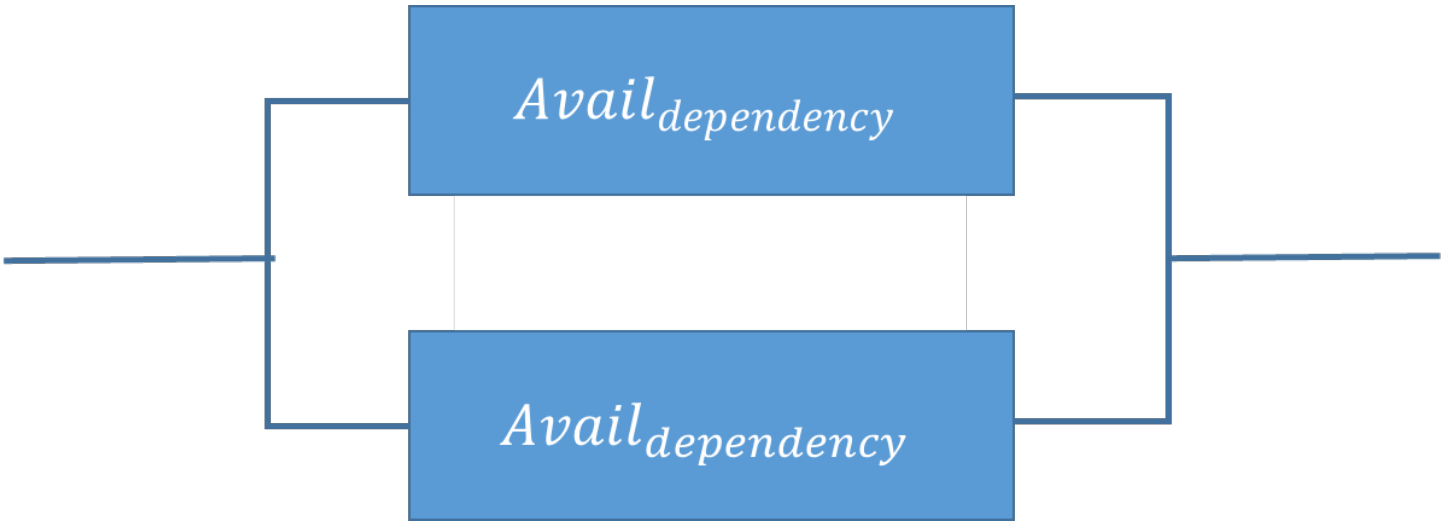
使用硬相依性計算可用性。許多系統對其他系統存有硬相依性，其中相依系統中的中斷會直接轉化為調用系統的中斷。這與軟相依性相反，後者在應用程式中補償了相依系統的故障。若發生這種硬相依性，調用系統的可用性是相依系統的可用性的乘積。例如，如果您有一個設計為 99.99% 可用性的系統，並且與其他兩個設計為 99.99% 可用性的獨立系統存在硬相依性，則該工作負載理論上可以實現 99.97% 的可用性：

$Avail_{invok} \times Avail_{dep1} \times Avail_{dep2} = Avail_{workload}$

99.99% × 99.99% × 99.99% = 99.97%

因此，在計算您自己的相依性及其可用性設計目標時，務必要對其有一定的了解。

使用冗餘元件計算可用性。當系統涉及使用獨立的備援元件 (例如，不同可用區域中的備援資源) 時，理論可用性的計算方式為 100% 減去元件故障率的乘積。例如，如果系統使用兩個獨立的元件，每個元件的可用性為 99.9%，則此相依性的有效可用性為 99.9999%：



$Avail_{effective} = Avail_{MAX} - ((100\% - Avail_{dependency}) \times (100\% - Avail_{dependency}))$

99.9999% = 100% − (0.1%×0.1%)

捷徑計算：如果您的計算中所有元件的可用性僅由數字 9 組成，則您可以將 9 數字的計數相加來得到您的答案。在上面範例中，兩個具有三個 9 可用性的備援獨立組件造成六個 9。

計算相依系統可用性。某些相依系統提供了有關其可用性的指引，包括許多 AWS 服務的可用性設計目標。但是，在無法使用此功能的情況下 (例如，製造商未發布可用性訊息的元件)，其中一種估算方法是確定平均故障間隔時間 (MTBF) 和平均復原時間 (MTTR)。可透過以下方式確定可用性估算值：

$$Avail_{EST} = \frac{MTBF}{MTBF + MTTR}$$

例如，如果 MTBF 為 150 天，MTTR 為 1 小時，則可用性估算值為 99.97%。

如需詳細資訊，請參閱 Availability and Beyond: Understanding and improving the resilience of distributed systems on AWS，該文件可協助您計算可用性。

可用性成本。設計具有較高可用性的應用程式通常會導致成本增加，因此在著手進行應用程式設計之前，有必要識別出真正的可用性需求。對於在詳盡的失敗情境下進行測試和驗證，高可用性會實施更嚴格的要求。他們需要自動化以從各種失敗中復原，並且要求系統營運的所有方面均以相似的方式進行建置和測試，已達到相同的標準。例如，必須進行容量的新增或刪除，更新軟體或組態變更的部署或還原，或者系統資料的移轉，以達到所需的可用性目標。在非常高的可用性水平上，軟體開發成本增加了，但由於在部署系統中需要更緩慢地移動，創新將會受到影響。因此，該指南詳述了如何套用標準，以及考慮操作系統的整個生命週期中的適當可用性目標。

在具有較高可用性設計目標的系統中，成本不斷攀升的另一種方式是選擇相依系統。在這些較高的目標下，可以選擇哪些軟體或服務作為相依系統，取決於這些服務中的哪一項具有我們前面所述的大量投資。隨著可用性設計目標的提高，通常會發現更少的多功能服務 (如關聯式資料庫) 和更多的專用服務。這是因為後者更易於評估、測試和自動化，並且可減少與包含但未使用的功能進行意外互動的可能性。

災難復原 (DR) 目標

除了可用性目標之外，您的彈性策略還應包括基於策略的災難復原 (DR) 目標，以在發生災難事件時復原您的工作負載。災難復原專注於回應自然災難、大規模技術故障，或攻擊或錯誤等人為威脅的一次性復原目標。這與可用性不同，其會測量一段時間內回應元件失敗、負載峰值或軟體錯誤的平均彈性。

復原時間點目標 (RTO) 由組織定義。RTO 是服務中斷與恢復服務之間的最大可接受延遲。這會決定可接受的服務無法使用之時間長度。

復原點目標 (RPO) 由組織定義。RPO 是自上次資料復原點之後的最大可接受時間長度。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。

RPO (復原點目標)、RTO (復原時間點目標) 和災難事件的關係。

RTO 與 MTTR (平均復原時間) 相似，兩者都會測量從中斷開始到工作負載復原的這段時間。不過，MTTR 是從一段時間內的多項影響可用性事件取得的平均值，而 RTO 則是單一影響可用性事件允許的目標或最大值。

了解可用性需求

通常最開始會將應用程式的可用性視為整個應用程式的單一目標。但是，在仔細檢查後，我們經常會發現應用程式或服務的某些方面具有不同的可用性要求。例如，某些系統可能會優先考慮在擷取現有資料之前接收和儲存新資料的能力。其他系統優先於即時營運，而不是變更系統組態或環境的營運。服務可能在一天中的某些時段有很高的可用性要求，但可以忍受非這些時段內更長的中斷時間。您可以透過下列幾種方法將單一應用程式分解為若干個組成部分，並評估每個部分的可用性要求。這樣做的好處是可以根據特定需求，將您的工作 (和費用) 聚焦於可用性上，而不是按照最嚴格的要求設計整個系統。

建議
嚴格評估應用程式的獨特方面，並在適當時區分可用性和災難復原設計目標，以反映您的業務需求。

在 AWS 內，我們通常將服務分為「資料平面」和「控制平面」。資料平面負責提供即時服務，而控制平面則用於設定環境。例如，Amazon EC2 執行個體、Amazon RDS 資料庫和 Amazon DynamoDB 資料表讀取/寫入操作均為資料平面操作。相反，啟動新的 EC2 執行個體或 RDS 資料庫，或在 DynamoDB 中新增或變更資料表中繼資料則均會被視為控制平面操作。儘管高可用性對於所有這些功能都很重要，但資料平面通常具有比控制平面更高的可用性設計目標。因此，具有高可用性要求的工作負載應避免控制平面操作上的執行時間相依性。

許多 AWS 客戶都採用類似的方法來嚴格評估其應用程式，並識別具有不同可用性需求的子元件。然後，針對不同方面客製化可用性設計目標，並執行適當的工作以對系統進行設計。AWS 擁有豐富的工程應用經驗，並且具有一系列的可用性設計目標，包括達到 99.999% 或更高可用性的服務。AWS解決方案架構師 (SA) 可協助您針對可用性目標進行適當的設計。在設計程序的早期階段引入 AWS 可以提高我們協助您實現可用性目標的能力。規劃可用性不僅是在工作負載啟動之前進行。在您獲得營運經驗、從實際事件中獲得經驗以及經受各種類型的失敗後，也需持續執行此項工作以完善您的設計。然後，您可以進行適當的工作以改善您的實作。

工作負載所需的可用性需求必須與業務需求和關鍵性一致。先以定義的 RTO、RPO 和可用性定義業務關鍵性框架後，您之後便可評估各工作負載。諸如此類的方法規定，參與工作負載實作的人員需精通該架構，及其工作負載對業務需求的影響。

基礎

基礎要求是其範圍超過單一工作負載或專案的要求。在建立任何系統架構之前，應確立會影響可靠性的基本要求。例如，您必須為資料中心提供足夠的網路頻寬。

在內部部署環境中，這些要求可能因相依性導致延長交付時間，因此必須在初始規劃中納入這些需求。但藉助 AWS，其中的大多數基礎要求已予以納入或可能按需要經過處理。設計的雲端近乎無限，因此 AWS 有責任滿足足夠的聯網和運算容量的要求，讓您可以根據需要自由變更資源大小和分配。

以下數節說明著重於這些可靠性考量因素的最佳實務。

主題

- 管理服務配額和限制
- 規劃您的網路拓撲

管理服務配額和限制

雲端型工作負載架構具有服務配額 (也稱為服務限制)。這些配額的存在是為了防止意外佈建超出所需數量的資源，並限制 API 操作的請求率，以保護服務免於遭到濫用。另外還有一些資源限制，例如可將位元下推到光纖纜線的速率，或實體磁碟的儲存量。

如果您使用 AWS Marketplace 應用程式，則必須了解這些應用程式的限制。如果您使用第三方 Web 服務或軟體即服務，也必須了解這些限制。

最佳實務

- REL01-BP01 了解服務配額和限制
- REL01-BP02 管理跨帳戶和區域的服務配額
- REL01-BP03 透過架構適應固定服務配額和限制
- REL01-BP04 監控和管理配額
- REL01-BP05 自動化配額管理
- REL01-BP06 確保目前配額與最大用量之間存在足夠的間隙，以適應容錯移轉

REL01-BP01 了解服務配額和限制

了解工作負載架構的預設配額和管理配額增加要求。知道哪些雲端資源限制 (例如，磁碟或網路) 具有潛在影響。

預期結果：客戶可以 AWS 帳戶 實作適當的準則來監控關鍵指標、基礎設施檢閱和自動化修復步驟，以驗證未達到可能導致服務降級或中斷的服務配額和限制，從而防止服務降級或中斷。

常見的反模式：

- 在部署工作負載時，未了解軟、硬體配額及其對所用服務的限制。
- 部署替換工作負載時，未事先分析及重新設定必要的配額或聯絡支援人員。
- 假設雲端服務沒有限制，且在使用服務時無須考量費率、限制、計數和數量。
- 假設配額會自動增加。
- 不知道配額要求的程序和時間表。
- 假設每個服務在不同區域間的預設雲端服務配額都是相同的。
- 假設可以違反服務限制，且系統會將限制自動擴展或增加到資源限制以上
- 未以尖峰流量測試應用程式，以製造資源使用率的壓力。
- 佈建資源時未分析必要的資源大小。
- 選擇遠超出實際需求或預期尖峰的資源類型，而過度佈建容量。
- 未在新的客戶事件或部署新技術之前事先評估新流量層級的容量要求。

建立此最佳實務的優勢：監控和自動化服務配額和資源限制的管理可以主動減少故障。若未遵循最佳實務，客戶服務的流量模式變更即可能導致中斷或降級。藉由在所有區域和所有帳戶間監控並管理這些值，應用程式在遇到不良或非計畫性事件時將會有更高的彈性。

未建立此最佳實務時的曝險等級：高

實作指引

Service Quotas 是一種 AWS 服務，可協助您從一個位置管理超過 250 個 AWS 服務的配額。除了查詢配額值之外，您也可以從 Service Quotas 主控台或使用 AWS SDK. AWS Trusted Advisor offer 來請求和追蹤配額增加，以顯示某些服務某些方面的用量和配額。每個服務的預設服務配額也位於每個個別服務的 AWS 文件中 （例如，請參閱 Amazon VPC Quotas ）。

某些服務限制，例如節流的速率限制APIs，是透過設定用量計畫在 Amazon API Gateway 本身內設定。在其各自服務上設定為組態的一些限制包括佈建的 IOPS、已配置的 Amazon RDS儲存體和 Amazon EBS磁碟區配置。Amazon Elastic Compute Cloud 擁有自己的服務限制儀表板，有助於您管理執行個體、Amazon Elastic Block Store 和彈性 IP 位址限制。如果您有服務配額影響應用程式效能的使用案例，且無法根據您的需求調整，請聯絡 支援 ，查看是否有緩解措施。

服務配額可能隨著區域而不同，或本質上是通用的。使用達到配額 AWS 的服務在正常使用中不會如預期般運作，並可能導致服務中斷或降級。例如，服務配額會限制區域中使用的 DL Amazon EC2執行個體數量。流量擴展事件期間，可以使用 Auto Scaling 群組 （ ） 達到此限制ASG。

每個帳戶的服務配額均應定期受到用量評估，以確認該帳戶的適當服務限制為何。這些服務配額可作為操作上的防護機制，以防止不慎佈建超過您所需的資源。它們還用於限制API操作的請求率，以保護服務免受濫用。

服務限制與服務配額不同。服務限制代表特定資源的類型為該資源定義的限制。這些可能是儲存容量（例如，gp2 的大小限制為 1 GB - 16 TB） 或磁碟輸送量。制定資源類型的限制，並持續評估用量是否可能超出限制，是很重要的。若意外超出限制，帳戶的應用程式或服務可能會降級或中斷。

如果存在服務配額影響應用程式效能的使用案例，且無法根據所需需求調整，請聯絡 支援 ，了解是否有緩解措施。如需有關調整固定配額的詳細資訊，請參閱 REL01-BP03 透過架構適應固定服務配額和限制。

有許多 AWS 服務和工具可協助監控和管理 Service Quotas 您應利用這些服務和工具，以提供配額層級的自動或手動檢查。

- AWS Trusted Advisor 提供服務配額檢查，顯示某些 服務的某些方面的用量和配額。這有助於識別接近配額的服務。
- AWS Management Console 提供顯示服務配額值、管理、請求新配額、監控配額請求狀態和顯示配額歷史記錄的方法。
- AWS CLI 和 CDKs提供程式設計方法，以自動管理和監控服務配額層級和用量。

實作步驟

對於 Service Quotas：

- 檢閱 AWS Service Quotas。
- 若要了解現有的服務配額，請判斷所使用的服務 （例如 IAM Access Analyzer）。大約有 250 個 AWS 服務由服務配額控制。然後，確認每個帳戶和區域內可能使用的特定服務配額名稱。每個區域約有 3000 個服務配額名稱。
- 使用 增強此配額分析 AWS Config ，以尋找 中使用的所有AWS 資源 AWS 帳戶。
- 使用 AWS CloudFormation 資料來判斷所使用的 AWS 資源。查看在 中建立或使用 `list-stack-resources` AWS CLI 命令 AWS Management Console 建立的資源。您也可以查看設定為自行在範本中部署的資源。
- 透過查看部署程式碼來確定工作負載所需的所有服務。

- 確認適用的服務配額。使用來自 Trusted Advisor 和 Service Quotas 的可程式設計存取資訊。
- 建立自動監控方法 (請參閱 REL01-BP02 管理跨帳戶和區域的服務配額 和 REL01-BP04 監控和管理配額)，以警示並通知服務配額是否接近或已達到上限。
- 建立自動化和程式化方法，以檢查某個區域中的服務配額是否已變更，但在同一帳戶的其他區域中未已變更 (請參閱 REL01-BP02 管理跨帳戶和區域的服務配額 和 REL01-BP04 監控和管理配額)。
- 自動執行掃描應用程式日誌和指標，以確認是否有任何配額或服務限制錯誤。若有這類錯誤存在，請傳送提醒到監控系統。
- 一旦確定特定服務需要更大的配額，請建立工程程序，以計算配額中所需的變更 (請參閱 REL01-BP05 自動化配額管理)。
- 建立佈建和核准工作流程，以要求變更服務配額。其中應包含要求遭拒絕或部分核准時的例外狀況工作流程。
- 建立工程方法，以在佈建和使用新 AWS 服務之前檢閱服務配額，然後再推出生產或載入環境。（例如，負載測試帳戶）。

針對服務限制：

- 建立監控和指標方法，針對接近資源限制的資源讀數發出提醒。 CloudWatch 適當利用指標或日誌監控。
- 為每個具有有效應用程式或系統限制的資源建立提醒閾值。
- 建立工作流程和基礎設施管理程序，以在限制接近使用率時變更資源類型。此工作流程應將負載測試納入作為最佳實務，以確認新類型是具有新限制的正確資源類型。
- 使用現有的程序和流程，將已識別的資源遷移至建議的新資源類型。

資源

相關的最佳實務：

- REL01-BP02 管理跨帳戶和區域的服務配額
- REL01-BP03 透過架構適應固定服務配額和限制
- REL01-BP04 監控和管理配額
- REL01-BP05 自動化配額管理
- REL01-BP06 確保目前配額與最大用量之間存在足夠的間隙，以適應容錯移轉
- REL03-BP01 選擇如何分割工作負載
- REL10-BP01 將工作負載部署至多個位置

- REL11-BP01 監控工作負載的所有元件以偵測故障
- REL11-BP03 將所有分層的修復自動化
- REL12-BP04 使用混沌工程測試彈性

相關文件：

- AWS Well-Architected Framework 的可靠性支柱：可用性
- AWS Service Quotas（先前稱為服務限制）
- AWS Trusted Advisor 最佳實務檢查 （請參閱服務限制區段）
- AWS 限制對 AWS 答案的監控
- Amazon EC2 Service 限制
- 什麼是 Service Quotas？
- 如何請求提高配額
- 服務端點和配額
- Service Quotas 使用者指南
- 的配額監控 AWS
- AWS 故障隔離界限
- 備援的可用性
- AWS 適用於資料
- 什麼是持續整合？
- 什麼是持續交付？
- APN 合作夥伴：可協助進行組態管理的合作夥伴
- 在 上管理 account-per-tenant SaaS 環境中的帳戶生命週期 AWS
- 管理和監控工作負載中的API限流
- 使用 大規模檢視 AWS Trusted Advisor 建議 AWS Organizations
- 使用 自動化服務限制增加和企業支援 AWS Control Tower

相關影片：

- AWS Live re：Inforce 2019 - Service Quotas
- 使用 AWS 服務配額檢視和管理Service Quotas

- AWS IAM Quotas 示範

相關工具：

- Amazon CodeGuru Reviewer
- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOpsGuru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP02 管理跨帳戶和區域的服務配額

如果您使用多個帳戶或區域，請在生產工作負載執行的所有環境中都要求合適的配額。

預期成果：對於跨帳戶或區域的組態，或使用地區、區域或帳戶容錯移轉具有彈性設計的組態，服務和應用程式不應受到服務配額耗盡的影響。

常見的反模式：

- 允許一個隔離區域內的資源用量增長，但無維持其他隔離區域中容量的機制。
- 在隔離區域中單獨手動設定所有配額。
- 未考量彈性架構 (例如主動或被動) 日後在非主要區域降級期間對配額需求產生的影響。
- 未定期評估配額，並在工作負載執行所在的每個區域和帳戶中進行必要的變更。
- 不要利用配額請求範本在多個區域和帳戶之間請求增加。
- 因誤認為增加配額會產生成本上的影響 (例如運算保留要求) 而未更新服務配額。

建立此最佳實務的優勢：確認如果區域服務無法使用時，您可以處理次要區域或帳戶中目前的負載。這有助於降低區域中斷期間發生的錯誤數量或降級程度。

未建立此最佳實務時的曝險等級：高

實作指引

系統會針對每個帳戶追蹤服務配額。除非另有說明，否則每個配額都是 AWS 區域特定的。除生產環境之外，也會在所有適用的非生產環境中管理配額，因此不會阻礙測試和開發。要維持高水準的彈性，必須持續評估服務配額 (無論自動還是手動)。

由於採用主動/主動、主動/被動 - 熱、主動/被動 - 冷，以及主動/被動- 指示燈方法實作設計，跨區域的工作負載越來越多，因此了解所有區域和帳戶配額級別至關重要。過去的流量模式不一定可明確指出服務配額是否正確設定。

同樣重要的是，每個區域的服務配額名稱限制不一定相同。在某個區域中，該值可能是五，而另一個區域中的值可能是十。這些配額的管理必須跨所有的相同服務、帳戶和區域，以在負載下提供一致的彈性。

在不同區域 (主動區域或被動區域) 間協調所有服務配額差異，並建立持續協調這類差異的程序。被動區域容錯移轉的測試計畫鮮少擴展至尖峰主動容量，意即演練日或桌面演練可能找不到區域之間的服務配額差異，因而無法維持正確的限制。

服務配額漂移是指特定指定配額的服務配額限制在一個區域而非所有區域中發生變更的情況，這對於追蹤和評估非常重要。您應考慮在具有流量甚或雲端承載流量的區域中變更配額。

- 根據您的服務要求、延遲、法規和災難復原 (DR) 要求，選取相關的帳戶和區域。
- 確定所有相關帳戶、區域和可用區域中的服務配額。限制範圍受限於帳戶和區域。您應比較這些值的差異。

實作步驟

- 審查可能超出使用風險等級的 Service Quotas 值。超出 80% 和 90% 閾值時， AWS Trusted Advisor 會提供提醒。
- 審查任何被動區域 (主動/被動設計中) 的服務配額值。確認在主要區域失敗時，負載將可在次要區域中成功執行。
- 自動評估相同帳戶中的區域之間是否發生了任何服務配額漂移，並採取因應措施以變更限制。
- 如果客戶的組織單位 (OU) 是以支援的方式建構的，則應更新服務配額範本，以反映應套用至多個區域和帳戶的任何配額中的變更。
  - 建立範本，並將區域關聯至配額變更。
  - 審查所有現有的服務配額範本，確認是否有任何必要的變更 (區域、限制和帳戶)。

資源

相關的最佳實務：

- REL01-BP01 了解服務配額和限制
- REL01-BP03 透過架構適應固定服務配額和限制
- REL01-BP04 監控和管理配額
- REL01-BP05 自動化配額管理
- REL01-BP06 確保目前配額與最大用量之間存在足夠的間隙，以適應容錯移轉
- REL03-BP01 選擇如何分割工作負載
- REL10-BP01 將工作負載部署至多個位置
- REL11-BP01 監控工作負載的所有元件以偵測故障
- REL11-BP03 將所有分層的修復自動化
- REL12-BP04 使用混沌工程測試彈性

相關文件：

- AWS Well-Architected Framework 的可靠性支柱：可用性
- AWS Service Quotas（先前稱為服務限制）
- AWS Trusted Advisor 最佳實務檢查 （請參閱服務限制區段）
- AWS 限制對 AWS 答案的監控
- Amazon EC2 Service 限制
- 什麼是 Service Quotas？
- 如何請求提高配額
- 服務端點和配額
- Service Quotas 使用者指南
- 的配額監控 AWS
- AWS 故障隔離界限
- 備援的可用性
- AWS 適用於資料
- 什麼是持續整合？
- 什麼是持續交付？

- APN 合作夥伴：可協助進行組態管理的合作夥伴
- 在 上管理 account-per-tenant SaaS 環境中的帳戶生命週期 AWS
- 管理和監控工作負載中的API限流
- 使用 大規模檢視 AWS Trusted Advisor 建議 AWS Organizations
- 使用 自動化服務限制增加和企業支援 AWS Control Tower

相關影片：

- AWS Live re：Inforce 2019 – Service Quotas
- 使用 AWS 服務配額檢視和管理Service Quotas
- AWS IAM Quotas 示範

相關服務：

- Amazon CodeGuru Reviewer
- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOpsGuru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP03 透過架構適應固定服務配額和限制

請注意不可變更的服務配額、服務限制和實際資源限制。設計應用程式和服務的架構以防止這些限制影響可靠性。

範例包括 API 閘道的網路頻寬、無伺服器函數叫用承載大小、調節爆量速率，以及資料庫的使用者同時連線數目。

預期成果：在正常和高流量條件下，應用程式或服務會如預期般執行。它們已設計為在該資源的固定限制或服務配額內運作。

常見的反模式：

- 選擇使用一項服務的一項資源的設計，但未注意到擴展時會導致此項設計失效的設計限制。
- 執行不切實際的基準並且在測試期間達到服務固定配額。例如，以爆量限制執行測試，但是進行擴充的時間量。
- 選擇若超過固定服務配額時無法擴展或修改的設計。例如，SQS 承載大小為 256KB。
- 未設計可觀測性並且實作以監控和提醒在高流量活動期間可能有風險之服務配額的閾值

建立此最佳實務的優勢：確認應用程式是否會在所有預計的服務負載層級下執行，而不會中斷或降級。

未建立此最佳實務時的曝險等級：中

實作指引

不同於以較高容量單位取代的軟性服務配額，AWS 服務的固定配額無法變更。這表示這裡所有類型的 AWS 服務都必須在使用於應用程式設計中時評估其潛在硬性容量限制。

硬性限制會顯示在 Service Quotas 主控台中。如果資料欄顯示 `ADJUSTABLE = No`，則服務有硬性限制。硬性限制也會顯示在一些資源組態頁面中。例如，Lambda 有無法調整的特定硬性限制。

例如，設計 Python 應用程式在 Lambda 函數中執行時，應用程式應該評估以判斷 Lambda 是否有機會執行超過 15 分鐘。如果程式碼可能執行超過此服務配額限制，則必須考慮替代技術或設計。如果在生產部署後達到此限制，應用程式會遭受降級和中斷直到可以矯正為止。與軟性配額不同，沒有任何方法可以變更這些限制，即使是在緊急嚴重性 1 活動下。

一旦應用程式部署到測試環境，應該使用策略來尋找是否達到任何硬性限制。壓力測試、負載測試和混亂測試應該是引入測試計劃的一部分。

實作步驟

- 審核可用於應用程式設計階段的 AWS 服務完整清單。
- 審核這些服務的軟性配額限制和硬性配額限制。並非所有限制都會顯示在 Service Quotas 主控台中。有些服務會在替代位置描述這些限制。
- 隨著您設計您的應用程式，審核您的工作負載的業務和技術驅動來源，例如業務成果、使用案例、相依系統、可用性目標和災難復原物件。讓您的業務和技術驅動來源引導程序以識別適合您的工作負載的分散式系統。

- 分析區域和帳戶之間的服務負載。許多硬性限制對於服務是區域型的。不過，某些限制是帳戶型。
- 分析區域 (Zonal) 失敗和區域 (Regional) 失敗期間資源用量的彈性架構。在使用主動/主動、主動/被動 – 熱、主動/被動 - 冷和主動/被動 - 指示燈方法的多區預設定進度中，這些失敗案例會導致較高的用量。這會建立達到硬性限制的潛在使用案例。

資源

相關的最佳實務：

- REL01-BP01 了解服務配額和限制
- REL01-BP02 管理跨帳戶和區域的服務配額
- REL01-BP04 監控和管理配額
- REL01-BP05 自動化配額管理
- REL01-BP06 確保目前配額與最大用量之間存在足夠的間隙，以適應容錯移轉
- REL03-BP01 選擇如何分割工作負載
- REL10-BP01 將工作負載部署至多個位置
- REL11-BP01 監控工作負載的所有元件以偵測故障
- REL11-BP03 將所有分層的修復自動化
- REL12-BP04 使用混沌工程測試彈性

相關文件：

- AWS Well-Architected Framework 的可靠性支柱：可用性
- AWS Service Quotas (先前稱為 Service Limits)
- AWS Trusted Advisor 最佳實務檢查 (請參閱 Service Limits 章節)
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 Service Limits
- 什麼是 Service Quotas？
- 如何請求提高配額
- 服務端點和配額
- Service Quotas 使用者指南
- Quota Monitor for AWS

- AWS 故障隔離界限
- 備援的可用性
- AWS for Data
- 什麼是持續整合？
- 什麼是持續交付？
- APN 合作夥伴：可以幫助進行組態管理的合作夥伴
- 管理 AWS 上每租用戶帳戶 SaaS 環境中的帳戶生命週期
- 管理和監控工作負載中的 API 限流
- 使用 AWS Organizations 大規模檢視 AWS Trusted Advisor 建議
- 使用 AWS Control Tower 自動提高服務限制和企業支援
- Service Quotas 的動作、資源和條件金鑰

相關影片：

- AWS Live re:Inforce 2019 - Service Quotas
- 使用 Service Quotas 檢視和管理 AWS 服務的配額
- AWS IAM 配額示範
- AWS re:Invent 2018：閉環與開放思維：如何取得大小型系統的控制權

相關工具：

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP04 監控和管理配額

評估潛在用量並適當地增加配額，以允許使用量按計劃增長。

預期成果：已經部署管理和監控的主動和自動化系統。這些操作解決方案可確保幾乎達到配額用量閾值。這些會由請求配額變更主動矯正。

常見的反模式：

- 未設定監控來檢查服務配額閾值
- 未設定監控硬性限制，即使這些值無法變更。
- 假設請求和保護軟性配額變更所需的時間量是立即或短期間。
- 設定了正在接近服務配額的警示，但無如何回應提醒的程序。
- 僅設定 AWS Service Quotas 所支援之服務的警示，而不監控其他服務 AWS 。
- 未考慮多個區域彈性設計的配額管理，例如主動/主動、主動/被動 – 熱、主動/被動 - 冷和主動/被動 - 指示燈方法。
- 未評估區域之間的配額差異。
- 未評估每個區域特定配額增加請求的需求。
- 不利用範本進行多區域配額管理。

建立此最佳實務的優點：自動追蹤 AWS Service Quotas，並根據這些配額監控用量，可讓您在接近配額限制時查看 。您也可以使用此監控資料來協助限制由於配額耗盡造成的任何降級。

未建立此最佳實務時的曝險等級：中

實作指引

針對支援的服務，您可以藉由設定可評估然後傳送提醒或警示的各種不同服務，來監控您的配額。這可協助監控用量並且可以在您接近配額時提醒您。這些警示可以從 AWS Config、Lambda 函數 CloudWatch、Amazon 或 從 叫用 AWS Trusted Advisor。您也可以在 CloudWatch 日誌上使用指標篩選條件來搜尋和擷取日誌中的模式，以判斷用量是否接近配額閾值。

實作步驟

針對監控：

- 擷取當前資源消耗 (例如，儲存貯體或執行個體)。使用 服務API操作，例如 Amazon EC2 `DescribeInstances` API來收集目前的資源消耗。

- 使用以下項目，擷取您目前基本且適用於服務的配額：
  - AWS Service Quotas
  - AWS Trusted Advisor
  - AWS 文件
  - AWS 服務特定頁面
  - AWS Command Line Interface (AWS CLI)
  - AWS Cloud Development Kit (AWS CDK)
- 使用 AWS Service Quotas ，這項 AWS 服務可協助您從單一位置管理超過 250 個 AWS 服務的配額。
- 使用 Trusted Advisor 服務限制來監控您目前在各種閾值的服務限制。
- 使用服務配額歷史記錄（主控台或 AWS CLI）來檢查區域增加。
- 比較每個區域和每個帳戶中的服務配額變更，視需要建立等值。

針對管理：

- 自動化：設定 AWS Config 自訂規則以跨區域掃描服務配額，並比較差異。
- 自動化：設定排定的 Lambda 函數來掃描區域之間的服務配額，並且比較是否有差異。
- 手動：透過 AWS CLI、 API或 AWS 主控台掃描服務配額，以跨區域掃描服務配額並比較差異。報告差異。
- 如果區域之間識別出配額的差異，請視需要請求配額變更。
- 檢閱所有請求的結果。

資源

相關的最佳實務：

- REL01-BP01 了解服務配額和限制
- REL01-BP02 管理跨帳戶和區域的服務配額
- REL01-BP03 透過架構適應固定服務配額和限制
- REL01-BP05 自動化配額管理
- REL01-BP06 確保目前配額與最大用量之間存在足夠的間隙，以適應容錯移轉
- REL03-BP01 選擇如何分割工作負載
- REL10-BP01 將工作負載部署至多個位置

- REL11-BP01 監控工作負載的所有元件以偵測故障
- REL11-BP03 將所有分層的修復自動化
- REL12-BP04 使用混沌工程測試彈性

相關文件：

- AWS Well-Architected Framework 的可靠性支柱：可用性
- AWS Service Quotas（先前稱為服務限制）
- AWS Trusted Advisor 最佳實務檢查 （請參閱服務限制區段）
- AWS 限制對 AWS 答案的監控
- Amazon EC2 Service 限制
- 什麼是 Service Quotas？
- 如何請求提高配額
- 服務端點和配額
- Service Quotas 使用者指南
- 的配額監控 AWS
- AWS 故障隔離界限
- 備援的可用性
- AWS 適用於資料
- 什麼是持續整合？
- 什麼是持續交付？
- APN 合作夥伴：可協助進行組態管理的合作夥伴
- 在 上管理 account-per-tenant SaaS 環境中的帳戶生命週期 AWS
- 管理和監控工作負載中的API限流
- 使用 大規模檢視 AWS Trusted Advisor 建議 AWS Organizations
- 使用 自動化服務限制增加和企業支援 AWS Control Tower
- Service Quotas 的動作、資源和條件金鑰

相關影片：

- AWS Live re：Inforce 2019 - Service Quotas
- 使用 AWS 服務配額檢視和管理Service Quotas

- AWS IAM Quotas 示範
- AWS re：Invent 2018：閉環和開場思維：如何控制大大小小的系統

相關工具：

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOpsGuru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

REL01-BP05 自動化配額管理

服務配額也稱為 AWS 服務中的限制，是您的 AWS 帳戶 中資源的最大值。每一項 AWS 服務都會定義一組配額及其預設值。若要讓您的工作負載存取所需的所有資源，您可能需要提高服務配額值。

若工作負載耗用的 AWS 資源增加，則可能在超出配額時威脅到工作負載的穩定性，並且影響使用者體驗。實作工具以在工作負載接近限制時提醒您，並考慮自動建立配額增加請求。

預期成果：每個 AWS 帳戶 和區域中執行的工作負載都設定了適當的配額。

常見的反模式：

- 您未能考慮並適當地調整配額以符合工作負載需求。
- 您用來追蹤配額和用量的方法可能過舊，例如使用試算表。
- 您僅依照定期排程更新服務限制。
- 您的組織缺少審查現有配額，以及在必要時請求增加服務配額的操作流程。

建立此最佳實務的優勢：

- 更富彈性的工作負載：您可避免超過 AWS 資源配額所造成的錯誤。
- 更簡單的災難復原：您在另一個 AWS 區域 中設定 DR 時，可重複使用在主要區域中建置的自動配額管理機制。

未建立此最佳實務時的曝險等級：中

實作指引

透過 AWS Service Quotas 主控台、AWS Command Line Interface (AWS CLI) 和 AWS SDK 等機制，檢視目前的配額並追蹤持續的配額使用量。您還可以將組態管理資料庫 (CMDB) 和 IT 服務管理 (ITSM) 系統與 AWS Service Quota API 整合。

如果配額用量達到定義的閾值，則產生自動提醒，並定義在收到提醒時提交配額增加請求的流程。如果基礎工作負載對您的業務至關重要，您可以自動化配額增加請求，但請仔細測試自動化程序，以避免增長回饋迴圈這類失控動作的風險。

通常會自動核准增加較少量的配額。較大的配額請求可能需要經過 AWS 支援人員手動處理，而且可能需要額外的時間來審查和處理。請預留額外的時間來處理多個請求或大量增加請求。

實作步驟

- 實作自動監控服務配額，並且在工作負載的資源使用率增長接近配額限制時發出警示。例如，適用於 AWS 的配額監控可自動監控服務配額。此工具會使用 Cloudformation StackSets 與 AWS Organizations 整合並進行部署，如此新帳戶就會在建立時自動受到監控。
- 使用 Service Quotas 請求範本或 AWS Control Tower 等功能簡化新帳戶的 Service Quotas 設定。
- 在所有 AWS 帳戶 和區域建置目前服務配額使用情形的儀表板，並視需要參考它們以避免超出配額。Trusted Advisor Organizational (TAO) 儀表板是 Cloud Intelligence 儀表板的一部分，可讓您快速開始使用此類儀表板。
- 追蹤服務限制增加請求。Consolidated Insights from Multiple Accounts (CIMA) 可提供所有請求的組織層級檢視內容。
- 透過在非實際執行帳戶中設定較低的配額閾值，以測試警示產生和任何配額增加請求的自動化程序。請勿在實際執行帳戶中進行這些測試。

資源

相關的最佳實務：

- OPS10-BP07 自動回應事件

相關文件：

- APN 合作夥伴：可以幫助進行組態管理的合作夥伴
- AWS Marketplace：可追蹤限制的 CMDB 產品
- AWS Service Quotas (先前稱為 Service Limits)
- AWS Trusted Advisor 最佳實務檢查 (請參閱 Service Limits 章節)
- AWS 上的配額監控解決方案 - AWS 解決方案
- 什麼是 Service Quotas？
- 什麼是 Service Quotas 請求範本？

相關影片：

- AWS Live re:Inforce 2019 - Service Quotas
- 使用 AWS Control Tower 自動提高服務限制和提供企業支援

相關工具：

- 適用於 AWS 的配額監控

REL01-BP06 確保目前配額與最大用量之間存在足夠的間隙，以適應容錯移轉

本文說明如何維護資源配額與使用量之間的空間，以及如何讓您的組織受益。在完成使用資源之後，使用量配額可能會繼續佔用該資源。這可能會導致資源失敗或無法存取。透過確認您的配額是否涵蓋無法存取資源及其替換項目的重疊，來防止資源失敗。計算此差距時，應考慮諸如網路失敗、可用區域失敗或區域失敗等案例。

預期成果：資源或資源可存取性中的小型或大型故障可涵蓋在目前的服務閾值內。已在資源規劃中考慮區域 (Zone) 失敗、網路失敗或甚至是區域 (Regional) 失敗。

常見的反模式：

- 根據目前的需求設定服務配額，而不考慮容錯移轉案例。
- 計算服務的尖峰配額時，未考慮靜態穩定性的主體。
- 計算每個區域所需的配額總計時，未考慮可能有無法存取的資源。

- 不考慮某些 AWS 服務的服務故障隔離界限及其潛在的異常使用模式。

建立此最佳實務的優勢：當服務中斷事件影響應用程式可用性時，請使用雲端來實作策略，以便從這些事件中復原。一個範例策略是建立額外的資源來取代無法存取的資源，以適應容錯移轉條件，而不會耗盡您的服務限制。

未建立此最佳實務時的曝險等級：中

實作指引

評估配額限制時，請考慮由於某些降級而可能發生的容錯移轉案例。請考慮下列容錯移轉情況。

- 已中斷或無法存取的 VPC。
- 無法存取的子網路。
- 影響資源可存取性的降級可用區域。
- 聯網路由或輸入和輸出點遭到封鎖或變更。
- 影響資源可存取性的降級區域。
- 受區域或可用區域中的失敗所影響的資源子集。

容錯移轉的決策對於每個情況都是獨一無二的，因為業務影響有所不同。在決定容錯移轉應用程式或服務之前，先處理容錯移轉位置中的資源容量規劃和資源的配額。

檢閱每個服務的配額時，請考慮高於正常的活動峰值。這些峰值可能與由於聯網或權限而無法存取但仍處於活動狀態的的資源相關。未終止的作用中資源會計入服務配額限制。

實作步驟

- 維持服務配額和最大用量之間的空間，以適應容錯移轉或可存取性的喪失。
- 確定服務配額。說明典型的部署模式、可用性需求和使用量增長。
- 視需要請求增加配額。預計配額增加請求的等待時間。
- 確定您的可靠性需求 (也稱為「幾個 9」)。
- 了解可能的故障案例，例如元件遺失、可用區域或區域。
- 建立您的部署方法 (範例包括 Canary、藍/綠、紅/黑或滾動)。
- 為當前配額限制新增適當的緩衝。範例緩衝為 15%。
- 適當時包含靜態穩定性的計算 (區域 (Zonal) 和區域 (Regional))。

- 規劃使用量增長並監控使用量趨勢。
- 考慮最關鍵工作負載的靜態穩定性影響。評估符合所有區域和可用區域中靜態穩定系統的資源。
- 考慮使用隨需容量保留，在任何容錯移轉之前排程容量。這是針對關鍵業務排程而實作的有用策略，可以降低在容錯移轉期間取得正確數量和資源類型的潛在風險。

資源

相關的最佳實務：

- REL01-BP01 了解服務配額和限制
- REL01-BP02 管理跨帳戶和區域的服務配額
- REL01-BP03 透過架構適應固定服務配額和限制
- REL01-BP04 監控和管理配額
- REL01-BP05 自動化配額管理
- REL03-BP01 選擇如何分割工作負載
- REL10-BP01 將工作負載部署至多個位置
- REL11-BP01 監控工作負載的所有元件以偵測故障
- REL11-BP03 將所有分層的修復自動化
- REL12-BP04 使用混沌工程測試彈性

相關文件：

- AWS Well-Architected Framework 的可靠性支柱：可用性
- AWS Service Quotas（先前稱為服務限制）
- AWS Trusted Advisor 最佳實務檢查 （請參閱服務限制區段）
- AWS 限制對 AWS 答案的監控
- Amazon EC2 Service 限制
- 什麼是 Service Quotas？
- 如何請求提高配額
- 服務端點和配額
- Service Quotas 使用者指南
- 的配額監控 AWS

- AWS 故障隔離界限
- 備援的可用性
- AWS 適用於資料
- 什麼是持續整合？
- 什麼是持續交付？
- APN 合作夥伴：可協助進行組態管理的合作夥伴
- 在 上管理 account-per-tenant SaaS 環境中的帳戶生命週期 AWS
- 管理和監控工作負載中的API限流
- 使用 大規模檢視 AWS Trusted Advisor 建議 AWS Organizations
- 使用 自動化服務限制增加和企業支援 AWS Control Tower
- Service Quotas 的動作、資源和條件金鑰

相關影片：

- AWS Live re：Inforce 2019 – Service Quotas
- 使用 AWS 服務配額檢視和管理Service Quotas
- AWS IAM Quotas 示範
- AWS re：Invent 2018：閉環和開場思維：如何控制大大小小的系統

相關工具：

- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOpsGuru
- AWS Config
- AWS Trusted Advisor
- AWS CDK
- AWS Systems Manager
- AWS Marketplace

規劃您的網路拓撲

工作負載通常存在於多個環境中。其中包括多個雲端環境 (可公開存取與私有)，也可能包含您現有的資料中心基礎設施。計畫必須包括系統內和系統間連接、公有 IP 位址管理、私有 IP 位址管理和網域名稱解析等網路考量因素。

使用基於 IP 位址的網路建立系統架構時，您必須規劃網路拓樸，以預期可能的失敗並適應未來成長，以及與其他系統及其網路整合。

Amazon Virtual Private Cloud (Amazon VPC) 可讓您佈建私有、隔離的 AWS 雲端部分，而您可以在該部分透過虛擬網路啟動 AWS 資源。

最佳實務

- REL02-BP01 針對工作負載公有端點使用高可用性網路連線
- REL02-BP02 佈建雲端和內部部署環境中私有網路之間的備援連線
- REL02-BP03 確保 IP 子網路配置考量擴充性和可用性
- REL02-BP04 偏好軸輻式拓撲而非多對多網狀拓撲
- REL02-BP05 在所有連線的私有地址空間中強制執行不重疊的私有 IP 地址範圍

REL02-BP01 針對工作負載公有端點使用高可用性網路連線

建置與您的工作負載公有端點的高度可用網路連線，可協助您減少由於遺失連線的停機時間，並且改善您的工作負載的可用性和 SLA。為達成此目的，請使用高度可用的 DNS、內容交付網路 (CDN)、API 閘道、負載平衡或反向代理。

預期成果：為您的公用端點規劃、建置和操作高可用性網路連線至關重要。如果您的工作負載由於遺失連線而無法連線，即使您的工作負載正在執行且可用，您的客戶還是會看到您的系統是停機。藉由結合工作負載公有端點高度可用和具彈性的網路連線與工作負載本身的彈性架構，為您的客戶提供可行的最佳可用性和服務水準。

AWS Global Accelerator、Amazon CloudFront、Amazon API Gateway、AWS Lambda Function URL、AWS AppSync API 以及 Elastic Load Balancing (ELB) 全都可提供高可用性公用端點。Amazon Route 53 針對網域名稱解析提供高可用性的 DNS 服務，以確認您的公有端點地址是否可以解析。

您也可以評估用於負載平衡和代理的 AWS Marketplace 軟體設備。

常見的反模式：

- 設計高度可用的工作負載，而未規劃 DNS 和網路連線以取得高可用性。
- 在個別執行個體或容器上使用公有網際網路地址，並透過 DNS 管理其連線。
- 使用 IP 位址，而非網域名稱來定位服務。
- 未測試您的公有端點已遺失連線的情境。
- 未分析網路輸送量需求和分發模式。
- 未測試和規劃您的工作負載公有端點的網際網路網路連線可能遭到中斷的情境。
- 提供內容 (例如網頁、靜態資產或媒體檔案) 到大型地理區域，而不使用內容交付網路。
- 未針對分散式阻斷服務 (DDoS) 攻擊加以規劃。DDoS 攻擊所存在的風險會將合法流量阻擋在外，並減少使用者的可用性。

建立此最佳實務的優勢：針對高可用性和高彈性的網路連線進行設計，確保您的工作負載可供使用者存取且可用。

未建立此最佳實務時的曝險等級：高

實作指引

建置與您的公有端點的高度可用網路連線的核心是流量的路由。若要確認您的流量可以連線到端點，DNS 必須能夠將網域名稱解析為它們的對應 IP 位址。使用 Amazon Route 53 等高可用且可擴展的網域名稱系統 (DNS) 來管理網域的 DNS 記錄。也可以使用 Amazon Route 53 提供的運作狀態檢查。運作狀態檢查會確認您的應用程式可連線、可用並且可運作，它們可以透過模仿您的使用者行為的方式進行設定，例如請求網頁或特定 URL。發生失敗時，Amazon Route 53 會回應 DNS 解析請求，並且僅將流量導向到健康的端點。您也可以考慮使用 Amazon Route 53 提供的 Geo DNS 和以延遲為基礎的路由功能。

若要確認您的工作負載本身具有高可用性，請使用 Elastic Load Balancing (ELB)。Amazon Route 53 可用於將流量鎖定到 ELB，這會將流量分配到目標運算執行個體。也可以將 Amazon API Gateway 以及 AWS Lambda 一起用於無伺服器解決方案。客戶也可以在多個 AWS 區域 中執行工作負載。透過多站台主動/主動模式，工作負載可以為來自多個區域的流量提供服務。使用多站台主動/被動模式時，工作負載可為來自主動區域的流量提供服務，同時將資料複製到次要區域，並在主要區域發生故障時變為作用中狀態。然後，Route 53 運作狀態檢查可用於控制從主要區域中任何端點到次要區域中的某個端點的 DNS 備援，從而確認您的工作負載是否可存取並可供使用者使用。

Amazon CloudFront 藉由使用全世界邊緣節點的網路為請求提供服務，以低延遲和高資料傳輸率提供簡易 API 來分發內容。內容交付網路 (CDN) 藉由在靠近使用者的位置提供放置或快取的內容來服務客戶。當內容負載從您的伺服器轉移到 CloudFront 的邊緣節點時，這也會改善應用程式的可用性。邊緣

節點和區域邊緣快取會將您的內容快取複本保存在靠近您觀眾的位置，以便快速擷取並且增加您的工作負載的連線能力和可用性。

針對具有分散各地使用者的工作負載，AWS Global Accelerator 可協助您改善應用程式的可用性和效能。AWS Global Accelerator 提供任播靜態 IP 位址，可做為一或多個 AWS 區域 中託管之應用程式的固定進入點。這可讓流量盡可能輸入到使用者附近的 AWS 全球網路，改善您的工作負載的連線能力和可用性。AWS Global Accelerator 也會使用 TCP、HTTP 和 HTTPS 運作狀態檢查來監控您的應用程式端點的運作狀態。您的端點的運作狀態或組態的任何變更都允許將使用者流量重新導向到健康的端點，為您的使用者交付最佳效能和可用性。此外，AWS Global Accelerator 有故障隔離設計，使用由獨立網路區域提供服務的兩個靜態 IPv4 地址，增加您的應用程式的可用性。

為了幫助保護客戶免受 DDoS 攻擊，AWS 會提供 AWS Shield Standard。Shield Standard 會自動開啟並防止一般基礎架構 (第 3 層和第 4 層) 攻擊，例如 SYN/UDP 泛洪和反射攻擊，以支援 AWS 上應用程式的高可用性。針對更複雜和更大型攻擊 (例如 UDP 泛洪)、狀態耗盡攻擊 (例如 TCP SYN 泛洪) 的額外保護，以及協助保護您的應用程式在 Amazon Elastic Compute Cloud (Amazon EC2)、Elastic Load Balancing (ELB)、Amazon CloudFront、AWS Global Accelerator 以及 Route 53 上執行，您可以考慮使用 AWS Shield Advanced。針對應用程式層攻擊的保護，例如 HTTP POST 或 GET 泛洪，請使用 AWS WAF。AWS WAF 可以使用 IP 位址、HTTP 標題、HTTP 本文、URI 字串、SQL injection 隱碼攻擊和跨網站指令碼條件來判斷應該封鎖或允許請求。

實作步驟

1. 設定高可用 DNS：Amazon Route 53 是一種可用性高、可擴展性強的網域名稱系統 (DNS) Web 服務。Route 53 會將使用者請求連接至在 AWS 上或內部部署中執行的網際網路應用程式。如需詳細資訊，請參閱 configuring Amazon Route 53 as your DNS service。
2. 設定運作狀態檢查：當使用 Route 53 時，請確認只有運作狀態良好的目標是可解析的。首先建立 Amazon Route 53 運作狀態檢查和設定 DNS 備援。以下是設定運作狀態檢查時要考慮的重要層面：
   a. Amazon Route 53 決定運作狀態檢查是否良好的方式
   b. 建立、更新和刪除運作狀態檢查
   c. 監控運作狀態檢查狀態和取得通知
   d. Amazon Route 53 DNS 的最佳實務
3. 將您的 DNS 服務連接到端點。
   a. 使用 Elastic Load Balancing 作為流量的目標時，請使用指向負載平衡器區域端點的 Amazon Route 53 建立別名記錄。建立別名記錄期間，將 [評估目標運作狀態] 選項設定為 [是]。
   b. 對於使用 API Gateway 時的無伺服器工作負載或私有 API，請使用 Route 53 將流量導向至 API Gateway。

4. 決定內容交付網路。
   a. 若要使用更接近使用者的邊緣節點交付內容，請先了解 CloudFront 如何交付內容。
   b. 簡單的 CloudFront 分佈入門。CloudFront 接著會知道您想要從哪裡交付內容，以及如何追蹤和管理內容交付的詳細資料。以下是設定 CloudFront 分發時要了解和考慮的重要層面：
      i. 快取如何與 CloudFront 邊緣節點搭配運作
      ii. 增加從 CloudFront 快取直接提供的請求比例 (快取命中率)
      iii. 使用 Amazon CloudFront Origin Shield
      iv. 透過 CloudFront 原始伺服器容錯移轉將高可用性最佳化
5. 設定應用程式層保護：AWS WAF 可協助您保護免受常見 Web 漏洞和機器人的攻擊，這些攻擊會影響可用性、危及安全性或導致消耗過多資源。若要深入了解，請審核 how AWS WAF works，當您準備實作來自應用程式層 HTTP POST AND GET 泛洪的保護時，請參閱 Getting started with AWS WAF。也可以搭配使用 AWS WAF 與 CloudFront，請參閱有關 how AWS WAF works with Amazon CloudFront features 的文件。
6. 設定額外 DDoS 保護：根據預設，所有 AWS 客戶都能透過 AWS Shield Standard 獲得以您的網站或應用程式為目標，對於常見、最常發生網路和傳輸層 DDoS 攻擊的保護，不需額外費用。如需有關在 Amazon EC2、Elastic Load Balancing、Amazon CloudFront、AWS Global Accelerator 以及 Amazon Route 53 上執行的面向網際網路應用程式的額外保護，您可以考慮 AWS Shield Advanced 並審核 DDoS 彈性架構的範例。若要保護您的工作負載和公有端點免受 DDoS 攻擊，請參閱 Getting started with AWS Shield Advanced。

資源

相關的最佳實務：

- REL10-BP01 將工作負載部署至多個位置
- REL11-BP04 在復原期間依賴資料平面而非控制平面
- REL11-BP06 當事件影響可用性時傳送通知

相關文件：

- APN 合作夥伴：可以幫助您規劃聯網的合作夥伴
- 適用於網路基礎設施的 AWS Marketplace
- 什麼是 AWS Global Accelerator？
- 什麼是 Amazon CloudFront？

- 什麼是 Amazon Route 53？
- 什麼是 Elastic Load Balancing？
- 網路連線能力 - 建立您的雲端基礎
- 什麼是 Amazon API Gateway？
- 什麼是 AWS WAF、AWS Shield 和 AWS Firewall Manager？
- 什麼是 Amazon 應用程式復原控制器？
- 設定 DNS 備援的自訂運作狀態檢查

相關影片：

- AWS re:Invent 2022 - 使用 AWS Global Accelerator 改善效能與可用性
- AWS re:Invent 2020：使用 Amazon Route 53 進行全球流量管理
- AWS re:Invent 2022 - 操作高可用性多可用區域應用程式
- AWS re:Invent 2022 – 深入了解 AWS 聯網基礎設施
- AWS re:Invent 2022 - 建置彈性網路

相關範例：

- 使用 Amazon 應用程式復原控制器 (ARC) 進行災難復原
- AWS Global Accelerator 研討會

REL02-BP02 佈建雲端和內部部署環境中私有網路之間的備援連線

在雲端和內部部署環境中的私有網路之間的連線中實作備援，以實現連線恢復能力。這可以透過部署兩個或多個連結和流量路徑來實現，從而在發生網路故障時保持連接。

常見的反模式：

- 您只依賴一個網路連線，這會產生單點故障。
- 您只能使用在相同可用區域中結束的一個VPN通道或多個通道。
- 您依賴一個 ISP 進行VPN連線，這可能會導致ISP中斷期間完全失敗。
- 未實作動態路由通訊協定，例如 BGP，這些通訊協定對於網路中斷期間重新路由流量至關重要。
- 您可以忽略VPN通道的頻寬限制，並高估其備份功能。

建立此最佳實務的優勢：透過在您的雲端環境與您的公司或內部部署環境之間實作備援連線，即可確保兩個環境之間的相依服務能夠可靠地進行通訊。

未建立此最佳實務時的曝險等級：高

實作指引

使用 AWS Direct Connect 將內部部署網路連線至 時 AWS，您可以使用在多個內部部署位置和多個 AWS Direct Connect 位置的不同裝置上結束的個別連線，來達到最大的網路彈性（SLA 99.99%）。此拓撲可針對裝置故障、連線問題和完全的位置中斷提供復原能力。或者，您也可以使用兩個個別連線至多個位置（SLA每個內部部署位置都連接至單一 Direct Connect 位置），以達到高彈性（ 99.9%）。此方法可防止因光纖斷裂或裝置故障而造成的連線中斷，並有助於減輕完全的位置故障。 AWS Direct Connect 復原工具組可協助設計您的 AWS Direct Connect 拓撲。

您也可以考慮在 上 AWS Site-to-Site VPN 結束 ， AWS Transit Gateway 作為主要 AWS Direct Connect 連線的成本效益備份。此設定可讓成本相同的多路徑（ECMP）路由跨多個VPN通道，即使每個VPN通道上限為 1.25 Gbps，輸送量仍高達 50Gbps。不過，請務必注意，這仍然 AWS Direct Connect 是將網路中斷降至最低並提供穩定連線的最有效選擇。

VPNs 使用網際網路將雲端環境連線至內部部署資料中心時，請將兩個VPN通道設定為單一 site-to-siteVPN連線的一部分。每個通道都應該在不同的可用區域結束，以獲得高可用性，並使用備援硬體來防止內部部署裝置故障。此外，請考慮您內部部署位置來自不同網際網路服務供應商（ISPs）的多個網際網路連線，以避免因單一ISP中斷而導致完全VPN連線中斷。選擇ISPs具有各種路由和基礎設施的 ，特別是具有 AWS 端點個別實體路徑的 ，可提供高度的連線可用性。

除了具有多個 AWS Direct Connect 連線和多個VPN通道（或兩者的組合）的實體備援之外，實作邊界閘道通訊協定（BGP）動態路由也很重要。Dynamic 會根據即時網路條件和設定的政策BGP，自動將流量從一個路徑重新路由到另一個路徑。這種動態行為對於在發生連結或網路故障時維持網路可用性和服務連續性特別有益。它可以快速選擇替代路徑，提高網路的彈性和可靠性。

實作步驟

- 在 AWS 與內部部署環境之間取得高可用性連線。
  - 在個別部署的私有網路之間使用多個 AWS Direct Connect 連線或VPN通道。
  - 使用多個 AWS Direct Connect 位置以獲得高可用性。
  - 如果使用多個 AWS 區域，請在其中至少兩個 中建立備援。
- AWS Transit Gateway盡可能使用 來結束VPN連線 。
- 評估 AWS Marketplace 設備以結束VPNs或擴展 SD-WAN 至 AWS。如果您使用 AWS Marketplace 設備，請在不同的可用區域中部署冗餘執行個體以實現高可用性。

- 提供內部部署環境的備援連線。
  - 您可能需要多個 的備援連線 AWS 區域 ，才能達成您的可用性需求。
  - 使用 AWS Direct Connect 彈性工具組以開始使用。

資源

相關文件：

- AWS Direct Connect 恢復力建議
- 使用備援 Site-to-SiteVPN連線提供容錯移轉
- 路由政策和BGP社群
- 中的主動/主動和主動/被動組態 AWS Direct Connect
- APN 合作夥伴：可協助規劃聯網的合作夥伴
- AWS Marketplace 適用於網路基礎設施
- Amazon Virtual Private Cloud 連線選項白皮書
- 建置可擴展且安全的多VPC AWS 網路基礎設施
- 使用備援 Site-to-SiteVPN連線提供容錯移轉
- 使用 AWS Direct Connect 彈性工具組開始
- VPC 端點和VPC端點服務 （ AWS PrivateLink ）
- 什麼是 AmazonVPC？
- 什麼是傳輸閘道？
- 什麼是 AWS Site-to-Site VPN？
- 使用 Direct Connect 閘道

相關影片：

- AWS re：Invent 2018：Amazon 的進階VPC設計和新功能 VPC
- AWS re：Invent 2019： AWS Transit Gateway reference architecture for many VPCs

REL02-BP03 確保 IP 子網路配置考量擴充性和可用性

Amazon VPC IP 地址範圍必須足夠大，以適應工作負載需求，包括考慮未來擴展，以及將 IP 地址配置到可用區域的子網路。這包括負載平衡器、EC2執行個體和容器型應用程式。

規劃您的網路拓樸時，首先要定義 IP 位址空間。應為每個 配置私有 IP 地址範圍（遵循 RFC 1918 年準則）VPC。在此流程中請滿足下列要求：

- 允許VPC每個區域的 IP 地址空間超過一個。
- 在 中VPC，為多個子網路留出空間，以便您可以涵蓋多個可用區域。
- 考慮將未使用的CIDR區塊空間保留在 中VPC，以供未來擴充。
- 確保有 IP 地址空間來滿足您可能使用的任何暫時性 Amazon EC2執行個體機群的需求，例如用於機器學習的 Spot Fleet、Amazon EMR叢集或 Amazon Redshift 叢集。由於每個 Kubernetes Pod 依預設都會從VPCCIDR區塊中指派一個可路由地址，因此應該對 Kubernetes 叢集進行類似的考量，例如 Amazon Elastic Kubernetes Service（Amazon EKS）。
- 請注意，每個子網路CIDR區塊的前四個 IP 地址和最後一個 IP 地址都已保留，且無法使用。
- 請注意，配置到 的初始VPCCIDR區塊VPC無法變更或刪除，但您可以將其他非重疊CIDR區塊新增至 VPC。不過，子網路IPv4CIDRs無法變更IPv6CIDRs。
- 最大的可能VPCCIDR區塊為 /16，最小區塊為 /28。
- 考慮其他連線網路（VPC、內部部署或其他雲端供應商），並確保 IP 地址空間不重疊。如需詳細資訊，請參閱 REL02-BP05 在所有連線的私有地址空間中強制執行不重疊的私有 IP 地址範圍。

預期成果：可擴展的 IP 子網路可以幫助您適應未來的成長，並避免不必要的浪費。

常見的反模式：

- 未考慮未來的成長，導致CIDR區塊太小且需要重新設定，進而可能導致停機時間。
- 錯誤預估 Elastic Load Balancer 可以使用的 IP 位址數量。
- 在相同的子網路中部署許多高流量負載平衡器
- 使用自動擴展機制，同時無法監控 IP 位址使用情況。
- 定義過大CIDR的範圍遠遠超過未來的成長預期，這可能會導致難以與其他地址範圍重疊的網路互連。

建立此最佳實務的優勢：如此可確保您可以適應工作負載的增長，並在向上擴展時繼續提供可用性。

未建立此最佳實務時的曝險等級：中

實作指引

規劃網路以適應增長、法規要求以及與其他網路整合。增長可能會被低估，合規要求可能會發生變化，並且如果沒有適當的規劃，採購或私有網路連線可能會難以實作。

- 根據您的服務需求、延遲、法規 AWS 帳戶 和災難復原 （DR） 需求選取相關 和 區域。
- 識別區域VPC部署的需求。
- 識別 的大小VPCs。
  - 判斷您是否要部署多VPC連線能力。
    - 什麼是 Transit Gateway？
    - 單一區域多VPC連線能力
  - 確定您是否需要區隔聯網以滿足法規要求。
  - VPCs 使用適當大小的CIDR區塊進行 ，以滿足您的目前和未來需求。
    - 如果您有未知的成長預測，您可能想要在較大的CIDR區塊側邊錯誤，以減少未來重新設定的可能性
  - 請考慮使用子網路IPv6定址作為雙堆疊 的一部分VPC。IPv6 非常適合用於包含暫時性執行個體或容器機群的私有子網路，否則需要大量IPv4地址。

資源

相關 Well-Architected 的最佳實務：

- REL02-BP05 在所有連線的私有地址空間中強制執行不重疊的私有 IP 地址範圍

相關文件：

- APN 合作夥伴：可協助規劃聯網的合作夥伴
- AWS Marketplace 適用於網路基礎設施
- Amazon Virtual Private Cloud 連線選項白皮書
- 多個資料中心 HA 網路連線能力
- 單一區域多VPC連線能力
- 什麼是 AmazonVPC？
- IPv6 在 上 AWS
- IPv6 在參考架構上
- Amazon Elastic Kubernetes Service 啟動IPv6支援
- 對您的 - Classic Load Balancer VPC 的建議
- 可用區域子網路 - Application Load Balancer

- 可用區域 - Network Load Balancer

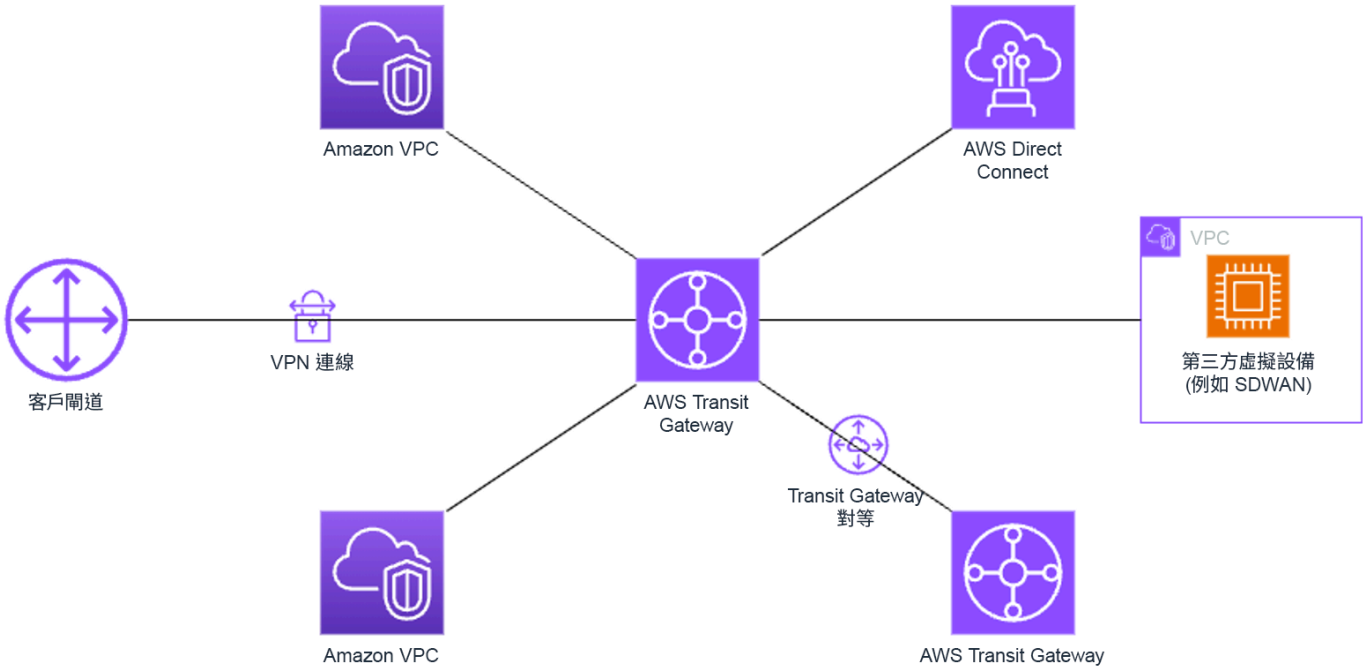
相關影片：

- AWS re：Invent 2018：Amazon 的進階VPC設計和新功能 VPC（NET303）
- AWS re：Invent 2019： AWS Transit Gateway 許多 VPCs（NET406-R1） 的參考架構
- AWS re：Invent 2023： AWS 準備下一步？ 為成長和靈活性設計網路 （NET310）

REL02-BP04 偏好軸輻式拓撲而非多對多網狀拓撲

連接多個私有網路 (例如虛擬私有雲端 (VPC) 和內部部署網路時，請選擇軸輻式拓撲而非網狀拓撲。與網狀拓撲不同，其中每個網路都直接連接到其他網路並增加了複雜性和管理開銷，軸輻式架構會透過單一中樞集中連線。這種集中化簡化了網路結構，並增強了其可操作性、可擴展性和控制能力。

AWS Transit Gateway 是一項受管理、可擴展且高可用性的服務，專為在 AWS 上建構軸輻式網路而設計。它可做為網路的中心樞紐，提供網路分段、集中式路由以及與雲端和內部部署環境的簡化連線。下圖說明如何使用 AWS Transit Gateway 來建置軸輻式拓撲。



預期成果：您已透過中樞連接虛擬私有雲端 (VPC) 和內部部署網路。您能透過中樞設定對等連線，而此中樞可做為高度擴展的雲端路由器。您不需要處理複雜的對等關係，因此路由更簡單。網路之間的流量會經過加密，而且可以隔離網路。

常見的反模式：

- 您會建置複雜的網路對等規則。
- 您在彼此不應相互通訊的網路之間提供路由 (例如，彼此互不相依的個別工作負載)。
- 中樞執行個體的管控無效。

建立此最佳實務的優勢：隨著連線網路數量的增加，網狀連線的管理和擴充變得越來越具有挑戰性。網格架構會帶來額外的挑戰，例如額外的基礎設施元件、組態需求和部署考量。網格也會帶來額外的負荷，因為需要管理和監控資料平面和控制平面元件。您必須思考如何提供高度可用的網格架構、如何監控網格運作狀態和效能，以及如何處理網格元件的升級。

另一方面來說，軸輻式模型會在多個網路之間建立集中式流量路由。該模型提供更簡單的方法來管理和監控資料平面和控制平面元件。

未建立此最佳實務時的曝險等級：中

實作指引

若還沒有網路服務帳戶，請先建立帳戶。在組織的網路服務帳戶中設置中樞。此方法可讓網路工程師集中管理中樞。

軸輻式模型的中樞是做為虛擬路由器，可讓流量在您的虛擬私有雲端 (VPC) 與內部部署網路之間傳遞。這種方法降低了網路的複雜性，同時也更容易對聯網問題進行故障診斷。

考慮您的網路設計，包括要互連的 VPC、AWS Direct Connect 和 Site-to-Site VPN 連線。

考慮針對每個傳輸閘道 VPC 連接使用個別子網路。對於每個子網路，請使用小型 CIDR (例如 /28)，以便擁有更多位址空間可供運算資源使用。此外，建立一個網路 ACL，並將它與中樞的所有關聯子網路建立關聯。在輸入和輸出方向上保持網路 ACL 開啟。

設計和實作您的路由表，以便僅在應通訊的網路之間提供路由。省略彼此不應相互通訊的網路之間的路由 (例如，彼此互不相依的個別工作負載之間)。

實作步驟

1. 規劃您的網路。決定要連線的網路，並確認這些網路提供的 CIDR 範圍不會重疊。
2. 建立 AWS Transit Gateway 並連接您的 VPC。
3. 如有需要，請建立 VPN 連線或 Direct Connect 閘道，並將其與 Transit Gateway 建立關聯。
4. 透過 Transit Gateway 路由表的組態，定義如何在連線的 VPC 和其他連線之間路由流量。

5. 使用 Amazon CloudWatch，視需要來監控和調整組態，以進行效能和成本最佳化。

資源

相關的最佳實務：

- REL02-BP03 確保 IP 子網路配置帳戶具有擴展性和可用性
- REL02-BP05 在連線的所有私有位址空間中強制使用不重疊的私有 IP 位址範圍

相關文件：

- 什麼是 Transit Gateway？
- 傳輸閘道設計最佳實務
- 建立可擴展且安全的多個 VPC AWS 網路架構
- 使用 AWS Transit Gateway 區域間對等互連建置全球網路
- Amazon Virtual Private Cloud 連線選項
- APN 合作夥伴：可以幫助您規劃聯網的合作夥伴
- 適用於網路基礎設施的 AWS Marketplace

相關影片：

- AWS re:Invent 2023 - AWS 聯網基礎
- AWS re:Invent 2023 - 進階 VPC 設計及新功能

相關研討會：

- AWS Transit Gateway 研討會

REL02-BP05 在所有連線的私有地址空間中強制執行不重疊的私有 IP 地址範圍

對等、透過 Transit Gateway 連線或透過 連線時，每個 的 IP 地址範圍VPCs不得重疊VPN。避免 IP 地址在 VPC和內部部署環境之間或您使用的其他雲端提供者之間發生衝突。您也必須有一種在需要時分配私有 IP 位址範圍的方法。IP 地址管理（IPAM）系統可協助自動化此作業。

預期成果：

- VPCs、內部部署環境或其他雲端提供者之間沒有 IP 地址範圍衝突。
- 適當的 IP 位址管理可以更輕鬆地擴展網路基礎設施，以適應網路需求的成長和變化。

常見的反模式：

- 在 中使用VPC與內部部署、公司網路或其他雲端供應商相同的 IP 範圍
- 不追蹤VPCs用於部署工作負載的 IP 範圍。
- 仰賴手動 IP 位址管理程序，例如試算表。
- 大小過大或過小的CIDR區塊，這會導致 IP 地址浪費或工作負載的地址空間不足。

建立此最佳實務的優勢：主動規劃網路，可確保在互連網路中不會出現多個相同的 IP 位址。這可防止使用不同應用程式的工作負載部分發生路由問題。

未建立此最佳實務時的曝險等級：中

實作指引

使用 IPAM，例如 Amazon VPC IP Address Manager ，來監控和管理您的CIDR使用。您也可以IPAMs從 取得數個 AWS Marketplace。在 上評估您的潛在用量 AWS，將CIDR範圍新增至現有VPCs，並建立 VPCs 以允許規劃的使用量增長。

實作步驟

- 擷取電流CIDR消耗 （例如 VPCs和 子網路）。
  - 使用 服務API操作來收集目前CIDR消耗。
  - 使用 Amazon VPC IP Address Manager 來探索資源 。
- 記錄當前的子網路用量。
  - 使用服務API操作來收集每個區域中每個 的子網路。 VPC
  - 使用 Amazon VPC IP Address Manager 來探索資源 。
- 記錄當前用量。
- 確定是否建立了任何重疊的 IP 範圍。
- 計算備用容量。
- 識別重疊的 IP 範圍。您可以遷移到新的地址範圍，也可以考慮使用私有NAT閘道之類的技術，或者AWS PrivateLink如果您需要連接重疊範圍。

資源

相關的最佳實務：

- 保護網路

相關文件：

- APN 合作夥伴：可協助規劃聯網的合作夥伴
- AWS Marketplace 適用於網路基礎設施
- Amazon Virtual Private Cloud 連線選項白皮書
- 多個資料中心 HA 網路連線能力
- 連線具有重疊 IP 範圍的網路
- 什麼是 AmazonVPC？
- 什麼是 IPAM？

相關影片：

- AWS re：Invent 2023 - 進階VPC設計和新功能
- AWS re：Invent 2019： AWS Transit Gateway reference 架構，適用於許多 VPCs
- AWS re：Invent 2023 - 為下一步做好準備？ 設計網路實現增長和靈活性
- AWS re：Invent 2021 - {New Launch} 在 上大規模管理您的 IP 地址 AWS

工作負載架構

可靠的工作負載始於對軟體和基礎設施的前期設計決策。您的架構選擇會對所有六大 Well-Architected 支柱的工作負載行為產生影響。為求可靠性，您必須依循特定模式。

以下數節說明使用這些可靠性模式的最佳實務。

主題

- 設計您的工作負載服務架構
- 在分散式系統中設計防止失敗的互動
- 設計分散式系統中的互動以緩解或承受失敗

設計您的工作負載服務架構

使用服務導向架構 (SOA) 或微型服務架構，建置擴展性與可靠性高的工作負載。服務導向架構 (SOA) 是透過服務界面讓軟體元件可重複使用的做法。微型服務架構則進一步讓元件變得更小、更簡單。

服務導向架構 (SOA) 界面使用常見的通訊標準，因此可以快速被納入新的工作負載。SOA 取代了建置整合型架構的做法，整合型架構是由互相依存、不可分割的單元組成。

在 AWS，雖然我們總是使用 SOA，但現在已接受使用微型服務建置系統的做法。儘管微型服務具有多種頗具吸引力的品質，但可用性的最重要益處是微型服務更為小巧簡單。藉助它們，您將能夠區分不同服務所需的可用性，進而更加注重在具有最大可用性需求的微型服務上進行投資。例如，為了在 Amazon.com 上交付產品資訊頁面 (「詳細頁面」)，我們將調用數百種微型服務，進而建置頁面的離散部分。雖然必須提供一些服務來展示價格和產品詳細資訊，但是如果該服務不可用，則可以將頁面上的絕大多數內容排除在外。甚至不需要相片和審查之類的東西來提供客戶可以購買產品的體驗。

最佳實務

- REL03-BP01 選擇如何分割工作負載
- REL03-BP02 建置專注於特定業務網域和功能的服務
- REL03-BP03 提供每個 的服務合約 API

REL03-BP01 選擇如何分割工作負載

在確認應用程式的彈性要求時，工作負載劃分是很重要的。應盡可能避免整合型架構。您應審慎考量哪些應用程式元件可分解為微型服務。根據您的應用程式需求，這可能最終會盡可能結合服務導向架構（SOA）與微服務。可以無狀態的工作負載較有能力部署為微型服務。

預期成果：工作負載應可受支援、可擴展，並且盡可能地鬆散耦合。

在選擇如何劃分工作負載時，請在效益與複雜性之間取得平衡。讓新產品能率先推出的正確做法，不同於打造可從最初需求擴展的工作負載的做法。重構現有的整合型時，您必須考量應用程式如何能支援以無狀態為方向的解構。將服務細分為較小的服務，可讓明確定義的小型團隊加以開發及管理。但較小的服務可能會帶來複雜性，包括延遲可能增加、偵錯更複雜，以及運作負擔增加。

常見的反模式：

- 微服務 Death Star 是一種特定情況：基本元件變得高度互相依賴，以致於只要有其中之一失敗，就會引發更加巨大的失敗，而導致元件像整合型一樣僵固且脆弱。

建立此實務的優勢：

- 更明確的劃分可造就更高的靈活性、組織彈性及可擴展性。
- 降低服務中斷的影響。
- 應用程式元件可能會有不同的可用性要求，這一點可藉由更細微的劃分來支應。
- 為支援工作負載的團隊明確定義責任。

未建立此最佳實務時的曝險等級：高

實作指引

根據劃分工作負載的方式，選擇您的架構類型。選擇 SOA或 微服務架構 （或在極少數情況下，選擇單片架構）。即使您選擇從整體架構開始，仍必須確保其為模組化，並隨著產品隨著使用者採用而擴展，最終可以發展為 SOA或 微服務。SOA 和 微服務分別提供較小的分割，這是現代可擴展且可靠的架構，但需要考慮權衡，特別是部署微服務架構時。

主要取捨之一，就是您現在擁有一種分散式運算架構，而其可能會增加您滿足使用者延遲要求的難度，並且在偵測和追蹤使用者互動方面還存在額外的複雜性。您可以利用 AWS X-Ray 來解決此問題。要考慮的另一個影響是，隨著您管理的應用程式數量增加，營運複雜性也隨之增加，因而需要部署多個獨立元件。

整合型、服務導向與微型服務架構

實作步驟

- 決定適當的架構以重構或建置您的應用程式。SOA 和 微服務分別提供較小的分割，這是現代可擴展且可靠的架構首選。SOA 對於實現更小的分割，同時避免微服務的一些複雜性，可能是一個很好的妥協。如需詳細資訊，請參閱 Microservice Trade-Offs。
- 如果您的工作負載適用於此類型，且您的組織可以提供支援，則應使用微型服務架構達成最佳的靈活性和可靠性。如需更多詳細資訊，請參閱在 上實作 Microservices AWS。
- 考慮遵循 Strangler Fig 模式，將整體重構為較小的組件。這涉及逐步以新的應用程式和服務取代特定的應用程式元件。AWS Migration Hub Refactor Spaces 可充當增量重構的起點。如需詳細資訊，親參閱 Seamlessly migrate on-premises legacy workloads using a strangler pattern。
- 實作微服務可能需要服務探索機制，以允許這些分散式服務彼此通訊。 AWS App Mesh 可與服務導向的架構搭配使用，以提供可靠的服務探索和存取。 AWS Cloud Map也可用於動態的 DNS型服務探索。
- 如果您要從整體遷移至 SOA，Amazon MQ 可以在重新設計雲端中的舊版應用程式時，協助將差距橋接為服務匯流排。
- 對於具有單一共用資料庫的現有整合型，請選擇如何將資料重新組織為較小的區段。此時可以按業務單位、存取模式或資料結構來劃分。在重構程序中，您應該選擇使用關聯式或非關聯式（否 SQL）類型的資料庫繼續。如需更多詳細資訊，請參閱從 SQL到否 SQL。

實作計劃的工作量：高

資源

相關的最佳實務：

- REL03-BP02 建置專注於特定業務網域和功能的服務

相關文件：

- Amazon API Gateway：RESTAPI使用 Open 設定API
- 什麼是服務導向架構？
- 有界限的環境 (領域驅動型設計的集中模式)
- 在 上實作 Microservices AWS
- 微型服務權衡
- 微型服務 - 此新架構術語的定義
- 上的微服務 AWS
- 什麼是 AWS App Mesh？

相關範例：

- 迭代應用程式現代化研討會

相關影片：

- 在 上使用 Microservices 實現卓越 AWS

REL03-BP02 建置專注於特定業務網域和功能的服務

服務導向架構（SOA）定義具有業務需求定義之詳細函數的服務。微型服務使用領域模型和有界限的環境，沿著業務環境界限繪製服務界限。專注於業務領域和功能，有助於團隊為其服務定義獨立的可靠性要求。有界限的環境可隔離和封裝商業邏輯，讓團隊更適切地推論如何處理失敗。

預期成果：工程師和業務利益相關者共同定義有界限的環境，並將其用來設計系統，作為滿足特定業務功能的服務。這些團隊使用既定的做法 (如事件風暴) 來定義要求。新的應用程式設計為服務妥善定義的界限和鬆散耦合。現有單體會分解為邊界內容，而系統設計則朝向 SOA或微服務架構邁進。整合型服務重構時，會套用已建立的方法 (如 Bubble 環境) 和整合型分解模式。

領域導向服務會以一或多個不共用狀態的程序執行。它們會單獨回應需求的波動，並根據領域的特定要求來處理錯誤情境。

常見的反模式：

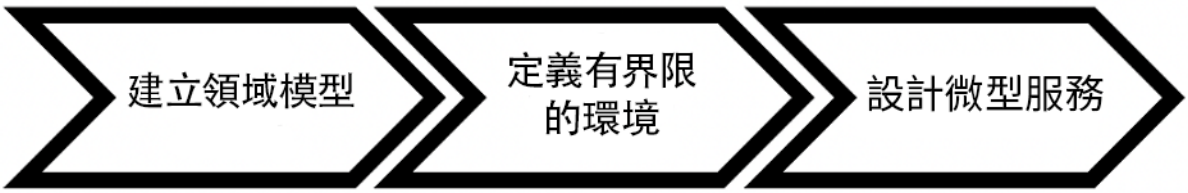
- 團隊是依據特定技術領域 (例如 UI 和 UX、中介軟體或資料庫) 組成的，而不是特定的業務領域。
- 應用程式跨多個領域責任。跨有界限環境的服務可能更難以維護，需要較大量的測試工作，且需要多個領域團隊參與軟體更新。
- 領域相依性 (例如領域實體程式庫) 會跨服務共用，因此一個服務領域出現變更時，需要變更其他服務領域
- 服務合約和商業邏輯無法以通用且一致的領域語言來表達實體，因此會導致翻譯層級使系統複雜化，並增加偵錯工作。

建立此最佳實務的優勢：應用程式設計為獨立的服務，受到業務領域限制，並使用共同的商務語言。服務可以單獨測試和部署。服務符合實作領域的特定恢復能力要求。

未建立此最佳實務時的曝險等級：高

實作指引

網域驅動設計（DDD）是圍繞業務網域設計和建置軟體的基礎方法。在建置專注於業務領域的服務時，使用現有架構將有所幫助。使用現有的整合型應用程式時，您可以利用分解模式提供已建立的技術，將應用程式現代化為服務。



領域驅動的設計

實作步驟

- 團隊可舉行事件風暴研討會，以便箋格式快速識別事件、命令、彙總和領域。
- 在網域環境中形成網域實體和函數之後，可以使用有界限的環境將網域劃分為服務，其中共用相似特徵和屬性的實體會分組在一起。隨著此模型劃分成多個環境，如何界定微型服務界限的範本便會浮現。

  - 例如，Amazon.com 網站實體可能包括包裝、交付、排程、價格、折扣和貨幣。
  - 包裝、交付和排程會分組到出貨環境中，而價格、折扣和貨幣則分組到訂價環境中。
- 將整合型服務分解為微型服務概述了重構微型服務的模式。按業務功能、子領域或交易使用分解的模式，會與領域驅動的方法保持一致。
- 泡泡內容等戰術技術可讓您DDD在現有或舊版應用程式中導入，而無需預先重寫和對 的完整承諾DDD。在 bubble 環境方法中，使用服務映射和協調或防損毀層來建立一個小的有界限環境，可保護新定義的網域模型免受外部影響。

在團隊執行網域分析和定義的實體和服務合約之後，他們可以利用 AWS 服務來實作其以雲端為基礎的服務之網域驅動設計。

- 藉由定義執行領域商務規則的測試來起始您的開發。測試驅動的開發（TDD）和行為驅動的開發（BDD）可協助團隊讓服務專注於解決業務問題。
- 選取最符合您的企業網域需求和微服務架構的 AWS 服務：
  - AWS 無伺服器可讓您的團隊專注於特定網域邏輯，而不是管理伺服器和基礎設施。
  - AWS的容器可簡化基礎設施的管理，讓您得以專注在您的網域要求上。
  - 專用資料庫協助您根據領域要求找出最適合的資料庫類型。
- 在 AWS上建置六邊形架構，它概述了一個框架，用以將業務邏輯建置到從業務領域回溯運作的服務中，以滿足功能要求，然後附加整合適配器。使用 AWS 服務將介面詳細資訊與業務邏輯分開的模式，可協助團隊專注於網域功能並改善軟體品質。

資源

相關的最佳實務：

- REL03-BP01 選擇如何分割工作負載
- REL03-BP03 提供每個 的服務合約 API

相關文件：

- AWS Microservices
- 在 上實作 Microservices AWS
- 如何將整合型服務分成微型服務
- 由 Legacy Systems 包圍DDD時入門

- 網域驅動設計：解決軟體核心的複雜性
- 在 上建置六邊形架構 AWS
- 將整合型服務分解為微型服務
- 事件風暴
- 有界限的環境之間的訊息
- 微型服務
- 測試驅動的開發
- 行為驅動的開發

相關範例：

- 在 AWS （從 DDD/EventStormingWorkshop） 上設計雲端原生微服務

相關工具：

- AWS 雲端 資料庫
- 上的無伺服器 AWS
- 的容器 AWS

REL03-BP03 提供每個 的服務合約 API

服務合約是機器可讀取API定義中定義的API生產者和消費者之間的書面協議。合約版本控制策略可讓消費者繼續使用現有的 ，API並在準備好API時將其應用程式遷移至較新的應用程式。只要遵守合約，就隨時可執行生產者部署。服務團隊可以使用他們選擇的 技術堆疊來滿足API合約。

預期結果：使用服務導向或微服務架構建置的應用程式可以獨立運作，同時具有整合的執行時間相依性。當雙方遵循共同API合約時，部署到API消費者或生產者的變更不會中斷整體系統的穩定性。透過服務通訊的元件APIs可以執行獨立的功能版本、升級至執行時間相依性，或容錯移轉至災難復原（DR） 站台，彼此幾乎沒有影響。此外，離散服務能夠獨立擴展而滿足資源需求，不需要其他服務一起擴展。

常見的反模式：

- 建立APIs不含強烈輸入結構描述的服務。這會導致 APIs無法用來產生無法以程式設計方式驗證的API繫結和承載。

- 不採用版本控制策略，這會強制API消費者在服務合約演進時更新和發行或失敗。
- 錯誤訊息會透露基礎服務實作的詳細資訊，而不是說明領域環境和語言中的整合失敗。
- 不使用API合約來開發測試案例和模擬API實作，以允許獨立測試服務元件。

建立此最佳實務的好處：由透過API服務合約通訊的元件組成的分散式系統可以提高可靠性。開發人員可以在開發程序的早期發現潛在的問題，並在編譯期間進行類型檢查，以確認請求和回應遵循API合約和必填欄位。API 合約為 APIs和 提供者提供了明確的自我記錄介面，讓不同的系統和程式設計語言之間具有更好的互通性。

未建立此最佳實務時的曝險等級：中

實作指引

識別業務網域並確定工作負載分割後，您就可以開發服務 APIs。首先，定義 的機器可讀取服務合約 APIs，然後實作API版本控制策略。當您準備好透過 REST、GraphQL 或非同步事件等常見通訊協定整合 服務時，您可以將 AWS 服務整合到您的架構中，以將元件與強式API合約整合。

AWS 服務API對象的服務

將 Amazon API Gateway 、 AWS AppSync和 Amazon EventBridge 等 AWS 服務整合到您的架構中，以在應用程式中使用API服務合約。Amazon API Gateway 可協助您與直接原生 AWS 服務和其他 Web 服務整合。API Gateway 支援 OpenAPI 規格和 versioning。 AWS AppSync 是您透過定義 GraphQL 結構描述來定義查詢、突變和訂閱的服務界面所設定的受管 GraphQL 端點。Amazon EventBridge 使用事件結構描述來定義事件，並為您的事件產生程式碼繫結。

實作步驟

- 首先，為您的 定義合約API。合約會表達 的功能API，並定義API輸入和輸出的強烈輸入資料物件和欄位。
- 當您APIs在 API Gateway 中設定 時，您可以匯入和匯出端點的開放API規格。
  - 匯入 OpenAPI 定義可簡化 的建立，API並可整合 AWS 基礎設施作為程式碼工具，例如 AWS Serverless Application Model和 AWS Cloud Development Kit (AWS CDK)。
  - 匯出API定義可簡化與API測試工具的整合，並為服務消費者提供整合規格。
- 您可以使用 APIs AWS AppSync 定義 GraphQL 結構描述檔案來定義和管理 GraphQL，以產生合約界面，並簡化與複雜REST模型、多個資料庫資料表或舊版服務的互動。
- AWS Amplify 與 整合的專案 AWS AppSync 會產生強式 JavaScript 查詢檔案，用於您的應用程式，以及 Amazon DynamoDB 資料表的 AWS AppSync GraphQL 用戶端程式庫。

- 當您從 Amazon 取用服務事件時 EventBridge，事件會遵守已存在於結構描述登錄檔中的結構描述，或是您使用 OpenAPI Spec 定義的結構描述。使用登錄中定義的結構描述時，您也可以從結構描述合約產生用戶端繫結，以將程式碼與事件整合。
- 擴展或版本您的 API。新增可使用選用欄位或必要欄位的預設值設定的欄位時，延伸 API 是更簡單的選項。
  - JSON REST和 GraphQL 等通訊協定的 型合約非常適合合約延伸。
  - XML 等通訊協定的 型合約SOAP應與 服務消費者進行測試，以判斷合約延伸的可行性。
- 版本控制 時API，請考慮實作 Proxy 版本控制，其中使用外觀來支援版本，以便在單一程式碼庫中維護邏輯。
  - 使用 API Gateway，您可以使用請求和回應映射，透過建立外觀來提供新欄位的預設值，或從請求或回應中刪除的欄位，以簡化吸收合約變更。透過此方法，基礎服務可以維護單一程式碼基底。

資源

相關的最佳實務：

- REL03-BP01 選擇如何分割工作負載
- REL03-BP02 建置專注於特定業務網域和功能的服務
- REL04-BP02 實作鬆散耦合相依性
- REL05-BP03 控制和限制重試呼叫
- REL05-BP05 設定用戶端逾時

相關文件：

- 什麼是 API（應用程式程式設計介面）？
- 在 上實作 Microservices AWS
- 微型服務權衡
- 微型服務 - 此新架構術語的定義
- 上的微服務 AWS
- 使用 API Gateway 擴充功能開啟API
- 開啟 API-規格
- GraphQL：結構描述和類型
- Amazon EventBridge 程式碼繫結

相關範例：

- Amazon API Gateway：設定RESTAPI使用開啟API
- 使用 Open 的 Amazon API Gateway 至 Amazon DynamoDB CRUD 應用程式API
- 無伺服器時代的現代應用程式整合模式：APIGateway Service Integration
- 使用 Amazon 實作標頭型API閘道版本控制 CloudFront
- AWS AppSync：建置用戶端應用程式

相關影片：

- 使用在 中開啟API AWS SAM 管理API閘道

相關工具：

- Amazon API Gateway
- AWS AppSync
- Amazon EventBridge

在分散式系統中設計防止失敗的互動

分散式系統仰賴通訊網路將伺服器或服務等元件互相連線。儘管這些網路中出現資料遺失或延遲，但工作負載仍必須可靠地運作。分散式系統的元件必須以不會對其他元件或工作負載造成負面影響的方式運作。這些最佳實務可防止故障，並改善平均故障間隔時間 (MTBF)。

最佳實務

- REL04-BP01 識別您依賴的分散式系統種類
- REL04-BP02 實作鬆散耦合相依性
- REL04-BP03 持續工作
- REL04-BP04 讓變異操作冪等

REL04-BP01 識別您依賴的分散式系統種類

分散式系統可以是同步、非同步或批次處理。同步系統必須盡快處理要求，並透過使用 HTTP/S、REST 或遠端程序呼叫 (RPC) 通訊協定進行同步請求和回應呼叫，以便相互通訊。非同步系統透過

中間服務以非同步方式交換資料來相互通訊，而不需要耦合個別系統。批處理系統接收大量輸入資料，無需人工介入即可執行自動化資料處理，並產生輸出資料。

預期成果：設計與同步、非同步和批次相依性有效互動的工作負載。

常見的反模式：

- 工作負載會無限期地等待來自其相依性的回應，這可能會導致工作負載用戶端逾時，而不知道是否已收到其請求。
- 工作負載使用可同步呼叫彼此的一系列相依系統。這需要每個系統都可供使用，並在整個系列成功之前成功處理請求，從而導致潛在的脆弱行為和整體可用性。
- 工作負載會以非同步方式與其相依性進行通訊，並依賴於僅保證傳遞訊息一次的概念，而且通常仍然可以接收重複的訊息。
- 工作負載不使用適當的批次排程工具，並允許同時執行相同的批次工作。

建立此最佳實務的優勢：對於特定的工作負載來說，在同步、非同步和批次之間實作一種或多種通訊方式很常見。此最佳實務可協助您識別與每種通訊方式相關的不同權衡，讓您的工作負載能夠容忍其任何相依性中斷。

未建立此最佳實務時的風險暴露等級：高

實作指引

下列各節包含每種相依性的一般和特定實作指引。

一般指引

- 請確定相依性所提供的效能與可靠性服務水準目標 (SLO) 符合工作負載的效能和可靠性需求。
- 使用 AWS 可觀測性服務來監控回應時間和錯誤率，以確保您的相依性在工作負載所需的層級提供服務。
- 識別工作負載在與其相依性通訊時可能面臨的潛在挑戰。分散式系統面臨各種挑戰，可能會增加架構複雜性、營運負擔和成本。常見的挑戰包括延遲、網路中斷、資料遺失、擴展和資料複寫延遲。
- 實作強大的錯誤處理和日誌記錄，以協助您在相依性遇到問題時對問題進行疑難排解。

同步相依性

在同步通訊中，工作負載會將請求傳送至其相依性，並封鎖等待回應的操作。當其相依性收到請求時，它會嘗試盡快處理它，並將回應傳送回您的工作負載。同步通訊的一個重大挑戰是它會導致時間耦合，

這需要您的工作負載及其相依性同時可用。當您的工作負載需要與其相依性同步通訊時，請考慮下列指引：

- 您的工作負載不應該依賴多個同步相依性來執行單一函數。此相依性系列增加了整體的脆弱性，因為路徑中的所有相依性都必須可用，才能順利完成請求。
- 當相依性狀態不良或無法使用時，請確定處理錯誤並重試策略。避免使用雙模態行為。雙模態行為是指工作負載在正常和故障模式下呈現不同行為的情況。如需雙模態行為的詳細資訊，請參閱 REL11-BP05 使用靜態穩定性來防止雙模態行為。
- 請記住，快速檢錯好於讓工作負載等待。例如，AWS Lambda 開發人員指南說明如何在呼叫 Lambda 函數時處理重試和失敗。
- 當工作負載呼叫其相依性時設定逾時。此技術可避免等待太長時間或無限期等待回應。如需此問題的有用討論，請參閱 Tuning AWS Java SDK HTTP request settings for latency-aware Amazon DynamoDB applications。
- 盡可能減少從工作負載到其相依性的呼叫次數，以滿足單一請求。在兩者之間進行聊天呼叫會增加耦合和延遲。

異步相依性

若要暫時將工作負載與其相依性分離，它們應該以非同步方式進行通訊。使用非同步方法，您的工作負載可以繼續執行任何其他處理，而不必等待其相依性或相依性系列來傳送回應。

當您的工作負載需要與其相依性異步通訊時，請考慮下列指引：

- 根據您的使用案例和需求來決定是使用訊息傳遞還是事件串流。訊息傳遞可讓您的工作負載透過訊息代理程式傳送和接收訊息，藉此與其相依性進行通訊。事件串流可讓您的工作負載及其相依性使用串流服務來發布和訂閱事件 (以連續資料串流形式傳送)，這些事件需要盡快處理。

- 訊息傳遞和事件串流處理訊息的方式不同，因此您需要根據下列內容做出權衡決策：
  - 訊息優先級：訊息代理程式可以在正常訊息之前處理高優先級訊息。在事件串流中，所有訊息都有相同的優先級。
  - 訊息消耗：訊息代理程式確保消費者收到消息。事件串流消費者必須追蹤他們讀取的最後一則訊息。
  - 訊息排序：使用訊息傳遞時，不能保證以傳送的確切順序接收訊息，除非您使用先進先出 (FIFO) 方法。事件串流永遠會保留產生資料的順序。
  - 訊息刪除：使用訊息傳遞時，消費者必須在處理訊息後刪除它。事件串流服務會將訊息附加至串流，並保留在其中，直到訊息的保留期過期為止。此刪除政策使得事件串流適合重播訊息。

- 定義工作負載如何知道其相依性何時完成其工作。例如，當工作負載以非同步方式調用 Lambda 函數時，Lambda 會將事件放在佇列中，並傳回成功回應，但沒有額外資訊。處理完成後，Lambda 函數可以將結果傳送至目的地，並根據成功或失敗進行設定。
- 透過利用冪等性來構建工作負載以處理重複訊息。冪等性意味著即使為相同訊息產生多次工作負載，工作負載的結果也不會變更。重要的是要指出，如果發生網路故障或尚未收到確認，訊息傳遞或串流服務將重新傳遞訊息。
- 如果您的工作負載未從其相依性中取得回應，則需要重新提交請求。請考慮限制重試次數，以保留工作負載的 CPU、記憶體和網路資源來處理其他請求。AWS Lambda 文件說明了如何處理非同步調用的錯誤。
- 運用適當的可觀測性、偵錯和追蹤工具，管理和操作工作負載的非同步通訊及其相依性。可以使用 Amazon CloudWatch 來監控訊息傳遞和事件串流服務。還可以使用 AWS X-Ray 檢測工作負載，以快速獲得疑難排解問題的洞見。

批處理相依性

批處理系統會取得輸入資料，啟動一系列作業來處理它，並產生一些輸出資料，無需人工介入。視資料大小而定，工作可能會執行幾分鐘，在某些情況下執行數天。當您的工作負載需要與其批處理相依性通訊時，請考慮下列指引：

- 定義工作負載執行批次工作的時間範圍。工作負載可以設定週期性模式來調用批次系統，例如，每小時或每月結束時。
- 確定資料輸入和已處理的資料輸出的位置。選擇可讓您的工作負載大規模讀取和寫入檔案的儲存服務，例如 Amazon Simple Storage Services (Amazon S3)、Amazon Elastic File System (Amazon EFS) 和 Amazon FSx for Lustre。
- 如果您的工作負載需要調用多個批次工作，可以利用 AWS Step Functions 簡化在 AWS 或內部部署中執行的批次工作的協同運作。此範例專案示範使用 Step Functions、AWS Batch 和 Lambda 協同運作批次任務。
- 監控批次任務以尋找異常情況，例如任務所花費的時間超過應該完成的時間。可以使用 CloudWatch Container Insights 等工具來監控 AWS Batch 環境和任務。在這種情況下，工作負載將從一開始就停止下一個任務，並通知相關人員有關例外情況。

資源

相關文件：

- AWS 雲端 操作：監控和可觀測性

- Amazon 建置者資料中心：分散式系統的挑戰
- REL11-BP05 使用靜態穩定性來防止雙模態行為
- AWS Lambda 開發人員指南：AWS Lambda 中的錯誤處理和自動重試
- 針對延遲感知的 Amazon DynamoDB 應用程式調整 AWS Java SDK HTTP 請求設定
- AWS 訊息傳遞
- 什麼是串流資料？
- AWS Lambda 開發人員指南：非同步調用
- Amazon Simple Queue Service 常見問答集：FIFO 佇列
- Amazon Kinesis Data Streams 開發人員指南：處理重複記錄
- Amazon Simple Queue Service 開發人員指南：適用於 Amazon SQS 的可用 CloudWatch 指標
- Amazon Kinesis Data Streams 開發人員指南：使用 Amazon CloudWatch 監控 Amazon Kinesis Data Streams 服務
- AWS X-Ray 開發人員指南：AWS X-Ray 概念
- GitHub 上的 AWS 範例：AWS Step 函數，複雜的協調器應用程式
- AWS Batch 使用者指南：AWS Batch CloudWatch Container Insights

相關影片：

- AWS Summit SF 2022 - 使用 AWS 獲得全堆疊可觀測性和應用程式監控 (COP310)

相關工具：

- Amazon CloudWatch
- Amazon CloudWatch Logs
- AWS X-Ray
- Amazon Simple Storage Services (Amazon S3)
- Amazon Elastic File System (Amazon EFS)
- Amazon FSx for Lustre
- AWS Step Functions
- AWS Batch

REL04-BP02 實作鬆散耦合相依性

佇列系統、串流系統、工作流程和負載平衡器之間具有鬆散耦合的相依性。鬆耦合有助於將某個元件的行為與依賴它的其他元件隔離，進而提高彈性和敏捷性。

解除相依性 (例如佇列系統、串流系統和工作流程) 有助於將變更或失敗對系統造成的影響降到最低。這種分離使組件的行為不會影響依賴它的其他組件，從而提高了彈性和敏捷性。

在緊耦合的系統中，對某個元件進行變更時，可能必須變更其他依賴此元件的元件，從而導致所有元件的效能降低。鬆耦合會破壞此相依性，因此相依元件只需要知道受版本控制的和已發布的界面。在相依性之間實作鬆耦合，可避免一個元件中的故障影響另一個元件。

鬆耦合可讓您修改程式碼或新增功能至某個元件，同時將依賴該元件的其他元件的風險降至最低。其還能讓您在元件層級提供細微的恢復能力，您可以橫向擴充，甚至是變更相依性的基礎實作。

若要透過鬆耦合進一步改善彈性，請盡可能讓元件採用非同步互動。此模型適用於不需要立即回應的任何互動，以及確認已註冊請求便以足夠的狀況。它涉及產生事件的一個元件和取用事件的另一個元件。這兩個元件不會透過直接 point-to-point互動整合，而是通常透過中繼耐用儲存層整合，例如 Amazon SQS佇列、串流資料平台，例如 Amazon Kinesis 或 AWS Step Functions。

圖 4：佇列系統和負載平衡器之間具有鬆散耦合的相依性

Amazon SQS佇列和 AWS Step Functions 只是新增中繼層以進行鬆散耦合的兩種方式。事件驅動的架構也可以 AWS 雲端 使用 Amazon 建置在 中 EventBridge，這可以從用戶端 （事件生產者） 仰賴的服務 （事件消費者） 中抽象用戶端 （事件生產者）。當您需要高輸送量、推送型訊息時， many-to-manyAmazon Simple Notification Service （Amazon SNS） 是有效的解決方案。使用 Amazon SNS 主題，您的發佈者系統可以將訊息散播到大量訂閱者端點以進行平行處理。

雖然佇列提供多項優勢，但在大多數硬式即時系統中，超過閾值時間 (通常為秒) 的請求應視為過時 (用戶端已放棄且不再等待回應) 且未處理。這樣才可以處理較新的 (且可能仍有效的) 請求。

預期成果：實作鬆耦合的相依性可將元件層級故障的影響降到最低，這有助於診斷和解決問題。它還能簡化開發週期，讓團隊在模組化層級實作變更，而不會影響依賴此元件之其他元件的效能。這種方法可讓您根據資源需求，以及對成本效益有所貢獻之元件的使用情況，在元件層級進行橫向擴充。

常見的反模式：

- 部署整合型工作負載。

- 直接在工作負載層APIs之間調用，而沒有容錯移轉或非同步處理請求的能力。
- 使用共用資料的緊耦合。鬆耦合系統應避免透過共用資料庫或其他形式的緊耦合資料儲存共用資料，這可能會重新引入緊耦合並阻礙可擴展性。
- 忽略反壓。當元件無法以相同的速率處理傳入的資料時，工作負載應該要有能力減緩或停止傳入的資料。

建立此最佳實務的優勢：鬆耦合有助於將某個元件的行為與依賴它的其他元件隔離，進而提高彈性和敏捷性。避免一個元件中的失敗影響其他元件。

未建立此最佳實務時的曝險等級：高

實作指引

實作鬆耦合相依性。有各種解決方案可讓您建置鬆耦合的應用程式。其中包括實作完全受管佇列、自動化工作流程、對事件做出反應APIs等服務，這些服務可協助隔離元件與其他元件的行為，進而提高復原能力和靈活性。

- 建置事件驅動架構：Amazon EventBridge 可協助您建置鬆散耦合和分散式事件驅動架構。
- 在分散式系統中實作佇列：您可以使用 Amazon Simple Queue Service（Amazon SQS）來整合和解耦分散式系統。
- 將元件容器化為微服務：微服務可讓團隊建置由小型獨立元件組成的應用程式，這些元件透過定義明確的 進行通訊APIs。Amazon Elastic Container Service（Amazon ECS）和 Amazon Elastic Kubernetes Service（Amazon EKS）可協助您更快開始使用容器。
- 使用 Step Functions 管理工作流程：Step Functions 可協助您將多項 AWS 服務協調為彈性工作流程。
- 利用發佈訂閱（pub/sub）訊息架構：Amazon Simple Notification Service（Amazon SNS）提供從發佈者到訂閱者（也稱為生產者和消費者）的訊息傳遞。

實作步驟

- 事件驅動架構中的元件會由事件啟動。事件是系統中發生的動作，例如使用者將某個商品新增至購物車。動作成功時會產生可啟動系統下一個元件的事件。
  - 使用 Amazon 建置事件驅動應用程式 EventBridge
  - AWS re：Invent 2022 - 使用 Amazon 設計事件驅動整合 EventBridge
- 分散式傳訊系統有三個需要針對佇列型架構來實作的主要部分。其中包括分散式系統的元件、用於解耦的佇列（在 Amazon SQS 伺服器上分佈），以及佇列中的訊息。典型的系統中有負責將訊息啟

動至佇列的生產者，以及從佇列接收訊息的取用者。佇列會在多個 Amazon SQS 伺服器之間存放訊息，以進行備援。

- 基本 Amazon SQS架構
- 使用 Amazon Simple Queue Service 在分散式應用程式之間傳送訊息

- 充分利用的微型服務會增強可維護性並提高可擴展性，因為鬆耦合元件由獨立團隊管理。其還能夠在發生變更時隔離單一元件的行為。
  - 在 上實作 Microservices AWS
  - 開始建構吧！使用容器建構微型服務
- AWS Step Functions 您可以透過 建置分散式應用程式、自動化程序、協調微服務等。將多個元件協同運作到自動化工作流程中可讓您解耦應用程式中的相依性。
  - 使用 AWS Step Functions 和 建立無伺服器工作流程 AWS Lambda
  - 入門 AWS Step Functions

資源

相關文件：

- Amazon EC2：確保錯位
- Amazon 建置者資料中心：分散式系統的挑戰
- Amazon 建置者資料中心：可靠性、持續工作以及咖啡時刻
- 什麼是 Amazon EventBridge？
- 什麼是 Amazon Simple Queue Service？
- 結束您的整合型架構
- 使用 AWS Step Functions 和 Amazon 協調佇列型 Microservices SQS
- 基本 Amazon SQS架構
- 佇列式架構

相關影片：

- AWS 2019 年紐約高峰會：事件驅動架構和 Amazon 簡介 EventBridge （MAD205）
- AWS re：Invent 2018：閉環和開場思維：如何控制系統，無論大小 ARC337（包括鬆散的耦合、固定工作、靜態穩定性）
- AWS re：Invent 2019：移至事件驅動的架構 （SVS308）

- AWS re：Invent 2019：使用 Amazon SQS和 Lambda 可擴展的無伺服器事件驅動應用程式
- AWS re：Invent 2022 - 使用 Amazon 設計事件驅動整合 EventBridge
- AWS re：Invent 2017：Elastic Load Balancing Deep Dive 和最佳實務

REL04-BP03 持續工作

負載大幅快速變更時，系統可能會發生故障。例如，如果您的工作負載正在執行運作狀態檢查，監控數千部伺服器的運作狀態，應該每次傳送相同大小的承載 (目前狀態的完整快照)。無論伺服器全無故障或全部出現故障，運作狀態檢查系統都會持續執行工作，而無大幅快速變更。

例如，如果運作狀態檢查系統正在監控 100,000 部伺服器，則在一般輕型伺服器失敗率下，其負載為額定值。不過，如果重大事件讓一半的伺服器運作狀況不良，則運作狀態檢查系統會因嘗試更新通知系統並向其用戶端溝通狀態，而承受不住負載。因此，運作狀態檢查系統應每次都傳送目前狀態的完整快照。100,000 個伺服器運作狀態 (每個以一位元表示) 只是 12.5 KB 的承載。無論沒有伺服器發生故障，還是全部發生故障，運作狀態檢查系統都會持續執行工作，而大型的快速變更也不會對系統穩定性造成威脅。實際上，這是 Amazon Route 53 處理端點運作狀態檢查 (例如 IP 位址) 的方式，以判斷最終使用者如何路由到端點。

未建立此最佳實務時的曝險等級：低

實作指引

- 執行持續工作，以便當負載大量快速變更時，系統不會發生故障。
- 實作鬆耦合相依性。佇列系統、串流系統、工作流程和負載平衡器之間具有鬆散耦合的相依性。鬆耦合有助於將某個元件的行為與依賴它的其他元件隔離，進而提高彈性和敏捷性。
  - Amazon 建置者資料中心：可靠性、持續工作以及咖啡時刻
  - AWS re：Invent 2018：閉環和開場思維：如何控制大大小小的系統 ARC337（包括持續工作）
    - 針對監控 100,000 部伺服器的運作狀態檢查系統範例，請設計工作負載，以便無論成功或失敗次數為何，承載大小都能保持不變。

資源

相關文件：

- Amazon EC2：確保錯位
- Amazon 建置者資料中心：分散式系統的挑戰
- Amazon 建置者資料中心：可靠性、持續工作以及咖啡時刻

相關影片：

- AWS 2019 年紐約高峰會：事件驅動架構和 Amazon 簡介 EventBridge （MAD205）
- AWS re：Invent 2018：閉環和開場思維：如何控制大大小小的系統 ARC337（包括持續工作）
- AWS re：Invent 2018：閉環和開場思維：如何控制系統，大大小小 ARC337（包括鬆散的耦合、固定工作、靜態穩定性）
- AWS re：Invent 2019：移至事件驅動的架構 （SVS308）

REL04-BP04 讓變異操作冪等

冪等服務確保每個請求只處理一次，因此發出多個相同請求的效果，與發出單一請求的效果相同。如此一來，用戶端可更輕鬆地重試，而不用擔心多次錯誤地處理請求。為此，用戶端可以使用冪等性字符發出 API 請求，每次重複請求時，都會使用這個權杖。冪等服務 API 會使用字符傳回回應，而該回應與第一次完成請求時傳回的回應相同，即使系統的基礎狀態已改變也一樣。

在分散式系統中，相對容易的做法是最多執行一次動作 (用戶端只發出一個請求) 或至少執行一次動作 (持續發出請求，直到用戶端確認成功)。但很難保證動作精準地執行一次，使得發出多個相同的請求與發出單一請求效果相同。透過在 API 中使用冪等性字符，服務可以收到一次或多次變異請求，而不需建立重複的記錄或產生副作用。

預期成果：您擁有一致、完整記錄且廣泛採用的方法，以確保所有元件和服務之間的冪等性。

常見的反模式：

- 即使不需要，您仍一視同仁地套用冪等性。
- 您導入過於複雜的邏輯來實作冪等性。
- 您使用時間戳記做為冪等性的索引鍵。這樣可能會因為時鐘誤差或多個用戶端使用相同的時間戳記來套用變更，而造成不準確。
- 您為了冪等性而儲存整個承載。採用這個方法時，您會為每個請求儲存完整的資料承載，並在每個新請求中覆寫它。這可能會導致效能降低並影響可擴展性。
- 您在服務之間產生不一致的索引鍵。若索引鍵不一致，服務可能無法辨識重複的請求，進而導致意外的結果。

建立此最佳實務的優勢：

- 可擴展性更大：系統可以處理重試和重複的請求，而不需執行額外的邏輯或複雜的狀態管理。

- 加強可靠性：冪等性可協助服務以一致的方式處理多個相同的請求，進而降低意外副作用或重複記錄的風險。這在經常發生網路故障和重試的分散式系統中尤其重要。
- 改善資料一致性：由於相同的請求會產生相同的回應，因此冪等性有助於在分散式系統之間保持資料一致性。這對於維護交易和操作的完整性相當重要。
- 錯誤處理：冪等性字符讓錯誤處理更簡單。如果用戶端因發生問題而未收到回應，它可以使用相同的冪等性字符安全地重新傳送請求。
- 操作透明度：冪等性可實現更有效的監控與記錄。服務可以透過冪等性字符來記錄請求，這樣更容易追蹤問題並進行偵錯。
- 簡化的 API 合約：它可以簡化用戶端和伺服器端系統之間的合約，並減少對資料處理錯誤的擔憂。

未建立此最佳實務時的曝險等級：中

實作指引

在分散式系統中，最多執行一次動作 (用戶端只發出一個請求) 或至少執行一次動作 (用戶端持續發出請求直到確認成功) 是相對容易的做法。不過，實作精確一次的行為並不容易。若要達成此目的，您的用戶端應針對每個請求產生並提供冪等性字符。

透過使用冪等性字符，服務就能區分新請求和重複的請求。當服務收到具有冪等性字符的請求時，它會檢查字符是否已使用。如果已使用字符，服務就會擷取並傳回預存回應。如果字符是新的，則服務會處理請求、一併儲存回應與字符，然後傳回回應。此機制會讓所有回應變成冪等，進而提高分散式系統的可靠性和一致性。

冪等性也是事件驅動架構的重要行為。這些架構通常由訊息佇列支援，例如 Amazon SQS、Amazon MQ、Amazon Kinesis Streams 或 Amazon Managed Streaming for Apache Kafka (MSK)。在某些情況下，只發布一次的訊息可能會意外傳遞超過一次。當發布者在訊息中產生並包含冪等性字符時，它會要求處理收到的任何重複訊息時，不會對相同訊息產生重複的動作。取用者應追蹤收到的每個字符，並忽略包含重複字符的訊息。

服務與取用者也應將收到的冪等性字符傳遞給其呼叫的任何下游服務。處理鏈中的每個下游服務都同樣負責確保實作冪等性，以免發生處理訊息超過一次的副作用。

實作步驟

1. 識別冪等操作

   確定哪些操作需要冪等性。這些通常包括 POST、PUT 和 DELETE HTTP 方法，以及資料庫插入、更新或刪除操作。不會改變狀態的操作 (例如唯讀查詢) 通常不需要冪等性，除非有副作用。

2. 使用唯一識別碼

   在傳送者傳送的每個冪等操作請求中包含唯一的字符，無論是直接在請求中，還是做為其中繼資料的一部分 (例如 HTTP 標頭)。這可讓接收者辨識和處理重複的請求或操作。常用於字符的識別碼包括通用唯一識別碼 (UUID) 和 K-Sortable Unique Identifiers (KSUID)。

3. 追蹤和管理狀態

   維護工作負載中每個操作或請求的狀態。這可以透過在資料庫、快取或其他持久性存放區中儲存冪等性字符及對應的狀態 (如待處理、已完成或失敗) 來達成。此狀態資訊可讓工作負載識別和處理重複的請求或操作。

   如有需要，可使用適當的並行控制機制來維持一致性和單元性，例如鎖定、交易或積極並行控制。包括記錄冪等性字符和執行所有與處理請求相關聯的改變操作程序。這樣做有助於防止競爭條件，並確認冪等操作正確執行。

   定期從資料儲存區移除舊的冪等性字符，以管理儲存空間和效能。如果您的儲存系統提供支援，請考慮針對資料使用過期時間戳記 (通常稱為存留時間或 TTL 值)。重複使用冪等性字符的可能性會隨著時間而降低。

   通常用於儲存冪等性字符和相關狀態的常見 AWS 儲存選項包括：

   - Amazon DynamoDB：DynamoDB 是一項 NoSQL 資料庫服務，可提供低延遲效能和高可用性，因此非常適合儲存冪等性相關資料。DynamoDB 的索引鍵值和文件資料模型具備高效率儲存和擷取冪等性字符與相關聯狀態資訊的能力。如果您的應用程式在插入時設定 TTL 值，則 DynamoDB 也會自動使冪等性字符過期。
   - Amazon ElastiCache：ElastiCache 可以儲存高輸送量、低延遲且成本低廉的冪等性字符。如果您的應用程式在插入時設定了 TTL 值，則 ElastiCache (Redis) 和 ElastiCache (Memcached) 也會自動讓冪等性字符過期。
   - Amazon Relational Database Service (RDS)：您可以使用 Amazon RDS 來存放冪等性字符和相關的狀態資訊，特別是您的應用程式已將關聯式資料庫用於其他用途時。
   - Amazon Simple Storage Service (S3)：Amazon S3 是高度可擴展且耐用的物件儲存服務，可用於存放冪等性字符和相關的中繼資料。S3 的版本控制功能在維護冪等操作的狀態方面特別實用。選擇儲存服務時，通常取決於以下因素：與冪等性相關的資料量、所需的效能特性、所須耐用性和可用性，以及冪等性機制與整體工作負載架構整合的方式。

4. 實作冪等操作

   將您的 API 和工作負載元件設計為冪等。將冪等性檢查納入工作負載元件中。在您處理請求或執行操作之前，請先檢查唯一識別碼是否已處理。若已處理，則傳回先前的結果，而非再次執行操作。

例如，若用戶端傳送建立使用者的請求，則檢查具有相同唯一識別碼的使用者是否已存在。如果使用者已存在，則應傳回現有的使用者資訊，而非建立新使用者。同樣地，如果佇列取用者收到的訊息中包含重複地冪等性字符，則取用者應忽略訊息。

建立全方位的測試套件，以確認請求的冪等性。這些套件應涵蓋各種不同的案例，例如成功請求、失敗請求和重複請求。

如果您的工作負載利用 AWS Lambda 函數，請考慮 Powertools for AWS Lambda。Powertools for AWS Lambda 是一套開發人員工具組，當您搭配使用 AWS Lambda 函數時，可協助實作無伺服器最佳實務，並加快開發人員速度。它特別提供公用程式將 Lambda 函數轉換為可安全重試的冪等操作。

5. 清楚傳達冪等性

   記錄您的 API 和工作負載元件，以清楚傳達操作的冪等性質。這樣有助於用戶端了解預期的行為，以及如何與您的工作負載進行可靠的互動。

6. 監控和稽核

   實作監控和稽核機制，以偵測任何與回應冪等性相關的問題，例如非預期的回應變化，或過多的重複請求處理。這可協助您偵測和調查工作負載中的任何問題或意外行為。

資源

相關的最佳實務：

- REL05-BP03 控制和限制重試呼叫
- REL06-BP01 監控工作負載的所有元件 (產生)
- REL06-BP03 傳送通知 (即時處理和警示)
- REL08-BP02 將功能測試整合為部署的一部分

相關文件：

- Amazon 建置者資料中心：透過冪等 API 安全地進行重試
- Amazon 建置者資料中心：分散式系統的挑戰
- Amazon 建置者資料中心：可靠性、持續工作以及咖啡時刻
- Amazon Elastic Container Service：確保冪等性
- 如何讓 Lambda 函數冪等？

- 確保 Amazon EC2 API 請求中的冪等性

相關影片：

- 使用事件驅動型架構建置分散式應用程式 - AWS 線上技術講座
- AWS re:Invent 2023 - 使用事件驅動型架構建置新一代應用程式
- AWS re:Invent 2023 - 鬆耦合系統的進階整合模式和權衡
- AWS re:Invent 2023 - 使用 Amazon EventBridge 的進階事件驅動型模式
- AWS re:Invent 2018 - 閉門造車與開放思維：如何取得大小型系統的控制權 (ARC337) (包括鬆耦合、持續工作、靜態穩定性)
- AWS re:Invent 2019：移至事件驅動型架構 (SVS308)

相關工具：

- AWS Lambda Powertools 的冪等性 (Java)
- AWS Lambda Powertools 的冪等性 (Python)
- AWS Lambda Powertools GitHub 頁面

設計分散式系統中的互動以緩解或承受失敗

分散式系統倚賴通訊網路來互連元件 (例如，伺服器或服務)。即使這些網路上的資料遺失或延遲，您的工作負載仍必須可靠運作。分散式系統的元件必須以不會對其他元件或工作負載造成負面影響的方式運作。這些最佳實務可讓工作負載承受壓力或故障，更快速地從其中復原，並減輕這類受損的影響。最終縮短平均復原時間 (MTTR)。

這些最佳實務可防止故障，並改善平均故障間隔時間 (MTBF)。

最佳實務

- REL05-BP01 實作適度降級，以將適用的硬相依性轉換為軟相依性
- REL05-BP02 限流請求
- REL05-BP03 控制和限制重試呼叫
- REL05-BP04 快速失敗並限制佇列
- REL05-BP05 設定用戶端逾時

- REL05-BP06 盡可能讓系統處於無狀態
- REL05-BP07 實作緊急槓桿

REL05-BP01 實作適度降級，以將適用的硬相依性轉換為軟相依性

即使相依性變得不可用，應用程式元件仍應繼續執行其核心功能。它們有可能提供稍微陳舊的資料、備用資料，甚至未提供任何資料。這可確保整體系統運作在本地化失敗時只會受到最低限度的阻礙，同時提供核心商業價值。

預期成果：當元件的相依性狀況不良，元件本身仍可運作，但以降級的方式運作。元件的失敗模式應被視為正常運作。工作流程應適當設計，使此類失敗不會導致完全失敗，或至少會進入可預測和可復原的狀態。

常見的反模式：

- 未識別所需的核心業務功能。即使在相依性失敗期間，也不測試元件是否正常運作。
- 發生錯誤時，或只有多個相依性的其中之一無法使用，且仍可傳回部分結果時，就不提供資料。
- 當交易部分失敗時建立不一致的狀態。
- 沒有替代方法可存取中央參數存放區。
- 因重新整理失敗而使本機狀態失效或清空，而未考量這麼做的後果。

建立此最佳實務的優勢：按正常程序降級可改善系統整體的可用性，並且讓最重要的功能保持運作，即使在失敗期間亦然。

未建立此最佳實務時的曝險等級：高

實作指引

按正常程序實作降級，有助於將相依性失敗對元件功能的影響降到最低。理想情況下，元件會偵測相依性失敗，並以對其他元件或客戶造成最小影響的方式解決這些問題。

按正常程序降級的架構，意味著在相依性設計期間會考量潛在的失敗模式。對於每種失敗模式，都有一種方法可至少將元件最關鍵的功能提供給呼叫者或客戶。這些考量可能會成為可供測試和驗證的其他要求。理想情況下，即使有一或多個相依性失敗，元件仍然能夠以可接受的方式執行其核心功能。

這在商業上和技術上都同樣值得討論。所有業務要求都很重要，都應盡可能地滿足。然而，若無法滿足各項要求將會如何，仍是值得提出的問題。一個系統可以設計成可用且一致的，但在必須放棄一項要

求的情況下，何者較重要？ 對於付款處理，可能應選擇一致性。對於即時應用程式，可能應選擇可用性。對於面向客戶的網站，答案可能取決於客戶的期望。

這意味著什麼，取決於元件的要求，以及應將哪些內容視為其核心功能。例如：

- 電子商務網站可能會顯示來自多個不同系統的資料，例如個人化推薦、排名最高的產品，以及客戶訂單在登陸網頁上的狀態。當一個上游系統失敗時，顯示其他所有內容，而不是向客戶顯示錯誤頁面，仍然是合理的。
- 如果個別作業之一失敗，執行批次寫入的元件仍然可以繼續處理批次。實作重試機制應該要很簡單。為此，您可以向呼叫者傳回關於哪些操作成功、哪些操作失敗及其為何失敗的資訊，或將失敗的請求放入無效字母佇列以實作非同步重試。失敗操作的相關資訊也應記錄下來。
- 處理交易的系統必須確認是否執行了所有更新，或完全未執行更新。對於分佈式交易，可使用 Saga 模式在相同交易的後續操作失敗的情況下回復先前的操作。在此，核心功能保有一致性。
- 具時間性的系統應該能夠處理未及時回應的相依性。在這類情況下，可以使用斷路器模式。若來自相依性的回應開始逾時，系統可以切換到不會進行其他呼叫的關閉狀態。
- 應用程式可從參數存放區讀取參數。使用一組預設的參數建立容器映像，並在參數存放區無法使用時使用這些參數，會很有效用。

請注意，在元件失敗的情況下採取的路徑需進行測試，且應遠比主要途徑簡單。一般來說，應避免使用備用策略。

實作步驟

識別外部和內部相依性。請考量其中可能會發生什麼樣的失敗。思考在這類失敗期間，將上游和下游系統以及客戶受到的負面影響降到最低的方法。

以下列出相依性，並說明如何在其失敗時按正常程序降級：

1. 相依性的部分失敗：一個元件可能會向下游系統提出多個請求，可以是對一個系統的多個請求，或者對多個系統各提出一個請求。視業務環境而定，對此可能會有不同的適當處理方式 (如需詳細資訊，請參閱實作指引中的先前範例)。
2. 下游系統因高負載而無法處理請求：如果對下游系統的請求一直失敗，則繼續重試是沒有意義的。這樣可能會對已過載的系統產生額外的負載，並使復原變得更加困難。此時可以使用斷路器模式，以監控對下游系統的失敗呼叫。若有大量呼叫失敗，將會停止向下游系統傳送更多請求，且偶爾才會讓呼叫通過，以測試下游系統是否已恢復可用性。
3. 參數存放區無法使用：若要轉換參數存放區，可以使用容器或機器映像中包含的軟相依性快取或有效的預設值。請注意，這些預設值需要保持最新狀態，並包含在測試套件中。

4. 監控服務或其他非功能性相依性無法使用：如果元件間歇性地無法傳送記錄、指標或追蹤給中央監控服務，最好還是照常執行業務功能。一般而言，長時間不日誌記錄或推送指標且未顯示任何訊息，是不可接受的。此外，某些使用案例可能需要完整的稽核項目才能滿足合規要求。
5. 關聯式資料庫的主要執行個體可能無法使用：Amazon Relational Database Service 與幾乎所有關聯式資料庫一樣，只能有一個主要寫入器執行個體。這會對寫入工作負載造成單一失敗點，並使擴展變得更加困難。透過使用多可用區域組態以獲得高可用性，或使用 Amazon Aurora Serverless 以獲得更好的擴展性，可以減輕部分問題。對於非常高的可用性要求，完全不依賴主要寫入器是有效用的。對於唯讀的查詢可以使用讀取複本，以提供備援和橫向擴展的能力，而不僅僅是縱向擴展。寫入可以緩衝處理 (例如，在 Amazon Simple Queue Service 佇列中)，如此，即使主要寫入器暫時無法使用，仍然可以接受客戶的寫入請求。

資源

相關文件：

- Amazon API Gateway：調節 API 請求以獲得更佳的輸送量
- CircuitBreaker (摘要說明「Release It!」書籍中的斷路器)
- AWS 中的錯誤重試與指數退避
- Michael Nygard “Release It! Design and Deploy Production-Ready Software”
- Amazon 建置者資料中心：避免分散式系統的備用
- Amazon 建置者資料中心：避免無法逾越的佇列待辦項目
- Amazon 建置者資料中心：快取挑戰和策略
- Amazon 建置者資料中心：逾時、重試、退避與抖動

相關影片：

- 重試、退避和抖動：AWS re:Invent 2019：Amazon 建置者資料中心簡介 (DOP328)

REL05-BP02 限流請求

限流請求以減輕因需求非預期地增加而耗盡資源。系統會處理低於限流速率的請求，並拒絕超過定義限制的請求，且傳回一則訊息，指出請求已遭到限流。

預期成果：由於客戶流量突然增加、洪水攻擊或重試風暴而造成的大量尖峰，可透過請求限流來緩解，讓工作負載能夠繼續正常處理支援的請求量。

常見的反模式：

- API 端點限流未實作或保留為預設值，而未考量預期的數量。
- API 端點未經過負載測試，或未測試限流限制。
- 限流請求率，而不考量請求大小或複雜性。
- 測試請求率上限或請求大小上限，但不同時測試兩者。
- 資源不會佈建在測試時建立的相同限制。
- 未針對應用程式對應用程式 (A2A) API 取用者設定或考量使用計畫。
- 水平擴展的佇列取用者未進行最大並行設定。
- 未實作個別 IP 位址的速率限制。

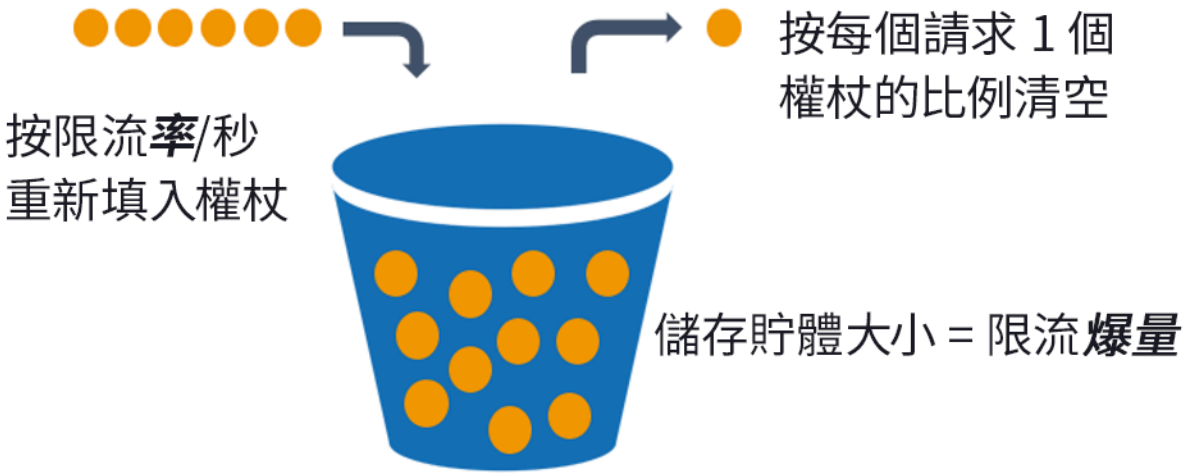
建立此最佳實務的優勢：設定限流限制的工作負載能夠在非預期的數量尖峰情況下正常運作，並成功處理已接受的請求負載。API 和佇列的請求若突然或持續激增將會受到限制，且不會耗盡請求處理資源。速率限制會限流個別請求者，使來自單一 IP 位址或 API 取用者的大量流量不會耗盡資源而影響到其他取用者。

未建立此最佳實務時的曝險等級：高

實作指引

服務應設計為處理已知的請求容量；此容量可透過負載測試來建立。如果請求到達率超過限制，會有適當的回應訊號指出請求已受到限流。這可讓取用者處理錯誤並於稍後重試。

當您的服務需要限流實作時，請考慮實作權杖儲存貯體演算法 (在此演算法中，權杖對於請求具重要性)。權杖會按每秒的限流率重新填入，並依照每個請求一個權杖的比例非同步清空。



權杖儲存貯體演算法。

Amazon API Gateway 會根據帳戶和區域限制實作權杖儲存貯體演算法，並且可根據使用計畫對個別用戶端進行設定。此外，Amazon Simple Queue Service (Amazon SQS) 和 Amazon Kinesis 可以緩衝請求以平滑請求速率，並為可解決的請求提供更高的限流。最後，可以使用 AWS WAF 實作速率限制，以針對會產生異常高負載的特定 API 取用者進行限流。

實作步驟

可以使用 API 的限流限制來設定 API Gateway，並在超出限制時傳回 `429 Too Many Requests` 錯誤。可以搭配使用 AWS WAF 與 AWS AppSync 和 API Gateway 端點，以按照 IP 位址啟用速率限制。此外，如果您的系統可容忍非同步處理，您可以將訊息放入佇列或串流中，以加快對服務用戶端的回應速度，進而提升到更高的限流率。

透過非同步處理，當您將 Amazon SQS 設定為 AWS Lambda 的事件來源時，可以設定並行上限，以避免高事件率消耗工作負載或帳戶中其他服務所需的可用帳戶並行執行配額。

雖然 API Gateway 提供了權杖儲存貯體的受管實作，在您無法使用 API Gateway 的情況下，您可以對服務使用權杖儲存貯體的語言特定開放原始碼實作 (請參閱「資源」中的相關範例)。

- 了解並設定每個區域帳戶層級的 API Gateway 節流限制、每個階段的 API 以及每個使用方案層級的 API 金鑰。
- 將 AWS WAF 速率限制規則套用至 API Gateway 和 AWS AppSync 端點，以防止泛洪並封鎖惡意 IP。此外也可在 A2A 取用者的 AWS AppSync API 金鑰上設定速率限制規則。
- 考慮您是否需要比 AWS AppSync API 的速率限制更高的限流控制，如果需要，請在 AWS AppSync 端點前面設定 API Gateway。
- 將 Amazon SQS 佇列設定為 Lambda 佇列取用者的觸發器時，請將並行上限設定為足以符合服務層級目標的值，但不會消耗影響其他 Lambda 函數的並行限制。當您透過 Lambda 使用佇列時，請考慮在相同帳戶和區域中的其他 Lambda 函數上設定預留並行。
- 使用 API Gateway 與 Amazon SQS 或 Kinesis 的原生服務整合來緩衝請求。
- 如果您無法使用 API Gateway，請查看語言特定程式庫，為您的工作負載實作權杖儲存貯體演算法。查看範例區段並自行研究，以尋找合適的程式庫。
- 測試您預計要設定的限制，或您打算允許增加的限制，並記錄已測試的限制。
- 請勿將限制提高到您在測試時建立的範圍外。增加限制時，請先確認佈建的資源已等同於或大於測試情境中的資源，然後再套用增加。

資源

相關的最佳實務：

- REL04-BP03 持續工作
- REL05-BP03 控制和限制重試呼叫

相關文件：

- Amazon API Gateway：調節 API 請求以獲得更佳的輸送量
- AWS WAF：以速率為基礎的規則陳述式
- 導入使用 Amazon SQS 作為事件來源時的 AWS Lambda 並行上限
- AWS Lambda：並行上限

相關範例：

- 三項最重要的 AWS WAF 以速率為基礎的規則
- Java Bucket4j
- Python 權杖儲存貯體
- 節點權杖儲存貯體
- .NET 系統執行緒速率限制

相關影片：

- 使用 AWS AppSync 實作 GraphQL API 安全最佳實務

相關工具：

- Amazon API Gateway
- AWS AppSync
- Amazon SQS
- Amazon Kinesis
- AWS WAF
- AWS 上的虛擬等候室

REL05-BP03 控制和限制重試呼叫

使用指數退避，在每次重試之間的間隔逐漸拉長後重試請求。在重試之間導入抖動以隨機化重試間隔。限制重試次數上限。

預期結果：分散式軟體系統中的典型元件包括伺服器、負載平衡器、資料庫和DNS伺服器。在正常操作期間，這些元件可以回應具有暫時性或有限錯誤的請求，以及無論是否重試都將持續存在的錯誤。當用戶端向服務發出請求時，請求會取用資源，包括記憶體、執行緒、連線、連接埠，或任何其他有限的資源。控制和限制重試是釋出資源並將資源耗用量降到最低的策略，可讓處於壓力下的系統元件不致不堪負荷。

當用戶端請求逾時或收到錯誤回應時，他們應該判斷是否要重試。如果執行重試，他們會使用具有抖動和最大重試值的指數退避來執行此作業。因此，後端服務和程序從負載中得到緩解並獲得自我修復的時間，進而更快速地復原和提供成功的請求服務。

常見的反模式：

- 在未新增指數退避、抖動和最大重試值的情況下實作重試。退避和抖動有助於避免因為在共用間隔內無意間進行協調重試而產生人為流量尖峰。
- 實作重試，而不測試其效果，或假設重試已內建到 中，SDK而不測試重試案例。
- 未能了解從相依性發布的錯誤代碼，因而重試了所有錯誤，包括有明確原因指出缺少權限的錯誤、組態錯誤，或其他預期必須要手動干預才能解決的狀況。
- 未解決可觀測性實務，包括對重複的服務失敗進行監控和提醒，使基礎問題廣為人知並且可以解決。
- 在內建或第三方重試功能堪用時，開發自訂重試機制。
- 以複合重試的方式在應用程式堆疊的多個層級重試，會嘗試在重試風暴中進一步耗用資源。請務必了解這些錯誤如何影響您所依賴的應用程式，然後僅在一個層級實作重試。
- 重試不是等冪的服務呼叫，導致非預期的副作用，例如重複的結果。

建立此最佳實務的優勢：重試可協助用戶端在請求失敗時獲得所需的結果，但也會耗用更多伺服器的時間來取得他們想要的成功回應。若失敗是罕見或暫時性的，重試可以有效運作。若失敗是由資源超載引起的，重試可能會使情況變得更糟。在用戶端重試中新增具有抖動的指數退避，可讓伺服器在資源超載導致失敗時進行復原。抖動可避免將請求對應到尖峰，而退避會減少將重試新增至正常請求負載所造成的負載上升。最後，請務必設定最大重試次數或經過時間，以避免建立會產生亞穩態失敗的積存。

未建立此最佳實務時的曝險等級：高

實作指引

控制和限制重試呼叫。使用指數退避以在逐漸延長間隔後重試。引進抖動來隨機化重試間隔，並限制重試次數上限。

有些 AWS SDKs依預設會實作重試和指數退避。在工作負載中適用的情況下，使用這些內建 AWS 實作。在呼叫等冪的服務時，以及重試可改善用戶端可用性時，在您的工作負載中實作類似的邏輯。根據您的使用案例確定逾時時間以及何時停止重試。為那些重試使用案例建置和模擬演練測試情境。

實作步驟

- 確認應用程式堆疊中的最佳層級，以針對您應用程式所依賴的服務實作重試。
- 請注意，現有的 SDKs 會針對您選擇的語言實作經過驗證的重試策略，並具有指數退避和抖動，並且比編寫您自己的重試實作更有利。
- 在實作重試之前，請確認服務是冪等的。實作重試後，請務必在生產環境中加以測試和定期模擬演練。
- 呼叫 AWS 服務 時APIs，請使用 AWS SDKs和 AWS CLI並了解重試組態選項。確認預設值是否適用於您的使用案例，並視需要進行測試和調整。

資源

相關的最佳實務：

- REL04-BP04 讓變異操作冪等
- REL05-BP02 限流請求
- REL05-BP04 快速失敗並限制佇列
- REL05-BP05 設定用戶端逾時
- REL11-BP01 監控工作負載的所有元件以偵測故障

相關文件：

- 中的錯誤重試和指數退避 AWS
- Amazon 建置者資料中心：逾時、重試、退避與抖動
- 指數退避和抖動
- 以無能方式確保重試安全 APIs

相關範例：

- Spring 重試
- Resilience4j 重試

相關影片：

- 重試、退避和抖動： AWS re：Invent 2019：介紹 Amazon Builders 程式庫（DOP328）

相關工具：

- AWS SDKs 和 工具：重試行為
- AWS Command Line Interface： AWS CLI 重試

REL05-BP04 快速失敗並限制佇列

在服務無法成功回應請求時快速檢錯。如此將可釋出與請求關聯的資源，並且使服務可在資源用盡時復原。快速檢錯是一種完善的軟體設計模式，可用來在雲端中建置高度可靠的工作負載。佇列也是一種完善的企業整合模式，可以平滑負載，並且讓用戶端在可容忍非同步處理時釋出資源。如果服務在正常情況下能夠成功回應，但在請求速率太高時會失敗，請使用佇列來緩衝請求。不過，請勿允許建置長佇列積存，這可能導致用戶端已放棄的過時請求受到處理。

預期成果：當系統遇到資源爭用、逾時、例外狀況或灰色失敗而使服務水準目標無法達成時，快速檢錯的策略可加快系統復原速度。必須吸納流量尖峰且能支應非同步處理的系統，可讓用戶端使用佇列緩衝處理後端服務的請求以快速釋出請求，藉此提升可靠性。緩衝處理要排入佇列的請求時，會實作佇列管理策略，以避免發生無法克服的積存。

常見的反模式：

- 實作訊息佇列，但不在DLQ磁碟區上設定無效字母佇列（DLQ）或警示，以偵測系統何時故障。
- 未測量訊息在佇列中的存留期，這是一種延遲測量，用以了解佇列取用者何時進度落後或發生錯誤導致重試。
- 當處理這些訊息沒有任何價值，且業務需求已不存在時，未清除佇列中已積存的訊息。
- 當最後的先出（FIFO）佇列更能滿足用戶端需求時，設定先出（LIFO）佇列，例如，當不需要嚴格排序，且待辦項目處理正在延遲所有新的和時間敏感請求，導致所有用戶端發生違反的服務層級時。
- 將內部佇列暴露至用戶端APIs，而不是將管理工作接收並將請求放置在內部佇列中的暴露。

- 藉由將資源需求分攤到不同請求型態，將過多的工作請求類型合併到可能加劇積存條件的單一佇列中。
- 儘管需要不同的監控、逾時和資源分配，仍在同一佇列中處理複雜而簡單的請求。
- 不驗證輸入或使用判斷提示在軟體中實作快速檢錯的機制，以對可用正常程序處理錯誤的較高層級元件快顯例外狀況。
- 不會從請求路由中移除錯誤的資源，特別是在失敗處於灰色地帶，因損毀和重新啟動、間歇性相依性失敗、容量減少或網路封包遺失而導致成功與失敗並存。

建立此最佳實務的優勢：快速檢錯的系統更容易偵錯和修正，並且在發布至生產環境之前，常會出現編碼和組態方面的問題。納入有效佇列策略的系統，可針對流量尖峰和間歇性系統失敗狀況提供更高的恢復能力和可靠性。

未建立此最佳實務時的曝險等級：高

實作指引

快速檢錯的策略可以編碼為軟體解決方案，並設定到基礎設施中。除了快速檢錯以外，佇列也是一種簡單而強大的架構技術，可將系統元件平滑負載分離。Amazon CloudWatch 提供功能來監控故障並發出警示。已知系統失敗時，可以調用緩解策略，包括背離受損的資源。當系統使用 Amazon SQS 和其他佇列技術實作佇列以順利載入時，他們必須考慮如何管理佇列待處理項目，以及訊息消耗失敗。

實作步驟

- 在軟體中實作程式化判斷提示或特定指標，並使用這些提示或指標來明確提醒系統問題。Amazon CloudWatch 可協助您根據應用程式日誌模式和SDK儀器建立指標和警示。
- 使用 CloudWatch 指標和警示來避免增加處理延遲或重複處理請求的受損資源。
- 使用非同步處理，方法是設計APIs接受請求，並使用 Amazon 將請求附加到內部佇列，SQS然後使用成功訊息回應訊息產生用戶端，以便用戶端可以在後端佇列取用者處理請求時釋出資源並繼續進行其他工作。
- 測量和監控佇列處理延遲，方法是在每次從佇列中除去訊息時產生 CloudWatch 指標，方法是立即與訊息時間戳記進行比較。
- 因失敗而無法成功處理訊息，或無法在服務水準協議內處理磁碟區中的流量尖峰時，請將較舊或過多的流量排除至溢滿佇列。這可讓您優先處理新工作，並且等到有可用的容量時再處理較舊的工作。此技術是LIFO處理方法的近似值，並允許所有新工作的正常系統處理。
- 使用無效字母或重新驅動佇列，將無法處理的訊息從積存移到可稍後再研究和解析的位置

- 進行重試，或在可接受的情況下，藉由比較目前時間與訊息時間戳記，捨棄與請求用戶端不再相關的訊息，將舊訊息捨棄。

資源

相關的最佳實務：

- REL04-BP02 實作鬆散耦合相依性
- REL05-BP02 限流請求
- REL05-BP03 控制和限制重試呼叫
- REL06-BP02 定義和計算指標 (彙總)
- REL06-BP07 監控 end-to-end透過您的系統追蹤請求

相關文件：

- 避免無法處理的佇列積存
- 快速檢錯
- 如何防止 Amazon SQS佇列中訊息的待處理項目增加？
- Elastic Load Balancing：區域轉移
- Amazon Application Recovery Controller：流量容錯移轉的路由控制

相關範例：

- 企業整合模式：無效字母通道

相關影片：

- AWS re：Invent 2022 - 操作高可用性的多可用區域應用程式

相關工具：

- Amazon SQS
- Amazon MQ
- AWS IoT Core
- Amazon CloudWatch

REL05-BP05 設定用戶端逾時

在連線和請求上妥善設定逾時、有系統地對其進行驗證，並且不要依賴預設值，因為它們不知道工作負載具體細節。

預期成果：用戶端逾時應考量與等待需要花費異常時間才能完成的請求相關的用戶端、伺服器和工作負載成本。由於無法知道任何逾時的確切原因，用戶端必須使用服務知識來找出對可能原因和適當逾時的期望

用戶端連線根據設定的值逾時。經歷逾時後，用戶端決定退回並重試，或開啟斷路器。這些模式可避免發出可能使基礎錯誤情況惡化的請求。

常見的反模式：

- 不知道系統逾時或預設逾時。
- 不知道正常的請求完成時間。
- 不知道完成請求異常耗時的可能原因，或是與等待這些作業完成相關聯的用戶端、服務或工作負載效能成本。
- 不知道受損的網路只有在達到逾時後才會造成請求失敗的可能性，以及未採用較短逾時的用戶端和工作負載效能的成本。
- 不測試連線和請求的逾時情境。
- 將逾時設定得太高，這可能會導致較長的等待時間，並增加資源使用率。
- 將逾時設定得太低，導致人為失敗。
- 忽略模式以處理遠端呼叫 (例如斷路器和重試) 的逾時錯誤。
- 不考慮監控服務呼叫錯誤率、延遲的服務水準目標，以及延遲離群值。這些指標可提供對積極或寬鬆逾時的洞見

建立此最佳實務的優勢：遠端呼叫逾時已設定，且系統設計為按正常程序處理逾時，以便在遠端呼叫回應異常緩慢，而逾時錯誤由服務用戶端正常處理時，可以保留資源。

未建立此最佳實務時的曝險等級：高

實作指引

針對任何服務相依性呼叫和任何跨程序的呼叫，同時設定連線逾時和請求逾時。許多架構都提供內建的逾時功能，但請注意，對您的服務目標而言，有些架構具有無限或過高的預設值。太高的值會降低逾時的實用性，因為當用戶端等待逾時發生時，資源會持續耗用。太低的值可能會增加後端流量和延遲，原因是重試的請求過多。在某些情況下，這可能導致完全停機，原因是正在重試所有請求。

決定逾時策略時，請考量下列事項：

- 由於請求的內容、目標服務受損或聯網分割失敗，處理請求的時間可能會比平常更長。
- 內容異常昂貴的請求可能會耗用不必要的伺服器和用戶端資源。在此情況下，讓這些請求逾時而不重試，可以保留資源。服務也應透過限流和伺服器端逾時，來保護自己免受異常昂貴的內容影響。
- 因服務受損而異常耗時的請求可能會逾時並重試。應考量請求和重試的服務成本，但如果原因是當地語系化的損害，則重試應該不會很昂貴，而且將可降低用戶端資源耗用量。逾時也可能會根據損害的性質釋出伺服器資源。
- 因網路傳遞請求或回應失敗而需要長時間才能完成的請求，可能會逾時並重試。由於請求或回應未傳遞，因此無論逾時長度為何，結果都是失敗。在此情況下，逾時不會釋出伺服器資源，但會釋出用戶端資源並改善工作負載效能。

利用如重試和斷路器等建立良好的設計模式，優雅地處理逾時，並支援快速失敗的方法。AWS SDKs 和 AWS CLI 允許設定連線和請求逾時，以及使用指數退避和抖動的重試。 AWS Lambda函數支援逾時組態，而使用 AWS Step Functions，您可以建置低程式碼斷路器，以利用與服務 AWS 和 預先建置的整合SDKs。 AWS App MeshEnvoy 提供逾時和斷路器功能。

實作步驟

- 設定遠端服務呼叫的逾時，並利用內建的語言逾時功能或開放原始碼逾時程式庫。
- 當您的工作負載使用 呼叫 時 AWS SDK，請檢閱文件，了解語言特定的逾時組態。
  - Python
  - PHP
  - .NET
  - Ruby
  - Java
  - Go
  - Node.js
  - C++
- 在工作負載中使用 或 AWS CLI 命令時 AWS SDKs，請設定 `connectTimeoutInMillis`和 的 AWS 組態預設值，以設定預設逾時值`tlsNegotiationTimeoutInMillis`。
- 將命令列選項 `cli-connect-timeout`和 套用至 AWS 服務`cli-read-timeout`，以控制一次性 AWS CLI 命令。
- 監控遠端服務呼叫是否有逾時，並對持續性錯誤設定警示，以便您可以主動處理錯誤案例。

- 對呼叫錯誤率、延遲的服務層級目標和延遲異常值實作CloudWatch 指標和CloudWatch 異常偵測，以深入了解如何管理過度激進或寬鬆的逾時。
- 設定 Lambda 函數的逾時。
- API Gateway 用戶端在處理逾時時必須實作自己的重試。API Gateway 支援下游整合的 50 毫秒至 29 秒整合逾時，在整合請求逾時時不會重試。
- 實作斷路器模式，以避免在逾時發生時進行遠端呼叫。開啟線路以避免呼叫失敗，並在呼叫正常回應時關閉線路。
- 對於基於容器的工作負載，請查看 App Mesh Envoy 功能以利用內建的逾時和斷路器。
- 使用 AWS Step Functions 建置用於遠端服務呼叫的低程式碼斷路器，特別是在呼叫 AWS 原生 SDKs和支援的 Step Functions 整合時，以簡化工作負載。

資源

相關的最佳實務：

- REL05-BP03 控制和限制重試呼叫
- REL05-BP04 快速失敗並限制佇列
- REL06-BP07 監控 end-to-end透過您的系統追蹤請求

相關文件：

- AWS SDK：重試和逾時
- Amazon 建置者資料中心：逾時、重試、退避與抖動
- Amazon API Gateway 配額和重要備註
- AWS Command Line Interface：命令列選項
- AWS SDK for Java 2.x：設定API逾時
- AWS 使用組態物件和組態參考的 Botocore
- 適用於 .NET 的 AWS SDK：重試與逾時
- AWS Lambda：設定 Lambda 函數選項

相關範例：

- 搭配 AWS Step Functions 和 Amazon DynamoDB 使用斷路器模式
- Martin Fowler： CircuitBreaker

相關工具：

- AWS SDKs
- AWS Lambda
- Amazon SQS
- AWS Step Functions
- AWS Command Line Interface

REL05-BP06 盡可能讓系統處於無狀態

系統不應要求狀態，或應該卸載狀態，以便在不同的用戶端請求之間，不依賴磁碟和記憶體中本機儲存的資料。這允許伺服器任意置換，而不會對可用性造成影響。

當使用者或服務與應用程式互動時，他們通常會執行形成工作階段的一系列互動。工作階段是使用者在使用應用程式時，在不同請求之間持續存在的唯一資料。無狀態應用程式是一種不需要了解先前互動，也不會儲存工作階段資訊的應用程式。

一旦設計為無狀態，您就可以使用無伺服器運算服務，例如 AWS Lambda 或 AWS Fargate。

除了伺服器替換之外，無狀態應用程式的另一個優點是他們可以水平擴展，因為任何可用的運算資源（例如EC2執行個體和 AWS Lambda 函數） 都可以服務任何請求。

建立此最佳實務的優勢：設計為無狀態的系統更適合水平擴展，因此可以根據波動的流量和需求來新增或移除容量。其本質上也具有抵抗故障的能力，並在應用程式開發中提供靈活性和敏捷性。

未建立此最佳實務時的曝險等級：中

實作指引

讓您的應用程式無狀態。無狀態應用程式支援水平擴展，並且可以容忍單個節點的失敗。分析並了解在架構中維持狀態的應用程式元件。這可協助您評估轉換為無狀態設計的潛在影響。無狀態架構會分離使用者資料並卸載工作階段資料。這提供了獨立擴展每個元件的彈性，以滿足不同的工作負載需求，並最佳化資源使用率。

實作步驟

- 識別並了解應用程式中的有狀態元件。
- 透過將使用者資料與核心應用程式邏輯進行分離和管理來解耦資料。

  - Amazon Cognito 可以使用身分池、使用者集區和 Amazon Cognito Sync 等功能，將使用者資料與應用程式的程式碼分離。
  - 可以將密碼儲存在安全的集中位置，使用 AWS Secrets Manager 來分離使用者資料。這意味著應用程式的程式碼不需要存儲密碼，這使得它更安全。
  - 請考慮使用 Amazon S3 來存放大型非結構化資料，例如影像和文件。應用程式可以在需要時擷取此資料，而無需將其存儲在記憶體中。
  - 使用 Amazon DynamoDB 來存放使用者設定檔等資訊。應用程式可以近乎即時的速度查詢這些資料。
- 將工作階段資料卸載至資料庫、快取或外部檔案。
  - Amazon ElastiCache、Amazon DynamoDB 、Amazon Elastic File System （Amazon EFS） 和 Amazon MemoryDB 是可用來卸載工作階段資料 AWS 的服務範例。
- 在確定需要使用所選儲存解決方案維持哪些狀態和使用者資料之後，設計一個無狀態架構。

資源

相關的最佳實務：

- REL11-BP03 在所有圖層上自動復原

相關文件：

- Amazon 建置者資料中心：避免分散式系統的備用
- Amazon 建置者資料中心：避免無法逾越的佇列待辦項目
- Amazon 建置者資料中心：快取挑戰和策略
- 上的無狀態 Web 層最佳實務 AWS

REL05-BP07 實作緊急槓桿

緊急控制桿是可緩解工作負載所受之可用性影響的快速程序。

緊急控制桿的運作方法是使用已知且經過測試的機制來停用、限流或變更元件或相依性的行為。這可以減輕因需求意外增加導致資源耗盡所造成的工作負載受損，並降低工作負載內非關鍵元件的故障影響。

預期成果：透過實作緊急控制桿，可以建立已知的良好流程，以維持工作負載中關鍵元件的可用性。在啟用緊急控制桿期間，工作負載應該會適度降級，並繼續執行其業務關鍵功能。如需優雅降級的詳細資訊，請參閱 REL05-BP01 實作優雅降級，將適用的硬相依性轉換為軟相依性 。

常見的反模式：

- 非關鍵相依性失敗會影響核心工作負載的可用性。
- 未在非關鍵元件受損期間測試或驗證關鍵元件的行為。
- 沒有為啟用或停用緊急控制桿定義明確且決定性的準則。

建立此最佳實務的優勢：實作緊急控制桿可透過為解析程式提供已確立的程序來應對意外的需求峰值或非關鍵相依性失敗，從而提高工作負載中關鍵元件的可用性。

未建立此最佳實務時的曝險等級：中

實作指引

- 識別工作負載中的關鍵元件。
- 設計和建構工作負載中的關鍵元件，以承受非關鍵元件的故障。
- 進行測試以驗證非關鍵元件失敗期間您關鍵元件的行為。
- 定義和監控相關指標或觸發器以啟動緊急控制桿程序。
- 定義構成緊急控制桿的程序 (手動或自動)。

實作步驟

- 識別工作負載中的業務關鍵元件。
  - 工作負載中的每個技術元件應對應到其相關業務功能，並將其排名為關鍵或非關鍵。如需 Amazon 關鍵和非關鍵功能的範例，請參閱 Any Day Can Be Prime Day: How Amazon.com Search Uses Chaos Engineering to Handle Over 84K Requests Per Second。
  - 這同時是技術和業務方面的決策，並且會因組織和工作負載而異。
- 設計和建構工作負載中的關鍵元件，以承受非關鍵元件的故障。
  - 在相依性分析期間，請考慮所有潛在的故障模式，並驗證您的緊急控制桿機制能為下游元件提供關鍵功能。
- 進行測試以驗證緊急控制桿啟動期間您關鍵元件的行為。
  - 避免雙模態行為。如需更多詳細資訊，請參閱 REL11-BP05 使用靜態穩定性來防止雙模式行為 。
- 定義、監控和警示相關指標，以啟動緊急控制桿程序。
  - 尋找適合監控的指標取決於您的工作負載。一些範例指標是延遲或失敗的相依性請求次數。
- 定義構成緊急控制桿的程序 (手動或自動)。

- 這可能包括諸如降載、限流請求或實作適度降級等機制。

資源

相關的最佳實務：

- REL05-BP01 實作優雅降級，將適用的硬相依性轉換為軟相依性
- REL05-BP02 節流請求
- REL11-BP05 使用靜態穩定性來防止雙模式行為

相關文件：

- 自動化安全、無人為介入的部署
- 任何一天都可能是黃金日：Amazon.com 搜尋功能如何使用混沌工程每秒處理 84K 以上的請求

相關影片：

- AWS re：Invent 2020：透過不可變性的可靠性、一致性和信心

變更管理

必須預期並因應工作負載或其環境的變更，才能實現可靠的工作負載操作。變更包括對工作負載強加的變更，例如需求峰值，以及內部的變更，例如功能部署和安全性修補程式。

下列各節說明變更管理的最佳實務。

主題

- 監控工作負載資源
- 設計工作負載以適應需求變更
- 實作變更

監控工作負載資源

日誌和指標是可深入洞察工作負載運作狀態的強大工具。您可以設定工作負載以監控日誌和指標，並在超過閾值或發生重大事件時傳送通知。監控可讓您的工作負載識別何時會超過低效能閾值或發生故障，以便自動復原來回應。

監控是確保您會滿足可用性要求的關鍵步驟。需高效監控，以偵測故障。最糟糕的失敗模式是「沉默」失敗，在這種情況下，功能不再發揮功用，但除了間接處理之外，無法偵測到該問題。您的客戶比您知悉的還要早。提醒問題出現的時間，是您監控的主要原因之一。您的提醒應盡量與您的系統解偶。若服務中斷讓您無法接收提醒，您的中斷期會延長。

在 AWS，我們在多個層級進行應用程式偵測。我們會記錄每個請求、所有相依性及流程中關鍵營運的延遲、錯誤率和可用性。我們還記錄成功營運的指標。這樣一來，我們就能在問題即將發生之前加以預防。我們不只考量平均延遲。我們更專注於延遲異常值，例如第 99.9 和 99.99 個百分位數。這是因為如果 1,000 或 10,000 中的一個請求進行緩慢，這仍是個差勁的體驗。此外，雖然您的平均值是可接受的，但如果 100 個請求中有一個造成極端延遲，最終會在流量增加時變成問題。

AWS 監控包含四個不同的階段：

1. 產生 – 監控工作負載的所有元件
2. 彙總 – 定義和計算指標
3. 即時處理和警示 – 傳送通知並將回應自動化
4. 儲存與分析

最佳實務

- REL06-BP01 監控工作負載的所有元件 (產生)
- REL06-BP02 定義和計算指標 (彙總)
- REL06-BP03 傳送通知 (即時處理和警示)
- REL06-BP04 自動化回應 (即時處理和警示)
- REL06-BP05 分析日誌
- REL06-BP06 定期審查監控範圍和指標
- REL06-BP07 監控 end-to-end透過您的系統追蹤請求

REL06-BP01 監控工作負載的所有元件 (產生)

使用 Amazon CloudWatch 或第三方工具監控工作負載的元件。使用 AWS Health 儀表板監控 AWS 服務。

工作負載的所有元件都應該受到監控，包括前端、商業邏輯和儲存層。定義關鍵指標，描述如何從日誌中擷取指標 (如果需要)，並設定調用對應警示事件的閾值。確保指標與工作負載的關鍵績效指標 (KPI) 相關，並使用指標和日誌來識別服務降級的早期警告訊號。例如，與業務成果相關的指標 (例如每分鐘成功處理的訂單數量) 可以比技術指標 (例如 CPU 使用率) 更快地指出工作負載問題。使用 AWS Health 儀表板可針對 AWS 資源下的 AWS 服務的效能和可用性，為您提供個人化檢視。

雲端監控提供新機遇。大多數雲端供應商都開發了可自訂的勾點，並且可以提供深入解析，協助您監控工作負載的多個層面。Amazon CloudWatch 等 AWS 服務會在使用者介入程度最低的情況下，套用統計和機器學習演算法來持續分析系統和應用程式的指標、判斷正常基準以及披露異常情況。異常偵測演算法會考慮指標的季節性和趨勢變化。

AWS 提供大量可供使用的監控和日誌資訊，可用於定義工作負載特定的指標、需求變更程序，並採用機器學習技術，而不論機器學習專業知識如何。

此外，監控所有外部端點，以確保它們獨立於基本實作。此主動監控可透過綜合交易 (有時稱為使用者 Canary，但請別與 Canary 部署混淆) 加以完成，它會定期運行工作負載的用戶端執行的許多常見任務匹配動作。在持續時間中讓這些任務保持簡單扼要，並確定在測試期間不會讓工作負載超載。Amazon CloudWatch Synthetics 讓您能夠建立綜合性 Canary，以監控您的端點和 API。您也可以將綜合性 Canary 用戶端節點與 AWS X-Ray 主控台結合，以指出綜合性 Canary 在所選時段內發生錯誤、故障或調節率等問題。

預期成果：

從工作負載的所有元件中收集並使用關鍵指標，以確保工作負載的可靠性和最佳的使用者體驗。偵測工作負載未達成業務成果，可讓您快速宣告災難並從事件中復原。

常見的反模式：

- 僅監控工作負載的外部界面。
- 不產生任何工作負載特定指標，只依賴工作負載使用的 AWS 服務提供給您的指標。
- 僅在工作負載中使用技術指標，而不監控與工作負載貢獻的非技術 KPI 相關的任何指標。
- 依賴生產流量和簡單的運作狀態檢查來監控和評估工作負載狀態。

建立此最佳實務的優勢：在工作負載中的所有層級進行監控，可讓您更快速地預測和解決構成工作負載的元件中的問題。

未建立此最佳實務時的曝險等級：高

實作指引

1. 在可用的地方開啟日誌記錄。應從工作負載的所有元件中取得監控資料。開啟其他日誌記錄，例如 S3 Access Logs，並允許您的工作負載記錄工作負載特定資料。從 Amazon ECS、Amazon EKS、Amazon EC2、Elastic Load Balancing、AWS Auto Scaling 和 Amazon EMR 等服務中收集 CPU、網路 I/O 和磁碟 I/O 平均值的指標。如需將指標發佈到 CloudWatch 的 AWS 服務清單，請參閱 AWS Services That Publish CloudWatch Metrics。
2. 審核所有預設指標，並探索任何資料收集漏洞。每個服務都會產生預設指標。收集預設指標可讓您進一步了解工作負載元件之間的相依性，以及元件可靠性和效能如何影響工作負載。您也可以使用 AWS CLI 或 API 建立自己的自訂指標並發佈到 CloudWatch。
3. 評估所有指標，以決定在工作負載中為每個 AWS 服務提醒哪些指標。您可以進行選擇，以選取對工作負載可靠性有重大影響的指標子集。專注於重要指標和閾值，可讓您調整提醒的數量，並協助將誤報率降至最低。
4. 在調用提醒後，定義工作負載的提醒和復原程序。定義提醒可讓您快速通知、呈報並遵循必要的步驟，以便從事件中復原並實現規定的復原時間點目標 (RTO)。可以使用 Amazon CloudWatch 警示來調用自動化工作流程，並根據定義的閾值啟動復原程序。
5. 探索如何使用綜合交易來收集有關工作負載狀態的相關資料。綜合監控遵循相同的路由並執行與客戶相同的動作，即使您的工作負載沒有任何客戶流量，也能持續驗證您的客戶體驗。透過使用綜合交易，您可以在客戶之前發現問題。

資源

相關的最佳實務：

- REL11-BP03 將所有分層的修復自動化

相關文件：

- 開始使用 AWS Health 儀表板 — 您的帳戶運作狀態
- 發佈 CloudWatch 指標的 AWS 服務
- Network Load Balancer 的存取日誌
- Application Load Balancer 的存取日誌
- 存取 AWS Lambda 的 Amazon CloudWatch Logs
- Amazon S3 伺服器存取日誌記錄
- 啟用 Classic Load Balancer 的存取日誌
- 將日誌資料匯出至 Amazon S3
- 在 Amazon EC2 執行個體上安裝 CloudWatch Agent
- 發佈自訂指標
- 使用 Amazon CloudWatch 儀表板
- 使用 Amazon CloudWatch 指標
- 使用 Canary (Amazon CloudWatch Synthetics)
- 什麼是 Amazon CloudWatch Logs？

使用者指南：

- 建立追蹤
- 監控 Amazon EC2 Linux 執行個體的記憶體和磁碟指標
- CloudWatch Logs 與容器執行個體搭配使用
- VPC 流程日誌
- 什麼是 Amazon DevOps Guru？
- 什麼是 AWS X-Ray？

相關部落格：

- 使用 Amazon CloudWatch Synthetics 和 AWS X-Ray 進行偵錯

相關範例：

- Amazon 建置者資料中心：偵測分散式系統，以了解運作狀態
- 可觀測性研討會

REL06-BP02 定義和計算指標 (彙總)

從工作負載元件收集指標和日誌，並從中計算相關的彙總指標。這些指標可提供廣泛且深入的工作負載可觀測性，並可大幅改善您的彈性狀態。

可觀測性不只是從工作負載元件收集指標以及能夠檢視指標並設定提醒。重點在於全面了解工作負載的行為。此行為資訊來自工作負載中的所有元件，包括其所依賴的雲端服務、精心製作的日誌及指標。此資料可讓您全面監督工作負載的行為，並了解每個元件與每個工作單元的互動及其細節。

預期成果：

- 您可以從工作負載元件和 AWS 服務相依性收集日誌，並將其發布到方便存取和處理的集中位置。
- 您的日誌包含高保真度且準確的時間戳記。
- 您的日誌包含處理內容的相關資訊，例如追蹤識別碼、使用者或帳戶識別碼，以及遠端 IP 位址。
- 您可以從日誌建立彙總指標，以便從高層級的角度來呈現工作負載的行為。
- 您可以查詢彙總日誌，以取得有關工作負載的深入分析，並識別實際和潛在的問題。

常見的反模式：

- 您未從工作負載執行所在的運算執行個體或其使用的雲端服務收集相關的日誌或指標。
- 您忽略了收集與業務關鍵績效指標 (KPI) 相關的日誌和指標。
- 您單獨分析工作負載相關的遙測，而未彙總和建立相互關聯。
- 您太快讓指標和日誌過期，導致阻礙了趨勢分析和週期性問題識別。

建立這些最佳實務的優勢：您可以偵測更多異常情況，並將工作負載的不同元件之間的事件與指標相互關聯。您可以根據日誌中通常無法單獨用於指標的資訊，從工作負載元件建立深入分析。您可以大規模查詢日誌來加速判斷失敗原因。

未建立這些最佳實務時的曝險等級：高

實作指引

識別與您的工作負載及其元件相關的遙測資料來源。此資料不僅來自發布指標的元件 (例如作業系統 (OS) 和 Java 等應用程式執行時期)，也來自應用程式和雲端服務日誌。例如，Web 伺服器通常會記錄每個請求的詳細資訊，例如時間戳記、處理延遲、使用者 ID、遠端 IP 位址、路徑和查詢字串。這些日誌中的詳細資訊層級可協助您執行詳細的查詢，並產生其他方式無法提供的指標。

使用適當的工具和程序收集指標和日誌。Amazon EC2 執行個體上執行的應用程式所產生的日誌，可透過 Amazon CloudWatch 代理程式等代理程式收集並發布至 Amazon CloudWatch Logs 等集中儲存服務。AWS 受管運算服務 (例如 AWS Lambda 和 Amazon Elastic Container Service) 會自動將日誌發布至 CloudWatch Logs。為 Amazon CloudFront、Amazon S3、Elastic Load Balancing 和 Amazon API Gateway 等工作負載所使用的 AWS 儲存和處理服務啟用日誌收集。

利用維度讓您的遙測資料更豐富，如此您就能更清楚地看見行為模式，並將相互關聯的問題隔離成相關元件的群組。新增後，您可以觀測更詳細的元件行為、偵測相互關聯的故障，並採取適當的補救措施。有用的維度範例包括可用區域、EC2 執行個體 ID，以及容器任務或 Pod ID。

收集指標和日誌後，您可以撰寫查詢並從中產生彙總指標，以針對正常和異常行為提供實用的深入分析。例如，您可以使用 Amazon CloudWatch Logs Insights 從您的應用程式日誌產生自訂指標、使用 Amazon CloudWatch Metrics Insights 大規模查詢您的指標、使用 Amazon CloudWatch Container Insights 從容器化應用程式和微服務收集、彙總及摘要整理指標與日誌，或者，如果您使用 AWS Lambda 函數，則可以使用 Amazon CloudWatch Lambda 洞察。若要建立彙總錯誤率指標，您可以在每次元件日誌中出現錯誤回應或訊息時讓計數器累進，或計算現有錯誤率指標的彙總值。您可以使用此資料來產生顯示結尾行為的長條圖，例如效能最差的請求或程序。您也可以使用 CloudWatch Logs 異常偵測等解決方案來即時掃描此資料中是否有異常模式。這些深入分析可以放置在儀表板上，並根據您的需求和偏好組織整理。

查詢日誌可協助您了解工作負載元件如何處理特定請求，並顯示請求模式或其他影響工作負載彈性的內容。根據您對應用程式和其他元件行為的了解，事先研究和準備查詢會很實用，讓您更方便視需要執行該程式或元件。舉例來說，若使用 CloudWatch Logs Insights，您能以互動方式搜尋和分析 CloudWatch Logs 中存放的日誌資料。您也可以使用 Amazon Athena 查詢多個來源的日誌，包括 PB 規模的許多 AWS 服務。

當您定義日誌保留政策時，請考慮歷史日誌的價值。歷史日誌有助於識別工作負載效能的長期使用情形及行為模式、迴歸與改善。日誌永久刪除後，即無法再供分析。然而，歷史日誌的價值往往會隨著時間而降低。選擇一項政策來適當平衡您的需求，並應付您可能須遵循的任何法律或合約要求。

實作步驟

1. 為您的可觀測性資料選擇收集、儲存、分析和顯示機制。

2. 在工作負載的適當元件上安裝並設定指標和日誌收集器 (例如，在 Amazon EC2 執行個體上和附屬容器中)。設定這些收集器，讓它們在意外停止時自動重新啟動。啟用收集器的磁碟或記憶體緩衝，不要讓臨時發布失敗影響您的應用程式或導致資料遺失。
3. 在做為工作負載的一部分使用的 AWS 服務上啟用記錄功能，並將這些日誌轉送到您選取的儲存服務 (如有需要)。如需詳細說明，請參閱個別服務的使用者或開發人員指南。
4. 根據您的遙測資料，定義與工作負載相關的營運指標。這些指標可以根據工作負載元件發出的直接指標，包括業務 KPI 相關指標，或彙總計算的結果，例如總和、速率、百分位數或長條圖。使用日誌分析器計算這些指標，並將其放在儀表板上適當的位置。
5. 視需要準備適當的日誌查詢，以分析工作負載元件、請求或交易行為。
6. 為您的元件日誌定義和啟用日誌保留政策。當日誌比政策允許的時間更早時，請定期刪除日誌。

資源

相關的最佳實務：

- REL06-BP01 監控工作負載的所有元件 (產生)
- REL06-BP03 傳送通知 (即時處理和警示)
- REL06-BP04 自動化回應 (即時處理和警示)
- REL06-BP05 分析日誌
- REL06-BP06 定期審查監控範圍和指標
- REL06-BP07 透過您的系統監控請求的端對端追蹤

相關文件：

- Amazon CloudWatch 的運作方式
- Amazon Managed Prometheus
- Amazon Managed Grafana
- 使用 CloudWatch Logs Insights 分析日誌資料
- Amazon CloudWatch Lambda 洞察
- Amazon CloudWatch Container Insights
- 使用 CloudWatch Metrics Insights 查詢您的指標
- AWS Distro for OpenTelemetry
- Amazon CloudWatch Logs Insights 範例查詢

- 使用 Amazon CloudWatch Synthetics 和 AWS X-Ray 進行偵錯
- 搜尋和篩選日誌資料
- 直接將日誌傳送至 Amazon S3
- Amazon 建置者資料中心：偵測分散式系統，以了解運作狀態

相關研討會：

- 一個可觀測性研討會

相關工具：

- AWS Distro for OpenTelemetry (GitHub)

REL06-BP03 傳送通知 (即時處理和警示)

當組織偵測到潛在問題時，他們會將即時通知和警示傳送給適當的人員和系統，以便快速有效地應對這些問題。

預期成果：根據服務和應用程式指標設定相關警示，就可以快速回應操作事件。違反警示閾值時，系統會通知適當的人員和系統，以便解決潛在問題。

常見的反模式：

- 將警示的閾值設得過高，會導致無法傳送重要通知。
- 將警示的閾值設得太低，導致使用者因通知過多的干擾而無法針對重要提醒採取行動。
- 當使用情況改變時，未更新警示及其閾值。
- 針對透過自動化動作解決的最佳警示，將通知傳送給人員而未引發自動化動作，會導致傳送過多的通知。

建立此最佳實務的優勢：將即時通知和警示傳送給適當的人員和系統，以便及早發現問題並快速回應操作事故。

未建立此最佳實務時的曝險等級：高

實作指引

工作負載應具備即時處理和警示功能，以改善可能影響應用程式可用性問題的可偵測性，並作為自動化回應的觸發程式。組織可以透過使用已定義的指標建立警示來執行即時處理和警示，以便在發生重大事件或指標超過閾值時收到通知。

Amazon CloudWatch 可讓您根據靜態閾值、異常偵測和其他條件，使用 CloudWatch 警示來建立指標和複合警示。如需有關可使用 CloudWatch 來設定的警示類型的詳細資訊，請參閱 CloudWatch 文件的警示一節。

您可以使用 CloudWatch 儀表板為團隊建構 AWS 資源的指標和警示的自訂檢視。CloudWatch 主控台中的可自訂首頁可讓您在單一檢視中監控多個區域的資源。

警示可執行一個或多個動作，例如向 Amazon SNS 主題傳送通知、執行 Amazon EC2 動作或 Amazon EC2 Auto Scaling 動作，或在 AWS Systems Manager 中建立 OpsItem 或事故。

Amazon CloudWatch 使用 Amazon SNS 於警示變更狀態時傳送通知，將訊息傳遞從發布者 (生產者) 提供給訂閱用戶 (消費者)。如需有關設定 Amazon SNS 通知的詳細資訊，請參閱 Configuring Amazon SNS。

每當建立、更新、刪除 CloudWatch 警示或變更其狀態時，CloudWatch 就會傳送 EventBridge 事件。您可以使用 EventBridge 搭配這些事件來建立執行動作的規則，例如，當警示狀態變更時就通知您，或使用 Systems Manager Automation 自動觸發您帳戶中的事件。

利用 AWS Health 隨時掌握新知。AWS Health 是 AWS 雲端 資源運作狀態的權威資訊來源。使用 AWS Health 來接收任何已確認服務事件的通知，方便您快速採取行動來緩解任何影響。透過 AWS 使用者通知 建立符合用途的 AWS Health 事件通知，以利用電子郵件和聊天管道傳送，並透過 Amazon EventBridge 以程式設計方式與您的監控和警示工具整合。如果使用 AWS Organizations，則跨帳戶彙總 AWS Health 事件。

應何時使用 EventBridge 或 Amazon SNS？

EventBridge 和 Amazon SNS 可用於開發事件驅動的應用程式，將視具體需求來做出選擇。

當您想要建立應用程式以回應您自己的應用程式、SaaS 應用程式和 AWS 服務中的事件時，建議使用 Amazon EventBridge。EventBridge 是唯一可直接與第三方 SaaS 合作夥伴整合的事件型服務。EventBridge 也會自動擷取來自 200 多個 AWS 服務的事件，而不需要開發人員在其帳戶中建立任何資源。

EventBridge 會將已定義的 JSON 架構用於事件，並協助您建立在整個事件內文中套用的規則，以選取要轉寄至目標的事件。EventBridge 目前支援超過 20 種 AWS 服務作為目標，包括 AWS Lambda、Amazon SQS、Amazon SNS、Amazon Kinesis Data Streams 和 Amazon Data Firehose。

針對需要高散發的應用程式 (數千或數百萬個端點)，建議使用 Amazon SNS。我們看到的常見模式是客戶使用 Amazon SNS 做為規則的目標，以篩選所需的事件並散發到多個端點。

訊息是非結構化的，可以是任何格式。Amazon SNS 支援將訊息轉寄到六種不同類型的目標，包括 Lambda、Amazon SQS、HTTP/S 端點、SMS、行動推送和電子郵件。Amazon SNS 一般的延遲時間短於 30 毫秒。透過將服務設定為傳送 Amazon SNS 訊息，各種 AWS 服務就能做到這一點 (超過 30 個，包括 Amazon EC2、Amazon S3 和 Amazon RDS)。

實作步驟

1. 使用 Amazon CloudWatch 警示建立警示。
   a. 指標警示會監控單一 CloudWatch 指標或與 CloudWatch 指標相依的表達式。與超過一段時間間隔的閾值相比，警示會根據指標或表達式的值起始一或多個動作。該動作可能包含將通知傳送至 Amazon SNS 主題、執行 Amazon EC2 動作或 Amazon EC2 Auto Scaling 動作，或在 AWS Systems Manager 中建立 OpsItem 或事故。
   b. 複合警示由規則表達式組成，該規則表達式會將您已建立的其他警示條件納入考量。只有在符合所有規則條件時，複合警示才會進入警示狀態。在複合警示規則表達式中指定的警示可能會包括指標警示和其他複合警示。複合警示可以在變更狀態時傳送 Amazon SNS 通知，並且可以在進入警示狀態時建立 Systems Manager OpsItems 或事故，但無法執行 Amazon EC2 動作或 Auto Scaling 動作。
2. 設定 Amazon SNS 通知。建立 CloudWatch 警示時，可以包含 Amazon SNS 主題，以便在警示變更狀態時傳送通知。
3. 在 EventBridge 中建立符合指定 CloudWatch 警示的規則。每個規則都支援多個目標，包括 Lambda 函數。例如，您可以定義在可用磁碟空間不足時啟動的警示，該警示會透過 EventBridge 規則觸發 Lambda 函數以清理空間。如需有關 EventBridge 目標的詳細資訊，請參閱 EventBridge targets。

資源

相關 Well-Architected 的最佳實務：

- REL06-BP01 監控工作負載的所有元件 (產生)
- REL06-BP02 定義和計算指標 (彙總)

- REL12-BP01 使用程序手冊調查失敗

相關文件：

- Amazon CloudWatch
- CloudWatch Logs 洞察
- 使用 Amazon CloudWatch 警示
- 使用 Amazon CloudWatch 儀表板
- 使用 Amazon CloudWatch 指標
- 設定 Amazon SNS 通知
- CloudWatch 異常偵測
- CloudWatch Logs 資料保護
- Amazon EventBridge
- Amazon Simple Notification Service

相關影片：

- 重塑 2022 年可觀測性影片
- AWS re:Invent 2022 - Amazon 的可觀測性最佳實務

相關範例：

- 一個可觀測性研討會
- Amazon EventBridge 到 AWS Lambda，由 Amazon CloudWatch 警示進行回饋控制

REL06-BP04 自動化回應 (即時處理和警示)

偵測到事件時，使用自動化以採取動作，例如取代故障的元件。

實作警示的自動即時處理，以便系統可以採取快速的糾正措施，並嘗試在觸發警示時防止故障或服務降級。對警示的自動回應可能包括替換故障元件、調整運算容量、將流量重新導向到運作狀態良好的主機、可用區域或其他區域，以及操作人員通知。

預期成果：識別即時警示，並設定警示的自動處理，以調用採取的適當動作，維護服務水準目標和服務水準協議 (SLA)。自動化的範圍從單一元件的自我修復活動到全站點的容錯移轉。

常見的反模式：

- 針對關鍵的即時警示沒有清晰的清單或目錄。
- 對關鍵警示沒有自動回應 (例如，當運算資源即將耗盡時，發生自動擴展)。
- 矛盾的警示回應動作。
- 操作人員在收到提醒通知時沒有可以遵循的標準作業程序 (SOP)。
- 未監控組態變更，因為未偵測到的組態變更可能會導致工作負載停機。
- 沒有復原意外組態變更的策略。

建立此最佳實務的優勢：自動化警示處理可改善系統復原能力。系統會自動採取糾正措施，減少人為介入時容易出錯的手動活動。工作負載運作符合可用性目標，並減少服務中斷。

未建立此最佳實務時的曝險等級：中

實作指引

為了有效管理提醒並自動化其回應，請根據提醒的重要性和影響來進行分類，記錄回應程序，並在為任務排名前規劃好回應。

識別需要特定動作的任務 (通常會在執行手冊中詳細說明)，並檢查所有執行手冊和程序手冊以確定哪些任務可以自動化。可以定義的動作通常也可以自動化。如果動作無法自動化，請在 SOP 中記錄手動步驟，並對操作人員進行相關培訓。持續挑戰手動程序以尋求自動化機會，以便您可以建立和維護用來自動化提醒回應的計畫。

實作步驟

1. 建立警示清單：若要取得所有警示的清單，可以透過 Amazon CloudWatch 命令 `describe-alarms` 來使用 AWS CLI。視設定的警示數量而定，您可能必須使用分頁來擷取每個呼叫的警示子集，或者也可以使用 AWS SDK 透過 API 呼叫來取得警示。
2. 記錄所有警報動作：更新執行手冊與所有警示及其動作，無論其為手動還是自動。AWS Systems Manager 可提供預先定義的執行手冊。如需有關執行手冊的詳細資訊，請參閱 Working with runbooks。如需有關如何檢視執行手冊內容的詳細資訊，請參閱檢視執行手冊內容。
3. 設定和管理警示動作：對於任何需要動作的警示，請使用 CloudWatch SDK 指定自動化動作。例如，您可以建立和啟用警示的動作，或停用警示的動作，以根據 CloudWatch 警示自動變更 Amazon EC2 執行個體的狀態。

   也可使用 Amazon EventBridge 來自動回應系統事件，例如應用程式可用性問題或資源變動。您可建立規則來指示您在意的事件，以及當事件符合規則時執行的動作。可以自動啟動的動作包括調

用 AWS Lambda 函數、調用 Amazon EC2 Run Command、將事件轉送至 Amazon Kinesis Data Streams 以及查看使用 EventBridge 自動化 Amazon EC2。

4. 標準操作程序 (SOP)：根據您的應用程式元件，AWS Resilience Hub 會建議多個 SOP 範本。您可以使用這些 SOP 記錄在發出提醒時操作員應遵循的所有程序。也可以根據 Resilience Hub 建議來建構 SOP，其中您需要具有相關彈性政策的 Resilience Hub 應用程式，以及針對該應用程式的歷史彈性評估。SOP 的建議會由彈性評估產生。

   Resilience Hub 與 Systems Manager 合作，透過提供許多 SSM 文件作為這些 SOP 的基礎來自動執行 SOP 的步驟。例如，Resilience Hub 可能會建議使用 SOP 來根據現有的 SSM 自動化文件新增磁碟空間。

5. 使用 Amazon DevOps Guru 執行自動化動作：可使用 Amazon DevOps Guru 來自動監控應用程式資源，以偵測異常行為並提供目標建議，以縮短問題識別和矯正時間。使用 DevOps Guru，可以近乎即時地監控多個來源的操作資料串流，包括 Amazon CloudWatch 指標、AWS Config、AWS CloudFormation 和 AWS X-Ray。也可以使用 DevOps Guru 在 OpsCenter 中自動建立 OpsItems，並將事件傳送至 EventBridge 以實現額外的自動化。

資源

相關的最佳實務：

- REL06-BP01 監控工作負載的所有元件 (產生)
- REL06-BP02 定義和計算指標 (彙總)
- REL06-BP03 傳送通知 (即時處理和警示)
- REL08-BP01 將執行手冊用於部署等標準活動

相關文件：

- AWS Systems Manager Automation
- 建立 EventBridge 規則，以透過 AWS 資源觸發事件
- 一個可觀測性研討會
- Amazon 建置者資料中心：偵測分散式系統，以了解運作狀態
- 什麼是 Amazon DevOps Guru？
- 使用自動化文件 (手冊)

相關影片：

- AWS re:Invent 2022 - Amazon 的可觀測性最佳實務
- AWS re:Invent 2020：使用 AWS Systems Manager 自動執行任何操作
- AWS Resilience Hub 簡介
- 為 Amazon DevOps Guru 通知建立自訂票證系統
- 使用 Amazon DevOps Guru 啟用多帳戶洞察彙總

相關範例：

- Amazon CloudWatch 和 Systems Manager 研討會

REL06-BP05 分析日誌

收集日誌檔和指標歷史記錄，並分析這些檔案和歷史記錄，以了解更廣泛的趨勢和工作負載洞見。

Amazon CloudWatch Logs Insights 支援簡單但功能強大的查詢語言，可用於分析日誌資料。Amazon CloudWatch Logs 也支援訂閱，允許資料順暢流至 Amazon S3，您可以在其中使用 或 Amazon Athena 查詢資料。它也支援對大量格式的查詢。如需詳細資訊，請參閱 Amazon Athena 使用者指南中的支援 SerDes 和資料格式。若要分析大型日誌檔案集，您可以執行 Amazon EMR叢集來執行 PB 規模分析。

AWS 合作夥伴和第三方提供許多工具，允許彙總、處理、儲存和分析。這些工具包括 New Relic、Splunk、Logly、Logstash CloudHealth、 和 Nagios。但是，系統和應用程式日誌之外的產生對於每個雲端提供者都是唯一的，並且通常對於每個服務也都是唯一的。

資料管理是監控程序中常常被忽略的部分。您需要確定監控資料的保留要求，然後相應地套用生命週期政策。Amazon S3 可支援 S3 儲存貯體層級的生命週期管理。該生命週期管理能以不同方式套用至儲存貯體中的不同路徑。在生命週期即將結束時，您可以將資料傳輸到 Amazon S3 Glacier 進行長期儲存，然後在保留期結束後到期。S3 智慧型分層儲存類別旨在透過自動將資料移至最經濟實惠的存取層來優化成本，而不會影響效能或營運開銷。

未建立此最佳實務時的曝險等級：中

實作指引

- CloudWatch Logs Insights 可讓您以互動方式搜尋和分析 Amazon Logs 中的 CloudWatch 日誌資料。
  - 使用 Logs Insights 分析 CloudWatch 日誌資料

  - Amazon CloudWatch Logs Insights 範例查詢
- 使用 Amazon CloudWatch Logs 將日誌傳送至 Amazon S3，您可以在其中使用 或 Amazon Athena 查詢資料。
  - 我要如何使用 Athena 來分析 Amazon S3 伺服器存取日誌？
    - 為您的伺服器存取日誌儲存貯體建立 S3 生命週期政策。設定生命週期政策以定期移除日誌檔案。這麼做可降低 Athena 分析每個查詢時的資料量。
      - 如何建立 S3 儲存貯體的生命週期政策？

資源

相關文件：

- Amazon CloudWatch Logs Insights 範例查詢
- 使用 Logs Insights 分析 CloudWatch 日誌資料
- 使用 Amazon CloudWatch Synthetics 和 進行偵錯 AWS X-Ray
- 如何建立 S3 儲存貯體的生命週期政策？
- 我要如何使用 Athena 來分析 Amazon S3 伺服器存取日誌？
- 一個可觀測性研討會
- Amazon 建置者資料中心：偵測分散式系統，以了解運作狀態

REL06-BP06 定期審查監控範圍和指標

經常檢閱工作負載監控的實作情形，並隨著工作負載及其架構的演進更新。定期稽核監控有助於降低遺漏或忽略問題指標的風險，並進一步協助工作負載達成其可用性目標。

有效的監控是以關鍵業務指標為基礎，這些指標會隨著業務優先事項的改變而演進。您的監控審查程序應強調服務層級指標 (SLI)，並納入基礎設施、應用程式、用戶端和使用者的深入分析。

預期成果：您擬訂一套有效的監控策略，它會定期審查和更新，以及在任何重大事件或變更之後更新。您確認隨著工作負載和業務需求的發展，關鍵應用程式運作狀態指標仍保持相關。

常見的反模式：

- 您只收集預設指標。
- 您擬訂了監控策略，但未曾檢閱它。

- 您未在部署重大變更時討論監控。
- 您相信過時的指標來判斷工作負載運作狀態。
- 由於指標和閾值過時，導致您的營運團隊因誤報而疲於奔命。
- 您缺乏未受監控之應用程式元件的可觀測性。
- 您只專注於低層級技術指標，並排除業務指標未加監控。

建立此最佳實務的優勢：若您定期審查監控，就可以預測潛在問題，並確認您能夠偵測到這些問題。它還能讓您發現在早期審查期間可能遺漏的盲點，藉此進一步改善您偵測問題的能力。

未建立此最佳實務時的曝險等級：中

實作指引

在您的營運整備度審查 (ORR) 程序中檢閱監控指標和範圍。依照一致的排程執行定期營運整備度審查，以評估目前工作負載與您設定的監控之間是否有任何差距。建立定期執行營運效能審查和知識共享的機制，以增強您從營運團隊獲得更高效能的能力。驗證現有的警示閾值是否仍適當，並確認是否發生營運團隊收到誤報，或未監控應監控之應用程式層面的情況。

彈性分析架構提供了實用的指引，可協助您進行整個程序。架構的重點在於識別潛在的失敗模式，以及您可採用哪些預防和修正控制來減輕其影響。這些知識可協助您確定要監控和警示的正確指標和事件。

實作步驟

1. 排程及定期審查工作負載儀表板。您對於檢查深度可能有不同規律。
2. 檢查指標中的趨勢。比較指標值與歷史值，以查看是否有趨勢可能指出某項需要調查的事務。這類範例包括：延遲增加、主要業務功能降低，以及失敗回應增加。
3. 檢查指標中是否有極端值和異常，這些可能被平均值或中位數掩蓋。查看時間範圍內的最高和最低值，並調查遠超出正常界限的觀測原因。當您繼續消除這些原因，您就可以設定更嚴格的預期指標界限，來回應獲得改善的工作負載效能一致性。
4. 尋找行為中的急劇變化。指標的數量或方向立即變更，可能表示應用程式有所變更，或您可能需要新增其他指標以追蹤的外部因素。
5. 檢閱目前的監控策略是否仍與應用程式相關。根據先前事件 (或彈性分析架構) 的分析，評估是否有其他應用程式層面應納入監控範圍。
6. 檢閱您的實際使用者監控 (RUM) 指標，以判斷應用程式功能涵蓋範圍是否有任何差距。
7. 檢閱您的變更管理程序。視需要更新您的程序，以納入應在核准變更之前執行的監控分析步驟。
8. 在營運整備度審查過程中確實檢閱監控並修正錯誤程序。

資源

相關的最佳實務

- REL06-BP01 監控工作負載的所有元件 (產生)
- REL06-BP02 定義和計算指標 (彙總)
- REL06-BP07 透過您的系統監控請求的端對端追蹤
- REL12-BP02 執行事件後分析
- REL12-BP06 定期進行演練日

相關文件：

- 為什麼您應該制定錯誤糾正 (COE)
- 使用 Amazon CloudWatch 儀表板
- 建置用於檢視營運狀況的儀表板
- 進階多可用區域彈性模式 - 微小故障
- Amazon CloudWatch Logs Insights 範例查詢
- 使用 Amazon CloudWatch Synthetics 和 AWS X-Ray 進行偵錯
- 一個可觀測性研討會
- Amazon 建置者資料中心：偵測分散式系統，以了解運作狀態
- 使用 Amazon CloudWatch 儀表板
- AWS 可觀測性最佳實務
- 彈性分析架構
- 彈性分析架構 - 可觀測性
- 營運整備度審查 - ORR

REL06-BP07 監控 end-to-end透過您的系統追蹤請求

在透過服務元件處理請求時追蹤請求，讓產品團隊可以更輕鬆地分析和偵錯問題，並改善效能。

預期結果：具有所有元件全面追蹤的工作負載易於偵錯，透過簡化根本原因探索來改善錯誤和延遲的平均解決時間（MTTR）。 End-to-end追蹤可減少探索受影響元件所需的時間，並深入探索錯誤或延遲的詳細根本原因。

常見的反模式：

- 追蹤可用於某些元件，但不適用於所有元件。例如，如果沒有追蹤 AWS Lambda，團隊可能無法清楚了解因頻繁工作負載冷啟動所造成的延遲。
- 合成 Canary 或真實使用者監控（RUM）未設定追蹤。如果沒有 Canary 或 RUM，追蹤分析會省略用戶端互動遙測，產生不完整的效能描述檔。
- 混合式工作負載同時包含雲端原生和第三方追蹤工具，但未採取相關步驟來選擇及完全整合單一追蹤解決方案。根據選擇的追蹤解決方案，雲端原生追蹤SDKs應用於非雲端原生或第三方工具的儀器元件，應設定為擷取雲端原生追蹤遙測。

建立此最佳實務的優勢：當開發團隊收到問題的提醒時，他們可以看到系統元件互動的全貌，包括個別元件與日誌記錄、效能和失敗的關聯性。由於追蹤可讓您輕鬆地以視覺化方式識別根本原因，調查根本原因的所需時間將可縮短。詳細了解元件互動的團隊，可在解決問題時做出更明智、更快速的決策。諸如何時應調用災難復原 (DR) 容錯移轉，或何處最適合實作自我修復策略之類的決策，可藉由分析系統追蹤來改善，最終提升客戶對服務的滿意度。

未建立此最佳實務時的曝險等級：中

實作指引

操作分散式應用程式的團隊，可使用追蹤工具來建立關聯性識別碼、收集請求追蹤，以及建置連網元件的服務圖。所有應用程式元件均應包含在請求追蹤中，包括服務用戶端、中介軟體閘道和事件匯流排、運算元件和儲存體 (包括鍵值存放區和資料庫)。在追蹤組態中 end-to-end包含合成 Canary 和真實使用者監控，以測量遠端用戶端互動和延遲，以便您可以根據服務層級協議和目標準確評估系統效能。

您可以使用 AWS X-Ray和 Amazon CloudWatch Application Monitoring 測試服務，在請求通過應用程式時提供完整的檢視。X-Ray 會收集應用程式遙測，並可讓您針對無程式碼或低程式碼的系統元件，將 APIs和 視覺化並進行篩選。 CloudWatch 應用程式監控包括 ServiceLens 整合您的追蹤與指標、日誌和警示。 CloudWatch 應用程式監控還包括用於監控端點和 的合成APIs，以及用於測試 Web 應用程式用戶端的真實使用者監控。

實作步驟

- 在 Amazon S3 和 Amazon Gateway 等所有支援的原生服務 AWS X-Ray 上使用 。 Amazon S3 AWS Lambda API AWS 這些服務使用基礎設施作為程式碼 AWS SDKs或 啟用具有組態切換的 X-Ray AWS Management Console。
- 檢測應用程式 AWS Distro for Open Telemetry 和 X-Ray 或第三方收集代理程式。
- 檢閱 AWS X-Ray 開發人員指南，了解程式設計語言特定實作。這些文件章節詳細說明如何針對您的應用程式程式設計語言執行HTTP請求、SQL查詢和其他程序。

- 使用適用於 Amazon CloudWatch Synthetic Canary 和 Amazon CloudWatch RUM 的 X-Ray 追蹤，透過下游 AWS 基礎設施來分析最終使用者用戶端的請求路徑。
- 根據資源運作狀態和 Canary 遙測設定 CloudWatch 指標和警示，以便團隊快速收到問題警示，然後可以透過 深入探索追蹤和服務地圖 ServiceLens。
- 如果將第三方工具用於主要追蹤解決方案，請為第三方追蹤工具 (例如 Datadog、New Relic 或 Dynatrace) 啟用 X-Ray 整合。

資源

相關的最佳實務：

- REL06-BP01 監控工作負載的所有元件 (產生)
- REL11-BP01 監控工作負載的所有元件以偵測故障

相關文件：

- 什麼是 AWS X-Ray？
- Amazon CloudWatch：應用程式監控
- 使用 Amazon CloudWatch Synthetics 和 進行偵錯 AWS X-Ray
- Amazon 建置者資料中心：偵測分散式系統，以了解運作狀態
- AWS X-Ray 與其他 AWS 服務整合
- AWS 適用於 OpenTelemetry 和 的 Distro AWS X-Ray
- Amazon CloudWatch：使用合成監控
- Amazon CloudWatch：使用 CloudWatch RUM
- 設定 Amazon CloudWatch 合成 Canary 和 Amazon CloudWatch 警示
- 可用性及更多：了解和改善 上分散式系統的復原能力 AWS

相關範例：

- 一個可觀測性研討會

相關影片：

- AWS re：Invent 2022 - 如何跨多個帳戶監控應用程式

- 如何監控您的 AWS 應用程式

相關工具：

- AWS X-Ray
- Amazon CloudWatch
- Amazon Route 53

設計工作負載以適應需求變更

可擴展的工作負載提供可自動新增或移除資源的彈性，讓資源能夠在任何特定時間點充分滿足目前的需求。

最佳實務

- REL07-BP01 取得或擴展資源時使用自動化
- REL07-BP02 在偵測到工作負載受損時取得資源
- REL07-BP03 偵測到工作負載需要更多資源時取得資源
- REL07-BP04 Load 測試您的工作負載

REL07-BP01 取得或擴展資源時使用自動化

雲端可靠性的基石，是基礎設施和資源的程式設計定義、佈建和管理。自動化可協助您簡化資源佈建、促進一致且安全的部署，以及在整個基礎設施中擴展資源。

預期成果：您管理自己的基礎設施即程式碼 (IaC)。您可以在版本控制系統 (VCS) 中定義和維護自己的基礎設施程式碼。您可以將佈建 AWS 資源的工作委派給自動化機制，並利用 Application Load Balancer (ALB)、Network Load Balancer (NLB) 和 Auto Scaling 群組等受管服務。您使用持續整合/持續交付 (CI/CD) 管道來佈建資源，以讓程式碼變更自動初始化資源更新，包括 Auto Scaling 組態的更新。

常見的反模式：

- 您使用命令列或在 AWS Management Console (也稱為 click-ops) 手動部署資源。
- 您緊密耦合應用程式元件或資源，因而建立了缺乏彈性的架構。
- 您實作缺乏彈性的擴展政策，因而無法適應不斷變化的業務需求、流量模式或新的資源類型。

- 您手動預估容量來應付預測的需求。

建立此最佳實務的優勢：基礎設施即程式碼 (IaC) 可讓您以程式設計方式定義基礎設施。這可協助您透過與應用程式變更相同的軟體開發生命週期來管理基礎設施變更，進而提高一致性和可重複性，並降低手動易出錯任務的風險。您可以使用自動化交付管道實作 IaC，以進一步簡化佈建和更新資源的程序。您採取可靠且有效率的方式部署基礎設施更新，而無需手動介入。在擴展資源以應付不斷變化的需求時，這種敏捷性尤其重要。

您可以結合 IaC 和交付管道來實現動態的自動化資源擴展。透過監控關鍵指標並套用預先定義的擴展政策，Auto Scaling 就可視需要自動佈建或取消佈建資源，進而改善效能和成本效益。這可降低手動錯誤或延遲的可能性，以回應應用程式或工作負載需求的改變。

IaC、自動化交付管道與 Auto Scaling 相互結合之下，可協助組織安心佈建、更新和擴展其環境。這種自動化對於維護反應靈敏、具彈性且高效管理的雲端基礎設施至關重要。

未建立此最佳實務時的曝險等級：高

實作指引

若要為您的 AWS 架構設定具有 CI/CD 管道和基礎設施即程式碼 (IaC) 的自動化，請選擇 Git 這類版本控制系統來儲存您的 IaC 範本和組態。這些範本可以使用 AWS CloudFormation 等工具撰寫。請從在這些範本內定義您的基礎設施元件開始 (例如 AWS VPC、Amazon EC2 Auto Scaling 群組和 Amazon RDS 資料庫)。

接著將這些 IaC 範本與 CI/CD 管道整合，以自動化部署程序。AWS CodePipeline 提供了順暢的 AWS 原生解決方案，您也可以使用其他第三方 CI/CD 解決方案。建立管道，當您的版本控制儲存庫發生變更時，就會啟用該管道。設定管道以包含檢查和驗證 IaC 範本的階段、將基礎設施部署到預備環境、執行自動化測試，最後再部署到實際執行環境。必要時納入核准步驟，以維持對變更的控制。此自動化管道不僅能加快部署速度，還能促進環境之間的一致性和可靠性。

在 IaC 中設定 Amazon EC2 執行個體、Amazon ECS 任務和資料庫複本等資源的 Auto Scaling，以視需要提供自動橫向擴充和縮減。此方法可增強應用程式可用性和效能，並根據需求動態調整資源來最佳化成本。如需支援的資源清單，請參閱 Amazon EC2 Auto Scaling 和 AWS Auto Scaling。

實作步驟

1. 建立並使用原始程式碼儲存庫來存放控制基礎設施組態的程式碼。對此儲存庫執行變更，以反映您要進行的任何持續變更。
2. 選取基礎設施即程式碼解決方案 (例如 AWS CloudFormation) 讓您的基礎設施保持最新狀態，並偵測與您預期狀態不一致 (偏離) 的情況。

3. 將 IaC 平台與您的 CI/CD 管道整合，以自動化部署。
4. 確定並收集適當的指標，以自動擴展資源。
5. 使用適合您工作負載元件的橫向擴充和縮減政策來設定自動擴展資源。考慮針對可預測的用量模式使用排程擴展。
6. 監控部署以偵測失敗和迴歸。在 CI/CD 平台內實作回復機制，以在必要時還原變更。

資源

相關文件：

- AWS Auto Scaling：擴展計畫的運作方式
- AWS Marketplace：可與 Auto Scaling 結合使用的產品
- 使用 DynamoDB Auto Scaling 功能自動管理輸送容量
- 將負載平衡器與 Auto Scaling 群組配合使用
- 什麼是 AWS Global Accelerator？
- 什麼是 Amazon EC2 Auto Scaling？
- 什麼是 AWS Auto Scaling？
- 什麼是 Amazon CloudFront？
- 什麼是 Amazon Route 53？
- 什麼是 Elastic Load Balancing？
- 什麼是 Network Load Balancer？
- 什麼是 Application Load Balancer？
- 將 Jenkins 與 AWS CodeBuild 和 AWS CodeDeploy 整合
- 使用 AWS CodePipeline 建立一個四階段的管道

相關影片：

- 回歸基礎：將程式碼部署至 Amazon EC2
- AWS 支援您 | 使用 AWS CloudFormation 範本開始使用您的基礎設施即程式碼解決方案
- 使用 AWS CodePipeline 簡化您的軟體版本程序
- 使用 Amazon CloudWatch 儀表板監控 AWS 資源
- 建立跨帳戶和跨區域 CloudWatch 儀表板 | Amazon Web Services

REL07-BP02 在偵測到工作負載受損時取得資源

在可用性受到影響時視需要主動擴展資源，以還原工作負載可用性。

您必須先設定運作狀態檢查和這些檢查的條件，以指出可用性因資源不足而受到影響的時間。然後，通知適當的人員手動擴展資源，或啟動自動化以自動調整資源規模。

您可以針對工作負載手動調整規模 （例如，變更 Auto Scaling 群組中的EC2執行個體數量，或透過 AWS Management Console 或 修改 DynamoDB 資料表的輸送量 AWS CLI）。但是，應盡可能使用自動化 (請參閱取得或擴展資源時使用自動化)。

預期成果：啟動擴展活動 (自動或手動)，以在偵測到故障或客戶體驗降級時恢復可用性。

未建立此最佳實務時的曝險等級：中

實作指引

在工作負載中的所有元件實作可觀測性和監控，以監控客戶體驗並偵測故障。定義手動或自動化程序，以擴展所需的資源。 o 如需詳細資訊，請參閱 REL11-BP01 監控工作負載的所有元件以偵測失敗。

實作步驟

- 定義會擴展所需資源的手動或自動程序。
  - 擴展程序取決於工作負載內不同元件的設計方式。
  - 擴展程序也會根據所使用的基礎技術而有所不同。
    - 使用 的元件 AWS Auto Scaling 可以使用擴展計畫來設定一組擴展資源的指示。如果您使用 AWS CloudFormation 或將標籤新增至 AWS 資源，您可以為每個應用程式設定不同資源集的擴展計劃。Auto Scaling 為針對每個資源自訂的擴展策略提供建議。建立擴展計畫之後，Auto Scaling 會將動態擴展和預測擴展方法結合在一起，以支援您的擴展策略。有關詳細資訊，請參閱 How scaling plans work。
    - Amazon EC2 Auto Scaling 會驗證您是否擁有正確數量的 Amazon EC2執行個體，以處理應用程式的負載。您可以建立EC2執行個體集合，稱為 Auto Scaling 群組。您可以在每個 Auto Scaling 群組中指定執行個體數量下限和上限，Amazon EC2 Auto Scaling 可確保您的群組永遠不會低於或超過這些限制。如需詳細資訊，請參閱什麼是 Amazon EC2 Auto Scaling ？
    - Amazon DynamoDB Auto Scaling 功能使用 Application Auto Scaling 服務代您動態調整佈建的輸送容量，以此回應實際流量模式。這可讓資料表或全域次要索引增加其佈建的讀取與寫入容量，以處理突然增加的流量，而不需限流。如需詳細資訊，請參閱 Managing throughput capacity automatically with DynamoDB auto scaling。

資源

相關的最佳實務：

- REL07-BP01 在取得或擴展資源時使用自動化
- REL11-BP01 監控工作負載的所有元件以偵測故障

相關文件：

- AWS Auto Scaling：擴展計劃的運作方式
- 使用 DynamoDB Auto Scaling 功能自動管理輸送容量
- 什麼是 Amazon EC2 Auto Scaling

REL07-BP03 偵測到工作負載需要更多資源時取得資源

雲端運算最有價值的功能之一，就是動態佈建資源的能力。

在傳統的內部部署運算環境中，您必須預先確定和佈建足夠的容量，以滿足尖峰需求。這點卻是問題所在，因為它很昂貴，而且如果您低估工作負載的尖峰容量需求，它可能會對可用性造成風險。

在雲端中，您就不需要這樣做。而是能夠視需要佈建運算、資料庫和其他資源容量，以滿足目前和預測的需求。Amazon EC2 Auto Scaling 和 Application Auto Scaling 等自動化解決方案可以根據您指定的指標，自動線上提供資源。這使得擴展程序更容易且可預測，同時可確保您隨時有足夠的資源可用，進而大幅提高工作負載的可靠性。

預期成果：您設定自動擴展運算和其他資源以滿足需求。您在擴展政策中提供足夠的預留空間，以便在其他資源上線時處理激增的流量。

常見的反模式：

- 您佈建固定數量的可擴展資源。
- 您選擇的擴展指標與實際需求無關。
- 您無法在擴展計畫中提供足夠的預留空間來應付需求激增的情況。
- 您的擴展政策太晚增加容量，以致於在其他資源上線時造成容量耗盡和服務降級的情況。
- 您未能正確設定資源計數的下限和上限，因而導致擴展失敗。

建立此最佳實務的優勢：擁有足夠的資源可滿足目前的需求，這點對於提供工作負載的高可用性並遵守您定義的服務層級目標 (SLO) 來說至關重要。自動擴展可讓您提供工作負載所需的確切運算、資料庫

和其他資源數量，以便處理目前和預測的需求。您不需要判斷尖峰容量需求，並靜態配置資源來處理這些需求。您可以改為隨著需求增加配置更多資源來應付這些需求，並且在需求減少之後停用資源以降低成本。

未建立此最佳實務時的曝險等級：中

實作指引

首先，確定工作負載元件是否適合自動擴展。這些元件稱為可水平擴展，因為它們提供相同的資源且行為相同。可水平擴展元件的範例包括相似設定的 EC2 執行個體、Amazon Elastic Container Service (ECS) 任務，以及在 Amazon Elastic Kubernetes Service (EKS) 上執行的 Pod。這些運算資源通常位於負載平衡器後方，稱為複本。

其他複寫資源可能包括資料庫讀取複本、Amazon DynamoDB 資料表和 Amazon ElastiCache (Redis OSS) 叢集。如需可支援資源的完整清單，請參閱可與 Application Auto Scaling 搭配使用的 AWS 服務。

對於容器型架構，您可能需要採用兩種不同的方式進行擴展。第一種方式是，您可能需要擴展提供可水平擴展服務的容器。第二種方式是，您可能需要擴展運算資源以騰出空間給新容器。每一層都有不同的自動擴展機制。若要擴展 ECS 任務，您可以使用 Application Auto Scaling。若要擴展 Kubernetes Pod，您可以使用 Horizontal Pod Autoscaler (HPA) 或 Kubernetes Event-driven Autoscaling (KEDA)。若要擴展運算資源，您可以針對 ECS 使用容量提供者，對於 Kubernetes，您可以使用 Karpenter 或 Cluster Autoscaler。

接著選取您要執行自動擴展的方式。有三個主要選項：指標型擴展、排程擴展和預測性擴展。

指標型擴展

指標型擴展會根據一或多個擴展指標的值佈建資源。擴展指標是指對應工作負載需求的指標。確定擴展指標是否適當的好方法，就是在非實際執行環境中執行負載測試。在負載測試期間，將可擴展資源的數量固定，並慢慢增加需求 (例如輸送量、並行或模擬的使用者)。然後尋找隨著需求增加 (或減少) 而增加的指標，以及隨著需求下降而減少 (或增加) 的指標。典型的擴展指標包括 CPU 使用率、工作佇列深度 (例如 Amazon SQS 佇列)、作用中使用者數量，以及網路輸送量。

> **Note**
>
> AWS 已觀測到，在大多數應用程式中，記憶體使用率會隨著應用程式暖機而增加，然後達到穩定值。當需求減少時，記憶體使用率通常會保持在升高處，而不會跟著減少。由於記憶體使用率不會隨著需求增加和減少而對應升降，因此在選取此指標用於自動擴展時，務必審慎考慮。

指標型擴展是延遲操作。利用率指標可能需要幾分鐘的時間才能傳播至自動擴展機制，而且這些機制通常會等待需求增加的明確訊號，然後才做出反應。於是，當自動擴展器建立新資源時，可能需要額外的時間才能提供完整服務。因此，請務必不要將擴展指標目標設定得太接近完全使用率 (例如 90% CPU 使用率)。這樣做可能產生在額外容量上線之前就耗盡現有資源容量的風險。典型的資源使用率目標範圍介於 50-70% 之間，此範圍可實現最佳可用性，具體取決於佈建其他資源的需求模式和所需的時間。

排程擴展

排程擴展會根據行事曆或一天當中的時間佈建或移除資源。它經常用於具有可預測需求的工作負載，例如工作日營業時間或銷售活動的尖峰使用率。Amazon EC2 Auto Scaling 和 Application Auto Scaling 都支援排程擴展。KEDA 的 cron 擴展器支援 Kubernetes Pod 的排程擴展。

預測性擴展

預測性擴展使用機器學習，根據預期的需求自動擴展資源。預測性擴展會分析您提供的使用率指標的歷史值，並持續預測其未來值。然後使用預測的值來向上擴展資源或縮減其規模。Amazon EC2 Auto Scaling 可執行預測性擴展

實作步驟

1. 確定工作負載元件是否適合自動擴展。
2. 確定哪一種擴展機制最適合工作負載：指標型擴展、排程擴展或預測性擴展。
3. 為元件選取適當的自動擴展機制。對於 Amazon EC2 執行個體，使用 Amazon EC2 Auto Scaling。對於其他 AWS 服務，使用 Application Auto Scaling。對於 Kubernetes Pod (例如在 Amazon EKS 叢集中執行的 Pod)，請考慮 Horizontal Pod Autoscaler (HPA) 或 Kubernetes Event-driven Autoscaling (KEDA)。對於 Kubernetes 或 EKS 節點，請考慮 Karpenter 和 Cluster Auto Scaler (CAS)。
4. 對於指標或排程擴展，請執行負載測試，以確定適合工作負載的擴展指標和目標值。對於排程擴展，確定您選取的日期和時間所需的資源數量。確定滿足預期的尖峰流量所需的資源數量上限。
5. 根據上面收集的資訊設定自動擴展器。如需詳細資訊，請參閱自動擴展服務的文件。確認已正確設定擴展的上限和下限。
6. 確認擴展組態依預期運作。在非實際執行環境中執行負載測試，以及觀察系統如何反應，並視需要調整。在實際執行環境中啟用自動擴展時，請設定適當的警報來通知您任何非預期的行為。

資源

相關文件：

- 什麼是 Amazon EC2 Auto Scaling？
- AWS 規範性指引：負載測試應用程式
- AWS Marketplace：可與 Auto Scaling 結合使用的產品
- 使用 DynamoDB Auto Scaling 功能自動管理輸送容量
- EC2 的預測擴展，採用機器學習技術
- Amazon EC2 Auto Scaling 排程擴展
- 說說「利特爾法則」的故事

REL07-BP04 Load 測試您的工作負載

採用負載測試方法來衡量擴展活動是否滿足工作負載要求。

重要的是執行持續的負載測試。負載測試應探索突破點，並測試工作負載的效能。 AWS 可讓您輕鬆設定臨時測試環境，以建立生產工作負載規模的模型。在雲端，您可隨需建立生產規模的測試環境、完成測試，然後再停用資源。因為您只為執行中的測試環境付費，所以能以與內部部署測試相較之下相當微小比例的成本來模擬即時環境。

在生產系統承受壓力的演練日，以及客戶使用量較低的時間內，您也應考慮在生產中進行負載測試，並且讓可用的所有人員解釋結果並解決所發生的任何問題。

常見的反模式：

- 在與生產組態不同的部署上執行負載測試。
- 僅對工作負載的個別部分執行負載測試，而非整個工作負載。
- 使用一部分請求而非代表性的一組實際請求來執行負載測試。
- 對高於預期負載的小型安全係數執行負載測試。

建立此最佳實務的優勢：您會知道架構中的哪些元件在負載時失敗，並能夠識別要監看哪些指標，指出您正在及時處理該負載來解決問題，避免受到該故障的影響。

未建立此最佳實務時的曝險等級：中

實作指引

- 執行負載測試，以識別工作負載的哪些層面指出您必須新增或移除容量。負載測試的代表性流量應與您在生產環境中收到的流量相似。在觀看您已檢測的指標時增加負載，以判斷哪些指標指出何時須新增或移除資源。

- 上的分散式負載測試 AWS：模擬數千個連線使用者
  - 識別請求混合。您可能會有不同的請求混合，因此您應該在識別流量混合時查看各種時間範圍。
  - 實作載入驅動程式。您可以使用自訂程式碼、開放原始碼或商業軟體實作載入驅動程式。
  - 最初使用小容量的負載測試。您將負載驅動到較小容量 (可能和單一執行個體或容器一樣小)，看到一些立即的影響。
  - 針對較大容量的負載測試。在分散式負載上的效果會有所不同，因此您必須盡可能在接近產品環境的條件下進行測試。

資源

相關文件：

- 上的分散式負載測試 AWS：模擬數千個連線使用者
- 負載測試應用程式

相關影片：

- AWS Summit ANZ 2023：透過 AWS 分散式負載測試，放心加速

實作變更

受控變更在執行以下操作時必不可少：部署新功能，以及確保工作負載和作業環境正在執行已知且經過適當修補的軟體。如果這些變更不受控制，那麼就很難預測這些變更的影響，也很難解決由於這些變更而產生的問題。

用於盡量降低風險的其他部署模式

功能標記 (也稱為功能切換) 是應用程式上的組態選項。您可以在功能關閉的情況下部署軟體，以讓客戶無法發現該功能。然後，您可以像啟用 Canary 部署一樣開啟該功能，也可以將變更速度設置為 100% 以查看效果。如果部署有問題，您可以輕鬆地關閉該功能而無須回復。

故障隔離區域部署：AWS 為自己的部署建立的最重要規則之一，是避免同時接觸一個區域內的多個可用區。這對於確保可用區域彼此獨立以便於計算可用性而言至關重要。我們建議您在部署中使用類似的考量事項。

營運準備度審查 (ORR)

AWS 發現執行營運準備度審查極為實用；這些審查可評估測試的完整性，能夠執行監控，重要的是，能夠針對其 SLA 稽核應用程式效能，並可在發生中斷或其他異常營運時提供資料。在初始生產部署之前，需要執行正式的 ORR。AWS 將定期 (每年一次，或在關鍵績效期之前) 重複執行 ORR，以確保不會偏離營運預期。如需營運準備度的詳細資訊，請參閱 AWS Well-Architected Framework 的卓越營運支柱。

最佳實務

- REL08-BP01 將執行手冊用於部署等標準活動
- REL08-BP02 將功能測試整合為部署的一部分
- REL08-BP03 整合彈性測試作為部署的一部分
- REL08-BP04 使用不可變基礎設施進行部署
- REL08-BP05 使用自動化部署變更

REL08-BP01 將執行手冊用於部署等標準活動

執行手冊是實現特定結果的預定義程序。使用執行手冊執行手動或自動進行的標準活動。範例包括部署工作負載、修補工作負載或進行 DNS 修改。

例如，實施程序以確保部署期間的回復安全性。確保您可以回復部署，且不會對客戶造成任何中斷，這對於打造可靠的服務而言至為關鍵。

對於執行手冊程序，從有效的手動過程開始，以程式碼實作它，並在適當時調用它以自動執行。

即使是高度自動化的複雜工作負載，執行手冊仍然適用於執行演練日或滿足嚴格的報告和稽核要求。

請注意，程序手冊用於回應特定事件，而執行手冊用於實現特定成果。執行手冊通常用於例行活動，而程序手冊則用於回應非例行事件。

常見的反模式：

- 在生產環境中對組態執行非計劃中的變更。
- 為了更快速地部署而略過計畫中的步驟，會導致部署失敗。
- 在不測試變更反轉的情況下進行變更。

建立此最佳實務的優勢：有效的變更規劃可提高您成功執行變更的能力，因為您知道所有受影響的系統。在測試環境中驗證變更可增強信心。

未建立此最佳實務時的曝險等級：高

實作指引

- 透過在執行手冊中記錄程序，對熟知的事件提供一致且迅速的回應。
  - AWS Well-Architected Framework：概念：執行手冊
- 使用基礎設施即程式碼的原則來定義您的基礎設施。透過使用 AWS CloudFormation (或受信任的第三方) 來定義您的基礎設施，您可以使用版本控制軟體對變更進行版本控制和追蹤。
  - 使用 AWS CloudFormation (或受信任的第三方供應商) 來定義您的基礎設施。
    - 什麼是 AWS CloudFormation？
  - 使用良好的軟體設計原則，建立單一且分離的範本。
    - 確定實作的許可、範本和負責方。
      - 使用 AWS Identity and Access Management 控制存取
    - 使用以 Git 等常用技術為基礎的託管原始程式碼管理系統，來儲存原始程式碼和基礎設施即程式碼 (IaC) 組態。

資源

相關文件：

- APN 合作夥伴：可以幫助您建立自動化部署解決方案的合作夥伴
- AWS Marketplace：可用於自動化部署的產品
- AWS Well-Architected Framework：概念：執行手冊
- 什麼是 AWS CloudFormation？

相關範例：

- 使用程序手冊和執行手冊將操作自動化

REL08-BP02 將功能測試整合為部署的一部分

使用可驗證所需功能的技術，例如單元測試和整合測試。

單元測試是您測試程式碼的最小功能單元以驗證其行為的程序。整合測試旨在驗證每一項應用程式功能是否根據軟體需求運作。單元測試著重於單獨測試應用程式的部分，整合測試則會考量副作用 (例如，

透過改變操作變更資料的影響)。無論在哪一種情況下，測試都應整合到部署管道中，如果不符合成功條件，則管道會停止或復原。這些測試會在生產前環境中執行，而且會在生產前暫存於管道中。

當這些測試做為建置和部署動作的一部分自動執行時，您會獲得最佳成果。例如，使用 AWS CodePipeline 時，開發人員會將變更遞交至來源儲存庫，而 CodePipeline 會在該儲存庫中自動偵測變更。系統會建置應用程式並執行單元測試。通過單元測試後，建置的程式碼會部署至預備伺服器以進行測試。CodePipeline 會從預備伺服器執行更多測試，例如整合或負載測試。成功完成這些測試後，CodePipeline 會將已測試及已核准的程式碼部署至生產執行個體。

預期成果：您會使用自動化來執行單元和整合測試，以驗證程式碼的行為如預期。這些測試會整合到部署程序中，若測試失敗，則會中止部署。

常見的反模式：

- 您為了加快部署時間表，而忽略或略過在部署過程中的測試失敗和計畫。
- 可以在部署管道之外手動執行測試。
- 您透過手動緊急工作流程，跳過自動化中的測試步驟。
- 您執行自動化測試的環境與實際執行環境並非十分相似。
- 您建置的測試套件不夠靈活，且不易隨著應用程式發展進行維護、更新或擴展。

建立此最佳實務的優勢：在部署過程中進行自動化測試可及早發現問題，進而降低將有錯誤或非預期行為的版本發布至實際執行環境的風險。單元測試會驗證程式碼的行為確實依照要求，並遵循 API 合約。整合測試會驗證系統確實根據指定的需求運作。這些類型的測試會驗證元件 (例如使用者介面、API、資料庫和原始程式碼) 的預定工作順序。

未建立此最佳實務時的風險暴露等級：高

實作指引

採用測試驅動的開發 (TDD) 方法來撰寫軟體，藉此開發測試案例來指定和驗證程式碼。若要開始進行，請為每個函數建立測試案例。如果測試失敗，您會撰寫新的程式碼來通過測試。此方法可協助您驗證每個函數的預期結果。在將程式碼遞交至原始程式碼儲存庫之前，先執行單元測試並確認其通過測試。

在 CI/CD 管道的建置、測試和部署階段，實作單元和整合測試。自動化測試，並在應用程式有新版本可進行部署時自動初始化測試。如果未符合成功條件，則會終止或回復管道。

如果應用程式為 Web 或行動應用程式，則在多個桌面瀏覽器或實際的裝置上執行自動化整合測試。此方法對於驗證行動應用程式在各種不同裝置之間的相容性和功能特別有用。

實作步驟

1. 在撰寫功能程式碼 (測試驅動的開發或 TDD) 之前，先撰寫單元測試。建立程式碼指引，讓寫入和執行單元測試成為非功能性的編碼需求。
2. 建立自動化整合測試套件，當中涵蓋已識別的可測試功能。這些測試應模擬使用者互動並驗證預期結果。
3. 建立必要的測試環境以執行整合測試。當中可能包括與實際執行環境十分相似的預備或實際執行前環境。
4. 使用 AWS CodePipeline 主控台或 AWS Command Line Interface (CLI) 設定來源、建置、測試和部署階段。
5. 程式碼建置並測試完成後，部署應用程式。AWS CodeDeploy 可以將它部署到您的預備 (測試) 和實際執行環境。這些環境可包括 Amazon EC2 執行個體、AWS Lambda 函數或內部部署伺服器。您應使用相同的部署機制將應用程式部署到所有環境。
6. 監控管道的進度和每個階段的狀態。使用品質檢查，根據測試的狀態封鎖管道。也可以接收任何管道階段失敗或管線完成的通知。
7. 持續監控測試結果，並尋找較需要關注的模式、迴歸或區域。利用這些資訊改進測試套件、找出應用程式中需要更完善測試的區域，以及最佳化部署程序。

資源

相關的最佳實務：

- REL07-BP04 對工作負載執行負載測試
- REL08-BP03 將彈性測試整合為部署的一部分
- REL12-BP04 使用混沌工程測試彈性

相關文件：

- AWS 規範性指引：測試自動化
- 持續交付和持續整合
- 功能測試指標
- 監控管道
- 搭配 AWS CodeBuild 使用 AWS CodePipeline 以測試程式碼並執行建置
- AWS Device Farm

REL08-BP03 整合彈性測試作為部署的一部分

通過有意識地在系統中引入故障來整合彈性測試，以在破壞性情況下衡量其能力。彈性測試與通常整合在部署週期中的單元和功能測試不同，因為它們專注於識別系統中的意外故障。雖然從生產前的彈性測試整合開始是安全的，但應設定一個目標，在生產環境中實作這些測試，作為演練日的一部分。

預期成果：彈性測試有助於建立系統承受生產降級能力的信心。實驗可識別會導致故障的弱點，這有助於您改善系統，以自動且有效地緩解故障和降級。

常見的反模式：

- 在部署過程中缺乏可觀測性和監控
- 依賴人類解決系統故障
- 品質不佳的分析機制
- 專注於系統中的已知問題，缺乏識別任何未知因素的實驗
- 可識別故障，但沒有解決方案
- 沒有調查結果和執行手冊文件

建立最佳實務的優勢：在部署中整合的彈性測試有助於識別系統中未知的問題，否則這些問題會被忽視，從而導致生產中停機。在系統中識別這些未知因素可協助您記錄調查結果、將測試整合到您的 CI/CD 程序中以及建置執行手冊，透過高效率、可重複的機制簡化緩解作業。

未建立此最佳實務時的曝險等級：中

實作指引

可以整合到系統部署中的最常見彈性測試表單是災難復原和混沌工程。

- 在任何重大部署中，包含災難復原計劃和標準操作程序（SOPs）的更新。
- 將可靠性測試整合至您的自動化部署管道。諸如 AWS Resilience Hub 的服務可整合到您的 CI/CD 管道中，以建立持續的彈性評估，並在每次部署中自動進行評估。
- 在 中定義您的應用程式 AWS Resilience Hub。復原能力評估會產生程式碼片段，協助您將復原程序建立為應用程式的 AWS Systems Manager 文件，並提供建議的 Amazon CloudWatch 監控器和警示清單。
- 您的 DR 計劃和SOPs更新後，請完成災難復原測試，以確認它們是否有效。災難復原可協助您判斷是否可以在事件發生後還原系統並恢復正常操作。您可以模擬各種災難復原策略，並確定您的規劃是否足以滿足您的正常運行時間需求。常見的災難復原策略包括備份與還原、指示燈、冷待命、暖

待命、熱待命和主動-主動式，而且成本和複雜性都有所不同。在災難復原測試之前，建議您定義復原時間目標（RTO）和復原點目標（RPO），以簡化要模擬的策略選擇。 AWS 提供災難復原工具AWS Elastic Disaster Recovery，例如協助您開始規劃和測試。

- 混沌工程實驗引入了系統中斷，例如網路中斷和服務故障。透過模擬受控故障，您可以發現系統的漏洞，同時控制注入故障的影響。就像其他策略一樣，在非生產環境中使用諸如 AWS Fault Injection Service 等服務執行受控故障模擬，以在部署到生產環境之前獲得信心。

資源

相關文件：

- 使用彈性測試進行故障實驗以建立復原備
- 使用 AWS Resilience Hub 和 持續評估應用程式復原能力 AWS CodePipeline
- 上的災難復原（DR）架構 AWS，第 1 部分：雲端復原的策略
- 使用混沌工程確認工作負載的彈性
- 混沌工程的原則
- 混沌工程研討會

相關影片：

- AWS re:Invent 2020：使用混沌工程測試彈性
- 透過 AWS Fault Injection Service 改善應用程式彈性
- 使用 準備和保護您的應用程式免受中斷 AWS Resilience Hub

REL08-BP04 使用不可變基礎設施進行部署

不可變基礎設施是一種模式，要求在生產工作負載上不進行現場的更新、安全性修補或組態變更。需要進行變更時，會在新的基礎設施上建置架構並部署到生產環境。

請遵循不可變基礎設施的部署策略，以提高工作負載部署中的可靠性、一致性和可重複性。

預期成果：使用不可變基礎設施，不允許就地修改以執行工作負載內的基礎設施資源。相反地，在需要變更時，會以平行方式與現有資源一起部署新的、包含所有必要變更的更新後基礎設施資源集。此部署會自動進行驗證，如果成功，流量會逐漸轉移到新的資源集。

此部署策略適用於軟體更新、安全修補程式、基礎設施變更、組態更新和應用程式更新等。

常見的反模式：

- 對執行中的基礎設施資源實作就地變更。

建立此最佳實務的優勢：

- 提高跨環境的一致性：由於跨環境的基礎設施資源沒有差異，因此可以提高一致性並簡化測試。
- 降低組態偏移：透過將基礎設施資源取代為已知且版本控制的組態，可將基礎設施設為已知、經過測試且可信狀態，以避免組態偏移。
- 可靠的不可分割部署：部署可以順利完成，也可以保持不變，從而提高部署流程的一致性和可靠性。
- 簡化部署：部署不需要支援升級，因此會得到簡化。升級只是新的部署。
- 利用快速的回復及復原程序打造更安全的部署：前一個運作版本並未變更，因此部署變得更加安全。如果偵測到錯誤，您可以回復至該版本。
- 增強的安全狀態：透過不允許變更基礎設施，可以停用遠端存取機制 (例如 SSH)。這可減少攻擊媒介，從而改善組織的安全狀態。

未建立此最佳實務時的曝險等級：中

實作指引

自動化

定義不可變基礎設施部署策略時，建議盡可能使用自動化來提高可重複性並將人為錯誤的可能性降至最低。如需詳細資訊，請參閱 REL08-BP05 透過自動化部署變更和安全且無需人為干預的自動化部署。

使用基礎設施即程式碼 (IaC) 時，基礎設施佈建、協同運作和部署步驟會以程式設計、描述性和宣告式的方式定義，並儲存在原始檔控制系統中。利用基礎設施即程式碼可讓您更輕鬆地自動部署基礎設施，並協助您實現基礎設施不可變性。

部署模式

需要變更工作負載時，不可變基礎設施的部署策略會要求您部署一組新的基礎設施資源，包括所有必要的變更。這組新資源必須遵循推出模式，以最大程度地降低使用者所受到的影響。此部署有兩個主要策略：

Canary 部署：將少量客戶導向至新版本的實務，通常會在單一服務執行個體 (Canary) 上執行。之後，您可以仔細檢查所產生的任何行為變更或錯誤。如果遇到嚴重問題，可以從 Canary 中刪除流量，然後將使用者傳送回以前的版本。如果部署成功，則您可以繼續以期望的速度進行部署，同時監控變更是否有錯誤，直到完全部署為止。可以使用允許 Canary 部署的部署組態來設定 AWS CodeDeploy。

藍/綠部署：與 Canary 部署類似，不同之處在於整個應用程式須並行部署。您可在兩個堆疊 (藍色和綠色) 之間交替部署。再次強調，您可以將流量傳送到新版本，且如果發現部署問題，則可以回復到舊版本。通常，會一次切換所有流量，但您也可以將一小部分的流量用於每個版本，以使用 Amazon Route 53 的加權 DNS 路由功能，提高新版本的採用率。可以使用允許藍/綠部署的部署組態來設定 AWS CodeDeploy 和 AWS Elastic Beanstalk。

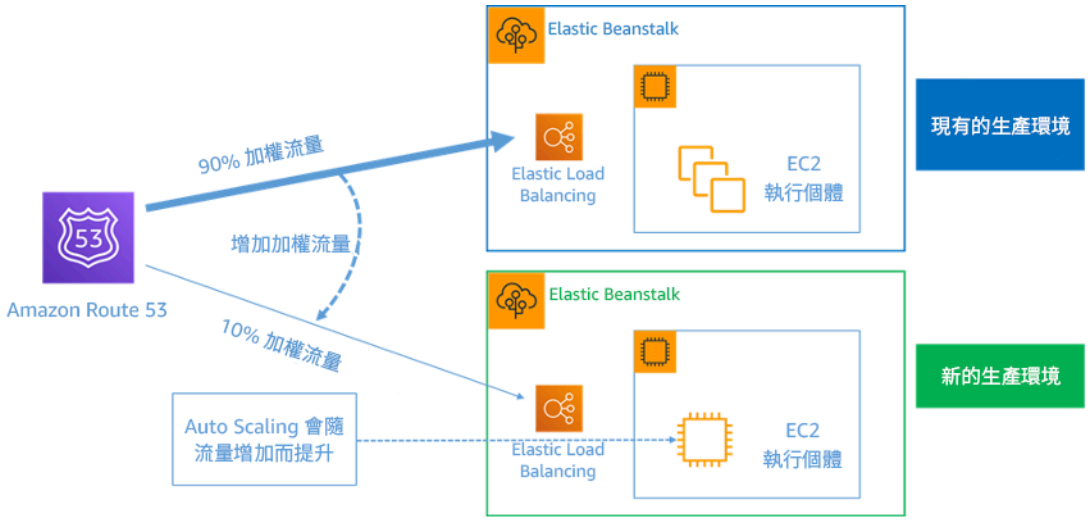


圖 8：使用 AWS Elastic Beanstalk 和 Amazon Route 53 進行藍/綠部署

漂移偵測

漂移被定義為導致基礎設施資源具有與預期不同的狀態或配置的任何變更。任何類型的未受管組態變更都違反了不可變基礎設施的概念，應加以偵測並修正，以便能成功實作不可變基礎設施。

實作步驟

- 禁止就地修改執行中的基礎設施資源。
  - 您可以使用 AWS Identity and Access Management (IAM) 指定誰或哪些服務可以存取 AWS 中的服務和資源、集中管理精細權限，以及分析存取以完善 AWS 中的權限。
- 自動部署基礎設施資源以提高可重複性，並最大限度地減少發生人為錯誤的可能性。
  - 如 DevOps on AWS 簡介中所述，自動化是 AWS 服務的基石，並在所有服務、功能和產品中獲得內部支援。
  - 預製您的 Amazon Machine Image (AMI) 可以加快啟動它們的時間。EC2 Image Builder 是一項全受管 AWS 服務，可協助您自動建立、維護、驗證、共用和部署自訂、安全且最新的 Linux 或 Windows 自訂 AMI。
  - 支援自動化的一些服務包括：

    - AWS Elastic Beanstalk 是一項服務，可在熟悉的伺服器 (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。
    - AWS Proton 協助平台團隊連接並協調開發團隊在基礎設施佈建、程式碼部署、監控和更新方面所需的所有不同工具。AWS Proton 可啟用無伺服器和容器型應用程式的自動化基礎設施即程式碼佈建和部署。
  - 利用基礎設施即程式碼可讓您輕鬆地自動部署基礎設施，並協助實現基礎設施的不可變性。AWS 會提供能以程式化、描述性和宣告式的方式建立、部署和維護基礎設施的服務。
    - AWS CloudFormation 幫助開發人員以有序和可預測的方式建立 AWS 資源。資源會使用 JSON 或 YAML 格式以文字檔撰寫。範本需要特定語法和結構，而這取決於所建立和管理的資源類型。您可以使用任何程式碼編輯器以 JSON 或 YAML 撰寫資源、將其簽入版本控制系統，然後 AWS CloudFormation 便會以安全、可重複的方式建置指定的服務。
    - AWS Serverless Application Model (AWS SAM) 是可用來在 AWS 上建置無伺服器應用程式的開放原始碼架構。AWS SAM 與其他 AWS 服務整合，是 AWS CloudFormation 的擴展功能。
    - AWS Cloud Development Kit (AWS CDK) 是一個開放原始碼軟體開發架構，可用來建模，並使用熟悉的程式設計語言來佈建您的雲端應用程式資源。您可以使用 AWS CDK 透過 TypeScript、Python、Java 和 .NET 來建模應用程式基礎設施。AWS CDK 會在背景中使用 AWS CloudFormation 以透過安全、可重複的方式佈建資源。
    - AWS 雲端控制 API 引入一組通用的「建立」、「讀取」、「更新」、「刪除」和「清單」(CRUDL) API，協助開發人員以簡單且一致的方式管理雲端基礎設施。Cloud Control API 通用 API 可讓開發人員統一管理 AWS 和第三方服務的生命週期。
- 實作可將使用者所受到的影響降到最低的部署模式。
  - Canary 部署：
    - 設定 API Gateway Canary 發行部署
    - 使用 AWS App Mesh 為 Amazon ECS 建立具有 Canary 部署的管道
  - 藍/綠部署：《AWS 上的藍/綠部署》白皮書說明了實作藍/綠部署策略的範例技術。
- 偵測組態或狀態的偏移。如需詳細資訊，請參閱 Detecting unmanaged configuration changes to stacks and resources。

資源

相關的最佳實務：

- REL08-BP05 使用自動化部署變更

相關文件：

- 自動化安全、無人為介入的部署
- 利用 AWS CloudFormation 在 Nubank 建立不可變基礎設施
- 基礎設施即程式碼
- 實作警示以自動檢測 AWS CloudFormation 堆疊中的漂移

相關影片：

- AWS re:Invent 2020：透過不變性實現可靠性、一致性和信心

REL08-BP05 使用自動化部署變更

部署和修補經過自動化以消除負面影響。

改變生產系統是許多組織的最大風險領域之一。我們認為，相較於軟體要解決的業務問題，部署才是我們要解決的首要問題。如今，這表示在營運中實際可行的地方使用自動化，包括測試和部署變更，新增或刪除容量以及移轉資料。

期望成果：可以透過廣泛的生產前測試、自動復原和交錯的生產部署，在發布過程中建置自動化部署安全性。這種自動化可將部署失敗所造成的潛在影響降到最低，而且開發人員不再需要主動觀察部署到生產環境的情況。

常見的反模式：

- 可以執行手動變更。
- 可以透過手動緊急工作流程，略過自動化中的步驟。
- 您不會遵循既定的計畫和流程，以加快時間表。
- 可以在不考慮封裝時間的情況下執行快速後續部署。

建立此最佳實務的優勢：當您使用自動化來部署所有變更時，可以移除導致人為錯誤的可能性，並能夠在變更生產之前進行測試。在生產推送之前執行此程序可驗證您的計畫是否已完成。此外，自動復原至您的發布程序可以識別生產問題，並將工作負載恢復到先前的作業狀態。

未建立此最佳實務時的曝險等級：中

實作指引

自動化您的部署管道。部署管道讓您可以調用自動測試、偵測異常，或者在生產部署之前的某個步驟中停止管道，或者自動回復變更。其中不可或缺的一部分就是採用持續整合和持續交付/部署 (CI/CD) 的文化，在這種文化中，提交或程式碼變更會經過各種自動化階段，從建置和測試階段到生產環境的部署。

儘管傳統觀點建議您將業內人員安排在營運程序中最困難的部分，但是出於這個原因，我們建議您能自動化最困難的程序。

實作步驟

可以依照下列步驟自動化部署以移除手動作業：

- 設定程式碼儲存庫以安全地儲存您的程式碼：使用以 Git 等常用技術為基礎的託管原始程式碼管理系統，來儲存原始程式碼和基礎設施即程式碼 (IaC) 組態。
- 設定持續整合服務以編譯原始程式碼、執行測試及建立部署成品：若要針對此目的設定建置專案，請參閱 Getting started with AWS CodeBuild using the console。
- 設定可自動化應用程式部署並處理應用程式更新複雜性的部署服務，而不需依賴容易出錯的手動部署：AWS CodeDeploy 可將軟體部署自動化至各種運算服務，例如 Amazon EC2、AWS Fargate、AWS Lambda 和內部部署伺服器。若要設定這些步驟，請參閱 Getting started with CodeDeploy。
- 設定持續交付服務，自動化您的發行管道，以便實現更快且更可靠的應用程式和基礎設施更新：請考慮使用 AWS CodePipeline 來協助您自動化發行管道。如需詳細資訊，請參閱 CodePipeline tutorials。

資源

相關的最佳實務：

- OPS05-BP04 使用建置和部署管理系統
- OPS05-BP10 完全自動化整合和部署
- OPS06-BP02 測試部署
- OPS06-BP04 自動化測試和復原

相關文件：

- 使用 AWS CodePipeline 連續交付巢狀 AWS CloudFormation 堆疊
- APN 合作夥伴：可以幫助您建立自動化部署解決方案的合作夥伴
- AWS Marketplace：可用於自動化部署的產品
- 使用 Webhook 自動化聊天訊息。
- Amazon 建置者資料中心：確保部署期間的回復安全
- Amazon 建置者資料中心：使用持續交付加快腳步
- 什麼是 AWS CodePipeline？
- 什麼是 CodeDeploy？
- AWS Systems Manager Patch Manager
- 什麼是 Amazon SES？
- 什麼是 Amazon Simple Notification Service？

相關影片：

- 2019 年 AWS 高峰會：AWS 上的 CI/CD

故障管理

> 故障是一定會發生的，一切最終都會隨著時間出現故障：從路由器到硬碟、從作業系統到毀損 TCP 封包的記憶體單位、從暫時性錯誤到永久故障，囊括方方面面。這是一定會發生的，無論您使用的是最高品質的硬體，還是成本最低的元件 – Werner Vogels，Amazon.com 技術長

低階硬體元件故障需每天在內部部署資料中心處理。不過，在雲端，您會受到保護，避免這類大多數的故障。例如，Amazon EBS 磁碟區放置在特定的可用區域中，會在該區域自動複寫，以保護您不受單一元件故障的影響。所有 EBS 磁碟區的設計都提供 99.999% 的可用性。Amazon S3 物件會跨至少三個可用區域存放，並在指定年度提供 99.999999999% 的物件耐久性。無論您的雲端供應商為何者，故障都有可能影響您的工作負載。因此，如果您需要工作負載擁有可靠性，則必須採取步驟來實作彈性。

套用此處所討論的最佳實務的先決條件是，您必須確保設計、實作和操作工作負載的人員清楚業務目標，以及實現這些目標的可靠性目標。這些人員必須了解這些可靠性要求並接受這些要求方面的培訓。

下列各節說明管理故障以避免對工作負載造成影響的最佳實務。

主題

- 備份資料
- 使用故障隔離來保護您的工作負載
- 設計工作負載以承受元件失敗
- 測試可靠性
- 災難復原 (DR) 計畫

備份資料

備份資料、應用程式和組態，以滿足復原時間點目標 (RTO) 和復原點目標 (RPO) 的要求。

最佳實務

- REL09-BP01 識別並備份所有需要備份的資料，或從來源複製資料
- REL09-BP02 保護和加密備份
- REL09-BP03 自動執行資料備份
- REL09-BP04 定期執行資料復原以驗證備份的完整性和程序

REL09-BP01 識別並備份所有需要備份的資料，或從來源複製資料

了解和使用工作負載所使用的資料服務和資源的備份功能。大部分服務都會提供備份工作負載資料的功能。

預期成果：已根據重要性識別並分類資料來源。然後，根據 RPO 建立資料復原的策略。此策略涉及備份這些資料來源，或具有從其他來源重現資料的能力。若遺失資料，實作的策略可讓您在定義的 RPO 和 RTO 內復原或重現資料。

雲端成熟度階段：基礎

常見的反模式：

- 未注意工作負載的所有資料來源及其關鍵性。
- 未備份關鍵資料來源。
- 只備份某些資料來源，而未使用關鍵性做為準則。
- 沒有已定義的 RPO，或備份頻率無法符合 RPO。
- 未評估是否需要備份，或是否可從其他來源重現資料。

建立此最佳實務的優勢：確定需要備份的位置並實作建立備份的機制，或是能夠從外部來源重製資料，這可提升在中斷時還原及復原資料的能力。

未建立此最佳實務時的曝險等級：高

實作指引

所有 AWS 資料存放區都會提供備份功能。Amazon RDS 和 Amazon DynamoDB 等服務會額外地支援啟用時間點復原 (PITR) 的自動備份，這可讓您將備份還原到目前時間之前最多五分鐘或更短的任何時間。許多 AWS 服務都提供將備份複製到另一個 AWS 區域的功能。AWS Backup 是一種工具，可讓您跨 AWS 服務集中化和自動化資料保護。AWS Elastic Disaster Recovery 可讓您複製完整的伺服器工作負載，並維護內部部署、跨可用區域或跨區域的持續資料保護，並以秒為單位測量復原點目標 (RPO)。

Amazon S3 可以用作自行管理和 AWS 管理的資料來源的備份目的地。Amazon EBS、Amazon RDS 和 Amazon DynamoDB 等 AWS 服務具有建立備份的內建功能。也可以使用第三方備份軟體。

可使用 AWS Storage Gateway 或 AWS DataSync，將內部部署資料備份至 AWS 雲端。Amazon S3 儲存貯體可用來在 AWS 中存放此資料。Amazon S3 提供多種儲存層級，例如 Amazon S3 Glacier 或 S3 Glacier Deep Archive，以降低資料儲存成本。

您能夠從其他資源重現資料來符合資料復原需求。例如，如果主節點遺失，您可以使用 Amazon ElastiCache 複本節點或 Amazon RDS 讀取複本來重現資料。如果這類來源可以用於滿足您的復原點目標 (RPO) 和復原時間點目標 (RTO)，則可能不需要備份。另一個範例是，如果使用 Amazon EMR，則可能不需要備份 HDFS 資料存放區，只要您可以從 Amazon S3 將資料複製到 Amazon EMR 即可。

選取備份策略時，請考慮復原資料所需的時間。復原資料所需的時間取決於備份的類型 (若有備份策略)，或資料重現機制的複雜性。此時間應該落在工作負載的 RTO 內。

實作步驟

1. 識別工作負載的所有資料來源。資料可以存儲在許多資源上，例如資料庫、磁碟區、檔案系統、日誌記錄系統和物件儲存。請參閱資源部分，尋找儲存資料之不同 AWS 服務的相關文件，以及這些服務提供的備份功能。
2. 根據重要性對資料來源進行分類。不同的資料集對工作負載具有不同的關鍵性等級，因此對彈性具有不同的要求。例如，有些資料可能至關重要，且需要接近零的 RPO，而其他資料可能不太重要，且可以容忍更高的 RPO 和一些資料遺失。同樣地，不同的資料集也可能具有不同的 RTO 要求。
3. 使用 AWS 或第三方服務來建立資料的備份。AWS Backup 是一項託管服務，它允許在 AWS 上建立各種資料來源的備份。AWS Elastic Disaster Recovery 可處理 AWS 區域 的自動次秒級資料複寫。大部分 AWS 服務也具有建立備份的原生功能。AWS Marketplace 具有許多也提供這些功能的解決方案。有關如何從各種 AWS 服務建立資料備份的資訊，請參閱下面列出的資源。
4. 對於未備份的資料，請建立資料複製機制。您可能基於各種原因選擇不備份可從其他來源重現的資料。可能有一種情況，即在需要時從來源重現資料比建立備份更便宜，因為可能有與儲存備份相關聯的成本。另一個範例是從備份中還原比從來源重現資料需要更長的時間，因而導致 RTO 中出現缺口。在這類情況下，考慮取捨並建立一個妥善定義的流程，其中指出在需要資料復原時如何從這些來源重現資料。例如，如果您已將資料從 Amazon S3 載入資料倉儲 (如 Amazon Redshift) 或 MapReduce 叢集 (如 Amazon EMR) 來分析該資料，這可以成為從其他來源複製的資料之範例。只要這些分析的結果存放在某處或可複製，您就不會因為資料倉儲或 MapReduce 叢集故障而遺失資料。其他可從來源複製的範例包括快取 (如 Amazon ElastiCache) 或 RDS 的僅供讀取複本。
5. 建立備份資料的節奏。建立資料來源的備份是一種定期流程，而且頻率應取決於 RPO。

實作計劃的工作量：中

資源

相關的最佳實務：

REL13-BP01 定義停機時間和資料遺失的復原目標

REL13-BP02 使用定義的復原策略來滿足復原目標

相關文件：

- 什麼是 AWS Backup？
- 什麼是 AWS DataSync？
- 什麼是 Volume Gateway？
- APN 合作夥伴：可以幫助備份的合作夥伴
- AWS Marketplace：可用於備份的產品
- Amazon EBS 快照
- 備份 Amazon EFS
- 備份 Amazon FSx for Windows File Server
- ElastiCache for Redis 的備份與還原
- 在 Neptune 中建立資料庫叢集快照
- 建立資料庫快照
- 建立依照排程觸發的 EventBridge 規則
- 使用 Amazon S3 進行跨區域複寫
- EFS 到 EFS AWS Backup
- 將日誌資料匯出至 Amazon S3
- 物件生命週期管理
- DynamoDB 的隨需備份與還原
- DynamoDB 的時間點復原
- 使用 Amazon OpenSearch Service 索引快照
- 什麼是 AWS Elastic Disaster Recovery？

相關影片：

- AWS re:Invent 2021 - 使用 AWS 進行備份、災難復原和勒索軟體保護
- AWS Backup 示範：跨帳戶和跨區域備份
- AWS re:Invent 2019：深入研究 AWS Backup, ft. Rackspace (STG341)

REL09-BP02 保護和加密備份

使用身分驗證和授權控制並偵測對備份的存取。使用加密來防止並檢測是否危及備份的資料完整性。

常見的反模式：

- 讓備份和還原自動化的存取權與資料的存取權相同。
- 不加密您的備份。

建立此最佳實務的優勢：保護您的備份可防止資料遭到竄改，加密資料可防止意外暴露時存取該資料。

未建立此最佳實務時的曝險等級：高

實作指引

使用身分驗證和授權控制並偵測對備份的存取，例如 AWS Identity and Access Management (IAM)。使用加密來防止並檢測是否危及備份的資料完整性。

Amazon S3 支援多種靜態資料的加密方法。使用伺服器端加密時，Amazon S3 會以未加密資料的形式接受物件，然後在儲存這些物件之前將其加密。使用用戶端加密時，您的工作負載應用程式需負責加密資料，然後將資料傳送至 Amazon S3。這兩種方法都可讓您使用 AWS Key Management Service (AWS KMS) 來建立和存放資料金鑰，或者您也可以提供自己的金鑰，之後由您對其負責。使用 AWS KMS 時，您可以透過 IAM 設定政策，設定誰可以和誰無法存取您的資料金鑰和解密資料。

對於 Amazon RDS，如果您已選擇加密資料庫，則備份也會加密。DynamoDB 備份一律會加密。使用 AWS Elastic Disaster Recovery 時，所有傳輸中的資料和靜態資料都會加密。使用 Elastic Disaster Recovery，靜態資料可以使用預設 Amazon EBS 加密磁碟區加密金鑰或自訂客戶受管金鑰進行加密。

實作步驟

1. 在每個資料存放區使用加密。如果來源資料已加密，則備份也會加密。
   - 在 Amazon RDS 中使用加密。您可以在建立 RDS 執行個體時，使用 AWS Key Management Service 設定靜態加密。
   - 在 Amazon EBS 磁碟區中使用加密。您可以在建立磁碟區時設定預設加密或指定唯一金鑰。
   - 使用必要的 Amazon DynamoDB 加密。DynamoDB 會加密所有靜態資料。您可以使用 AWS 自有的 AWS KMS 金鑰或 AWS 受管 KMS 金鑰，指定帳戶中儲存的金鑰。
   - 加密存放在 Amazon EFS 中的資料。在建立檔案系統時設定加密。
   - 在來源和目的地區域設定加密。您可以使用 KMS 中存放的金鑰來設定 Amazon S3 中的靜態加密，但金鑰是區域限定的。您可以在設定複寫時指定目的地金鑰。

- 選擇使用預設還是自訂 Amazon EBS 加密進行彈性災難復原。此選項會在模擬區域子網路磁碟和複寫磁碟上加密您的複寫靜態資料。

2. 實作存取備份的最低權限。遵循最佳實務，以根據安全最佳實務限制對備份、快照和複本的存取。

資源

相關文件：

- AWS Marketplace：可用於備份的產品
- Amazon EBS 加密
- Amazon S3：使用加密保護資料
- CRR 其餘組態：複寫使用 AWS KMS 中儲存的加密金鑰，透過伺服器端加密 (SSE) 所建立的物件
- DynamoDB 靜態加密
- 加密 Amazon RDS 資源
- 在 Amazon EFS 中加密資料和中繼資料
- AWS 中的備份加密
- 管理加密資料表
- 安全支柱 – AWS Well Architected Framework
- 什麼是 AWS Elastic Disaster Recovery？

REL09-BP03 自動執行資料備份

設定備份以根據復原點目標 (RPO) 所通知的定期排程或資料集中的變更自動執行。資料遺失要求低的關鍵資料集需要經常自動備份，而可以接受一些遺失的不太重要資料可以較不頻繁地備份。

預期成果：一種自動化過程，可以按既定的節奏建立資料來源的備份。

常見的反模式：

- 手動執行備份。
- 使用具有備份功能的資源，但不包含您的自動化中的備份。

建立此最佳實務的優勢：自動化備份會驗證是否根據您的 RPO 定期進行備份，並在未進行備份時提醒您。

未建立此最佳實務時的曝險等級：中

實作指引

AWS Backup 可用於建立各種 AWS 資料來源的自動資料備份。Amazon RDS 執行個體幾乎可以持續每五分鐘備份一次，而且 Amazon S3 物件幾乎可以持續每十五分鐘備份一次，同時將時間點復原 (PITR) 提供至備份歷史記錄內的特定時間點。針對其他 AWS 資料來源，例如 Amazon EBS 磁碟區、Amazon DynamoDB 資料表或 Amazon FSx 檔案系統，AWS Backup 可以頻繁地每小時執行自動備份。這些服務也提供原生備份功能。AWS 服務提供具有時間點復原功能的自動備份，包括 Amazon DynamoDB、Amazon RDS 和 Amazon Keyspaces (適用於 Apache Cassandra) — 這些服務可以還原到備份歷史記錄中的特定時間點。大部分其他 AWS 資料儲存服務都會提供定期備份排程的能力，頻率為每小時備份一次。

Amazon RDS 和 Amazon DynamoDB 提供具有時間點復原功能的連續備份。Amazon S3 版本控制一旦開啟，便會自動執行。Amazon Data Lifecycle Manager 可用來自動化建立、複製和刪除 Amazon EBS 快照。其也可以自動建立、複製、棄用和取消註冊 Amazon EBS 支援的 Amazon Machine Image (AMI) 及其基礎 Amazon EBS 快照。

AWS Elastic Disaster Recovery 提供從來源環境 (內部部署或 AWS) 到目標復原區域的持續區塊層級複寫。時間點 Amazon EBS 快照會由服務自動建立及管理。

為了集中檢視備份自動化和歷史記錄，AWS Backup 提供全受管的、基於政策的備份解決方案。它使用 AWS Storage Gateway 在雲端和內部部署中跨多個 AWS 服務，自動集中進行資料備份。

除版本控制之外，Amazon S3 還具有複寫功能。整個 S3 儲存貯體可自動複寫至相同或不同 AWS 區域 中的另一個儲存貯體。

實作步驟

1. 識別目前正在手動備份的資料來源。如需詳細資訊，請參閱REL09-BP01 識別並備份所有需要備份的資料，或從來源複製資料。
2. 確定工作負載的 RPO。如需詳細資訊，請參閱REL13-BP01 定義停機時間和資料遺失的復原目標。
3. 使用自動備份解決方案或受管服務。AWS Backup 是一項全受管的服務，可輕鬆在雲端和內部部署環境內集中管理並自動化各 AWS 服務間的資料保護。使用 AWS Backup 中的備份計劃建立規則，定義要備份的資源，以及應以何種頻率建立這些備份。此頻率應由步驟 2 中建立的 RPO 通知。如需有關如何使用 AWS Backup 建立自動化備份的實作指引，請參閱 Testing Backup and Restore of Data。大多數存放資料的 AWS 服務都會提供原生備份功能。例如，可以利用 RDS 搭配時間點復原 (PITR) 進行自動備份。

4. 對於自動備份解決方案或受管服務不支援的資料來源 (例如內部部署資料來源或訊息佇列)，請考慮使用受信任的第三方解決方案來建立自動備份。或者，您可以使用 AWS CLI 或 SDK 建立自動化來執行此動作。您可以使用 AWS Lambda Functions 或 AWS Step Functions，定義涉及建立資料備份的邏輯，並使用 Amazon EventBridge，以根據 RPO 的頻率調用它。

實作計劃的工作量：低

資源

相關文件：

- APN 合作夥伴：可以幫助備份的合作夥伴
- AWS Marketplace：可用於備份的產品
- 建立依照排程觸發的 EventBridge 規則
- 什麼是 AWS Backup？
- 什麼是 AWS Step Functions？
- 什麼是 AWS Elastic Disaster Recovery？

相關影片：

- AWS re:Invent 2019：深入研究 AWS Backup, ft. Rackspace (STG341)

REL09-BP04 定期執行資料復原以驗證備份的完整性和程序

透過執行復原測試，驗證您的備份程序實作是否符合復原時間點目標 (RTO) 和復原點目標 (RPO)。

預期成果：備份中的資料會使用良好定義的機制定期復原，以驗證是否可以在工作負載的既定復原時間點目標 (RTO) 內進行復原。驗證從備份中還原是否會導致資源包含原始資料 (而其中沒有任何損壞或無法存取)，但在復原點目標 (RPO) 內發生資料遺失。

常見的反模式：

- 還原備份，但不查詢或擷取任何資料，以檢查還原可用。
- 假設備份存在。
- 假設系統的備份可以完全運作，而且可以從中復原資料。
- 假設從備份中還原或復原資料的時間落在工作負載的 RTO 內。

- 假設備份上包含的資料落在工作負載的 RPO 內
- 在不使用執行手冊的情況下，或在建立的自動化程序外部，視需要還原。

建立此最佳實務的優勢：測試備份復原可驗證資料是否可以在需要時還原，而不必擔心資料可能遺失或損毀，可在工作負載的 RTO 內進行還原和復原，而且任何資料遺失都屬於工作負載的 RPO 範圍。

未建立此最佳實務時的曝險等級：中

實作指引

測試備份和還原功能可以提高能夠在中斷期間執行這些動作的信心。定期將備份還原至新位置，並執行測試以驗證資料的完整性。某些應該執行的常用測試會檢查所有資料是否可用、未損毀、可存取，且任何資料遺失落在工作負載的 RPO 內。此類測試也可以協助確定，復原機制是否足夠快到適應工作負載的 RTO。

使用 AWS 時，您可以建立一個測試環境，還原備份來評估 RTO 和 RPO 功能，並針對資料內容和完整性執行測試。

此外，Amazon RDS 和 Amazon DynamoDB 允許時間點復原 (PITR)。使用持續備份時，您可以將資料集還原到指定日期和時間當時的狀態。

所有資料是否可用、未損壞、可存取，並且任何資料遺失都落在工作負載的 RPO 內。此類測試也可以協助確定，復原機制是否足夠快到適應工作負載的 RTO。

AWS Elastic Disaster Recovery 提供 Amazon EBS 磁碟區的持續時間點復原快照。複寫來源伺服器時，會根據設定的政策，隨著時間的推移記錄時間點狀態。彈性災難復原可透過啟動執行個體進行測試和演練，而無須重新導向流量，從而協助您驗證這些快照的完整性。

實作步驟

1. 識別目前正在備份的資料來源，以及這些備份的儲存位置。如需實作指引，請參閱 REL09-BP01 識別並備份所有需要備份的資料，或從來源複製資料。
2. 針對每個資料來源建立資料驗證條件。不同類型的資料將具有不同的屬性，可能需要不同的驗證機制。在您自信可於生產環境中使用此資料之前，請考慮如何驗證它。一些驗證資料的常用方法是使用資料和備份屬性，例如資料類型、格式、檢查總和、大小，或這些屬性與自訂驗證邏輯的組合。例如，這可能是建立備份時所還原資源與資料來源之間的檢查總和值比較。
3. 建立 RTO 和 RPO，以根據資料重要性還原資料。如需實作指引，請參閱 REL13-BP01 定義停機時間和資料遺失的復原目標。

4. 評估您的復原能力。審核您的備份和還原策略，以了解它是否可以符合您的 RTO 和 RPO，並視需要調整策略。使用 AWS Resilience Hub，可以執行工作負載的評估。此評定會針對彈性政策評估您的應用程式組態，並報告您的 RTO 和 RPO 目標是否可以實現。
5. 使用生產中用於資料還原的當前已建立的流程進行測試還原。這些程序取決於原始資料來源的備份方式、備份本身的格式和儲存位置，或是否已從其他源重現資料。例如，如果您使用的是諸如 AWS Backup 等受管服務，這可能就像將備份還原到新資源一樣簡單。如果使用 AWS Elastic Disaster Recovery，則可以啟動復原演練。
6. 根據先前建立的資料驗證條件，從已還原的資源中驗證資料復原。還原和復原的資料是否包含備份時最新的記錄/項目？ 此資料是否落在工作負載的 RPO 內？
7. 測量還原和復原所需的時間，並將其與已確定的 RTO 進行比較。此程序是否落在工作負載的 RTO 內？ 例如，比較從還原程序開始到復原驗證完成的時間戳記，以計算此程序需要多長時間。所有 AWS API 呼叫都有時間戳記，而且此資訊在 AWS CloudTrail 中可用。儘管此資訊可以提供有關還原程序何時開始的詳細資訊，但驗證完成時的結束時間戳記應由驗證邏輯記錄。如果使用自動流程，則可以使用 Amazon DynamoDB 之類的服務來存放此資訊。此外，許多 AWS 服務會提供事件歷史記錄，其中提供特定動作何時發生的時間戳記資訊。在 AWS Backup 中，備份和還原動作稱為工作，而這些工作包含時間戳記資訊做為其中繼資料的一部分，可用來測量還原和復原所需的時間。
8. 如果資料驗證失敗，或如果還原和復原所需的時間超過工作負載的既定 RTO，請通知利益相關者。實作自動化以執行此操作時，例如在此實驗室中，可以使用 Amazon Simple Notification Service (Amazon SNS) 等服務向利益相關者傳送推送通知，例如電子郵件或 SMS。這些訊息也可以發佈到傳訊應用程式 (例如 Amazon Chime、Slack 或 Microsoft Teams)，或用於使用 AWS Systems Manager OpsCenter 將任務建立為 OpsItems。
9. 將此流程自動化以定期執行。例如，諸如 AWS Lambda 或 AWS Step Functions 中的狀態機器等服務可以用來將還原和復原程序自動化，而且 Amazon EventBridge 可以用來定期調用此自動化工作流程，如下面架構圖所示。了解如何使用 AWS Backup 自動化資料復原驗證。此外，此 Well-Architected 實驗室為這裡的幾個步驟提供了一種自動化方法的實際操作體驗。

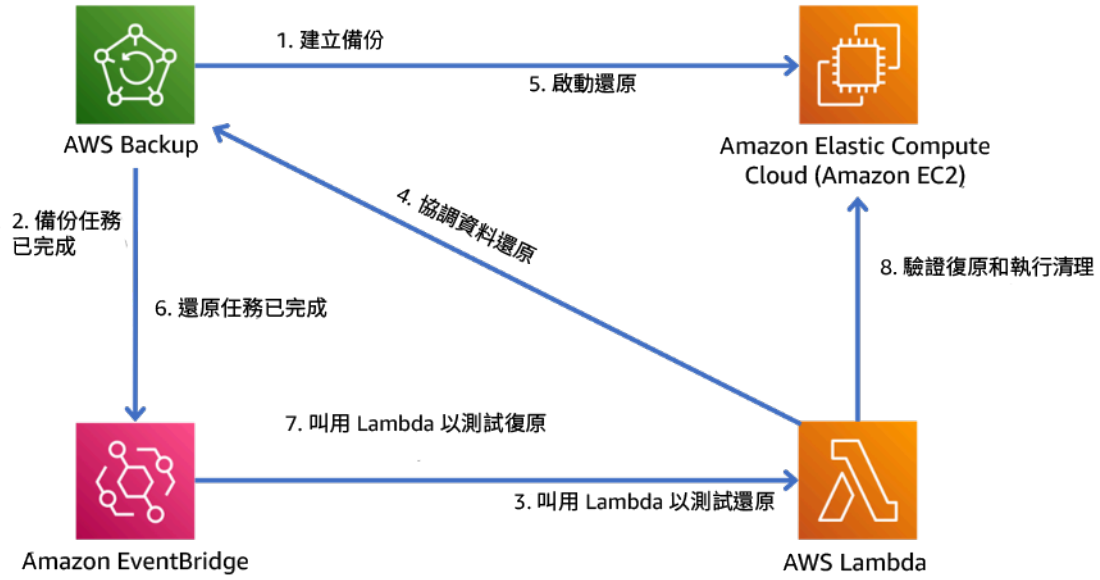


圖 9. 自動備份和還原流程

實施計劃的工作量：中到高，取決於驗證條件的複雜性。定。

資源

相關文件：

- 使用 AWS Backup 自動化資料復原驗證
- APN 合作夥伴：可以幫助備份的合作夥伴
- AWS Marketplace：可用於備份的產品
- 建立依照排程觸發的 EventBridge 規則
- DynamoDB 的隨需備份與還原
- 什麼是 AWS Backup？
- 什麼是 AWS Step Functions？
- 什麼是 AWS Elastic Disaster Recovery
- AWS Elastic Disaster Recovery

使用故障隔離來保護您的工作負載

故障隔離會將元件或系統故障的影響侷限在定義的界限內。藉由適當的隔離，界限外部的元件就不會受到故障影響。只要能跨多個故障隔離界限執行工作負載，就可更靈活地應對故障。

最佳實務

- REL10-BP01 將工作負載部署至多個位置
- REL10-BP02 針對限制在單一位置的元件將復原自動化
- REL10-BP03 使用隔板架構限制影響範圍

REL10-BP01 將工作負載部署至多個位置

跨多個可用區域或視需要跨 AWS 區域，分配工作負載資料和資源。

AWS 中服務設計的基本原則是避免單一故障點，包括基礎實體基礎設施。AWS 在全球多個地理位置 (稱為區域) 提供雲端運算資源和服務。每個區域實際上和邏輯上都各自獨立，並且由三個以上的可用區域 (AZ) 所組成。可用區域在地理位置上彼此接近，但實際上是分開且隔離的。若您將工作負載分配到不同的可用區域和區域，就可降低發生火災、洪水、天災、地震和人為錯誤等威脅的風險。

建立位置策略來提供適合您工作負載的高可用性。

預期成果：實際執行工作負載會分散到多個可用區域 (AZ) 或區域，以實現容錯能力和高可用性。

常見的反模式：

- 您的實際行工作負載只存在單一可用區域中。
- 當多可用區域架構可滿足業務需求時，您卻實作多區域架構。
- 您的部署或資料變得不同步，進而導致組態偏離或資料複寫不足。
- 您未考量在這些元件之間的彈性和多位置需求不同的情況下，應用程式元件之間的相依性。

建立此最佳實務的優勢：

- 您的工作負載對於影響 AZ 或整個區域的事件較有彈性，例如電力或環境控制故障、天災、上游服務故障或網路問題。
- 您可以存取更廣泛的 Amazon EC2 執行個體庫存，並降低啟動特定 EC2 執行個體類型時發生 InsufficientCapacityExceptions (ICE) 的可能性。

未建立此最佳實務時的曝險等級：高

實作指引

至少在區域中的兩個可用區域 (AZ) 中部署和操作所有實際執行工作負載。

使用多個可用區域

可用區域是託管資源的位置，這些位置實際上彼此分隔，以避免因火災、洪水和龍捲風等風險而導致相互關聯失敗。每個可用區域都有單獨的實體基礎設施，包括公用電源連接、備用電源、機械服務，以及網路連線。此配置可將上述任一元件中的故障限制在只有受影響的可用區域。例如，若 AZ 範圍的事件使得受影響的可用區域中的 EC2 執行個體無法使用，您在其他可用區域中的執行個體仍然可用。

雖然實際上彼此分隔，但相同 AWS 區域 中的可用區域彼此接近，因此能提供高輸送量、低延遲 (僅幾毫秒) 聯網。您可以在可用區域之間同步複寫大多數工作負載的資料，而不會顯著影響使用者體驗。這表示，您可以在主動/主動或主動/待命組態中，與區域中使用可用區域。

所有與工作負載相關聯的運算都應分散到多個可用區域。這包括 Amazon EC2 執行個體、AWS Fargate 任務和 VPC 連接的 AWS Lambda 函數。包括 EC2 Auto Scaling、Amazon Elastic Container Service (ECS) 和 Amazon Elastic Kubernetes Service (EKS) 在內的 AWS 運算服務，為您提供了在不同可用區域之間啟動和管理運算的方法。將它們設定為在不同的可用區域中視需要自動取代運算，以維持可用性。若要將流量引導至可用的可用區域，請在運算前放置負載平衡器，例如 Application Load Balancer 或 Network Load Balancer。AWS 負載平衡器可以在可用區域受損時，將流量重新路由至可用的執行個體。

您也應該複寫工作負載的資料，並且在多個可用區域中提供。有些 AWS 受管資料服務預設會在多個可用區域中複寫資料，例如 Amazon S3、Amazon Elastic File Service (EFS)、Amazon Aurora、Amazon DynamoDB、Amazon Simple Queue Service (SQS) 和 Amazon Kinesis Data Streams，這些服務面對可用區域受損的情形時相當可靠。使用其他 AWS 受管資料服務，例如 Amazon Relational Database Service (RDS)、Amazon Redshift 和 Amazon ElastiCache 時，您必須啟用多可用區域複寫。啟用後，這些服務會自動偵測可用區域受損、將請求重新導向至可用的可用區域，並在復原後視需要重新複寫資料，而不需客戶介入。詳讀您使用的每一項 AWS 受管資料服務的使用者指南，以了解其多可用區域功能、行為和操作。

如果您使用的是自我管理儲存體，例如 Amazon Elastic Block Store (EBS) 磁碟區或 Amazon EC2 執行個體儲存體，您必須自行管理多可用去複寫。

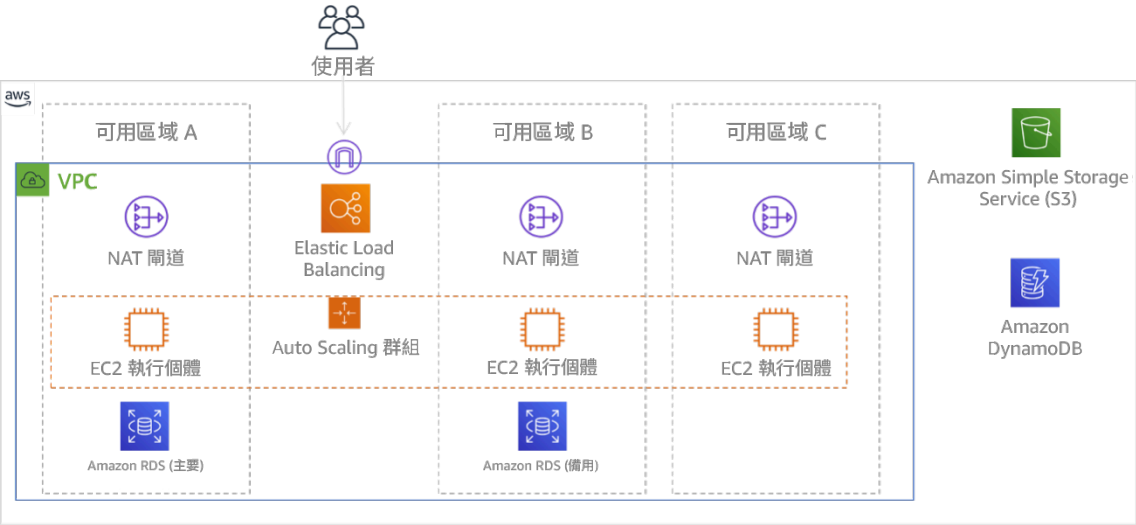


圖 9：跨三個可用區域部署的多層架構。請注意，Amazon S3 和 Amazon DynamoDB 一律自動採用異地同步備份策略。ELB 也會部署至全部三個區域。

使用多個 AWS 區域

如果您的工作負載需要相當大的彈性 (例如，關鍵基礎設施、運作狀態相關應用程式，或是有嚴格的客戶或強制性可用性需求的服務)，則您可能需要額外的可用性，而這並非單一 AWS 區域 能夠提供。在此情況下，您應該至少在兩個 AWS 區域 (假設您的資料落地要求允許) 中部署和操作工作負載。

AWS 區域 位於全球不同的地理區域和多個不同的大陸上。單是 AWS 區域 的實際分隔和隔離距離甚至就比可用區域還大。AWS 服務 (除了少數例外) 利用此設計在不同區域 (也稱為區域服務) 之間完全獨立運作。AWS 區域 服務的故障設計為，不會影響不同區域中的服務。

當您在多個區域中操作工作負載時，應考慮其他需求。由於不同區域中的資源彼此分隔且獨立，因此您必須在每個區域中複寫工作負載的元件。除了運算和資料服務之外，還包括基本的基礎設施，例如 VPC。

注意：當您考慮多區域設計時，務必確認您的工作負載能夠在單一區域中執行。如果您在區域之間建立相依性，讓某一個區域中的元件依賴另一個區域中的服務或元件，則可能會使得失敗的風險升高，並顯著降低可靠性狀態。

為了簡化多區域部署並維持一致性，AWS CloudFormation StackSets 可將整個 AWS 基礎設施複寫到多個區域。AWS CloudFormation 還可以偵測組態偏離，並且在區域中的 AWS 資源不同步時通知您。許多 AWS 服務都為重要的工作負載資產提供多區域複寫。例如，EC2 Image Builder 可以在每次建置之後，將 EC2 機器映像 (AMI) 發布到您使用的每個區域。Amazon Elastic Container Registry (ECR) 可以將容器映像複寫到您選取的區域。

您也必須將資料複寫到您選擇的每一個區域。有許多 AWS 受管資料服務都提供跨區域複寫功能，包括 Amazon S3、Amazon DynamoDB、Amazon RDS、Amazon Aurora、Amazon Redshift、Amazon Elasticache 和 Amazon EFS。Amazon DynamoDB 全域資料表接受寫入任何支援的區域中，並且會將資料複寫到您設定的所有其他區域中。使用其他服務時，您必須指定用於寫入的主要區域，因為其他區域包含唯讀複本。對於工作負載使用的每個 AWS 受管資料服務，請參閱其使用者指南和開發人員指南，以了解其多區域功能和限制。請特別注意寫入的方向、交易功能和限制、複寫執行方式，以及如何監控區域之間的同步作業。

AWS 還提供了將請求流量路由至區域部署的能力，帶來了極大的靈活度。例如，您可以使用 Amazon Route 53 設定 DNS 記錄，將流量導向距離使用者最近的可用區域。您也可以在主動/待命組態中設定您的 DNS 記錄，在其中指定一個區域做為主要區域，並且僅在主要區域運作狀態不佳時，容錯移轉至另一個區域複本。您可以設定 Route53 運作狀態檢查，以偵測運作狀態不佳的端點，並執行自動容錯移轉，並視需要額外使用 Amazon Application Recovery Controller (ARC) 來提供高度可用的路由控制，以手動重新路由流量。

即使您選擇不在多個區域中操作以實現高可用性，仍務必將多個區域納入您的災難復原 (DR) 策略中。如果可能，請在次要區域中，於暖待命或指示燈組態中複寫工作負載的基礎設施元件和資料。在此設計中，您會從主要區域複寫基準基礎設施，例如 VPC、Auto Scaling 群組、容器協調器和其他元件，但您會將待命區域中的可變大小的元件 (例如 EC2 執行個體和資料庫複本的數量) 設定為最小可操作大小。您也可以安排從主要區域持續複寫資料至待命區域。如果有事件發生，您可以橫向擴充或擴大待命區域中的資源，然後將其提升為主要區域。

實作步驟

1. 與業務利害關係人和資料落地專家合作，確定哪些 AWS 區域 可以用來託管您的資源和資料。
2. 與業務和技術利害關係人合作，評估您的工作負載，並判斷多可用區域方法 (單一 AWS 區域) 是否可滿足其彈性需求，或者需要多區域方法 (若允許多個區域)。使用多個區域可以實現更高的可用性，但可能也會帶來額外的複雜性和成本。評估時，請考慮下列因素：
   a. 業務目標和客戶需求：在可用區域或區域中發生影響工作負載的事件時，允許多久的停機時間？如 REL13-BP01 定義停機時間和資料遺失的復原目標中所述，評估復原點目標。
   b. 災難復原 (DR) 需求：您希望確實防範哪種可能的災難？ 請考慮從單一可用區域到整個區域的不同影響範圍下，資料遺失或長期無法使用的可能性。如果您將資料和資源複寫到不同的可用區域，而某一個可用區域發生持續故障，您可以在另一個可用區域中復原服務。如果您將資料和資源複寫到不同的區域，就可以在另一個區域中復原服務。
3. 將運算資源部署到多個可用區域中。

a. 在 VPC 中，於不同可用區域建立多個子網路。設定每個子網路，使其足以容納處理工作負載所需的資源，即使在發生事件時也一樣。如需詳細資訊，請參閱 REL02-BP03 確保 IP 子網路配置帳戶具有擴展性和可用性。

b. 如果您使用 Amazon EC2 執行個體，請使用 EC2 Auto Scaling 來管理執行個體。建立 Auto Scaling 群組時，請指定您在上一個步驟中選擇的子網路。

c. 如果您針對 Amazon ECS 或 Amazon EKS 使用 AWS Fargate 運算，請在建立 ECS Service、啟動 ECS 任務，或是為 EKS 建立 Fargate 設定檔時，選取您在第一個步驟中選擇的子網路。

d. 如果您使用的是需要在 VPC 中執行的 AWS Lambda 函數，請在建立 Lambda 函數時選取您在第一個步驟中選擇的子網路。對於沒有 VPC 組態的任何函數，AWS Lambda 會自動為您管理可用性。

e. 將負載平衡器等流量導向器放在運算資源前面。如果啟用跨區域負載平衡，AWS Application Load Balancer 和 Network Load Balancer 會偵測 EC2 執行個體和容器等目標何時因可用區域受損而無法連線，並將流量重新路由至運作狀態良好的可用區域中的目標。如果您停用跨區域負載平衡，請使用 Amazon Application Recovery Controller (ARC) 來提供區域轉移功能。如果您使用第三方負載平衡器，或已實作自己的負載平衡器，請在不同的可用區域中為它們設定多個前端。

4. 在多個可用區域中複寫工作負載的資料。

   a. 如果您使用 AWS 受管資料服務，例如 Amazon RDS、Amazon ElastiCache 或 Amazon FSx，請詳讀其使用者指南，以了解其資料複寫和復原功能。視需要啟用跨可用區複寫和容錯移轉。

   b. 如果您使用 AWS 受管儲存服務，例如 Amazon S3、Amazon EFS 和 Amazon FSx，請避免針對需要高耐久性的資料使用單一可用區域或單一區域組態。針對這些服務使用多可用區域組態。詳閱個別服務的使用者指南，判斷多可用區域複寫為預設啟用，或是您必須啟用它。

   c. 如果您執行自我管理的資料庫、佇列或其他儲存服務，請根據應用程式的說明或最佳實務安排多可用區域複寫。熟悉應用程式的容錯移轉程序。

5. 設定您的 DNS 服務，使其偵測 AZ 受損情形，並將流量重新路由至運作狀態良好的可用區域。Amazon Route 53 與 Elastic Load Balancer 搭配使用時，可以自動執行此操作。Route 53 也可以設定為擁有容錯移轉記錄，這些記錄使用運作狀態檢查來回應查詢，並且只提供運作狀態良好的 IP 位址。對於用於容錯移轉的任何 DNS 記錄，請指定短存留時間 (TTL) 值 (例如 60 秒或更短)，這樣做有助於防止記錄快取阻礙復原 (Route 53 別名記錄會為您提供適當的 TTL)。

使用多個 AWS 區域 時的其他步驟

1. 複寫工作負載在所選區域中使用的所有作業系統 (OS) 和應用程式程式碼。視需要使用 Amazon EC2 Image Builder 等解決方案複寫 EC2 執行個體所使用的 Amazon Machine Image (AMI)。使用

Amazon ECR 跨區域複寫等解決方案複寫存放在登錄檔中的容器映像。為任何用於儲存應用程式資源的 Amazon S3 儲存貯體啟用區域複寫。

2. 將運算資源和組態中繼資料 (例如存放在 AWS Systems Manager 參數存放區的參數) 部署到多個區域。使用先前步驟所述的相同程序，但是複寫您用於工作負載的每個區域的組態。使用基礎設施即程式碼解決方案 (例如 AWS CloudFormation)，在區域之間統一重複產生組態。如果您在災難復原的指示燈組態中使用次要區域，您可以將運算資源的數量減少到最小值以節省成本，並相應地增加復原時間。
3. 將您的資料從主要區域複寫到次要區域。
   a. Amazon DynamoDB 全域資料表提供可從任何支援的區域寫入的全域資料複本。使用其他 AWS 受管資料服務時 (例如 Amazon RDS、Amazon Aurora 和 Amazon Elasticache)，您可以指定主要 (讀取/寫入) 區域和複本 (唯讀) 區域。如需區域複寫的詳細資訊，請參閱個別服務的使用者指南和開發人員指南。
   b. 如果您執行自我管理的資料庫，請根據應用程式的說明或最佳實務安排多區域複寫。熟悉應用程式的容錯移轉程序。
   c. 如果您的工作負載使用 AWS EventBridge，您可能需要將所選事件從主要區域轉送至次要區域。若要這麼做，請將次要區域中的事件匯流排指定為主要區域中相符事件的目標。
4. 考慮是否要在不同區域使用相同的加密金鑰，以及使用的程度。在安全性和易用性之間取得平衡的典型方法，是針對區域本機資料和身分驗證使用區域範圍金鑰，以及使用全域範圍金鑰來加密在不同區域之間複寫的資料。AWS Key Management Service(KMS) 支援多區域金鑰，它可安全地分配和保護在區域之間共用的金鑰。
5. 考慮使用 AWS Global Accelerator，透過將流量導向包含運作狀態良好之端點的區域來改善應用程式的可用性。

資源

相關的最佳實務：

- REL02-BP03 確保 IP 子網路配置帳戶具有擴展性和可用性
- REL11-BP05 使用靜態穩定性來防止雙模態行為
- REL13-BP01 定義停機時間和資料遺失的復原目標

相關文件：

- AWS 全球基礎設施
- 白皮書：AWS 故障隔離界限

- Amazon EC2 Auto Scaling 的恢復能力
- Amazon EC2 Auto Scaling：範例：將執行個體發布到不同的可用區域
- EC2 Image Builder 的運作方式
- Amazon ECS 如何將任務放置在容器執行個體上 (包括 Fargate)
- AWS Lambda 中的恢復能力
- Amazon S3：複寫物件概觀
- Amazon ECR 中的私有映像複寫
- 全域資料表：使用 DynamoDB 進行多區域複寫
- Amazon Elasticache for Redis OSS：使用全域資料儲存在 AWS 區域 之間複寫
- Amazon RDS 的恢復能力
- 使用 Amazon Aurora 全球資料庫
- AWS Global Accelerator 開發人員指南
- AWS KMS 中的多區域金鑰
- Amazon Route 53：設定 DNS 備援
- Amazon Application Recovery Controller (ARC) 開發人員指南
- 在 AWS 區域之間傳送和接收 Amazon EventBridge 事件
- 使用 AWS Services 部落格系列建立多區域應用程式
- AWS 上的災難復原 (DR) 架構，第一部分：雲端復原策略
- AWS 上的災難復原 (DR) 架構，第 III 部分：指示燈和暖待命

相關影片：

- AWS re:Invent 2018：適用於多區域主動-主動應用程式的架構模式
- AWS re:Invent 2019：AWS 全球網路基礎設施的創新和營運

REL10-BP02 針對限制在單一位置的元件將復原自動化

如果工作負載的元件只能在單一可用區域或內部部署資料中心執行，在定義的復原目標內實作完整重建工作負載的功能。

未建立此最佳實務時的曝險等級：中

實作指引

如果因為技術限制而無法實作將工作負載部署至多個位置的最佳實務，您必須實作彈性的替代路徑。您必須將以下能力自動化：重新建立必要基礎設施、重新部署應用程式，以及針對這些案例重新建立必要資料。

例如，Amazon EMR 會在相同可用區域中啟動指定叢集的所有節點，因為在相同區域執行叢集可以提供更高的資料存取速率，從而能提高任務流程的效能。如果為實現工作負載彈性而需要此元件，您必須要有方法重新部署叢集及其資料。此外，對於 Amazon EMR，您還應以異地同步備份以外的方式佈建冗餘。可以佈建多個節點。使用 EMR 檔案系統 (EMRFS) 時，EMR 中的資料可存放在 Amazon S3 中，然後可複寫至多個可用區域或 AWS 區域。

同樣地，對於 Amazon Redshift，它預設會將叢集佈建在您所選 AWS 區域 內隨機選取的可用區域中。所有叢集節點將佈建在相同的區域中。

針對部署到內部部署資料中心的有狀態的伺服器型工作負載，您可以使用 AWS Elastic Disaster Recovery 在 AWS 中保護您的工作負載。如果已經在 AWS 中託管，則可以使用彈性災難復原將工作負載保護到備選可用區域或區域。彈性災難復原使用輕量型暫存區的持續區塊層級複寫，以提供內部部署應用程式和雲端應用程式的快速且可靠的復原。

實作步驟

1. 實作自我修復。盡可能使用 Automatic Scaling 來部署執行個體或容器。如果無法使用 Automatic Scaling，請對 EC2 執行個體使用自動復原，或者根據 Amazon EC2 或 ECS 容器生命週期事件實作自我修復自動化。
   - 對於不需要單個執行個體 IP 位址、私有 IP 位址、彈性 IP 位址和執行個體中繼資料的執行個體和容器工作負載，使用 Amazon EC2 Auto Scaling 群組。
     - 啟動範本使用者資料可用於實現自動自我修復大多數工作負載。
   - 對於需要單個執行個體 IP 位址、私有 IP 位址、彈性 IP 位址和執行個體中繼資料的工作負載，使用 Amazon EC2 執行個體的自動復原。
     - 在偵測到執行個體失敗時，自動復原會將提醒傳送到 SNS 主題。
   - 在無法使用 Auto Scaling 或 EC2 復原的情況下，使用 Amazon EC2 執行個體生命週期事件或 Amazon ECS 事件自動執行自我修復。
     - 使用事件來調用自動化，以根據您所需的過程邏輯來修復您的元件。
   - 使用 AWS Elastic Disaster Recovery 保護僅限於單一位置的有狀態工作負載。

資源

相關文件：

- Amazon ECS 事件
- Amazon EC2 Auto Scaling lifecycle hooks
- 復原您的執行個體。
- 服務自動擴展
- 什麼是 Amazon EC2 Auto Scaling？
- AWS Elastic Disaster Recovery

REL10-BP03 使用隔板架構限制影響範圍

實作隔板架構 (也稱為小組型架構) 將工作負載內的失敗效應限制為有限數量的元件。

預期成果：小組型架構使用工作負載的多個獨立執行個體，其中每個執行個體稱為小組。每個小組都是獨立的，不會與其他小組共用狀態，並且處理整體工作負載請求的子集。這會對個別小組和它處理的請求降低失敗的潛在影響，例如不良的軟體更新。如果工作負載使用 10 個小組為 100 個請求提供服務，發生失敗時，整體請求中 90% 不會受到失敗影響。

常見的反模式：

- 允許小組成長，沒有界限。
- 將程式碼更新或部署同時套用到所有小組。
- 在小組之間共用狀態或元件 (路由器層例外)。
- 將複雜商業或路由邏輯新增至路由器層。
- 不將跨小組互動降至最低。

建立此最佳實務的優勢：使用小組型架構時，小組本身包含許多常見的故障類型，從而提供額外的故障隔離。這些故障界限可針對難以控制的失敗類型提供復原能力，例如失敗的程式碼部署或已損毀或叫用特定失敗模式 (也稱為毒藥請求) 的請求。

未建立此最佳實務時的曝險等級：高

實作指引

在船上，隔板可確保船體破口包含在船體的其中一個區段內。在複雜的系統中，通常會複寫這個模式以實現故障隔離。故障隔離界限會在工作負載內將失敗影響限制為有限數量的元件。界限外部的元件不會受到故障影響。您可以使用多個錯誤隔離界限來限制對工作負載的影響。在 AWS 上，客戶可以使用多個可用區域或區域來提供故障隔離，但是故障隔離的概念也可以延伸為您的工作負載的架構。

整體工作負載是依分割區索引鍵的分割區小組。此索引鍵需要與服務的精細度保持一致，否則服務的工作負載會自然地透過最小的跨小組互動進行細分。分割區索引鍵的範例為客戶 ID、資源 ID 或可在大部分 API 呼叫中輕易存取的其他任何參數。小組路由層會根據分割區索引鍵將請求分散到個別小組，並且對用戶端呈現單一端點。

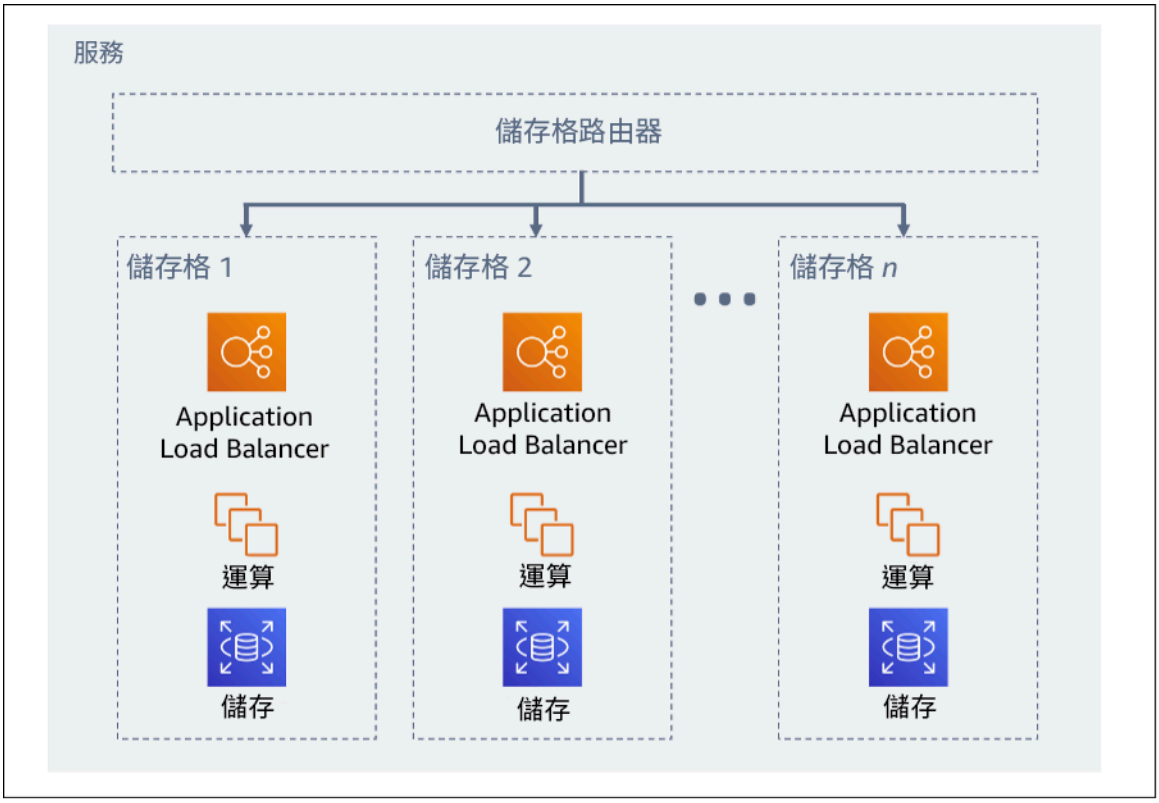


圖 11：小組型架構

實作步驟

設計小組型架構時，要考慮數個設計考量：

1. 分割區索引鍵：選擇分割區索引鍵時，應特別考慮。
   - 應該與服務的精細度保持一致，否則服務的工作負載會自然地透過最小跨小組互動進行細分。例如 `customer ID` 或 `resource ID`。

- 分割區索引鍵必須在所有請求中都可供使用，無論是直接或由其他參數確定性地推斷。

2. 持久性小組映射：上游服務應該僅在其資源的生命週期內與單個小組進行互動。
   - 依據工作負載而定，可能需要小組遷移策略，以便從其中一個小組將資料遷移到另一個小組。可能需要小組遷移的可能情境是，如果您的工作負載中特定使用者或資源變得太大並且要求它具備專有小組。
   - 小組不應該在小組之間共用狀態或元件。
   - 因此，應該避免跨小組互動或保持在最低程度，因為這些互動會建立小組之間的相依性，因而消滅故障隔離改善。
3. 路由器層：路由器層是儲存格之間的共用元件，因此無法遵循與儲存格相同的區隔策略。
   - 建議路由器層以有效率運算的方式使用分割區對應演算法將請求分發到個別小組，例如結合加密雜湊函數和模組化算術以將分割區索引鍵對應至小組。
   - 若要避免多小組影響，路由層必須保持簡單並且盡可能水平擴展，如此才能避免此層級內的複雜商業邏輯。這樣有增加的優點，隨時都容易了解其預期行為，以獲得徹底的可測試性。正如 Colm MacCárthaigh 在可靠性、持續工作以及一杯好咖啡中所述，簡單的設計和持續的工作模式會產生可靠的系統並降低抗脆弱性。
4. 儲存格大小：儲存格應具有最大的大小，且不允許超過它。
   - 最大大小應該藉由執行徹底測試來識別，直到觸及中斷點並且建立安全的操作邊距。如需如何實作測試實務的詳細資訊，請參閱 REL07-BP04 Load 測試您的工作負載
   - 整體工作負載應該透過新增額外小組來成長，讓工作負載隨著需求的增加而擴展。
5. 多可用區域或多區域策略：應利用多層彈性來防範不同的故障網域。
   - 對於彈性，您應該使用建置防禦層的方法。一層透過使用多個 AZ 建置高度可用的架構來防範更小、更常見的中斷。另一防禦層旨在防範發生罕見事件，例如廣泛的自然災害和區域級中斷。這第二層涉及架構您的應用程式以跨越多個 AWS 區域。針對您的工作負載實作多區域策略有助於其防範影響國家一大片地理區域的廣泛自然災害，或整個區域範圍的技術失敗。請注意，實作多區域架構可能相當複雜，並且通常對於大多數工作負載而言不是必要的。如需詳細資訊，請參閱REL10-BP01 將工作負載部署至多個位置。
6. 程式碼部署：交錯的程式碼部署策略應優先於同時將程式碼變更部署到所有單元格。
   - 這樣可協助將多個小組由於不良部署或人為錯誤的潛在失敗降至最低。如需詳細資訊，請參閱安全且無需人為干預的自動化部署。

資源

相關的最佳實務：

- REL07-BP04 Load 測試您的工作負載
- REL10-BP01 將工作負載部署至多個位置

相關文件：

- 可靠性、持續工作以及一杯好咖啡
- AWS 和分隔
- 使用隨機切換分區隔離工作負載
- 自動化安全、無人為介入的部署

相關影片：

- AWS re:Invent 2018：閉環與開放思維：如何取得大小型系統的控制權
- AWS re:Invent 2018：AWS 如何最大程度地減小故障的影響範圍 (ARC338)
- 隨機切換分區：AWS re:Invent 2019：Amazon 建置者資料中心簡介 (DOP328)
- AWS Summit ANZ 2021 - 故障總在情理之中、意料之外 (Everything fails, all the time)：專為彈性而設計

設計工作負載以承受元件失敗

須架構具高可用性和低平均復原時間 (MTTR) 需求的工作負載以實現彈性。

最佳實務

- REL11-BP01 監控工作負載的所有元件以偵測故障
- REL11-BP02 容錯移轉至運作良好的資源
- REL11-BP03 將所有分層的修復自動化
- REL11-BP04 在復原期間依賴資料平面而非控制平面
- REL11-BP05 使用靜態穩定性來防止雙模式行為
- REL11-BP06 當事件影響可用性時傳送通知
- REL11-BP07 建立您的產品架構以符合可用性目標和運行時間服務水準協議 (SLA)

REL11-BP01 監控工作負載的所有元件以偵測故障

持續監控工作負載的運作狀態，讓您和自動化系統在發生故障或效能降低時能夠察覺。根據商業價值監控關鍵績效指標 (KPI)。

所有復原和修復機制首先都必須能夠快速偵測問題。應該先偵測技術故障，以便解決問題。不過，可用性取決於工作負載提供商業價值的能力，因此測量此需求的關鍵績效指標 (KPI) 必須成為偵測和修復策略的一部分。

預期成果：工作負載的基本元件會單獨監控，以偵測故障發生的時機和位置並發出警示。

常見的反模式：

- 未設定任何警報，因此會在未發出通知的情況下發生中斷。
- 警示存在，但在此閾值下無法提供足夠的回應時間。
- 收集的指標經常不足以符合復原時間點目標 (RTO)。
- 只主動監控面對客戶的工作負載介面。
- 只收集技術指標，未收集業務功能指標。
- 無測量工作負載使用者體驗的指標。
- 建立了太多監控。

建立此最佳實務的優勢：在各層級內進行適當的監控，可讓您減少偵測時間，進而減少復原時間。

未建立此最佳實務時的曝險等級：高

實作指引

確定將要審核以進行監控的所有工作負載。確定需要監控的所有工作負載元件之後，您現在需要確定監控間隔。根據偵測故障所需的時間而定，監控間隔會直接影響復原的速度。平均偵測時間 (MTTD) 是指從發生故障到開始修復作業經過的時間。服務清單應盡可能廣泛且完整。

監控必須涵蓋應用程式堆疊的所有層級，包括應用程式、平台、基礎設施和網路。

您的監控策略應考慮微小故障的影響。如需微小故障的詳細資訊，請參閱《進階多可用區域彈性模式》白皮書中的 Gray failures。

實作步驟

- 您的監控間隔取決於復原必須多快完成。您的復原時間取決於所需的復原時間，因此您必須考量此時間和復原時間點目標 (RTO)，藉以決定收集頻率。

- 設定元件和受管服務的詳細監控。
  - 判斷 EC2 執行個體和 Auto Scaling 是否需要詳細監控。詳細監控提供 1 分鐘的間隔指標，預設監控則提供 5 分鐘的間隔指標。
  - 判斷 RDS 是否需要增強型監控。增強型監控使用 RDS 執行個體上的代理程式，以取得不同處理程序或執行緒的實用資訊。
  - 判斷 Lambda、API Gateway、Amazon EKS、Amazon ECS 和所有類型的負載平衡器的關鍵無伺服器元件的監控需求。
  - 確定 Amazon S3、Amazon FSx、Amazon EFS 和 Amazon EBS 的儲存元件的監控需求。
- 建立自訂指標以測量業務關鍵績效指標 (KPI)。工作負載會實作重要的業務功能，這些功能應做為 KPI，以利確定間接問題發生的時間。
- 以使用者 Canary 監控使用者的故障體驗。可執行和模擬客戶行為的綜合交易測試 (也稱為 Canary 測試，但請別與 Canary 部署混淆)，是最重要的測試程序之一。針對來自不同遠端位置的工作負載端點持續執行這些測試。
- 建立追蹤使用者體驗的自訂指標。如果您可以檢測客戶的體驗，則可以判斷消費者體驗何時變差。
- 設定警示以偵測工作負載的任何部分何時未正常運作，並指示何時自動擴展資源。警示會在儀表板上以視覺化方式顯示、透過 Amazon SNS 或電子郵件傳送提醒，以及搭配使用 Auto Scaling 來擴展或縮減工作負載資源。
- 建立儀表板以視覺化指標。儀表板可以讓您以視覺化方式查看趨勢、極端值和其他潛在問題的指標，或指出您可能想要調查的問題。
- 為您的服務建立分散式追蹤監控。透過分散式監控，您可以了解應用程式及其基礎服務的執行方式，以確定和疑難排解效能問題與錯誤的根本原因。
- 在單獨的區域和帳戶中建立監控系統 (使用 CloudWatch 或 X-Ray) 儀表板和資料收集。
- 透過 AWS Health 隨時掌握服務降級的相關資訊。透過 AWS 使用者通知 建立符合用途的 AWS Health 事件通知，以利用電子郵件和聊天管道傳送，並透過 Amazon EventBridge 以程式設計方式與您的監控和警示工具整合。

資源

相關的最佳實務：

- 可用性定義
- REL11-BP06 當事件影響可用性時傳送通知

相關文件：

- Amazon CloudWatch Synthetics 可讓您建立使用者 Canary
- 為執行個體啟用或停用詳細監控
- Enhanced Monitoring (增強型監控)
- 使用 Amazon CloudWatch 監控 Auto Scaling 群組和執行個體
- 發佈自訂指標
- 使用 Amazon CloudWatch 警示
- 使用 CloudWatch 儀表板
- 使用跨區域跨帳戶 CloudWatch 儀表板
- 使用跨區域跨帳戶 X-Ray 追蹤
- 了解可用性

相關影片：

- 減少微小故障

相關範例：

- 一個可觀測性研討會：探索 X-Ray

相關工具：

- CloudWatch
- CloudWatch X-Ray

REL11-BP02 容錯移轉至運作良好的資源

如果發生資源失敗，運作良好的資源應繼續處理請求。對於位置受損 (例如可用區域或 AWS 區域)，請確保您的系統已就位，可容錯移轉至未受影響位置中運作良好的資源。

設計服務時，請將負載分散到各個資源、可用區域或區域。因此，可以透過將流量轉移到剩餘運作狀態良好的資源來減輕個別資源故障或損害的影響。請考慮發生故障時，如何找到服務及其路由。

設計服務時，務必考慮故障復原。在 AWS，我們設計服務以盡可能減少從故障復原的時間並減輕對資料的影響。我們的服務主要使用的資料存放區，會在請求持久儲存於區域內的多個複本中之後，才確認

請求。經過建構後，它們會使用以儲存格為基礎的隔離，以及使用可用區域提供的故障隔離。我們在營運程序中廣泛使用自動化。我們還將取代-重啟功能最佳化，以期從中斷快速復原。

允許容錯移轉的模式和設計會隨著各 AWS 平台服務而有所不同。許多 AWS 原生受管服務本身就是多個可用區域 (例如 Lambda 或 API Gateway)。其他 AWS 服務 (例如 EC2 和 EKS) 需要特定的最佳實務設計，以支援在 AZ 的各資源或資料儲存容錯移轉。

監控應設定為確認容錯移轉資源是否正常運作、追蹤資源容錯移轉的進度，以及監控業務程序復原。

預期成果：系統能夠自動或手動使用新資源，以從降級恢復。

常見的反模式：

- 故障計畫不是規劃和設計階段的一部分。
- 未建立 RTO 和 RPO。
- 監控不足，無法偵測出失敗的資源。
- 正確隔離故障網域。
- 未考慮多區域容錯移轉。
- 決定進行容錯移轉時，失敗偵測太過敏感或積極。
- 未測試或驗證容錯移轉設計。
- 進行自動修復自動化，但未通知需要修復。
- 缺少緩衝期，以避免過早容錯恢復。

建立此最佳實務的優勢：您可以建置更具彈性的系統，在發生故障時透過適當降級並快速復原來維持可靠性。

未建立此最佳實務時的曝險等級：高

實作指引

諸如 Elastic Load Balancing 和 Amazon EC2 Auto Scaling 等 AWS 服務可協助跨資源和可用區域分配負載。因此，可以透過將流量轉移到剩餘運作狀態良好的資源來緩解個別資源 (例如 EC2 執行個體) 的失敗或可用區域的損害。

對於多區域工作負載，設計會更複雜。例如，跨區域僅供讀取複本可讓您將資料部署到多個 AWS 區域。不過仍需要容錯移轉，才能將僅供讀取複本提升為主要複本，然後將流量指向新端點。Amazon

Route 53、Amazon 應用程式復原控制器 (ARC)、Amazon CloudFront 和 AWS Global Accelerator 可協助在 AWS 區域 之間路由流量。

諸如 Amazon S3、Lambda、API Gateway、Amazon SQS、Amazon SNS、Amazon SES、Amazon Pinpoint、Amazon ECR、AWS Certificate Manager、EventBridge 或 Amazon DynamoDB 等 AWS 服務由 AWS 自動部署到多個可用區域。如果發生故障，這些 AWS 服務會自動將流量路由到運作良好的位置。資料以冗餘方式存放在多個可用區域中，並且仍然可用。

對於 Amazon RDS、Amazon Aurora、Amazon Redshift、Amazon EKS 或 Amazon ECS 而言，多可用區域是一種組態選項。如果啟動容錯移轉，則 AWS 可將流量導向運作良好的執行個體。此容錯移轉動作可由 AWS 執行，或依客戶要求執行

對於 Amazon EC2 執行個體、Amazon Redshift 或 Amazon ECS 任務或 Amazon EKS Pod，您可以選擇要部署到哪個可用區域。對於某些設計，Elastic Load Balancing 會提供解決方案，以偵測運作狀態不佳區域中的執行個體，並將流量路由至運作良好的區域。Elastic Load Balancing 也可將流量路由至內部部署資料中心內的元件。

對於多區域流量容錯移轉，重新路由可利用 Amazon Route 53、Amazon 應用程式復原控制器、AWS Global Accelerator、適用於 VPC 的 Route 53 私有 DNS 或 CloudFront 來提供定義網際網路網域和指派路由政策 (包括運作狀態檢查) 的方法，以便將流量路由到運作狀態良好的區域。AWS Global Accelerator 提供靜態 IP 位址，做為應用程式端點的固定進入點，然後使用 AWS 全球網路 (而不是網際網路) 路由至您所選 AWS 區域 中的端點，以獲得更好的效能和可靠性。

實作步驟

- 為所有適當的應用程式和服務建立容錯移轉設計。隔離每個架構元件，並為每個元件建立符合 RTO 和 RPO 的容錯移轉設計。
- 設定較低的環境 (例如開發或測試)，且其中所有服務都需要有容錯移轉計畫。使用基礎設施即程式碼 (IaC) 來部署解決方案，以確保可重複性。
- 設定復原站台 (例如第二個區域)，以實作和測試容錯移轉設計。如有必要，可以臨時設定測試的資源，以限制額外的成本。
- 判斷哪些容錯移轉計畫是由 AWS 自動執行、哪些可由 DevOps 程序自動執行，以及哪些可能要手動執行。記錄並測量每一項服務的 RTO 和 RPO。
- 建立容錯移轉程序手冊，並包括容錯移轉每個資源、應用程式和服務的所有步驟。
- 建立容錯恢復程序手冊，並包括容錯恢復 (含時程) 每個資源、應用程式和服務的所有步驟
- 制定計畫來啟動和演練程序手冊。使用模擬和混亂測試來測試程序手冊的步驟和自動化。
- 對於位置受損 (例如可用區域或 AWS 區域)，請確保您的系統已就位，可容錯移轉至未受影響位置中運作良好的資源。在容錯移轉測試之前，檢查配額、自動擴展層級和執行的資源。

資源

相關 Well-Architected 的最佳實務：

- REL13 - 災難復原 (DR) 計畫
- REL10 - 使用故障隔離來保護您的工作負載

相關文件：

- 設定 RTO 和 RPO 目標
- 使用 Route 53 加權路由進行容錯移轉
- 使用 Amazon 應用程式復原控制器進行災難復原
- 具有自動擴展的 EC2
- EC2 部署 - 多可用區域
- ECS 部署 - 多可用區域
- 使用 Amazon 應用程式復原控制器切換流量
- 具有 Application Load Balancer 和容錯移轉的 Lambda
- ACM 複寫和容錯移轉
- 參數存放區複寫和容錯移轉
- ECR 跨區域複寫和容錯移轉
- Secrets Manager 跨區域複寫組態
- 針對 EFS 和容錯移轉啟用跨區域複寫
- EFS 跨區域複寫和容錯移轉
- 聯網容錯移轉
- 使用 MRAP 的 S3 端點容錯移轉
- 為 S3 建立跨區域複寫
- AWS 上的跨區域容錯移轉和寬限容錯恢復指南
- 使用多區域 Global Accelerator 進行容錯移轉
- 透過 DRS 進行容錯移轉

相關範例：

- AWS 上的災難復原

- AWS 上的彈性災難復原

REL11-BP03 將所有分層的修復自動化

偵測到失敗時，使用自動化功能執行動作來進行修復。降級可能透過內部服務機制自動修復，或需要透過矯正動作重新啟動或移除資源。

對於自我管理的應用程式和跨區域修復，復原設計和自動修復程序可從現有最佳實務中提取。

重新啟動或移除資源是修復故障的重要工具。最佳實務是盡可能讓服務無狀態。這可防止資源重新啟動時遺失資料或可用性。在雲端，您可以 (且通常應該) 在重新啟動時取代整個資源 (例如，運算執行個體或無伺服器函數)。重新啟動本身是從故障中復原的一個簡單、可靠方法。工作負載中會發生許多不同類型的故障。硬體、軟體、通訊和營運可能會發生故障。

重新啟動或重試也適用於網路請求。對網路逾時和相依系統故障 (其中相依系統會返回錯誤) 套用相同的復原方法。這兩個事件對系統具有類似的影響，因此，不要嘗試讓任何一個事件成為特殊情況，而是藉由指數退避和抖動來採用類似的限制重試策略。重新啟動的能力是復原導向運算和高可用性叢集架構中的一種復原機制。

預期成果：執行自動化動作來矯正錯誤偵測。

常見的反模式：

- 佈建資源，但無自動擴展。
- 個別部署執行個體或容器中的應用程式。
- 部署不透過自動復原就無法部署到多個位置的應用程式。
- 手動復原自動擴展和自動復原無法修復的應用程式。
- 未自動化資料庫容錯移轉。
- 缺乏自動化方法可將流量重新路由至新端點。
- 沒有儲存複寫。

建立此最佳實務的優勢：自動修復可減少您的平均復原時間，並提高可用性。

未建立此最佳實務時的曝險等級：高

實作指引

Amazon EKS 或其他 Kubernetes 服務的設計應包括最小和最大複本或有狀態的集合，以及最小叢集和節點群組規模調整。這些機制提供了最少量的連續可用處理資源，同時會使用 Kubernetes 控制平面自動修復任何失敗。

透過使用運算叢集的負載平衡器存取的設計模式應利用 Auto Scaling 群組。Elastic Load Balancing (ELB) 會自動將傳入的應用程式流量分配到一或多個可用區域 (AZ) 中的多個目標和虛擬設備。

未使用負載平衡的叢集式運算設計，其大小設計應考量至少遺失一個節點。這可讓服務在復原新節點的同時，維持在可能減少的容量中自行執行。範例服務包括 Mongo、DynamoDB Accelerator、Amazon Redshift、Amazon EMR、Cassandra、Kafka、MSK-EC2、Couchbase、ELK 和 Amazon OpenSearch Service。其中許多服務都可以設計為納入額外的自動修復功能。某些叢集技術必須在節點遺失時產生警示，才能觸發自動或手動工作流程來重新建立新節點。此工作流程可以使用 AWS Systems Manager 自動化，以快速修復問題。

Amazon EventBridge 可用來監控及篩選事件，例如 CloudWatch 警示，或其他 AWS 服務的狀態變更。根據事件資訊，它接著可以調用 AWS Lambda、Systems Manager Automation 或其他目標，在您的工作負載上執行自訂修復邏輯。Amazon EC2 Auto Scaling 可設定為檢查 EC2 執行個體的運作狀態。如果執行個體處於執行中以外的任何狀態，或系統狀態為受損，Amazon EC2 Auto Scaling 會將執行個體視為運作狀態不佳，並啟動替代執行個體。對於大規模替換 (例如遺失整個可用區域)，靜態穩定性是高可用性的首選。

實作步驟

- 使用 Auto Scaling 群組在工作負載中部署分層。Auto Scaling 可以對無狀態應用程式進行自我修復，並新增或移除容量。
- 對於先前提及的運算執行個體，請使用負載平衡並選擇適當的負載平衡器類型。
- 考慮修復 Amazon RDS。對於待命執行個體，請設定待命執行個體的自動容錯移轉。對於 Amazon RDS 僅供讀取複本，須有自動化工作流程才能將僅供讀取複本設為主要。
- 對已部署無法在多個位置中部署之應用程式且可以容忍失敗後重新開機的 EC2 執行個體，實作自動復原。無法將應用程式部署到多個位置時，自動復原可以用來取代失敗的硬體並重新啟動執行個體。執行個體中繼資料和相關聯的 IP 位址，以及 EBS 磁碟區和 Amazon Elastic File System 或 File Systems for Lustre 以及 Windows 的掛載點皆會保留。使用 AWS OpsWorks，可在層級中設定 EC2 執行個體的自動修復功能。
- 當您無法使用自動擴展或自動復原，或自動復原失敗時，則使用 AWS Step Functions 和 AWS Lambda 實作自動復原。當您無法使用自動擴展，且無法使用自動復原或自動復原失敗時，則可以使用 AWS Step Functions 和 AWS Lambda 將修復作業自動化。

- Amazon EventBridge 可用來監控及篩選事件，例如 CloudWatch 警示或其他 AWS 服務的狀態變更。根據事件資訊，它接著可以調用 AWS Lambda (或其他目標)，在您的工作負載上執行自訂修復邏輯。

資源

相關的最佳實務：

- 可用性定義
- REL11-BP01 監控工作負載的所有元件以偵測故障

相關文件：

- AWS Auto Scaling 的運作方式
- Amazon EC2 自動復原
- Amazon Elastic Block Store (Amazon EBS)
- Amazon Elastic File System (Amazon EFS)
- 什麼是 Amazon FSx for Lustre？
- 什麼是 Amazon FSx for Windows File Server？
- AWS OpsWorks：使用自動修復來替換出現故障的執行個體
- 什麼是 AWS Step Functions？
- 什麼是 AWS Lambda？
- 什麼是 Amazon EventBridge？
- 使用 Amazon CloudWatch 警示
- Amazon RDS 容錯移轉
- SSM - Systems Manager Automation
- 彈性架構最佳實務

相關影片：

- 自動佈建及擴展 OpenSearch Service
- Amazon RDS 自動容錯移轉

相關範例：

- Amazon RDS 容錯移轉研討會

相關工具：

- CloudWatch
- CloudWatch X-Ray

REL11-BP04 在復原期間依賴資料平面而非控制平面

控制平面提供APIs用於建立、讀取和描述、更新、刪除和列出（CRUDL）資源的管理，而資料平面則處理 day-to-day服務流量。對可能影響彈性的事件實作復原或緩解回應時，請盡量使用最少數量的控制平面操作來復原、重新擴展、還原、修復或容錯移轉服務。資料平面動作應取代這些降級事件期間的任何活動。

例如，以下全都是控制平面動作：啟動新的運算執行個體、建立區塊儲存，以及說明佇列服務。啟動運算執行個體時，控制平面必須執行多項工作，例如尋找具有容量的實體主機、配置網路介面、準備本機區塊儲存磁碟區、產生憑證，以及新增安全規則。控制平面往往是複雜的協同運作。

預期成果：當資源進入受損狀態時，系統能夠將流量從受損資源轉移到健康狀況良好的資源，來自動或手動復原。

常見的反模式：

- 取決於變更DNS記錄以重新路由流量。
- 依賴控制平面擴展操作來取代因佈建資源不足而受損的元件。
- 依賴廣泛的多服務多API控制平面動作來修復任何類別的損害。

建立此最佳實務的優勢：提高自動化修復的成功率可減少平均復原時間，並改善工作負載的可用性。

未建立此最佳實務時的風險暴露等級：中。對於某些類型的服務降級，則會影響控制平面。廣泛使用控制平面進行修復的相依性可能會增加復原時間（RTO）和平均復原時間（MTTR）。

實作指引

若要限制資料平面動作，請評估每一項服務還原時所需的動作。

利用 Amazon Application Recovery Controller 來轉移DNS流量。這些功能會持續監控應用程式從故障中復原的能力，並可讓您控制跨多個 AWS 區域、可用區域和內部部署的應用程式復原。

Route 53 路由政策使用控制平面，因此不要依賴它進行復原。Route 53 資料平面會回答DNS查詢，並執行和評估運作狀態檢查。它們是全域分佈的，專為 100% 可用性服務層級協議 （ SLA ） 而設計。

您建立、更新和刪除 Route 53 資源的 Route 53 管理和APIs主控台會在控制平面上執行，這些平面旨在優先考慮管理 時所需的強大一致性和耐用性DNS。為了實現此目標，控制平面位於單一區域中：美國東部 (維吉尼亞北部)。雖然這兩個系統都建置得非常可靠，但 中不包含控制平面SLA。在極少數情況下，資料平面的彈性設計允許它保持可用性，而控制平面則不允許。對於災難復原和容錯移轉機制，使用資料平面功能提供可能最好的可靠性。

將運算基礎設施設計為靜態穩定，以避免在事件期間使用控制平面。例如，如果您使用 Amazon EC2 執行個體，請避免手動佈建新執行個體，或指示 Auto Scaling 群組新增執行個體作為回應。為獲得最高層級的彈性，請在用於容錯移轉的叢集中佈建足夠的容量。如果必須限制此容量閾值，請在整體 end-to-end系統上設定限流，以安全地限制到達有限資源集的總流量。

對於 Amazon DynamoDB 、Amazon API Gateway、負載平衡器和無 AWS Lambda 伺服器等服務，使用這些服務會利用資料平面。不過，建立新的函數、負載平衡器、API閘道或 DynamoDB 資料表是控制平面動作，應該在降級之前完成，作為事件的準備和容錯移轉動作的演練。對於 Amazon RDS，資料平面動作允許存取資料。

如需資料平面、控制平面以及如何 AWS 建置服務以滿足高可用性目標的詳細資訊，請參閱使用可用區域 的靜態穩定性。

了解哪些作業位於資料平面，哪些位於控制平面。

實作步驟

針對需要在降級事件之後還原的每個工作負載，評估容錯移轉執行手冊、高可用性設計、自動修復設計，或 HA 資源還原計畫。找出可能視為控制平面動作的每個動作。

考慮將控制動作變更為資料平面動作：

- Auto Scaling （ 控制平面 ） 到預先擴展的 Amazon EC2 資源 （ 資料平面 ）
- Amazon EC2執行個體擴展 （ 控制平面 ） 到 AWS Lambda 擴展 （ 資料平面 ）
- 使用 Kubernetes 評估任何設計，以及控制平面動作的性質。新增 Pod 是 Kubernetes 中的資料平面動作。動作應限於新增 Pod 而不是新增節點。使用過度佈建的節點是限制控制平面動作的慣用方法

請考慮可讓資料平面動作影響相同修復措施的替代方法。

- Route 53 記錄變更（控制平面）或 Amazon Application Recovery Controller（資料平面）
- Route 53 運作狀態檢查以進行更多自動化更新

如果服務具任務關鍵性，請考慮次要區域中的某些服務，以便在未受影響的區域中執行更多控制平面和資料平面動作。

- 主要區域中EKS的 Amazon EC2 Auto Scaling 或 Amazon，相較於次要區域中EKS的 Amazon EC2 Auto Scaling 或 Amazon，並將流量路由至次要區域（控制平面動作）
- 將僅供讀取複本設為主要，或在主要區域中嘗試相同的動作 (控制平面動作)

資源

相關的最佳實務：

- 可用性定義
- REL11-BP01 監控工作負載的所有元件以偵測故障

相關文件：

- APN 合作夥伴：可協助自動化容錯能力的合作夥伴
- AWS Marketplace：可用於容錯的產品
- Amazon 建置者資料中心：控管較小服務，避免分散式系統過載
- Amazon DynamoDB API（控制平面和資料平面）
- AWS Lambda 執行（分割至控制平面和資料平面）
- AWS Elemental MediaStore 資料平面
- 使用 Amazon Application Recovery Controller 建置高彈性的應用程式，第 1 部分：單一區域堆疊
- 使用 Amazon Application Recovery Controller 建置高彈性的應用程式，第 2 部分：多區域堆疊
- 使用 Amazon Route 53 建立災難復原機制
- 什麼是 Amazon Application Recovery Controller
- Kubernetes 控制平面和資料平面

相關影片：

- 回歸基礎 - 使用靜態穩定性

- 使用 AWS 全球服務建置彈性的多站台工作負載

相關範例：

- Amazon Application Recovery Controller 簡介
- Amazon 建置者資料中心：控管較小服務，避免分散式系統過載
- 使用 Amazon Application Recovery Controller 建置高彈性的應用程式，第 1 部分：單一區域堆疊
- 使用 Amazon Application Recovery Controller 建置高彈性的應用程式，第 2 部分：多區域堆疊
- 使用可用區域實現靜態穩定性

相關工具：

- Amazon CloudWatch
- AWS X-Ray

REL11-BP05 使用靜態穩定性來防止雙模式行為

工作負載應該是靜態穩定的，且只在單一正常模式下運作。雙模態行為是指工作負載在正常和故障模式下呈現不同行為的情況。

例如，您可能在不同的可用區域中啟動新的執行個體，嘗試回復可用區域故障。這可能會導致在故障模式期間產生雙模態回應。您應改為建置靜態穩定且僅以一種模式操作的工作負載。在此範例中，這些執行個體應該在發生故障之前已佈建在第二個可用區域。此靜態穩定設計可以確保工作負載僅在單一模式下運作。

預期成果：工作負載不會在正常和故障模式出現雙模態行為。

常見的反模式：

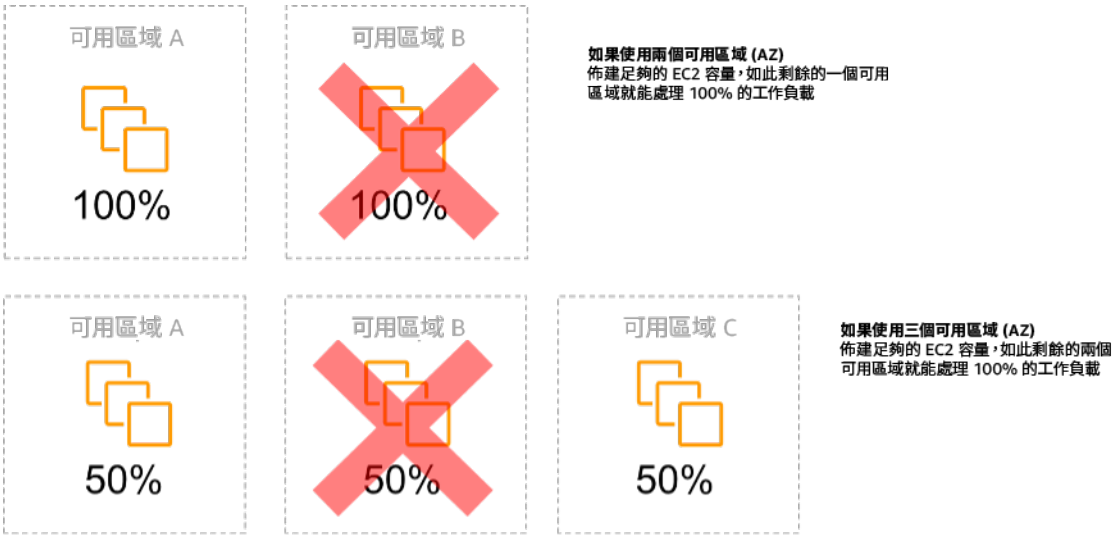
- 假設無論故障範圍，一律可以佈建資源。
- 嘗試在故障期間動態取得資源。
- 在發生故障之前，請勿在多個區域佈建適度的資源。
- 僅考慮運算資源的靜態穩定設計。

建立此最佳實務的優勢：使用靜態穩定設計執行的工作負載，能夠在正常和故障事件發生時產生可預測的結果。

未建立此最佳實務時的曝險等級：中

實作指引

雙模態行為是指您的工作負載在正常和故障模式下展現出不同的行為，例如，當可用區域故障時，仰賴啟動新的執行個體。雙模式行為的範例是當穩定的 Amazon EC2設計在每個可用區域中佈建足夠的執行個體，以便在移除一個 AZ 時處理工作負載負載。Elastic Load Balancing 或 Amazon Route 53 運作狀態會進行檢查，將負載從受損的執行個體中移出。流量轉移後，請使用 AWS Auto Scaling 以非同步方式取代失敗區域中的執行個體，並在運作狀態良好的區域中啟動執行個體。運算部署的靜態穩定性（例如EC2執行個體或容器）會產生最高的可靠性。



跨可用區域的EC2執行個體靜態穩定性

這必須在所有彈性情況下，與此模型的成本以及維護工作負載的商業價值互相衡量。佈建較少運算容量並在故障時啟動新執行個體的成本較低，但是對於大規模故障 (例如可用區域損壞)，這種方法的效率較低，因為它同時仰賴作業平面，以及未受影響區域中的足夠資源。

您的解決方案也應該權衡可靠性與工作負載的成本需求。靜態穩定性架構適用於各種架構，包括跨可用區域分佈的運算執行個體、資料庫僅供讀取複本設計、Kubernetes（AmazonEKS）叢集設計和多區域容錯移轉架構。

若在每個區域使用更多資源，也可以實施更靜態的穩定設計。透過新增更多區域，您可以降低靜態穩定性所需的額外運算量。

雙模態行為範例之一是網路逾時，網路逾時可能導致系統嘗試重新整理整個系統的組態狀態。這樣一來，即會給另一個元件新增意外負載，且可能導致其發生故障，從而引發其他意外後果。這種負面意見回饋迴圈會影響工作負載的可用性。反之，您可以建置靜態穩定且僅以一種模式操作的系統。靜態穩定

的設計是執行持續工作，並始終以固定的規律重新整理組態狀態。呼叫失敗時，工作負載會使用先前的快取數值，並啟動警示。

另一個雙模態行為範例是允許用戶端在發生失敗時繞過您的工作負載快取。這看起來可能是滿足用戶端需求的解決方案，但會大幅變更工作負載的需求，且可能導致故障。

評估關鍵工作負載，決定哪些工作負載需要此類彈性設計。針對關鍵工作負載，必須檢視每個應用程式元件。需要靜態穩定性評估的服務類型範例如下：

- 運算：Amazon EC2、 EKS-EC2、 ECS-EC2、 EMR-EC2
- 資料庫：Amazon Redshift、Amazon RDS、Amazon Aurora
- 儲存體：Amazon S3 （單區域）、Amazon EFS（安裝）、Amazon FSx（安裝）
- 負載平衡器：在某些設計下

實作步驟

- 建置靜態穩定且僅以一種模式操作的系統。在此情況下，請在每個可用區域佈建足夠的執行個體，以處理移除一個可用區域時的工作負載容量。許多服務皆可用於路由到運作狀態良好的資源，例如：
  - 跨區域DNS路由
  - MRAP Amazon S3 MultiRegion Routing
  - AWS Global Accelerator
  - Amazon Application Recovery Controller
- 設定資料庫讀取複本以考慮單一主要執行個體或讀取複本的遺失情況。若僅供讀取複本為流量提供服務，則每個可用區域中的數量應等同於區域故障時的整體需求。
- 在 Amazon S3 儲存中設定重要資料，以便可用區域故障時，能針對所儲存的資料保持靜態穩定。如果使用 Amazon S3 One Zone-IA 儲存類別，則不應將其視為靜態穩定，因為該區域的遺失會最小化此儲存資料的存取權。
- Load balancers 有時會設定錯誤，或本來就設定為供特定可用區域使用。在這種情況下，靜態穩定的設計可能是在更複雜的設計AZs中將工作負載分散到多個。出於安全性、延遲或成本考量，可以使用原始設計來減少區域間流量。

資源

相關 Well-Architected 的最佳實務：

- 可用性定義

- REL11-BP01 監控工作負載的所有元件以偵測故障
- REL11-BP04 在復原期間依賴資料平面而非控制平面

相關文件：

- 在災難復原計畫中盡可能減少相依關係
- Amazon 建置者資料中心：使用可用區域實現靜態穩定性
- 故障隔離界限
- 使用可用區域實現靜態穩定性
- 多區域 RDS
- 在災難復原計畫中盡可能減少相依關係
- 跨區域DNS路由
- MRAP Amazon S3 MultiRegion Routing
- AWS Global Accelerator
- Amazon Application Recovery Controller
- 單區域 Amazon S3
- 跨區域負載平衡

相關影片：

- AWS： AWS re：Invent 2019 中的靜態穩定性：介紹 Amazon Builders' Library （DOP328）

REL11-BP06 當事件影響可用性時傳送通知

當偵測到閥值超標時傳送通知，即使問題造成的事件已自動解決。

自動修復功能可讓您的工作負載變得可靠。不過，也可能會遮蔽需要解決的潛在問題。實作適當的監控和事件，讓您能夠偵測到問題模式 (包括自動修復功能處理的問題模式)，以解決根本原因問題。

具有韌性的系統可將降級事件立即傳達給權責團隊。這些通知應該透過一個或多個通訊管道傳送。

預期成果：當超過閾值 (例如錯誤率、延遲或其他關鍵績效指標 (KPI)) 時，營運團隊會立即收到警示，以盡快解決問題，避免或將使用者負面影響降至最低。

常見的反模式：

- 傳送太多警示。
- 傳送不可採取行動的警示。
- 警示閾值設置太高 (太敏感) 或太低 (太遲鈍)。
- 不傳送外部相依性的警示。
- 在設計監控和警示時，不考慮微小故障。
- 進行修復自動化，但不通知權責團隊需要修復。

建立此最佳實務的優勢：回復通知可讓營運和業務團隊注意到服務降級，讓他們可以立即反應，將平均偵測時間 (MTTD) 和平均復原時間 (MTTR) 降至最低。回復事件的通知也會確認您不會忽略不常發生的問題。

未建立此最佳實務時的風險暴露等級：中。若無法實作適當的監控和事件通知機制，您可能就無法偵測到問題模式 (包括自動修復功能處理的問題模式)。只有當使用者聯絡客服或偶然情況下，團隊才會注意到系統降級。

實作指引

定義監控策略時，觸發警示是常見的事件。此事件可能包含警示的識別碼、警示狀態 (例如 `IN ALARM` 或 `OK`) 以及觸發原因詳情。在許多情況下，系統應檢測到警示事件並傳送電子郵件通知。這是警示動作範例。警示通知對於可觀測性至關重要，因為它會通知權責人員有問題發生。然而，當可觀測性解決方案對事件的回應措施夠熟練後，便可以自動修復問題，無須人為介入。

建立 KPI 監控警示後，閾值超過時就應會向權責團隊傳送警示。這些警示也可用於觸發嘗試修復降級的自動化程序。

針對更複雜的閾值監控，則應考慮使用複合警示。複合警示會使用數個 KPI 監控警示，根據作業商務邏輯建立警示。CloudWatch 警示可設定為傳送電子郵件，或使用 Amazon SNS 整合或 Amazon EventBridge 在第三方事件追蹤系統中記錄事件。

實作步驟

根據監控工作負載的方式建立各種警示類型，例如：

- 應用程式警示可用來偵測工作負載任何無法正常運作的部分。
- 基礎設施警示會指出何時擴展資源。警示會在儀表板上以視覺化方式顯示、透過 Amazon SNS 或電子郵件傳送提醒，以及搭配使用 Auto Scaling 來擴展或縮減工作負載資源。
- 可建立簡單的靜態指示，以監控指標在指定評估期間內超過靜態閾值的時間。
- 複合警示可以涵蓋來自多個來源的複雜警示。

- 建立警示後，請建立適當的通知事件。可以直接調用 Amazon SNS API 來傳送通知，並連結任何自動化以進行修復或通訊。
- 透過 AWS Health 隨時掌握服務降級的相關資訊。透過 AWS 使用者通知 建立符合用途的 AWS Health 事件通知，以利用電子郵件和聊天管道傳送，並透過 Amazon EventBridge 以程式設計方式與您的監控和警示工具整合。

資源

相關 Well-Architected 的最佳實務：

- 可用性定義

相關文件：

- 根據靜態閾值建立 CloudWatch 警示
- 什麼是 Amazon EventBridge？
- 什麼是 Amazon Simple Notification Service？
- 發佈自訂指標
- 使用 Amazon CloudWatch 警示
- 設定 CloudWatch 複合警示
- re:Invent 2022 中 AWS Observability 的最新消息

相關工具：

- CloudWatch
- CloudWatch X-Ray

REL11-BP07 建立您的產品架構以符合可用性目標和運行時間服務水準協議 (SLA)

建立您的產品架構以符合可用性目標和運行時間服務水準協議 (SLA)。如果您發佈或私下同意可用性目標或運行時間 SLA，請確認您的架構和操作程序的設計可以支援。

預期成果：每個應用程式都有一個已定義的可用性目標和效能指標的 SLA，可以進行監控和維護，以達到業務成果。

常見的反模式：

- 設計和部署工作負載，而未設定任何 SLA。
- SLA 指標設定為高，而沒有合理或業務要求。
- 設定 SLA 但未考慮相依性及其基礎 SLA。
- 建立應用程式設計而未考慮彈性的共同責任模型。

建立此最佳實務的優勢：根據關鍵彈性目標設計應用程式，可協助您達成業務目標和客戶期望。這些目標可協助推動應用程式設計程序，評估不同的技術和考慮各種權衡。

未建立此最佳實務時的曝險等級：中

實作指引

應用程式設計必須將多元的要求納入考慮，這些要求是從業務、營運和財務目標衍生而來。在營運要求內，工作負載必須有特定彈性指標目標，才能適當地監控和支援。彈性指標不應該在部署工作負載之後設定或衍生。它們應該在設計階段期間定義，協助引導各種決策和權衡。

- 每個工作負載都應該有自己的一組彈性指標。這些指標可能與其他業務應用程式不同。
- 降低相依性對可用性有正面影響。每個工作負載都應該考慮其相依性及其 SLA。一般而言，選取可用性目標等於或大於工作負載目標的相依性。
- 請考慮鬆散耦合設計，讓您的工作負載在可行時不論是否有相依性受損，都可以正確操作。
- 減少控制平面相依性，特別是復原或降級期間。評估針對任務關鍵性工作負載靜態穩定的設計。使用資源節省來增加工作負載中這些相依性的可用性。
- 可觀測性和檢測對於透過降低平均偵測時間 (MTTD) 和平均修復時間 (MTTR) 來達成 SLA 相當關鍵。
- 低頻率失敗 (MTBF 較長)、較短的失敗偵測時間 (較短 MTTD) 和較短的修復時間 (較短 MTTR)，是用來在分散式系統中改善可用性的三個因素。
- 建立和符合工作負載的彈性指標，是任何有效設計的基礎。這些設計必須考慮到設計複雜性、服務相依性、效能、擴展和成本的權衡。

實作步驟

- 請考慮下列問題，審核和記載工作負載設計：
  - 控制平面用於工作負載的哪個地方？
  - 工作負載如何實作容錯能力？

  - 擴展、自動擴展、備援和高可用性元件的設計模式是什麼？
  - 資料一致性和可用性的要求是什麼？
  - 資源節省或資源靜態穩定性是否有任何考慮？
  - 服務相依性是什麼？
- 與利益相關者合作時根據工作負載架構定義 SLA 指標。請考慮工作負載所使用所有相依性的 SLA。
- 一旦設定 SLA 目標，最佳化架構以符合 SLA。
- 一旦設定可符合 SLA 的設計，實作營運變更、處理自動化以及也會著重在降低 MTTD 和 MTTR 的執行手冊。
- 部署之後，監控和報告 SLA。

資源

相關的最佳實務：

- REL03-BP01 選擇如何分割工作負載
- REL10-BP01 將工作負載部署至多個位置
- REL11-BP01 監控工作負載的所有元件以偵測故障
- REL11-BP03 將所有分層的修復自動化
- REL12-BP04 使用混沌工程測試彈性
- REL13-BP01 定義停機時間和資料遺失的復原目標
- 了解工作負載運作狀態

相關文件：

- 備援的可用性
- 可靠性支柱 - 可用性
- 測量可用性
- AWS 故障隔離界限
- 彈性的共同責任模型
- 使用可用區域實現靜態穩定性
- AWS 服務水準協議 (SLA)
- AWS 上小組型架構指南

- AWS 基礎設施
- 《進階多可用區域彈性模式》白皮書

相關服務：

- Amazon CloudWatch
- AWS Config
- AWS Trusted Advisor

測試可靠性

在將工作負載設計為可彈性應對生產壓力之後，測試是確保其依設計運作並交付您預期之彈性的唯一方法。

測試以驗證您的工作負載是否滿足功能與非功能性要求，因為錯誤或效能瓶頸可能會影響工作負載的可靠性。測試工作負載彈性以協助您找出僅在生產中出現的延遲錯誤。請定期進行這些測試。

最佳實務

- REL12-BP01 使用程序手冊調查失敗
- REL12-BP02 執行事件後分析
- REL12-BP03 測試可擴展性和效能需求
- REL12-BP04 使用混沌工程測試彈性
- REL12-BP05 定期進行演練日

REL12-BP01 使用程序手冊調查失敗

藉由在程序手冊中記錄調查程序，對無法充分理解的失敗情境進行快速一致的回應。程序手冊是為識別造成失敗情境的因素所執行的預先定義步驟。在識別或呈報問題之前，任何程序步驟的結果都會用來決定要採取的後續步驟。

程序手冊是您必須進行的主動規劃，以便能夠有效地採取回應動作。在生產環境中遇到程序手冊未涵蓋的故障情境時，請先解決問題 (解決燃眉之急)。然後返回並查看您為解決問題所採取的步驟，並使用這些步驟在程序手冊中新增新的項目。

請注意，程序手冊用於回應特定事件，而執行手冊則用於實現特定成果。執行手冊通常用於例行活動，而程序手冊則用於回應非例行事件。

常見的反模式：

- 在不知道診斷問題或回應事件的程序之情況下，規劃部署工作負載。
- 調查事件時，未規劃即決定要向哪些系統收集日誌和指標。
- 指標和事件的保留時間過短，無法用以擷取資料。

建立此最佳實務的優勢：擷取程序手冊可確保流程得到一致遵循。有系統地編纂您的程序手冊可限制手動活動引入錯誤。程序手冊自動化可免除團隊成員介入的需要，或在介入開始時提供其他資訊，從而縮短事件回應時間。

未建立此最佳實務時的曝險等級：高

實作指引

- 使用程序手冊識別出問題。程序手冊是調查問題的書面程序。透過在程序手冊中記錄程序，對失敗情境做出一致且迅速的回應。程序手冊包含的資訊和指南必須能夠讓技能嫻熟的人員得以收集適用資訊、識別潛在的失敗來源、隔離故障，以及判斷成因 (執行事件後分析)。
  - 將程序手冊實做為程式碼。透過撰寫程序手冊指令碼，以程式碼形式執行操作，確保一致性並限制和減少手動程序引起的錯誤。程序手冊可由多個指令碼組成，這些指令碼代表識別成因時可能需要的不同步驟。執行手冊活動可以做為程序手冊活動的一部分被調用或執行，或者可能提示執行程序手冊，以回應已識別的事件。
    - 透過 AWS Systems Manager 自動化您的操作程序手冊
    - AWS Systems Manager 執行命令
    - AWS Systems Manager Automation
    - 什麼是 AWS Lambda？
    - 什麼是 Amazon EventBridge？
    - 使用 Amazon CloudWatch 警示

資源

相關文件：

- AWS Systems Manager Automation
- AWS Systems Manager 執行命令
- 透過 AWS Systems Manager 自動化您的操作程序手冊

- 使用 Amazon CloudWatch 警示
- 使用 Canary (Amazon CloudWatch Synthetics)
- 什麼是 Amazon EventBridge？
- 什麼是 AWS Lambda？

相關範例：

- 使用程序手冊和執行手冊將操作自動化

REL12-BP02 執行事件後分析

審查影響客戶的事件，並識別成因和預防性行動項目。使用此資訊來開發緩解措施，以限制或防止事件再次發生。制定可快速有效回應的程序。適當地傳達成因和為目標受眾量身打造的糾正措施。建立一種可以根據需要將這些原因傳達給其他人的方法。

評估現有測試找不到問題的原因。如果測試尚未存在，請為此案例新增測試。

預期成果：您的團隊擁有一致且商定的方法來處理事件後分析。一種機制是錯誤糾正 (COE) 流程。COE 程序可幫助您的團隊識別、了解和解決事件的根本原因，同時還能建置機制和防護機制，以限制相同事件再次發生的可能性。

常見的反模式：

- 尋找成因，但未繼續深入尋找其他潛在問題和減輕方法。
- 僅確定人為錯誤原因，不未嘗試可防止人為錯誤發生的任何培訓或或自動化。
- 專注於追究責任，而不是了解根本原因，造成恐懼文化並阻礙開放的溝通
- 無法分享見解，只讓一小群人知道事件分析調查結果，讓其他人無法從學到的教訓中受益
- 沒有機制可擷取機構知識而失去寶貴的見解，因為組織不會以更新過的最佳實務形式保存所學到的教訓，並導致重複發生相同或類似根本原因的事件

建立此最佳實務的優勢：進行事件後分析並分享結果，以讓其他實作了相同成因的工作負載減輕風險，並讓工作負載能夠在事件發生前實作減輕措施或自動復原。

未建立此最佳實務時的曝險等級：高

實作指引

良好的事件後分析提供了機會，為系統中其他地方使用的架構模式問題提出通用解決方案。

COE 程序的基石是記錄和解決問題。建議您定義標準化方式來記錄關鍵的根本原因，並確保加以檢視和解決。為事件後分析程序指派明確的擁有權。指定負責監督事件調查和後續跟進的團隊或個人。

鼓勵專注於學習和改進的文化，而不是追究責任的文化。強調目標是預防未來的事件，而不是懲罰個人。

開發用於進行事件後分析的明確定義程序。這些程序應概述要採取的步驟、要收集的資訊，以及要在分析期間解決的關鍵問題。徹底調查事件，跳脫出直接原因以找出根本原因和成因。使用諸如五個為什麼等技巧深入研究潛在問題。

維護事件分析所學教訓的儲存庫。此機構知識可以做為未來事件和預防工作的參考。分享事件後分析的調查結果和見解，並考慮舉行公開邀請的事件後檢討會議，以討論學到的教訓。

實作步驟

- 在進行事件後分析時，請確保事件後分析不會讓相關人員受到責備。這可讓事件中的相關人員平心靜氣看待建議的糾正措施，並促進誠實地自我評估與跨團隊合作。
- 定義標準化方式來記錄重要問題。這類文件的範例結構如下：
  - 發生了什麼？
  - 對客戶和您的業務有什麼影響？
  - 根本原因是什麼？
  - 您擁有什麼可以提供支援的資料？
    - 例如，指標和圖表
  - 對關鍵支柱的影響有哪些 (尤其是安全性)？
    - 建立工作負載的架構時，您可依照業務環境，在各支柱之間作出權衡。這些業務決策可以讓您了解工程設計的優先順序。您可以選擇在開發環境中以可靠性做為代價最佳化成本，或者針對關鍵任務解決方案，以較高成本達到可靠性的最佳化。安全始終是首要工作，因為您必須保護客戶。
  - 您獲得了什麼教訓？
  - 您正在採取什麼糾正措施？
    - 動作項目
    - 相關項目
- 建立用於進行事件後分析的明確定義標準作業程序。
- 設定標準化的事件報告程序。全面記錄所有事件，包括初始事件報告、日誌、通訊，以及事件期間採取的行動。
- 請記住，發生事件時不見得會有中斷情形。事件也可能是幾乎錯過的情況，或是系統雖以意想不到的方式執行，卻仍可履行其業務功能。

- 請根據意見回饋和學到的教訓，持續改善事件後分析程序。
- 擷取知識管理系統中的關鍵調查結果，並考慮任何應新增至開發人員指南或部署前檢查清單的模式。

資源

相關文件：

- 為什麼您應該制定錯誤糾正 (COE)

相關影片：

- Amazon 成功失敗的方法
- AWS re:Invent 2021 - Amazon 建置者資料中心：Amazon 的卓越營運

REL12-BP03 測試可擴展性和效能需求

使用諸如負載測試等技術來驗證工作負載符合擴展和效能需求。

在雲端，您可以隨需為工作負載建立正式作業規模的測試環境。您不必依賴縮減規模的測試環境，這可能會導致正式作業行為的預測不準確，而是可以使用雲端來佈建測試環境，以確切反映您預期的正式作業環境。此環境可協助您以更準確模擬您的應用程式面臨的真實狀況來進行測試。

除了效能測試之外，務必確認您的基本資源、擴展設定、服務配額和彈性設計能夠在負載下如預期運作。這種整體方法可確認您的應用程式即使在最嚴苛的條件下，仍能可靠地擴展和執行。

預期成果：您的工作負載即使在承受尖峰負載的情況下，仍維持其預期行為。您主動解決可能隨應用程式發展和演進而出現的任何效能相關問題。

常見的反模式：

- 您使用的測試環境與正式作業環境相似程度不高。
- 您將負載測試視為單獨的一次性活動，而不是部署持續整合 (CI) 管道整體的一部分。
- 您未定義明確且可衡量的效能需求，例如回應時間、輸送量和可擴展性目標。
- 您在負載情況不切實際或不足的情況下執行測試，而且未能測試尖峰負載、突發性峰值和持續高負載。
- 您未透過超過預期的負載限制來對工作負載進行壓力測試。

- 您使用不適當或不適合的負載測試和效能分析工具。
- 您缺乏全方位的監控和警示系統可用來追蹤效能指標並偵測異常。

建立此最佳實務的優勢：

- 負載測試可協助您在付諸正式作業之前，找出系統中潛在的效能瓶頸。當您模擬正式作業層級流量和工作負載時，就可找出系統處理負載時可能遇到困難層面，例如，回應時間緩慢、資源限制或系統故障。
- 當您在各種負載條件下測試系統時，您可以更了解支援工作負載所需的資源需求。此資訊可協助您做出有關資源分配的明智決策，並防止資源過度佈建或佈建不足。
- 若要找出潛在的故障點，您可以觀察工作負載在高負載條件下的運作情形。此資訊可協助您透過適時實作容錯機制、容錯移轉策略和備援措施，來改善工作負載的可靠性和彈性。
- 您可及早找出和解決效能問題，這有助於避免系統中斷、回應時間緩慢和使用者不滿意所帶來代價高昂的後果。
- 測試期間收集的詳細效能資料和分析資訊，可協助您解決正式作業時可能出現的效能相關問題。這樣就能加快回應和解決事件，進而降低對使用者和組織營運的影響。
- 在某些產業中，主動效能測試可協助讓工作負載符合合規標準，進而降低遭受懲罰或產生法律問題的風險。

未建立此最佳實務時的曝險等級：高

實作指引

第一步是定義全面的測試策略，涵蓋擴展和效能需求的所有層面。首先，請根據您的業務需求，例如輸送量、延遲長條圖和錯誤率，清楚定義工作負載的服務層級目標 (SLO)。接著設計一套測試，可模擬從平均用量到突發尖峰和持續尖峰負載的各種負載案例，並確認工作負載的行為符合您的 SLO。這些測試應該自動化，並整合到您的持續整合和部署管道中，以及早發現開發過程中的效能迴歸。

為了有效測試擴展和效能，請投資正確的工具和基礎設施。這包括可產生實際使用者流量的負載測試工具、識別瓶頸的效能分析工具，以及追蹤關鍵指標的監控解決方案。重要的是，您應確認您的測試環境在基礎設施和環境條件方面盡量與正式作業環境相符，以使測試結果盡可能準確。為了更容易可靠地複寫和調整類似正式作業的設定，請使用基礎設施即程式碼和容器型應用程式。

擴展和效能測試是持續進行的程序，而不是一次性的活動。實作全面監控和警示，以追蹤應用程式正式作業時的效能，並使用這些資料持續改進您的測試策略和最佳化工作。定期分析效能資料，以找出新興問題、測試新的擴展策略並實作最佳化，以提高應用程式的效率和可靠性。只要您採用迭代方法並持續

從正式作業資料中學習，就可以確認您的應用程式能夠適應不斷變化的使用者需求，並隨著時間維持彈性和最佳效能。

實作步驟

1. 確立明確且可衡量的效能需求，例如回應時間、輸送量和可擴展性目標。這些需求應以工作負載的使用模式、使用者期望和業務需求為基礎。
2. 選取並設定負載測試工具，以準確地模擬正式作業環境中的負載模式和使用者行為。
3. 設定盡量與正式作業環境相符的測試環境，包括基礎設施和環境條件，以提高測試結果的準確性。
4. 建立測試套件，涵蓋從平均用量模式到尖峰負載、快速尖峰和持續高負載等各種案例。將這些測試整合到您的持續整合和部署管道中，以及早發現開發過程中的效能迴歸。
5. 執行負載測試以模擬實際使用者流量，並了解應用程式在不同負載條件下的行為。若要對您的應用程式進行壓力測試，請超過預期的負載並觀察其行為，例如回應時間變慢、資源耗盡或系統故障，這樣做有助於識別應用程式的中斷點並納入擴展策略。透過逐步增加負載來評估工作負載的可擴展性，並衡量效能影響以了解擴展限制並規劃未來容量需求。
6. 實作全面的監控和警示，以追蹤效能指標、偵測異常情況，並在超出閾值時初始化擴展動作或通知。
7. 持續監控和分析效能資料，以識別需要改進的層面。對您的測試策略和最佳化工作採取迭代方法。

資源

相關的最佳實務：

- REL01-BP04 監控和管理配額
- REL06-BP01 監控工作負載的所有元件 (產生)
- REL06-BP03 傳送通知 (即時處理和警示)

相關文件：

- 負載測試應用程式
- AWS 上的分散式負載測試
- 應用程式效能監控
- Amazon EC2 測試政策

相關範例：

- AWS 上的分散式負載測試 (GitHub)

相關工具：

- Amazon CodeGuru Profiler
- Amazon CloudWatch RUM
- Apache JMeter
- K6
- Vegeta
- Hey
- ab
- wrk
- AWS 上的分散式負載測試

REL12-BP04 使用混沌工程測試彈性

定期在位於或盡可能鄰近生產環境的環境中執行混沌試驗，以了解您的系統因應不良狀況的能力。

預期成果：

除了以彈性測試驗證您的工作負載在某事件期間的已知預期行為以外，還可以藉由在故障注入試驗中套用混沌工程或注入非預期的負載，來定期驗證工作負載的彈性。結合混沌工程與彈性測試，您將可確信工作負載在經歷元件失敗後仍可存留，並且可在 (幾乎) 不受影響的情況下從非預期的中斷復原。

常見的反模式：

- 針對彈性進行設計，但未確認工作負載在錯誤發生時的整體運作情形。
- 未曾在真實的情況和預期的負載下試驗。
- 未將試驗視為程式碼或透過開發週期加以維護。
- 未在 CI/CD 管道中與部署以外執行混沌試驗。
- 在決定要以哪些錯誤進行試驗時，未使用過去的事故後分析。

建立此最佳實務的優勢：注入錯誤以驗證工作負載的彈性，可讓您確信在發生真正的錯誤時，彈性設計的復原程序將可發揮作用。

未建立此最佳實務時的曝險等級：中

實作指引

混沌工程可讓您的團隊有能力以受控的方式，持續在服務供應商、基礎架構、工作負載和元件層級注入真實的中斷 (模擬)，且對客戶 (幾乎) 不會造成影響。它可讓您的團隊從錯誤中學習，並且觀察、測量及改善工作負載的彈性，以及驗證在事件發生時會引發提醒，且團隊會收到通知。

持續執行時，混沌工程可能會凸顯您工作負載中的缺陷，且若未解決，可能會對可用性與操作產生負面影響。

> **Note**
>
> 混沌工程是在系統中進行試驗的專業領域，旨在建立對系統承受生產環境中紊亂情況的能力的信心。– 混沌工程的原則

如果系統能夠承受這些中斷，則應將混沌試驗視為自動化迴歸測試來維護。此時，您應在系統開發生命週期 (SDLC) 和 CI/CD 管道中執行混沌試驗。

若要確定您的工作負載可以承受元件失敗，請在試驗中注入真實事件。例如，進行遺失 Amazon EC2 執行個體或容錯移轉主要 Amazon RDS 資料庫執行個體的試驗，並驗證您的工作負載未受影響 (或僅受到些微影響)。使用元件錯誤的組合，模擬可用區域的中斷可能導致的事件。

對於應用程式層級的錯誤 (例如當機)，您可以從記憶體和 CPU 用盡之類的壓力源開始著手。

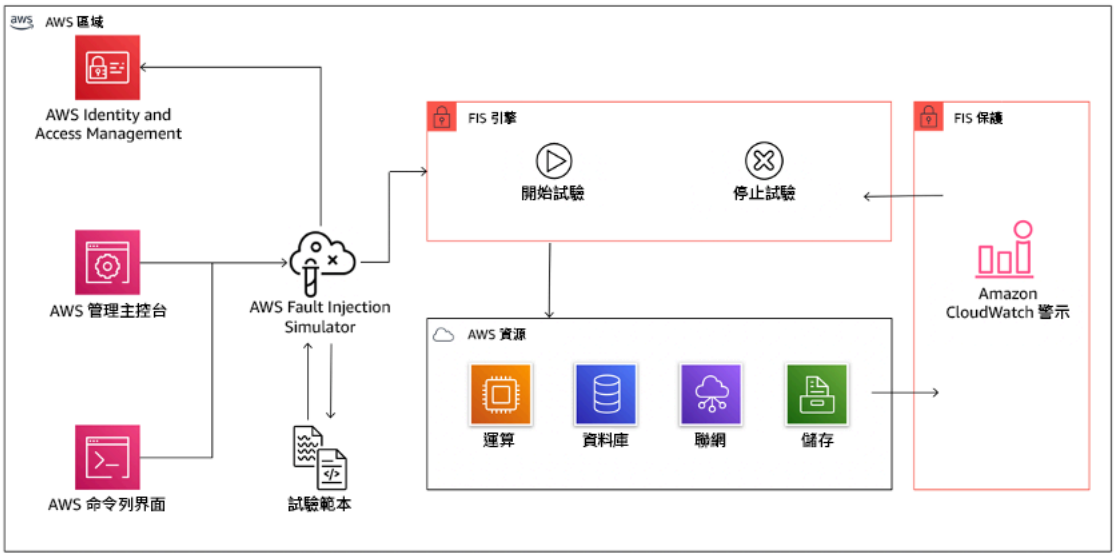
為了驗證由於間歇性網路中斷導致的外部相依性的備用或容錯移轉機制，您的元件應在指定的時間內 (可延續數秒到數小時) 阻止對第三方供應商的存取來模擬這類事件。

其他降級模式可能會導致功能降低和回應速度緩慢，而往往會導致服務中斷。這種降級的常見原因是關鍵服務的延遲增加和不可靠的網路通訊 (丟包)。以這些錯誤 (包括延遲、已捨棄訊息和 DNS 失敗等聯網影響) 進行的試驗，可包含無法解析名稱、無法連線到 DNS 服務，或無法建立相依服務的連線等情境。

混沌工程工具：

AWS Fault Injection Service (AWS FIS) 是用來執行故障注入試驗的全受管服務，這些試驗可作為 CD 管道的一部分，或未於管道以外。AWS FIS 很適合在混沌工程演練日期間使用。它支援跨不同類型的資源同時引入故障，包括 Amazon EC2、Amazon Elastic Container Service (Amazon ECS)、Amazon Elastic Kubernetes Service (Amazon EKS) 和 Amazon RDS。這些錯誤包括終止資源、強制執行容錯

移轉、施壓於 CPU 或記憶體、限流，以及封包遺失。由於它與 Amazon CloudWatch 警示整合，因此您可以將停止條件設定為防護機制，以在試驗導致非預期的影響時將其回復。



AWS Fault Injection Service 與 AWS 資源整合，讓您可對工作負載執行故障注入試驗。

故障注入試驗也有數個第三方選項。其中包括開放原始碼工具，例如 Chaos Toolkit、Chaos Mesh 和 Litmus Chaos，以及諸如 Gemlin 之類的商業選項。為了擴大可以在 AWS 上注入的故障範圍，AWS FIS 會與 Chaos Mesh 和 Litmus Chaos 整合，使您可以在多種工具之間協調故障注入工作流程。例如，您可以在使用 AWS FIS 錯誤動作終止隨機選定百分比的叢集節點時，使用 Chaos Mesh 或 Litmus 錯誤對 Pod 的 CPU 執行壓力測試。

實作步驟

1. 決定要將哪些錯誤用於試驗。

   評估您的工作負載設計是否有彈性。這類設計 (使用 Well-Architected Framework 的最佳實務建立) 會根據關鍵相依性、過去的事件、已知問題和合規性要求來考量風險。列出要用來維護彈性的每個設計元素，及其依設計要減輕的錯誤。如需有關建立此類清單的詳細資訊，請參閱 Operational Readiness Review 白皮書，指導您如何建立程序以防止先前事件再次發生。Failure Modes and Effects Analysis (FMEA) 程序提供了相關架構，讓您執行失敗的元件層級分析，並說明失敗對於工作負載有何影響。Adrian Cockcroft 在 Failure Modes and Continuous Resilience 中詳盡地闡述了 FMEA。

2. 指派每個錯誤的優先順序。

   請從粗略的分類開始著手，例如高、中或低。若要評估優先順序，請考量錯誤的頻率，以及失敗對整體工作負載的影響。

考量特定錯誤的頻率時，請分析此工作負載過去的資料 (如果可用)。如果無法使用，請使用在類似環境中執行的其他工作負載所包含的資料。

考量特定錯誤的影響時，錯誤的範圍愈大，通常影響就愈大。另請考量工作負載的設計和用途。例如，對執行資料轉換和分析的工作負載而言，存取來源資料存放區的能力至關重要。在此情況下，您應優先執行存取錯誤以及限流存取和延遲注入的試驗。

事故後分析是您了解失敗模式的頻率與影響的理想資料來源。

請使用指派的優先順序，決定要先以哪些錯誤進行試驗，以及要以何種順序開發新的故障注入試驗。

3. 對於您所執行的每個試驗，均應依循下圖中的混沌工程和連續彈性飛輪操作。

混沌工程和連續彈性飛輪，採用 Adrian Hornsby 的科學方法。

a. 將穩定狀態定義為顯示出正常行為之工作負載的某種可測量輸出。

工作負載的運作若可靠且符合預期，就會呈現穩定狀態。因此，在定義穩定狀態前，請先驗證工作負載的運作狀態良好。穩定狀態不一定表示在錯誤發生時完全不會影響到工作負載，因為有特定百分比的錯誤可能會在可接受的限制內。穩定狀態是您在試驗期間將觀察到的基準，如果您在下一步定義的假設未符合預期，就會凸顯異常。

例如，支付系統的穩定狀態可定義為 300 TPS、成功率 99%、且來回時間為 500 毫秒的處理。

b. 形成關於工作負載將如何回應錯誤的假設。

良好的假設奠基於工作負載應如何減輕錯誤以維護穩定狀態。假設指出，在發生特定類型的錯誤時，系統或工作負載將持續保有穩定狀態，因為工作負載設有特定緩解機制。特定類型的錯誤和緩解機制應指定於假設中。

以下是可用於假設的範本 (但也接受其他措辭)：

> **Note**
>
> 如果發生####，######工作負載將########，以維持##########。

例如：

- 若 Amazon EKS 節點群組中有 20% 的節點遭到關閉，Transaction Create API 會在 100 毫秒以內繼續提供第 99 個百分位數的請求 (穩定狀態)。Amazon EKS 節點將在五分鐘內復原，而 Pod 將在試驗起始後的八分鐘內進入排程並處理流量。提醒將在三分鐘內引發。
- 單一 Amazon EC2 執行個體失敗發生時，訂單系統的 Elastic Load Balancing 運作狀態檢查將使 Elastic Load Balancing 僅將請求傳送至其餘運作狀態良好的執行個體，而 Amazon EC2 Auto Scaling 會取代失敗的執行個體，將伺服器端 (5xx) 錯誤的增量維持在 0.01% 以內 (穩定狀態)。
- 主要 Amazon RDS 資料庫執行個體失敗時，供應鏈資料收集工作負載將進行容錯移轉並連線至待命 Amazon RDS 資料庫執行個體，以維持不到 1 分鐘的資料庫讀取或寫入錯誤 (穩定狀態)。

c. 藉由注入錯誤來執行試驗。

試驗依預設應處於安全模式，並獲得工作負載的容許。如果您確知工作負載將失敗，請不要執行試驗。混沌工程應該用來尋找已知的未知或未知的未知。已知的未知是您知道但不完全理解的事情，未知的未知是您不知道也不完全理解的事情。對您確知已失效的工作負載執行試驗，將不會為您帶來新的見解。試驗應經過審慎規劃、具有明確的影響範圍，並且提供在非預期的錯亂發生

時可供套用的回復機制。如果您的盡職調查顯示工作負載應可承受試驗，請繼續執行試驗。有數種選項可用來注入錯誤。對於 AWS 上的工作負載，AWS FIS 提供許多預先定義的錯誤模擬，稱為動作。您也可以定義在 AWS FIS 中執行的自訂動作 (使用 AWS Systems Manager 文件)。

我們不鼓勵使用自訂指令碼來執行混沌試驗，除非指令碼有能力理解工作負載目前的狀態、能夠發出日誌，並且提供回復機制和停止條件 (若情況允許)。

支援混沌工程的有效架構或工具集，應追蹤試驗目前的狀態、發出日誌，並提供回復機制以支援受控制的試驗執行。請先從 AWS FIS 這類已建立的服務著手，以便能以明確定義的範圍執行試驗，並且有安全機制可在試驗導入非預期的錯亂時回復試驗。要了解更多有關使用 AWS FIS 的實驗，請參閱 Resilient and Well-Architected Apps with Chaos Engineering lab。此外，AWS Resilience Hub 會分析您的工作負載，並建立可供您選擇在 AWS FIS 中實作並執行的試驗。

> **Note**
>
> 對於每一項試驗，您都應明確了解其範圍與影響。我們建議，錯誤應先在非生產環境中模擬，再於生產環境中執行。

在可行的情況下，實驗應該使用 Canary 部署在實際負載下在生產環境中執行，這可加速控制和實驗系統部署。在非尖峰時段執行試驗是很好的做法，可以減少首次在生產環境中試驗時的潛在影響。此外，如果使用實際的客戶流量會伴隨太高的風險，您可以對控制和試驗部署使用生產基礎架構上的綜合流量，來執行試驗。無法使用生產環境時，請在盡可能接近生產環境的生產前環境中執行試驗。

您必須建立防護機制並加以監控，以確定試驗不會超出可接受的限制而影響到生產流量或其他系統。請建立停止條件，以在試驗達到您定義的防護機制指標閾值時，將試驗停止。其中應包括工作負載的穩定狀態指標，以及您對其注入錯誤的元件所適用的指標。綜合監測 (也稱為使用者 Canary)，是您在一般情況下應納入作為使用者代理的指標之一。AWS FIS 的停止條件被視為試驗範本的一部分受到支援，每個範本最多可啟用五個停止條件。

混沌的準則之一，是盡可能縮小試驗的範圍與影響：

儘管容許某些短期負面影響是必要的，但混沌工程師有責任和義務將試驗的副作用控制在最低限度。

驗證範圍和潛在影響的方法之一，是先在非生產環境中執行試驗，驗證停止條件的閾值在試驗期間會依預期啟動，且有可觀測性會捕捉例外狀況，而不是直接在生產環境中試驗。

執行故障注入試驗時，請驗證所有的責任方都會及時獲得通知。請與營運團隊、服務可靠性團隊和客戶支援等適當的團隊通訊，讓他們知道試驗將於何時執行，且預期會有何情況。請為這些團隊提供通訊工具，以便他們在試驗執行期間發現任何不利影響時發出通知。

您必須將工作負載及其基礎系統還原為原始的已知良好狀態。工作負載的彈性設計通常具有自癒能力。但某些錯誤設計或失敗的試驗可能會使您的工作負載處於非預期的失敗狀態。試驗結束時，您必須察覺到這一點，並還原工作負載和系統。透過 AWS FIS，您可以在動作參數內設定回復組態 (也稱為後置動作)。後置動作會將目標回復為動作執行前原有的狀態。無論是自動 (例如使用 AWS FIS) 還是手動，這些後續動作皆應為程序手冊的一部分，以說明如何偵測及處理失敗。

d. 驗證假設。

混沌工程的原則提供了下列關於如何驗證工作負載穩定狀態的指引：

著重於可測量的系統輸出，而不是系統的內部屬性。這類輸出在一段時間內的測量，會構成系統穩定狀態的代理。整體系統的輸送量、錯誤率和延遲百分位數，全都可能成為呈現穩定狀態行為的相關指標。著重於試驗期間的系統行為模式，混沌工程會驗證系統是否可運作，而非試著驗證其運作情形。

在先前的兩個範例中，我們納入了伺服器端 (5xx) 錯誤的增量低於 0.01% 的穩定狀態指標，以及資料庫讀取和寫入錯誤不到一分鐘的穩定狀態指標。

5xx 錯誤是工作負載的用戶端在失敗模式下將直接經歷的結果，因此可說是良好的指標。資料庫錯誤測量是錯誤的直接產物，因此有其效用，但應同時輔以用戶端影響測量，例如失敗的客戶請求或用戶端遇到的錯誤。此外，請在工作負載的用戶端直接存取的任何 API 或 URI 納入綜合監控 (也稱為使用者 Canary)。

e. 改善工作負載設計的彈性。

如果未維持穩定狀態，請採用 AWS Well-Architected 可靠性支柱的最佳實務，調查如何改進工作負載設計來緩解故障。可以在 AWS 建置者資料中心中找到其他指引和資源，其中包含有關如何改善運作狀態檢查或在應用程式碼中透過輪詢進行重試等文章。

這些變更實作完成後，請再次執行試驗 (在混沌工程飛輪中以虛線表示) 以判斷其有效性。若驗證步驟指出假設成立，則工作負載將處於穩定狀態，且週期會繼續。

4. 請定期執行試驗。

混沌試驗是一個週期，而試驗應被視為混沌工程的一部分定期執行。當工作負載符合試驗的假設後，即應將試驗自動化，以將其視為 CI/CD 管道的迴歸部分持續執行。要了解如何執行此操作，請

參閱有關如何使用 AWS CodePipeline 執行 AWS FIS 實驗的部落格。這個關於 CI/CD 管道中的經常性 AWS FIS 實驗的實驗室可讓您親手操作。

故障注入試驗也是演練日的一部分 (請參閱 REL12-BP05 定期進行演練日)。演練日會模擬失敗或事件，以驗證系統、程序和團隊的應變。目的是實際執行在異常事件發生時團隊將要執行的動作。

5. 擷取並儲存試驗結果。

故障注入試驗的結果必須擷取並保存。請納入所有必要資料 (例如時間、工作負載和條件)，以便後續能分析試驗結果和趨勢。舉例來說，結果可包括儀表板的螢幕擷取畫面、指標的資料庫產生的 CSV 傾印，或是試驗中的事件與觀察的手寫記錄。使用 AWS FIS 試驗日誌記錄可以是此資料擷取的一部分。

資源

相關的最佳實務：

- REL08-BP03 整合彈性測試作為部署的一部分
- REL13-BP03 測試災難復原實作以驗證實作

相關文件：

- 什麼是 AWS Fault Injection Service？
- 什麼是 AWS Resilience Hub？
- 混沌工程的原則
- 混沌工程：規劃您的第一個試驗
- 彈性工程：學習接受故障
- 混沌工程案例
- 避免分散式系統的備用
- 混沌試驗的 Canary 部署

相關影片：

- AWS re:Invent 2020：使用混沌工程測試彈性 (ARC316)
- AWS re:Invent 2019：透過混沌工程提升彈性 (DOP309-R1)
- AWS re:Invent 2019：在無伺服器環境中執行混沌工程 (CMY301)

相關工具：

- AWS Fault Injection Service
- AWS Marketplace：Gremlin 混沌工程平台
- Chaos Toolkit
- Chaos Mesh
- Litmus

REL12-BP05 定期進行演練日

舉辦演練日，以定期執行程序來回應影響工作負載的事件和損害。讓可能負責處理實際執行案例的相同團隊參與。這些練習有助於強制執行措施，以防止使用者因實際執行事件而受到影響。若您在實際情況下練習回應程序，就可以在實際事件發生之前識別和解決任何差距或弱點。

演練日會在類似實際執行環境中模擬事件，以測試系統、程序和團隊回應。目的是實際執行在異常事件發生時，團隊會執行的動作。這些練習可幫助您了解何處有改善空間，並能協助發展組織處理事件和損害的經驗。這些練習應該定期進行，讓您的團隊能夠培養應對的肌肉記憶。

演練日可讓團隊更有信心地處理實際發生的事件。熟悉演練的團隊更能夠快速偵測和回應各種案例。這可大幅改善整備度和彈性狀態。

預期成果：您定期一致地進行彈性演練日。這些演練日會做為預期的正常業務營運。您的組織袍養出整備文化，當實際執行問題發生時，您的團隊已準備好有效地回應、有效率地解決問題，並減輕對客戶的影響。

常見的反模式：

- 您記載您的程序，但未曾進行演練。
- 您未讓業務決策者參與測試練習。
- 您進行演練日，但未通知所有相關利害關係人。
- 您只專注於技術失敗，但未納入業務利害關係人。
- 您未將演練日所學到的經驗納入復原程序中。
- 您在發生失敗或錯誤時責怪團隊。

建立此最佳實務的優勢：

- 增強回應技能：在演練日，團隊會在模擬事件的過程中練習其職責並測試其溝通機制，從而在實際情境中建立更協調且更有效率的回應。
- 識別和解決相依性：複雜的環境通常涉及各種系統、服務和元件之間繁複的相依性。演練日可協助您識別並解決這些相依性，以及確認您的關鍵系統和服務確實涵蓋在您的執行手冊程序內，並且能夠及時向上擴展或復原。
- 培養彈性的文化：演練日有助於培養組織內的彈性思維。當您納入跨職能團隊和利害關係人時，這些練習就能提升整個組織對彈性重要性的意識、協作和同理。
- 持續改進和適應：定期演練日可協助您持續評估和調整彈性策略，以便在面對不斷變化的情況時，保持關聯性和有效性。
- 提高對系統的信心：成功的演練日可協助您建立對系統承受中斷並從中復原能力的信心。

未建立此最佳實務時的曝險等級：中

實作指引

設計並實作必要的彈性措施後，請舉行演練日來確認實際執行時，一切是否如規畫運作。特別是第一次的演練日，應納入所有團隊成員，且所有利害關係人和參與者都應事先收到有關日期、時間和模擬案例的通知。

在演練日，參與的團隊會根據既定程序模擬各種事件和可能的案例。參與者會密切監控和評估這些模擬事件的影響。如果系統依設計運作，則自動偵測、擴展和自我修復機制應會啟動，而且對使用者的影響極少或無影響。如果團隊觀察到任何負面影響，則會透過自動化的方式或適用的執行手冊中記錄的手動介入方式，來復原測試並補救找到的問題。

為了持續改善彈性，記錄並納入學到的經驗至關重要。此程序是一種回饋循環，採取系統化的方式從演練日獲得洞察，並利用它們來增強系統、程序和團隊功能。

為了協助您重現系統元件或服務可能意外失敗的真實案例，請將模擬錯誤植入演練日練習中。團隊可以測試其系統的彈性和容錯能力，並在受控的環境中模擬其事件回應和復原程序。

在 AWS 中，您的演練日可以使用基礎設施即程式碼，透過實際執行環境的複本來執行。透過此程序，您可以在類似實際執行環境的安全環境中進行測試。考慮使用 AWS Fault Injection Service 來建立不同的失敗案例。使用 Amazon CloudWatch 和 AWS X-Ray 等服務來監控演練日的系統行為。使用 AWS Systems Manager 來管理和執行程序手冊，並使用 AWS Step Functions 來協調週期性演練日工作流程。

實作步驟

- 擬訂演練日計畫：擬訂結構化的計畫，以定義演練日的頻率、範圍和目標。規畫和執行這些練習時，讓主要利害關係人和主題專家參與。
- 準備演練日：
  1. 指定做為演練日重點的重要關鍵業務服務。將支援這些服務的人員、程序和技術造冊並對應。
  2. 設定演練日的流程，並讓參與的團隊準備好參與活動。備妥自動化服務，以模擬規劃的案例並執行適當的復原程序。AWS Fault Injection Service、AWS Step Functions 和 AWS Systems Manager 等 AWS 服務可協助您自動進行演練日的各個層面，例如植入錯誤和啟動復原動作。
- 執行模擬：在演練日當天，執行規劃的案例。觀察並記錄人員、程序和技術對模擬事件的反應。
- 進行練習後檢討：在演練日結束後，舉行回顧會議來檢討學到的經驗。識別需要改進的層面，以及改善營運彈性所需的任何動作。記錄您的調查結果，並追蹤任何必要的變更，以增強彈性策略並完成準備。

資源

相關的最佳實務：

- REL12-BP01 使用程序手冊調查失敗
- REL12-BP04 使用混沌工程測試彈性
- OPS04-BP01 識別關鍵績效指標
- OPS07-BP03 使用執行手冊執行程序
- OPS10-BP01 使用程序進行事件、事故和問題管理

相關文件：

- 什麼是 AWS GameDay？
- AWS Well-Architected 概念 - 演練日

相關影片：

- AWS re：Invent 2023 - 寓教於樂：Amazon 如何將彈性擴展到新境界

相關範例：

- AWS 研討會 - 駕馭風暴：充分利用彈性系統的受控混沌
- 打造自己的演練日以支援營運彈性

災難復原 (DR) 計畫

備妥備份和冗餘工作負載元件是 DR 策略的開始。RTO 和 RPO 是您還原工作負載的目標。根據業務需求設定這些目標。實作策略以滿足這些目標，考量工作負載資源和資料的位置和功能。發生中斷的可能性和復原成本也是重要因素，可反映為工作負載提供災難復原的商業價值。

可用性和災難復原都依賴相同的最佳實務，例如監控故障、部署至多個位置以及自動容錯移轉。然而，可用性著重於工作負載的元件，而災難復原則著重於整個工作負載的離散副本。災難復原的目標與可用性不同，著重於災難後復原的時間。

最佳實務

- REL13-BP01 定義停機時間和資料遺失的復原目標
- REL13-BP02 使用定義的復原策略來滿足復原目標
- REL13-BP03 測試災難復原實作以驗證實作
- REL13-BP04 管理 DR 站點或區域的組態偏移
- REL13-BP05 自動化回復

REL13-BP01 定義停機時間和資料遺失的復原目標

故障可能會以多種形式影響您的業務。首先，故障可能會導致服務中斷 (停機時間)。其次，故障可能會導致資料遺失、不一致或過時。為了引導您回應故障並從故障中復原，請為每個工作負載定義復原時間點目標 (RTO) 和復原點目標 (RPO)。復原時間點目標 (RTO) 是服務中斷與服務還原之間的可接受延遲上限。復原點目標 (RPO) 是從上次資料復原點之後起算，可接受的最長時間。

預期成果：每個工作負載都根據技術考量和業務影響指定了 RTO 和 RPO。

常見的反模式：

- 您尚未指定復原目標。
- 您選取任意復原目標。
- 您選取的復原目標過於寬鬆且不符合業務目標。
- 您尚未評估停機時間和資料遺失的影響。

- 您選取的復原目標不切實際，例如零復原時間或零資料損失，這對於您的工作負載組態而言可能無法實現。
- 您選取的復原目標比實際業務目標更嚴格。這會強制進行比工作負載所需更昂貴和更複雜的復原實作。
- 您選取的復原目標與相依的工作負載不相容。
- 您未將法規和合規要求納入考量。

建立此最佳實務的優勢：設定工作負載的 RTO 和 RPO 之後，您就可以根據業務需求訂定清楚且可衡量的復原目標。設定這些目標之後，您就可以制定專為實現這些目標所打造的災難復原 (DR) 計畫。

未建立此最佳實務時的曝險等級：高

實作指引

建構對照表或工作表，以協助引導您的災難復原規劃。在對照表中，根據個別的業務影響 (例如嚴重、高、中和低) 以及要做為個別目標的相關聯 RTO 和 RPO，建立不同的工作負載類別或分級。下列對照表為您提供可遵循的範例 (請注意，您的 RTO 和 RPO 值可能不同)：

災難復原方法						
		復原點目標				
		< 1 分鐘	< 1 小時	< 6 小時	< 1 天	+ 1 天
復原時間目標	< 10 分鐘	嚴重	嚴重	高	中	中
	< 2 小時	嚴重	高	中	中	低
	< 8 小時	高	中	中	低	低
	< 24 小時	中	中	低	低	低
	24 + 小時	中	低	低	低	低

災難復原對照表範例

針對每個工作負載，調查並了解停機時間和資料遺失對業務造成的影響。影響通常會隨著停機時間和資料遺失而增強，但影響的程度會根據工作負載類型而有所不同。例如，長達一小時的停機時間可能會產生低程度的影響，但隨後影響可能會迅速增強。影響可能有多種形式，包括財務影響 (例如收益損失)、聲譽影響 (包括失去客戶信任)、營運影響 (例如發不出薪資或生產力降低)，以及法規風險。完成後，將工作負載指派至適當的分級。

當您分析故障的影響時，請考慮下列問題：

1. 在對業務造成無法接受的影響之前，可以容忍的最長工作負載停機時間是多久？
2. 工作負載中斷會對業務產生多大程度的影響以及何種影響？ 請考慮各種影響，包括財務、聲譽、營運和法規。
3. 在對業務造成無法接受的影響之前，可以容忍的遺失或無法復原的最大資料量是多少？
4. 是否可從其他來源重新建立遺失的資料 (也稱為衍生資料)？ 如果可以，請同時考量用來重新建立工作負載資料的所有來源資料的 RPO。
5. 此工作負載所依賴的工作負載 (下游) 的復原目標和可用性期望為何？ 工作負載的目標必須在其下游相依的工作負載具有復原能力的情況下才能達成。請考慮可能改善此工作負載復原能力的下游相依性因應措施或緩解措施。
6. 依賴此工作負載的工作負載 (上游) 的復原目標和可用性期望為何？ 上游工作負載目標可能要求此工作負載具有比首次出現更嚴格的復原能力。
7. 是否取決於事件類型而有不同的復原目標？ 例如，根據事件影響的是可用區域或整個區域而定，您可能有不同的 RTO 和 RPO。
8. 您的復原目標是否在一年中的某些事件或某段時間改變？ 例如，您可能在節日購物季節、體育賽事、特別促銷和新產品推出時，採用不同的 RTO 和 RPO。
9. 復原目標如何與您可能擁有的任何業務範圍和組織災難復原策略保持一致？
10. 是否需要考慮任何法律或合約後果？ 例如，您是否在合約方面有義務要提供具有特定 RTO 或 RPO 的服務？ 若未履行義務，可能會受到哪些懲罰？
11. 是否需要維護資料完整性以符合法規或合規要求？

下列工作表可協助您評估每個工作負載。您可以修改此工作表以滿足您的特定需求，例如新增其他問題。

步驟 2：主要問題	是否適用於工作負載？	工作負載 RTO	工作負載 RPO	RTO 調整。	RPO 調整。	簡介
[1] 工作負載可以關閉的最長時間						以開始中斷到復原的時間進行測量
[2] 可以遺失的資料數量上限						以自從上次已知良好的可還原資料集後的時間進行測量
[3a] 上游相依性						輸入最嚴格的上游復原目標
[3b] 下游相依性						輸入最不嚴格的下游復原目標
[3a] 達成一致的上游相依性						如果上游值小於目前值，而下游值更大，則使用相依性來達成一致，
[3b] 達成一致的下游相依性						並在這裡輸入達成一致的值
[3] 相依性						請降低值以符合上游相依性，或根據下游相依性功能提高這些值
步驟 2：其他問題						指出問題是否適用。如果不適用，則略過它
基底 RTO/RPO						將上面的 RTO 和 RPO 值帶至這裡
[4] 中斷類型	[ ]Y / [ ]N					為具有最嚴格需求的事件類型輸入復原目標
[5] 特定時間型目標	[ ]Y / [ ]N					為具有最嚴格需求的時間輸入復原目標
[6] 顛覆客戶	[ ]Y / [ ]N					透過圖表以停機時間或資料遺失的函數表示受影響的客戶。 使用該函數，根據客戶影響輸入最大允許的 RTO 和 RPO
[7] 信譽影響	[ ]Y / [ ]N					與企業合作，根據對信譽的影響決定最大 RTO 和 RPO
[8] 營運影響	[ ]Y / [ ]N					根據營運影響輸入最大 RTO 和 RPO
[9] 組織遵循	[ ]Y / [ ]N					根據 LOB 和組織需求輸入此類型的最大工作負載 RTO 和 RPO
[10] 合約義務	[ ]Y / [ ]N					根據合約義務輸入最大 RTO 和 RPO
[11] 法規合規	[ ]Y / [ ]N					根據適用的法規合規輸入最大 RTO 和 RPO
以其他問題為基礎的目標						從 Q’s 4-11 取得並在這裡輸入最小值 (更嚴格的值)
調整後的目標						如果無法滿足上行的目標，請與利害關係人合作放寬限制，並在這裡輸入新的最小值
調整後的 RTO/RPO						輸入基底 RPO/RTO 值或調整後的目標，以較低者為準
步驟 3						
對應至預先定義的類別或層級						向下調整這兩個值 (更嚴格) 以與最接近的定義層級一致

工作表

實作步驟

1. 確定負責每個工作負載的業務利害關係人和技術團隊，並與他們互動。
2. 在組織中建立工作負載影響的關鍵性類別或分級。範例類別包括：嚴重、高、中和低。為每個類別選擇合乎您業務目標和需求的 RTO 和 RPO。
3. 將您在上一個步驟中建立的其中一個影響類別指派給每個工作負載。若要決定工作負載對應至類別的方式，請考慮工作負載對業務的重要性，以及中斷或資料遺失的影響，並利用上述問題來引導您。這樣就能找到每個工作負載的 RTO 和 RPO。
4. 考慮上一個步驟中為每個工作負載確定的 RTO 和 RPO。讓工作負載的業務和技術團隊參與，以判斷是否應調整目標。例如，業務利害關係人可判斷是否需要更嚴格的目標。或者，技術團隊可以判斷應修改目標，以透過可用的資源和技術限制來實現目標。

資源

相關的最佳實務：

- REL09-BP04 定期執行資料復原以驗證備份的完整性和程序

- REL12-BP01 使用程序手冊調查失敗
- REL13-BP02 使用定義的復原策略來滿足復原目標
- REL13-BP03 測試災難復原實作以驗證實作

相關文件：

- AWS 架構部落格：災難復原系列
- AWS 上工作負載的災難復原：在雲端中復原 (AWS 白皮書)
- 使用 AWS Resilience Hub 管理彈性政策
- APN 合作夥伴：可協助進行災難復原的合作夥伴
- AWS Marketplace：可用於災難復原的產品

相關影片：

- AWS re:Invent 2018：適用於多區域主動-主動應用程式的架構模式
- AWS 上工作負載的災難復原

REL13-BP02 使用定義的復原策略來滿足復原目標

定義一個符合工作負載復原目標的災難復原 (DR) 策略。選擇如下策略：備份與還原；待命 (主動/被動)；或是主動/主動。

預期成果：對於每個工作負載，都有已定義並實作的 DR 策略，可讓工作負載實現災難復原目標。工作負載之間的 DR 策略會利用可重複使用模式 (例如上述策略)，

常見的反模式：

- 針對具有類似 DR 目標的工作負載實作不一致的復原程序。
- 災難發生時臨時實作 DR 策略。
- 沒有災難復原的計劃。
- 復原期間依賴控制平面操作。

建立此最佳實務的優勢：

- 使用定義的復原策略可讓您使用常用的工具和測試程序。

- 使用定義的復原策略可改善在團隊之間分享知識，並更輕鬆地在他們擁有的工作負載上實作 DR。

未建立此最佳實務時的風險暴露等級：高。若沒有事先規劃、實作和測試災難復原策略，您就不可能在發生災難時實現復原目標。

實作指引

如果您的主要位置變成無法執行工作負載，則災難復原策略會依賴在復原站點中支持您工作負載的能力。最常見的復原目標為 RTO 和 RPO，如 REL13-BP01 定義停機時間和資料遺失的復原目標 中所討論。

單一 AWS 區域 內跨多個可用區域 (AZ) 的 DR 策略可以緩解火災、洪水和重大停電等災難事件。如果需要實作保護，以防範阻止您的工作負載在給定 AWS 區域 中執行且不太可能發生的事件，您可以使用一個使用多個區域的 DR 策略。

在跨多個區域架構 DR 策略時，您應該選擇下列其中一個策略。其按成本和複雜性的升序以及 RTO 和 RPO 的降序排列。復原區域是指用於工作負載的主要區域以外的 AWS 區域。

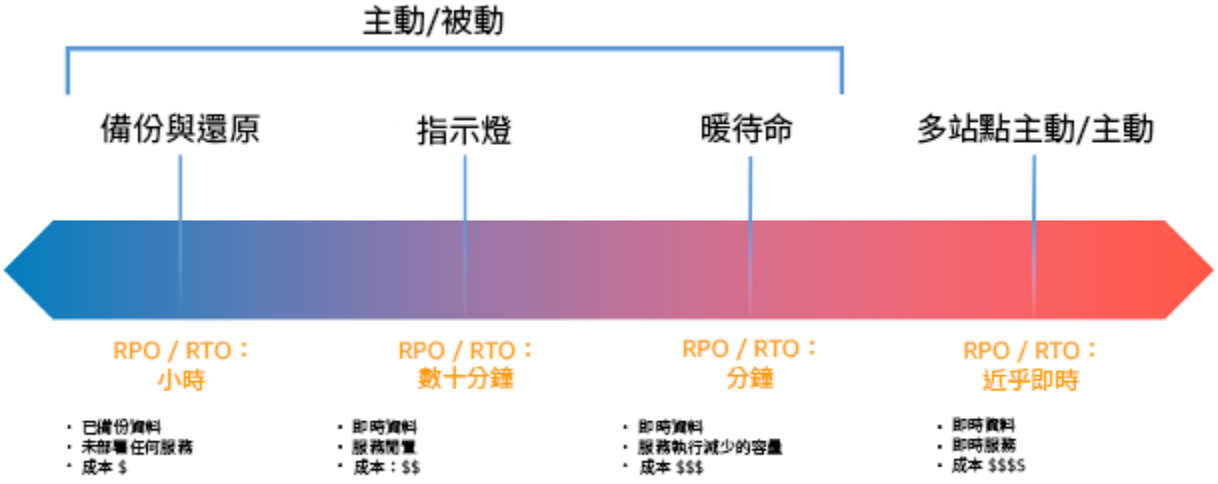


圖 17：災難復原 (DR) 策略

- 備份與還原 (RPO 以小時為單位，24 小時以內的 RTO)：將您的資料和應用程式備份至復原區域。使用自動或連續備份將允許時間點復原 (PITR)，在某些情況下可以將 RPO 降低至 5 分鐘。如果發生災難，您將部署您的基礎設施 (使用基礎設施架構即程式碼來減少 RTO)、部署您的程式碼，並還原備份的資料以從復原區域中的災難中復原。

- 指示燈 (RPO 以分鐘為單位，RTO 以十分鐘為單位)：在復原區域佈建核心工作負載基礎設施的副本。將您的資料複寫到復原區域並在該處建立其備份。支援資料複寫和備份所需的資源 (例如資料庫和物件儲存) 始終處於開啟狀態。其他元素 (例如應用程式伺服器或無伺服器運算) 未部署，但可在需要時使用必要的組態和應用程式碼建立。
- 暖待命 (RPO 以秒為單位，RTO 以分鐘為單位)：維持始終在復原區域中執行，規模縮減但功能完整的工作負載版本。業務關鍵系統會完全複製且持續開啟，但叢集會縮小。資料會被複寫並存在於復原區域中。當需要復原時，系統會迅速擴展以處理生產負載。暖待命的縱向擴增越多，對 RTO 和控制平面的依賴就越低。當完全擴展時，這稱為熱待命。
- 多區域 (多站台) 主動-主動式 (RPO 接近零，RTO 可能為零)：您的工作負載會部署到多個 AWS 區域，並可主動為其流量提供服務。此策略需要您跨區域同步資料。必須避免或處理在兩個不同區域複本中寫入同一記錄所引起的可能衝突，這可能很複雜。資料複寫對於資料同步很有用，而且可以保護您防範某些類型的災難，但它不能保護您防範資料損毀或破壞，除非您的解決方案也包括時間點復原的選項。

> **Note**
>
> 指示燈和暖待命之間的差異有時可能很難理解。這兩者都在您的復原區域中包含一個環境，其中具有主要區域資產的副本。區別在於，若未先採取額外動作，指示燈無法處理請求，而暖待命可以立即處理流量 (容量層級降低)。指示燈將需要您開啟伺服器，可能會部署額外 (非核心) 基礎設施並向上擴展，而暖待命只需要您向上擴展 (一切都已部署並執行中)。根據您的 RTO 和 RPO 需求在這兩者之間進行選擇。
> 當成本是一大顧慮時，且想要達到與暖待命策略所定義類似的 RPO 和 RTO 目標，您可以考慮雲端原生解決方案，例如 AWS Elastic Disaster Recovery，它會採取指示燈方法並且提供改善的 RPO 和 RTO 目標。

實作步驟

1. 決定將滿足此工作負載復原需求的 DR 策略。

   選擇 DR 策略是在減少停機時間和資料遺失 (RTO 和 RPO) 與實作策略的成本和復雜性之間進行取捨。您應該避免實作比其所需更嚴格的策略，因為這會產生不必要的成本。

   例如，在下圖中，企業已確定其最大允許的 RTO 以及其可以在服務還原策略上花費的限制。鑑於業務目標，DR 策略指示燈或暖待命將同時滿足 RTO 和成本準則。

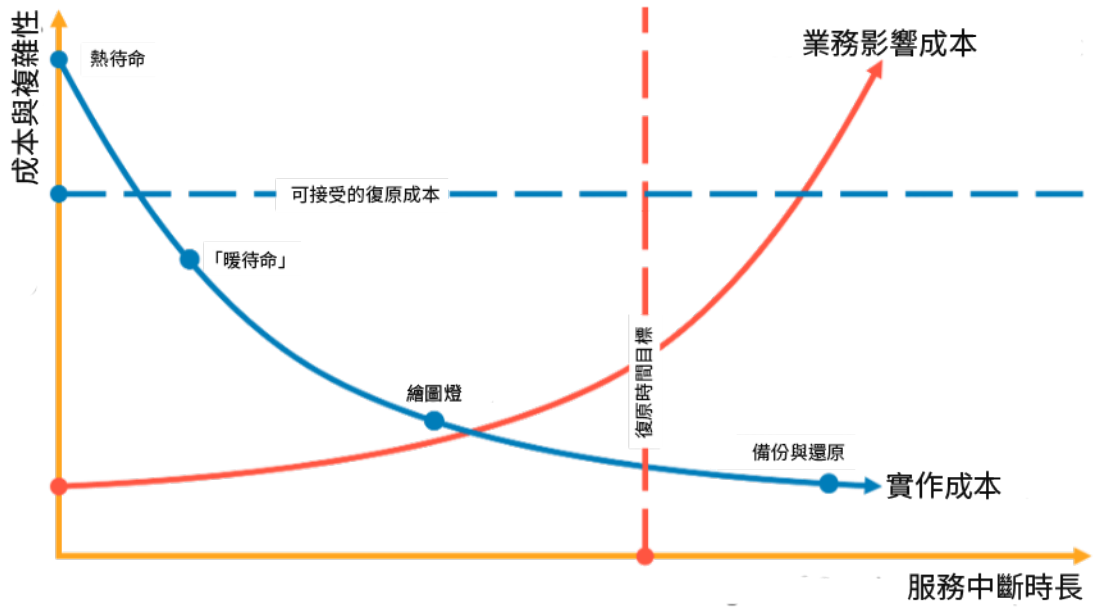


圖 18：根據 RTO 和成本選擇 DR 策略

如需進一步了解，請參閱業務持續性計劃 (BCP)。

2. 審核如何實作所選 DR 策略的模式。

   此步驟在於了解您將如何實作所選策略。使用 AWS 區域 做為主要站點和復原站點來解釋這些策略。不過，您也可以選擇使用單一區域內的可用區域，做為您的 DR 策略，這會利用其中多個策略的元素。

   在下列步驟中，您可以將策略套用到您的特定工作負載。

   備份和還原

   備份和還原是最不複雜的實作策略，但還原工作負載需要更多時間和精力，從而導致更高的 RTO 和 RPO。始終對資料進行備份並將其複製到另一個站點 (例如另一個 AWS 區域) 是一種很好的做法。

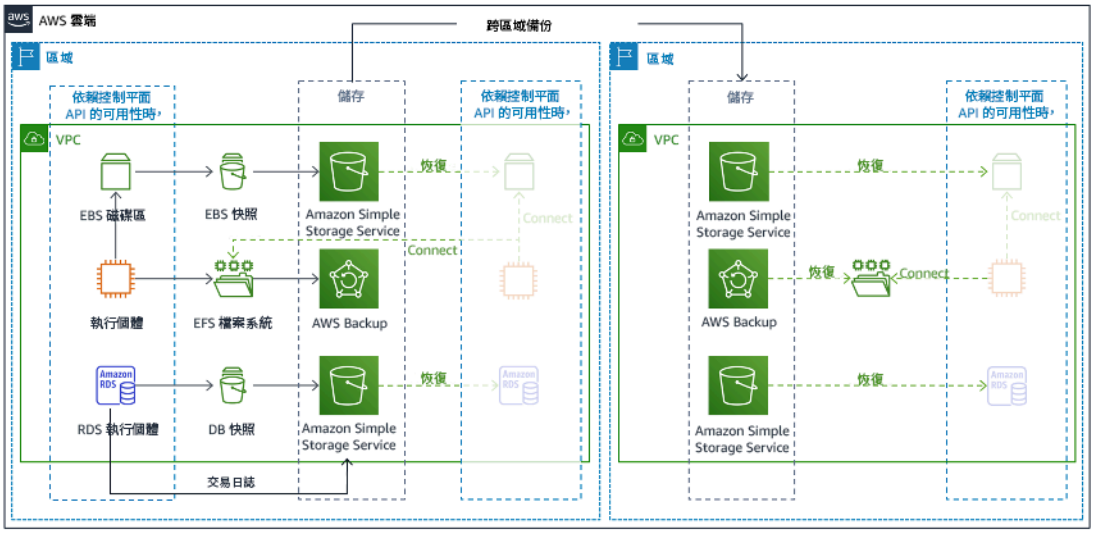


圖 19：備份和還原架構

如需此策略的詳細資訊，請參閱 [Disaster Recovery (DR) Architecture on AWS, Part II: Backup and Restore with Rapid Recovery]。

指示燈

使用指示燈方法，可以將資料從主要區域複製到復原區域。用於工作負載基礎設施的核心資源會部署在復原區域中，不過，仍需要額外的資源和任何相依性，才能使其成為功能堆疊。例如，在圖 20 中，未部署任何運算執行個體。

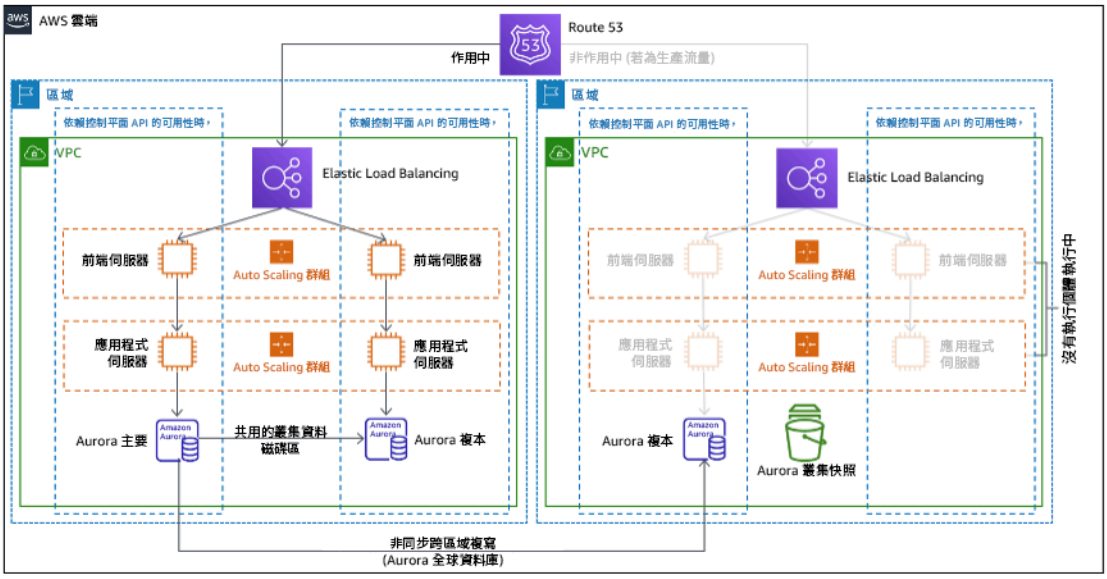


圖 20：指示燈架構

有關此策略的詳細資訊，請參閱 AWS 上的災難復原 (DR) 架構，第 III 部分：指示燈和暖待命。

暖待命

暖待命方法包括確保在另一個區域中有規模縮減但功能完整的生產環境副本。這種方法擴充了指示燈概念並減少了復原時間，因為您的工作負載始終在另一個區域中開啟。如果復原區域已部署全部容量，則稱為熱待命。

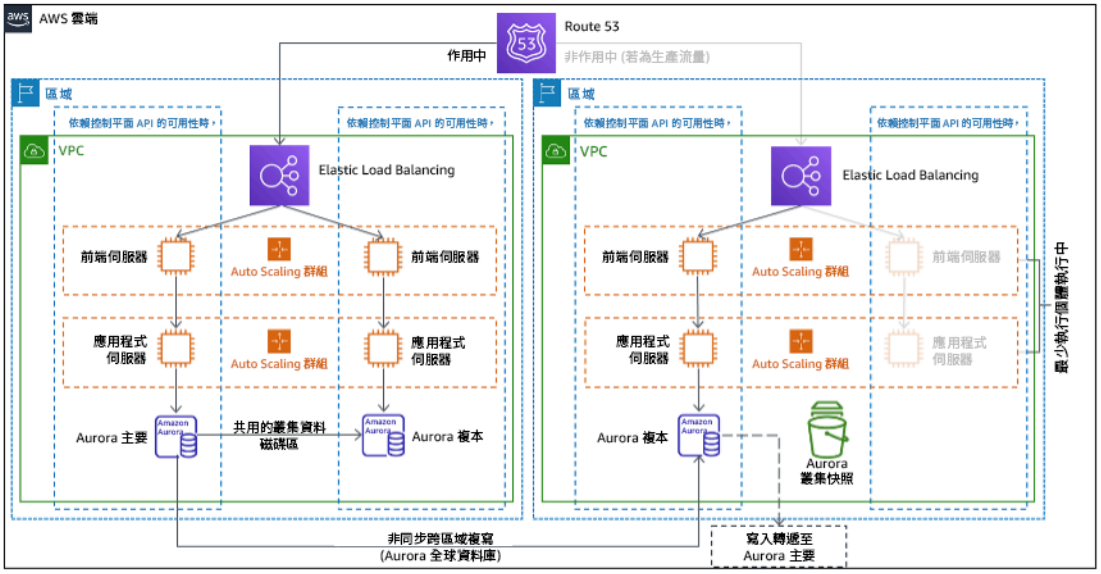


圖 21：暖待命架構

使用暖待命或指示燈需要縱向擴展復原區域中的資源。若要確認容量在需要時可用，請考慮使用 EC2 執行個體的容量保留。如果使用 AWS Lambda，佈建並行可以提供執行時期環境，以便立即回應函數的調用。

有關此策略的詳細資訊，請參閱 AWS 上的災難復原 (DR) 架構，第 III 部分：指示燈和暖待命。

多站點主動/主動

作為多站台主動/主動策略的一部分，您可以在多個區域同時執行工作負載。多站點主動/主動會為來自其部署至的所有區域的流量提供服務。客戶可以出於 DR 以外的原因選擇此策略。它可以用來提高可用性，或在將工作負載部署至全球對象 (使端點更靠近使用者和/或將本地化的堆疊部署到該區域的對象) 時使用它。作為 DR 策略，如果工作負載無法在其部署至的其中一個 AWS 區域中得到支

援，則會撤離該區域，而其餘區域則會用來維護可用性。多站點主動/主動是災難復原策略中操作最複雜的策略，因此只有在業務要求有此需要時才應選取它。

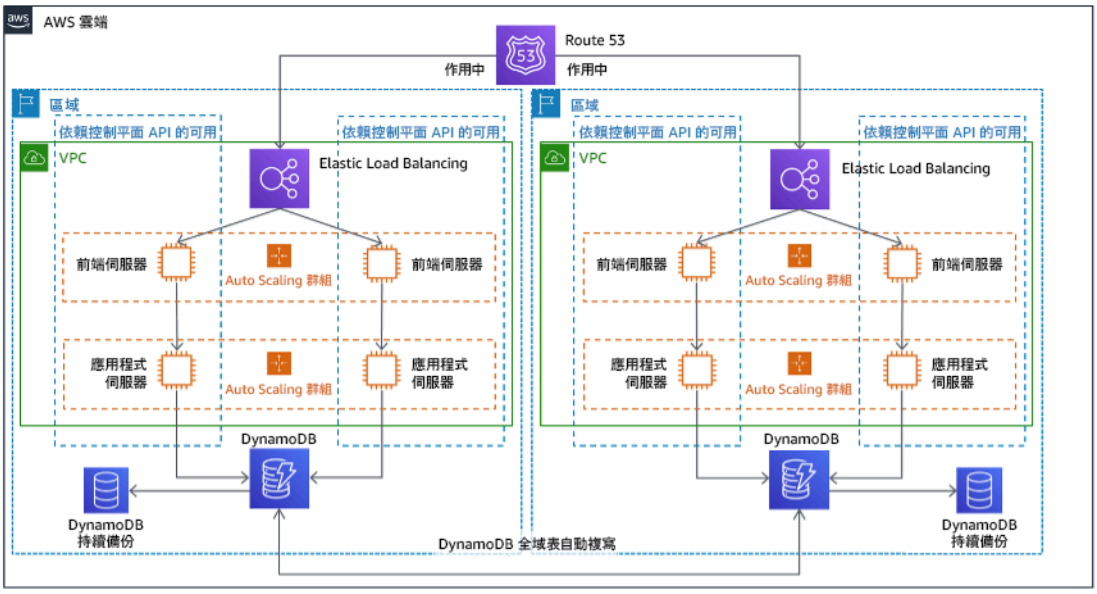


圖 22：多站點主動/主動架構

如需此策略的詳細資訊，請參閱 Disaster Recovery (DR) Architecture on AWS, Part IV: Multi-site Active/Active。

AWS Elastic Disaster Recovery

如果您正在考慮用於災難復原的指示燈或暖待命策略，AWS Elastic Disaster Recovery 可以提供具有改進效益的替代方法。Elastic Disaster Recovery 可以提供類似於暖待命的 RPO 和 RTO 目標，但維持指示燈的低成本方法。Elastic Disaster Recovery 會使用持續的資料保護功能，實現以秒為單位測量的 RPO，以及以幾分鐘為單位測量的 RTO，將您的資料從主要區域複製到復原區域。只有複寫資料所需的資源會在復原區域中部署，保持低成本，類似於指示燈策略。使用 Elastic Disaster Recovery 時，服務會在容錯移轉或練習過程中啟動時進行協調。

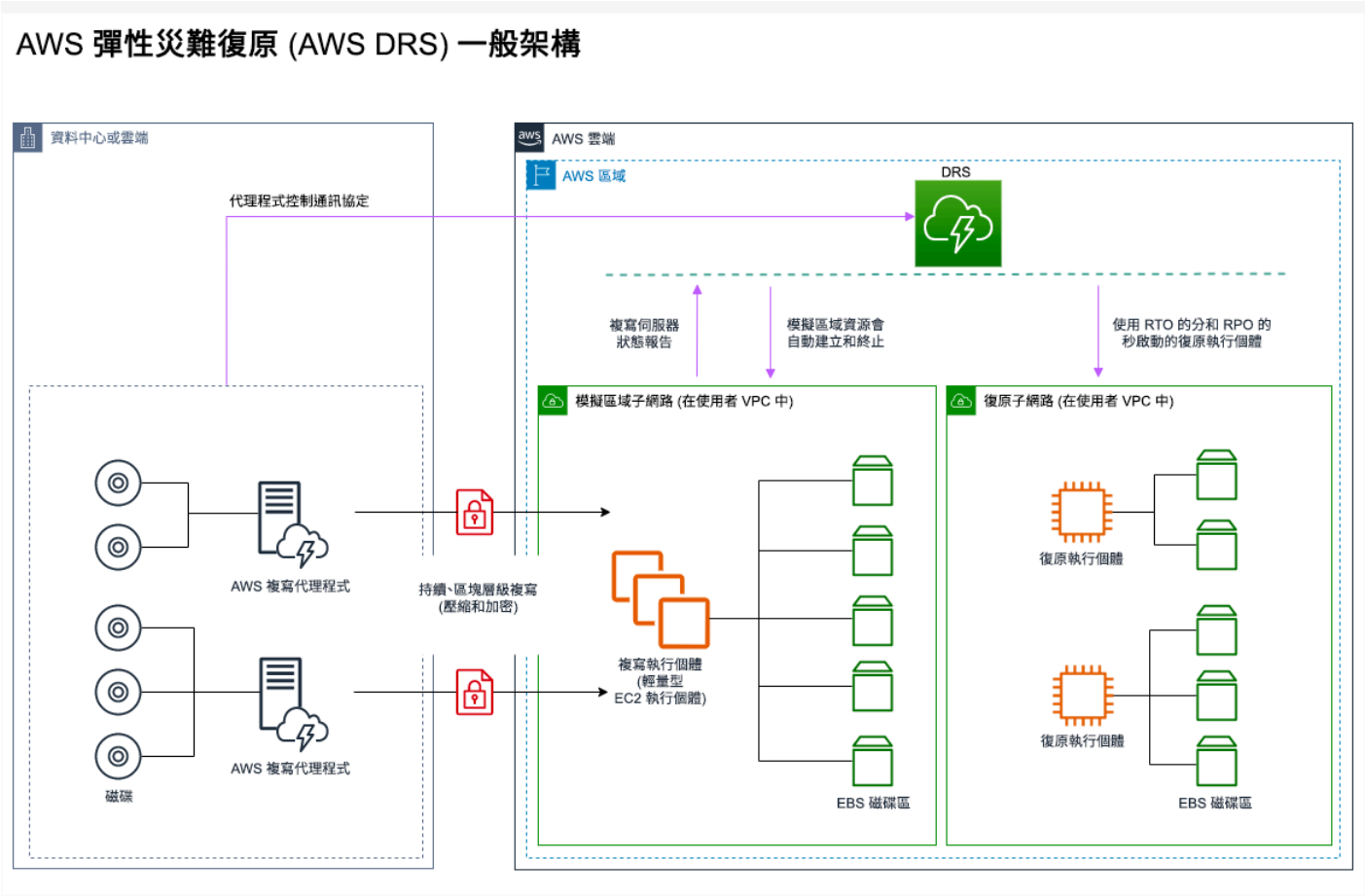


圖 23：AWS Elastic Disaster Recovery 架構

保護資料的其他實務

使用所有策略時，您還必須緩解資料災難。持續資料複寫可以保護您防範某些類型的災難，但它不能保護您防範資料損毀或破壞，除非您的策略也包括所存放資料的版本控制，或時間點復原的選項。除了複本之外，您還必須備份復原站點中的複寫資料，以建立時間點備份。

在單一 AWS 區域內使用多個可用區域 (AZ)

在單一區域內使用多個 AZ 時，您的 DR 實作會使用上述策略的多個元素。首先，您必須建立高可用性 (HA) 架構，使用多個 AZ，如圖 23 所示。由於 Amazon EC2 執行個體和 Elastic Load Balancer 已在多個 AZ 中部署了資源，因此架構可使用多站點主動/主動方法，以積極處理請求。該架構也示範熱待命，如果主要 Amazon RDS 執行個體發生故障 (或 AZ 本身失敗)，則待命執行個體會升級為主執行個體。

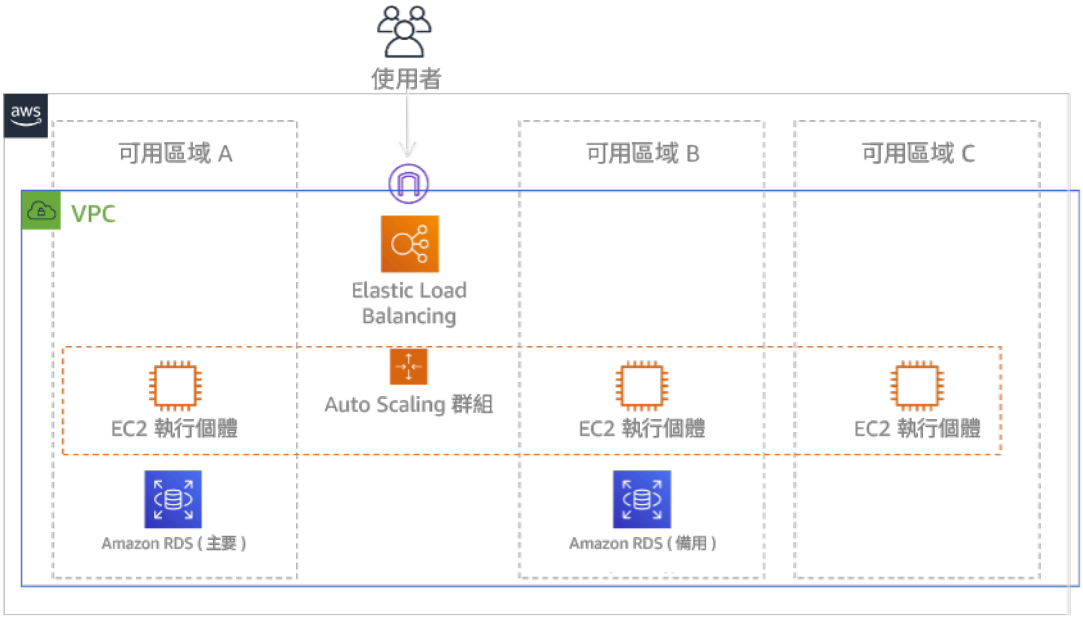


圖 24：多可用區域架構

除了這種 HA 架構之外，您還需要新增執行工作負載所需之所有資料的備份。這對於受限於單一區域的資料 (例如 Amazon EBS 磁碟區或 Amazon Redshift 叢集) 尤其重要。如果 AZ 失敗，您需要將此資料還原至另一個 AZ。可能的話，您也應該將資料備份複製到另一個 AWS 區域，做為額外的保護層。

部落格文章使用 Amazon 應用程式復原控制器建置高彈性應用程式，第 1 部分：單一區域堆疊中說明了單一區域、多可用區域災難復原的不常見替代方法。在這裡，策略是盡可能地在 AZ 之間保持隔離，就像區域的操作方式一樣。使用這種替代策略，您可以選擇主動/主動或主動/被動方法。

> **Note**
>
> 某些工作負載具有法規資料落地要求。如果在目前只有一個 AWS 區域的區域性中，這適用於您的工作負載，則多區域將不適合您的業務需求。異地同步備份策略提供良好的保護，可防範大部分災難。

3. 在容錯移轉之前 (在正常操作期間)，評估工作負載的資源，以及其在復原區域中的組態。

   對於基礎設施和 AWS 資源，請使用基礎設施即程式碼 (例如 AWS CloudFormation) 或諸如 Hashicorp Terraform 之類的第三方工具。若要透過單一操作在多個帳戶和區域中進行部署，可以使用 AWS CloudFormation StackSets。對於多站點主動/主動和熱待命策略，您的復原區域中部署的基礎設施具有與您主要區域相同的資源。對於指示燈和暖待命策略，部署的基礎設施將需要額外的動作，才能為生產做好準備。使用 CloudFormation 參數和條件邏輯，可透過單一範本來控制已部署

的堆疊是處於作用中還是待命。使用 Elastic Disaster Recovery 時，服務會複寫和協調應用程式組態和運算資源的還原。

所有 DR 策略都需要在 AWS 區域 中備份資料來源，然後將這些備份複製到復原區域。AWS Backup 提供集中式檢視，您可以在其中設定、排程和監控這些資源的備份。對於指示燈、暖待命和多站點主動/主動式，您還應該將資料從主要區域複寫到復原區域中的資料資源，例如 Amazon Relational Database Service (Amazon RDS) 資料庫執行個體或 Amazon DynamoDB 資料表。因此，這些資料資源是即時的，而且可以為復原區域中的請求提供服務。

若要深入了解 AWS 服務如何跨區域運作，請參閱 Creating a Multi-Region Application with AWS Services 部落格系列。

4. 確定並實作如何讓復原區域在需要時 (在災難事件發生時) 做好容錯移轉的準備。

   對於多站點主動/主動，容錯移轉意味著撤離一個區域，並依賴剩餘的主動區域。通常，這些區域已準備好接受流量。對於指示燈和暖待命策略，您的復原動作將需要部署遺漏的資源，例如圖 20 中的 EC2 執行個體，以及任何其他遺漏的資源。

   對於上述所有策略，您可能需要提升資料庫的唯讀執行個體，以變成主要讀取/寫入執行個體。

   對於備份和還原，從備份還原資料會為該資料建立資源，例如 EBS 磁碟區、RDS 資料庫執行個體和 DynamoDB 資料表。您也需要還原基礎設施和部署程式碼。您可以使用 AWS Backup，還原復原區域中的資料。如需詳細資訊，請參閱REL09-BP01 識別並備份所有需要備份的資料，或從來源複製資料。重建基礎設施包括建立 EC2 執行個體等資源，以及所需的 Amazon Virtual Private Cloud (Amazon VPC)、子網路和安全群組。您可以將大部分還原程序自動化。若要了解如何操作，請參閱此部落格文章。

5. 確定並實作如何在需要時 (在災難事件發生時) 重新路由流量以進行容錯移轉。

   此容錯移轉作業可以自動或手動啟動。應謹慎使用根據運作狀態檢查或警示自動啟動的容錯移轉，因為不必要的容錯移轉 (誤報) 會產生非可用性和資料遺失等成本。因此通常使用手動啟動的容錯移轉。在此情況下，您仍應將容錯移轉的步驟自動化，讓手動啟動就像按下按鈕一樣簡易。

   使用 AWS 服務時，有數個流量管理選項需要考慮。一種選擇是使用 Amazon Route 53。使用 Amazon Route 53，您可以將一個或多個 AWS 區域中的 IP 端節與一個 Route 53 網域名稱建立關聯。若要實作手動啟動的容錯移轉，您可以使用 Amazon 應用程式復原控制器，它可提供高可用性的資料平面 API，將流量重新路由到復原區域。實作容錯移轉時，使用資料平面操作並避免控制平面操作，如 REL11-BP04 在復原期間依賴資料平面而非控制平面 中所述。

   若要進一步了解此選項和其他選項，請參閱《災難復原白皮書》中的此章節。

6. 設計一個工作負載故障恢復計畫。

容錯恢復是指在災難事件減弱後將工作負載操作回復到主要區域。將基礎設施和程式碼佈建到主要區域通常遵循最初使用的相同步驟，依賴基礎設施即程式碼和程式碼部署管道。容錯恢復的挑戰是還原資料存放區，並確保它們與操作中的復原區域保持一致。

在容錯移轉狀態下，復原區域中的資料庫是即時的並具有最新資料。後續目標是從復原區域重新同步到主要區域，確保它是最新的。

有些 AWS 服務將會自動執行此動作。如果使用 Amazon DynamoDB 全域表，即使主要區域中的資料表變得無法使用，當它重新上線時，DynamoDB 會繼續傳播任何擱置的寫入。如果使用 Amazon Aurora 全球資料庫並使用受管計劃容錯移轉，則維護 Aurora 全球資料庫的現有複寫拓撲。因此，主要區域中先前的讀取/寫入執行個體將成為複本，並從復原區域中接收更新。

如果這不是自動的，您將需要在主要區域中重建資料庫，做為復原區域中資料庫的複本。在許多情況下，這將涉及刪除舊的主要資料庫並建立新的複本。

容錯移轉後，如果您可以繼續在復原區域中執行，請考慮使其成為新的主要區域。您仍會執行上述所有步驟，使先前的主要區域成為復原區域。有些組織會執行排程輪換，定期 (例如每三個月) 交換其主要區域和復原區域。

容錯移轉和復原所需的所有步驟都應保持在可供所有團隊成員使用的程序手冊中，並定期進行審查。

使用 Elastic Disaster Recovery 時，該服務會協助協調和自動化容錯恢復程序。如需詳細資訊，請參閱 Performing a failback。

實作計劃的工作量：高

資源

相關的最佳實務：

- the section called “REL09-BP01 識別並備份所有需要備份的資料，或從來源複製資料”
- the section called “REL11-BP04 在復原期間依賴資料平面而非控制平面”
- the section called “REL13-BP01 定義停機時間和資料遺失的復原目標”

相關文件：

- AWS 架構部落格：災難復原系列

- AWS 上工作負載的災難復原：在雲端中復原 (AWS 白皮書)
- 雲端中的災難復原選項
- 一小時建置無伺服器的多區域、主動-主動後端解決方案
- 多區域無伺服器後端 - 重新載入
- RDS：跨區域複寫僅供讀取複本
- Route 53：設定 DNS 備援
- S3：跨區域複寫
- 什麼是 AWS Backup？
- 什麼是 Amazon 應用程式復原控制器？
- AWS 彈性災難復原
- HashiCorp Terraform：入門 - AWS
- APN 合作夥伴：可以幫助災難復原的合作夥伴
- AWS Marketplace：可用於災難復原的產品

相關影片：

- AWS 上工作負載的災難復原
- AWS re:Invent 2018：適用於多區域主動-主動應用程式的架構模式 (ARC209-R2)
- AWS Elastic Disaster Recovery 入門 | Amazon Web Services

REL13-BP03 測試災難復原實作以驗證實作

定期測試容錯移轉到您的復原站點以確認它正常操作，並符合 RTO 和 RPO。

常見的反模式：

- 切勿在生產環境中執行容錯移轉。

建立此最佳實務的優勢：定期測試您的災難復原計畫，可驗證該計畫能在需要時運作，也能讓您的團隊知道如何執行策略。

未建立此最佳實務時的曝險等級：高

實作指引

要避免的模式是：開發鮮少執行的復原路徑。例如，您可能有一個次要資料存放區，只供唯讀查詢之用。當您寫入資料存放區而主資料存放區發生故障時，您可能需要容錯移轉到次要資料存放區。如果您不經常測試此容錯移轉，則可能會發現您對次要資料存放區的功能的假設不正確。次要資料存放區的容量 (在您上次測試時可能已經足夠) 在這種情況下可能無法再容忍負載。我們的經驗顯示，唯一能發揮功用的錯誤復原，是您經常測試的路徑。因此，最好擁有少量的復原路徑。您可建立復原模式，並定期進行測試。若擁有複雜或關鍵復原路徑，您還是需要定期在生產環境中執行該故障，說服自己該復原路徑能發揮功用。在我們剛剛討論的範例中，無論是否需要，您都應定期容錯移轉到備用資料庫。

實作步驟

1. 為復原設計您的工作負載。定期測試您的復原路徑。復原導向運算可識別系統中能增強復原能力的特性：隔離和備援，系統範圍內的回復變更能力，監控和確定運行狀態的能力，提供診斷、自動復原和模組化設計的能力，以及重新啟動的能力。練習復原路徑，以確認您可以在指定時間內完成復原到指定狀態。在復原過程中使用您的執行手冊，以記錄問題並在下一次測試前找出其解決方案。
2. 對於以 Amazon EC2 為基礎的工作負載，請使用 AWS Elastic Disaster Recovery 為 DR 策略實作和啟動演練執行個體。AWS Elastic Disaster Recovery 提供有效執行演練的功能，協助您準備容錯移轉事件。您也可以使用 Elastic Disaster Recover 頻繁啟動您的執行個體進行測試和演練，而不需要重新導向流量。

資源

相關文件：

- APN 合作夥伴：可以幫助災難復原的合作夥伴
- AWS 架構部落格：災難復原系列
- AWS Marketplace：可用於災難復原的產品
- AWS Elastic Disaster Recovery
- AWS 上工作負載的災難復原：在雲端中復原 (AWS 白皮書)
- AWS Elastic Disaster Recovery 準備容錯移轉
- 柏克萊加州大學/史丹佛大學復原導向的運算專案
- 什麼是 AWS Fault Injection Simulator？

相關影片：

- AWS re:Invent 2018：適用於多區域主動-主動應用程式的架構模式
- AWS re:Invent 2019：使用 AWS 的備份與還原和災難復原解決方案

REL13-BP04 管理 DR 站點或區域的組態偏移

若要執行成功的災難復原 (DR) 程序，您的工作負載必須能夠及時恢復正常運作，而且在 DR 環境上線後，不會發生相關的功能喪失或資料損失。若要實現此目標，請務必在 DR 環境與主要環境之間維護一致的基礎設施、資料和組態。

預期成果：您的災難復原站點的組態和資料與主要站點相同，這有助於在需要時快速且完整的復原。

常見的反模式：

- 當主要位置發生變更時，您未能更新復原位置，這會導致組態過時，進而阻礙復原工作。
- 您未考量潛在的限制，例如，主要和復原位置之間的服務差異，這可能會導致容錯移轉期間發生非預期的失敗。
- 您依賴手動程序來更新和同步 DR 環境，這會增加人為錯誤和不一致的風險。
- 您未能偵測到組態偏移，這會導致在事件發生之前對 DR 站點整備度產生錯誤的認知。

建立此最佳實務的優勢：DR 環境與主要環境之間的一致性可大幅改善事件發生後成功復原的可能性，並降低復原程序失敗的風險。

未建立此最佳實務時的曝險等級：高

實作指引

一套完整的組態管理和容錯移轉整備方法，可協助您確認 DR 站點持續更新，並準備好在主要站點故障時接管。

若要實現主要與災難復原 (DR) 環境之間的一致性，請確定您的交付管道會將應用程式同時分配到主要和 DR 網站。經過適當的評估期 (也稱為交錯部署) 之後，將變更推展至 DR 站點，以偵測主要站點的問題，並在問題擴大之前停止部署。實作監控來偵測組態偏移，並追蹤整個環境的變更和合規性。在 DR 站點中執行自動修復，使其完全一致，並準備好在事件發生時接管。

實作步驟

1. 確定 DR 區域包含成功執行 DR 計畫所需的 AWS 服務和功能。

2. 使用基礎設施即程式碼 (IaC)。保持實際執行基礎設施和應用程式組態範本的準確性，並定期將其套用至災難復原環境。AWS CloudFormation 可偵測 CloudFormation 範本所指定內容與實際部署內容之間的偏移。
3. 設定 CI/CD 管道，將應用程式和基礎設施更新部署到所有環境，包括主要和 DR 站點。CI/CD 解決方案 (例如 AWS CodePipeline) 可以自動化部署程序，進而降低組態偏移的風險。
4. 在主要環境和 DR 環境之間交錯部署。此方法允許一開始在主要環境中部署和測試更新，這樣會隔離主要站點中的問題，避免後續傳播到 DR 站點。此方法可防止同時將瑕疵推送至實際執行環境和 DR 站點，並維護 DR 環境的完整性。
5. 同時主要和 DR 環境中持續監控資源組態。AWS Config 這類解決方案可協助強制執行組態合規性並偵測偏移，這有助於在環境之間維持組態的一致性。
6. 實作警示機制，以追蹤任何組態偏移或資料複寫中斷或延遲的情形，並發出通知。
7. 自動修復偵測到的組態偏移。
8. 排程定期稽核和合規檢查，以確認主要和 DR 組態之間持續保持一致。定期審查可協助您保持符合既定規則，並識別任何需要解決的差異。
9. 檢查 AWS 佈建的容量、服務配額、限流限制是否不相符，以及組態和版本的差異。

資源

相關的最佳實務：

- REL01-BP01 了解服務配額和限制
- REL01-BP02 管理帳戶與區域之間的服務配額
- REL01-BP04 監控和管理配額
- REL13-BP03 測試災難復原實作以驗證實作

相關文件：

- 依 AWS Config 規則 修補不合規的 AWS 資源
- AWS Systems Manager Automation
- AWS CloudFormation：偵測堆疊和資源的未受管組態變更
- AWS CloudFormation：在整個 CloudFormation 堆疊上偵測偏差
- AWS Systems Manager Automation
- AWS 上工作負載的災難復原：在雲端中復原 (AWS 白皮書)
- 如何在 AWS 上實作基礎設施組態管理解決方案？

- 依 AWS Config 規則 修補不合規的 AWS 資源

相關影片：

- AWS re:Invent 2018：適用於多區域主動-主動應用程式的架構模式 (ARC209-R2)

相關範例：

- AWS CloudFormation 登錄
- 適用於 AWS 的配額監控
- 使用 Amazon CloudWatch 和 AWS Lambda 實作自動修復 AWS CloudFormation 的偏移
- AWS 架構部落格：災難復原系列
- AWS Marketplace：可用於災難復原的產品
- 自動化安全、無人為介入的部署

REL13-BP05 自動化回復

實作經測試、可靠、可觀測且可重現的自動復原機制，以減少故障的風險和業務影響。

預期成果：您已針對復原程序實作記錄完整、標準化且經過徹底測試的自動化工作流程。您的復原自動化會自動修正資料遺失或無法使用屬於低風險的次要問題。您可針對嚴重事件快速調用復原程序、觀察執行修復過程時的行為，並且在發現危險情況或故障時結束程序。

常見的反模式：

- 您依賴處於失敗或降級狀態的元件或機制做為復原計畫的一部分。
- 您的復原程序需要手動介入，例如主控台存取 (也稱為點按操作)。
- 您在資料遺失或無法使用處於高風險的情況下，自動初始化復原程序。
- 您未納入中止無法運作或會產生其他風險的復原程序的機制 (例如 Andon cord 或 大型紅色停止按鈕)。

建立此最佳實務的優勢：

- 復原操作擁有更高的可靠性、可預測性和一致性。
- 能夠實現更嚴格的復原目標，包括復原時間點目標 (RTO) 和復原點目標 (RPO)。

- 降低在事件期間復原失敗的可能性。
- 降低容易發生人為錯誤的手動復原程序伴隨的失敗風險。

未建立此最佳實務時的曝險等級：中

實作指引

若要實作自動復原程序，您需要一套使用 AWS 服務和最佳實務的全方位方法。若要開始進行，請找出工作負載中的關鍵元件和潛在的失敗點。制定可在不需要人為介入的情況下復原您的工作負載和資料的自動化程序。

使用基礎設施即程式碼 (IaC) 原則來制定復原自動化。這可讓復原環境與來源環境保持一致，並允許復原程序的版本控制。若要協調複雜的復原工作流程，請考慮 AWS Systems Manager Automations 或 AWS Step Functions 等解決方案。

自動化復原程序可提供顯著的優勢，並可協助您更輕鬆地達成復原時間點目標 (RTO) 和復原點目標 (RPO)。不過，這些程序可能會遇到非預期的情況，而導致程序失敗或本身產生新風險，例如額外的停機時間和資料遺失。為了降低此風險，請提供可快速停止進行中復原自動化的功能。停止後，您就可以調查並採取修正步驟。

對於支援的工作負載，請考慮採用 AWS 彈性災難復原 (AWS DRS) 這類解決方案來提供自動容錯移轉。AWSDRS 會持續將您的機器 (包括作業系統、系統狀態組態、資料庫、應用程式和檔案) 複寫至您的目標 AWS 帳戶 和慣用區域中的預備區域。如果事件發生，AWS DRS 會自動將複寫的伺服器轉換到 AWS 上復原區域中完整佈建的工作負載。

自動復原的維護和改善是一個持續進行的過程。根據學到的經驗持續測試和精進復原程序，並隨時掌握可增強復原功能的新 AWS 服務和功能。

實作步驟

1. 規劃自動復原
   a. 徹底檢閱工作負載架構、元件和相依性，以識別和規劃自動復原機制。將工作負載的相依性分類為硬和軟相依性。硬相依性是指工作負載無法在沒有此相依性的情況下運作，且此相依性無法替代。軟相依性是指工作負載平常使用的相依性，此相依性可使用臨時替代系統或程序取代，也可以透過正常降級處理。
   b. 建立程序以識別和復原缺少或損毀的資料。
   c. 定義在復原動作完成後，用來確認復原穩定狀態的步驟。
   d. 考慮讓復原的系統就緒可提供完整服務所需的任何動作，例如預熱和填入快取。

   e. 考慮在復原過程中可能遇到的問題，以及如何偵測和修復這些問題。
   f. 考慮無法存取主要站點及其控制平面的情況。確認復原動作可以在不依賴主要站點的情況下單獨執行。考慮採用 Amazon 應用程式復原控制器 (ARC) 這類解決方案來重新導向流量，而不需要手動改變 DNS 記錄。
2. 制定自動復原程序
   a. 實作自動故障偵測和容錯移轉機制，不需手動介入即可自動復原。建置像是 Amazon CloudWatch 中的儀表板，用來報告自動復原程序的進度和運作狀態。納入確認成功復原的程序。提供中止進行中復原程序的機制。
   b. 針對無法自動復原的故障，製作程序手冊做為備援程序，並考量您的災難復原計畫。
   c. 依照 REL13-BP03 中所述，測試復原程序。
3. 準備復原
   a. 評估復原站點的狀態，並提前部署關鍵元件。如需詳細資訊，請參閱 REL13-BP04。
   b. 定義明確的角色、責任，以及復原操作的決策流程，並且讓整個組織的利害關係人和團隊參與其中。
   c. 定義初始化復原程序的條件。
   d. 建立計畫來還原復原程序，並視需要或在安全時恢復您的主要站點。

資源

相關的最佳實務：

- REL07-BP01 取得或擴展資源時使用自動化
- REL11-BP01 監控工作負載的所有元件以偵測故障
- REL13-BP02 使用定義的復原策略來滿足復原目標
- REL13-BP03 測試災難復原實作以驗證實作
- REL13-BP04 管理 DR 站點或區域的組態偏移

相關文件：

- AWS 架構部落格：災難復原系列
- AWS 上工作負載的災難復原：在雲端中復原 (AWS 白皮書)
- 使用 Amazon Route 53 ARC 和 AWS Step Functions 協調災難復原自動化
- 使用 AWS CDK 建置 AWS Systems Manager Automation 執行手冊

- AWS Marketplace：可用於災難復原的產品
- AWS Systems Manager Automation
- AWS 彈性災難復原
- 使用彈性災難復原進行容錯移轉和容錯恢復
- AWS 彈性災難復原資源
- APN 合作夥伴：可協助進行災難復原的合作夥伴

相關影片：

- AWS re:Invent 2018：適用於多區域主動-主動應用程式的架構模式 (ARC209-R2)
- AWS re:Invent 2022: AWS On Air 搭配 AWS 彈性災難復原的 AWS 容錯恢復

結論

無論您是剛接觸可用性和可靠性主題的新人，還是尋求能最大程度地提高關鍵任務工作負載可用性的洞見之經驗豐富的資深人士，我們都希望本白皮書能挑戰的思維，提出新的想法或引入新的質疑方式。我們希望藉此幫助您根據業務需求更深入地了解正確的可用性層級，以及如何設計可靠性來達成此目標。我們鼓勵您利用此處提供的設計、操作和復原導向建議，以及 AWS 解決方案架構師的知識和經驗。我們很想聽聽您的意見，尤其是您在 上達到高水準可用性的成功案例 AWS。請聯絡您的客戶團隊或使用我們網站上的「聯絡我們」。

貢獻者

本文件的貢獻者包括：

- Michael Fischer，Amazon Web Services 首席解決方案架構師
- Seth Eliot，Amazon Web Services 首席開發人員支援主管
- Mahanth Jayadeva，Amazon Web Services Well-Architected 解決方案架構師
- Amulya Sharma，Amazon Web Services 首席解決方案架構師
- Jason DiDomenico，Amazon Web Services Cloud Foundations 資深解決方案架構師
- Marcin Bednarz，Amazon Web Services 首席解決方案架構師
- Tyler Applebaum，Amazon Web Services 資深解決方案架構師
- Rodney Lester，Amazon Web Services 首席解決方案架構師
- Joe Chapman，Amazon Web Services 資深解決方案架構師
- Adrian Hornsby，Amazon Web Services 首席系統開發工程師
- Kevin Miller，Amazon Web Services S3 副總裁
- Shannon Richards，Amazon Web Services 首席技術計畫經理
- Laurent Domb，Amazon Web Services Fed Fin 首席技術專家
- Kevin Schwarz，Amazon Web Services 資深解決方案架構師
- Rob Martell，Amazon Web Services 首席雲端恢復能力架構師
- Priyam Reddy，Amazon Web Services 資深解決方案架構師經理 DR
- Jeff Ferris，Amazon Web Services 首席技術專家
- Matias Battaglia，Amazon Web Services 資深解決方案架構師

深入閱讀

如需其他資訊，請參閱：

- AWS 建構良好的架構
- AWS 架構中心

文件修訂

若要收到此白皮書更新的通知，請訂閱 RSS 摘要。

變更	描述	日期
已更新最佳實務指引	最佳實務更新中增加了下列領域的新指引：REL 1、REL 2、REL 4、REL 6、REL 7、REL 8、REL 10、REL 12 和 REL 13。已在整個支柱中擴充和澄清指引。REL10-BP02 和 REL12-BP03 已將其指引合併到其他最佳實務中。已更新整個支柱的資源。	2024 年 11 月 6 日
已更新最佳實務指引	對 REL 2、4、5、6、7 和 8 的最佳做法進行小幅更新。	2024 年 6 月 27 日
已更新最佳實務指引	最佳實務已更新，納入了下列方面的新指引：在分散式系統中設計防止失敗的互動、設計分散式系統中的互動以緩解或承受失敗、監控工作負載資源、設計工作負載以適應需求變更、實作變更和測試可靠性。	2023 年 12 月 6 日
已更新最佳實務指引	最佳實務已更新，納入了下列方面的新指引：監控工作負載資源和設計工作負載以承受元件失敗。	2023 年 10 月 3 日
已更新最佳實務指引	最佳實務已更新，納入了下列方面的新指引：設計您的工作負載服務架構、設計分散式系	2023 年 7 月 13 日

	統中的互動以緩解或承受失敗和監控工作負載資源。	
次要更新	移除非包容性語言。	2023 年 4 月 13 日
新框架的更新	最佳實務已更新，納入了規範性指引，並增加了新的最佳實務。	2023 年 4 月 10 日
白皮書已更新	最佳實務更新了新的實作指引。	2022 年 12 月 15 日
次要更新	更正了圖片編號號和整體小幅變更。	2022 年 11 月 17 日
白皮書已更新	已擴充最佳實務並新增了改善計畫。	2022 年 10 月 20 日
白皮書已更新	已將兩個新的最佳實務新增至可靠性支柱的以下章節：使用故障隔離來保護您的工作負載和設計工作負載以承受元件失敗。	2022 年 5 月 5 日
白皮書已更新	更新災難復原指引以包括 Route 53 應用程式復原控制器。將參考新增至 DevOps Guru。更新數個資源連結，以及其他小幅度編輯變更。	2021 年 10 月 26 日
次要更新	已新增有關 AWS Fault Injection Service (AWS FIS) 的資訊。	2021 年 3 月 15 日
次要更新	小幅度文字更新。	2021 年 1 月 4 日

白皮書已更新	已更新附錄 A 來更新 Amazon SQS、Amazon SNS 和 Amazon MQ 的可用性設計目標；重新排序資料表中的資料列以便於更輕鬆的查詢；改善可用性與災難復原之間差異的解釋，以及它們兩者如何促進彈性；擴大多區域架構 (適用於可用性) 和多區域策略 (適用於災難復原) 的覆蓋範圍；將參考書更新到最新版本；擴展可用性計算以包括請求型計算和捷徑計算；改善演練日的描述	2020 年 12 月 7 日
次要更新	已更新附錄 A 來更新 AWS Lambda 的可用性設計目標	2020 年 10 月 27 日
次要更新	已更新附錄 A 來新增 AWS Global Accelerator 的可用性設計目標	2020 年 7 月 24 日

新框架的更新	重大更新和新/修訂內容，包括：新增「工作負載架構」最佳實務章節、將最佳實務重新組織到「變更管理」和「故障管理」章節、更新了「資源」章節、更新以納入最新的 AWS 資源和服務，例如 AWS Global Accelerator、AWS Service Quotas 和 AWS Transit Gateway，新增/更新了可靠性、可用性、彈性的定義，讓白皮書與用於 Well-Architected 審查的 AWS Well-Architected Tool (問題和最佳實務) 更保持一致，重新排序設計原則，將自動從故障中復原移到測試復原程序之前，更新了方程式的圖表和格式，移除了「重要服務」章節，並將重要 AWS 服務的參考整合到了最佳實務中。	2020 年 7 月 8 日
次要更新	修正了中斷的連結	2019 年 10 月 1 日
白皮書已更新	更新了附錄 A	2019 年 4 月 1 日
白皮書已更新	新增了特定的 AWS Direct Connect 聯網建議和其他服務設計目標	2018 年 9 月 1 日
白皮書已更新	新增了「設計原則」和「限制管理」章節。更新了連結，消除了上游/下游術語的歧義，並在可用性情境中新增了對其餘「可靠性支柱」主題的明確引用。	2018 年 6 月 1 日

白皮書已更新	將 DynamoDB 跨區域解決方案變更為了 DynamoDB 全域資料表。新增了服務設計目標	2018 年 3 月 1 日
次要更新	對可用性計算的細微糾正，以包括應用程式可用性	2017 年 12 月 1 日
白皮書已更新	更新以提供有關高可用性設計的指南，包括概念、最佳實務和範例實作。	2017 年 11 月 1 日
初次出版	可靠性支柱 – AWS Well Architected Framework 已發布。	2016 年 11 月 1 日

注意

客戶有責任對本文件中的資訊進行自己的獨立評定。本文件：（a）僅供參考，（b）代表目前的 AWS 產品產品和實務，這些產品和實務可能隨時變更，恕不另行通知，且（c）不會從 AWS 及其附屬公司、供應商或授權方提供「原樣」的任何承諾 AWS 或保證，而沒有任何明示或暗示的保證、陳述或條件。 AWS 對其客戶的責任和責任受 AWS 協議控制，本文件不屬於 與其 AWS 客戶之間的任何協議，也未對其進行修改。

© 2023 Amazon Web Services, Inc. 或其附屬公司。保留所有權利。

AWS 詞彙表

如需最新的 AWS 術語，請參閱AWS 詞彙表 參考 中的AWS 詞彙表。