AWS Well-Architected Framework

# 可靠性支柱

# 可靠性支柱: AWS Well-Architected Framework

# Table of Contents

# 可靠性支柱 - AWS Well-Architected Framework。

出版日期： 2024 年 6 月 27 日 (文件修訂)

本白皮書的重點是 AWS Well-Architected Framework。本文提供了相關指導，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

此 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，並找出需要改進的方面。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：

- 卓越營運
- 安全性
- 可靠性
- 效能達成效率
- 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，提供最佳實務指導。

主題



## 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。此基礎設施包含硬體、軟體、網路和執行 AWS 雲端服務的設施。AWS 會使用商業上合理的工作量讓這些 AWS 雲端 服務可供使用，確保服務可用性符合或超過 AWS 服務水準協議 (SLA)。

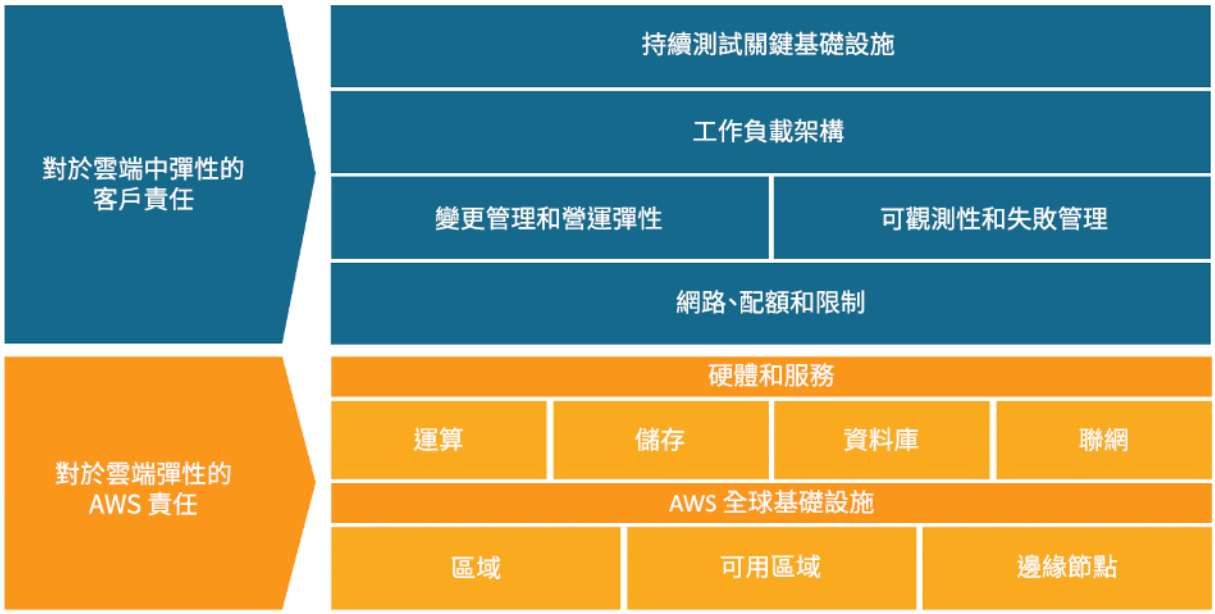
AWS 全球雲端基礎設施的設計目的是讓客戶可以建置具有高度彈性的工作負載架構。每個 AWS 區域都完全隔離並且包含多個可用區域，是基礎設施的實體隔離分割區。可用區域會隔離可能影響工作負載彈性的故障，防止這些故障影響區域中的其他區域。但是於此同時 AWS 區域中的所有區域都是使用完全備援的專用都會光纖 (在區域之間提供高輸送量、低延遲網路)，搭配高頻寬、低延遲網路來互連。區域之間的所有流量都會加密。網路效能足以完成區域之間的同步複寫。應用程式跨 AZ 分割時，公司可以獲得更好的隔離和保護，讓您免於停電、雷擊、龍捲風、颶風等問題。

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、操作系統和平台，而且客戶會存取端點以存放和擷取資料。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。如需特定各個服務的詳細資訊，請參閱 AWS 文件。

網路、配額和限制

- 此區域的共同責任模型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觸發自動化。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 Service Quotas 和限制) 建立持續整合/持續部署 (CI/CD) 管道。
- 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运。
- 安全性，包括防止惡意動作項目損毀資料或基礎架构，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，包括实现工作負載最大請求率和最小延遲的設計。
- 成本優化，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。

其他四個層面也很重要，並都有各自的 AWS Well-Architected 架構支柱白皮書說明。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：

$$可用性 = \frac{可供使用時間}{時間總計}$$

- 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：

$$可用性 = \frac{成功的回應}{有效要求}$$

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叫用系統的中斷。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，叫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：

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

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%：


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

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南，包括許多 AWS 服務的可用性設計目標 建立持續整合/持續部署 (CI/CD) 管道。但是，在無法使用此功能的情況下 (例如，製造商未發佈可用性訊息的元件)，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：

$$可用_{EST} = \frac{MTBF}{MTBF + MTTR}$$

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，請參閱可用性和超越：了解和改善 AWS 上分散式系統的彈性，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

雲端型工作負載架構會有服務配額 (也稱為服務限制)。這些配額旨在用於防止不慎佈建超過您所需的資源，並限制 API 操作上的請求率，以防止服務遭到濫用。此外也會有資源限制，例如，您可將位元壓入光纖電纜的速率或實體磁碟上的儲存量會受到限制。

若您使用 AWS Marketplace 應用程式，則必須了解這些應用程式的限制。若您使用第三方 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

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。

預期成果：客戶可實作適當的指導方針來監控關鍵指標、基礎設施審查和自動矯正步驟，確認未達到可能導致服務降級或中斷的服務配額與限制，藉以防止其 AWS 帳戶 中的服務降級或中斷。

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。

建立此最佳實務的效益：監控及自動管理服務配額和資源限制，可主動減少失敗的狀況。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劃性事件時將會有更高的彈性。

未建立此最佳實務時的風險暴露等級：高

## 實作指引

Service Quotas 是一項 AWS 服務，可協助您從單一位置管理超過 250 個 AWS 服務的配額。除了查閱配額值外，您也可以從 Service Quotas 主控台或使用 AWS SDK 要求和追蹤配額增長。AWS Trusted Advisor 提供服務配額檢查功能，會顯示部分服務某些層面的用量和配額。每項服務的預設服務配額也根據各自服務列於 AWS 文件中 (如需範例，請參閱 Amazon VPC 配額)。

某些服務限制 (例如用於調節 API 的速率限制) 可藉由設定用量計畫在 Amazon API Gateway 中設定。在其各自服務上設為組態的某些限制包括佈建 IOPS、分配的 Amazon RDS 儲存體，以及 Amazon EBS 磁碟區分配。Amazon Elastic Compute Cloud 具有專門的 Service Limits 儀表板，有助於您管理執行個體、Amazon Elastic Block Store 和彈性 IP 地址限制。如果在您的使用案例中，服務配額會影響您的應用程式效能且無法根據您的需求調整，請聯絡 AWS Support 以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。使用達到配額的 AWS 服務，將無法作為預期的正常用法，且可能導致服務中斷或降級。例如，服務配額會限制在區域中使用的 DL Amazon EC2 數目，而該限制可能會在使用 Auto Scaling 群組 (ASG) 的流量擴展事件期間達到。

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。此外也可用來限制 API 操作的要求率，以保護服務免於濫用。

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。這有可能是儲存容量 (例如，gp2 的大小限制為 1 GB - 16 TB) 或磁碟輸送量 (10,0000 iops)。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。

如果在某個使用案例中，服務配額會影響到應用程式的效能，且無法根據需求進行調整，請聯絡 AWS Support 以查看是否有緩解措施。如需關於調整固定配額的詳細資料，請參閱 REL01-BP03 透過架構適應固定服務配額和限制。

有許多 AWS 服務和工具可協助您監控及管理 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 提供服務配額檢查功能，會顯示部分服務某些層面的用量和配額。這有助於識別接近配額的服務。
- AWS Management Console 提供了相關方法，用以顯示服務配額值、管理及要求新配額、監控配額要求的狀態，以及顯示配額的歷史。
- AWS CLI 和 CDK 提供了程式化方法，可自動管理及監控服務配額層級與用量。

實作步驟

針對 Service Quotas：

- 審查 AWS Service Quotas。
- 若要得知現有的服務配額，請確認使用的服務為何 (例如 IAM Access Analyzer)。約有 250 個 AWS 服務受到服務配額控制。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 AWS Config 擴大此配額分析，以尋找在您的 AWS 帳戶 中使用的所有 AWS 資源。
- 使用 AWS CloudFormation 資料來確認您使用的 AWS 資源。查看在 AWS Management Console 中或透過 `list-stack-resources` AWS CLI 命令建立的資源。您也可以查看設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。
- 決定適用的服務配額。從 Trusted Advisor 和 Service Quotas 使用可以程式設計方式存取的資訊。
- 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)，以在服務配額接近或已達限制時發出提醒和通知。
- 建立自動化和程式化方法，以檢查服務配額是否在相同帳戶中的某個區域有所變更，但在其他區域並未變更 (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。
- 建立工程程序，以在發現特定服務需要更大的配額時計算配額的必要變更 (請參閱 REL01-BP05 自動配額管理)。
- 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之前審查服務配額，並在推行至生產環境之前使用新的 AWS 服務。(例如，載入測試帳戶)。

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。適當利用 CloudWatch 進行指標或日誌監控。
- 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 自動配額管理
- REL01-BP06 確保目前配額與最大使用量之間存在足夠差距以適應容錯移轉
- REL03-BP01 選擇如何劃分工作負載

- 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-BP05 使用混沌工程測試彈性

相關文件：

- AWS Well-Architected Framework 的可靠性要素：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 一節)
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 Service Limits
- 什麼是 Service Quotas？
- 如何要求增加配額
- 服務端點和配額
- Service Quotas 使用者指南
- AWS 的配額監視器
- AWS 故障隔離界限
- 可用性與備援性
- 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
- 在 AWS 上管理每租用戶一帳戶 SaaS 環境中的帳戶生命週期
- 管理和監控工作負載中的 API 調節
- 使用 AWS Organizations 大規模檢視 AWS Trusted Advisor 建議
- 使用 AWS Control Tower 自動執行服務限制增加和企業支援

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

相關工具：

- Amazon CodeGuru Reviewer
- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- 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：在跨帳戶或區域的組態，或有彈性設計使用了區域或帳戶容錯移轉的組態中，服務和應用程式應該不受服務配額用盡的影響。

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。
- 未利用配額要求範本在多個區域和帳戶間要求增加配額。
- 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。

建立此最佳實務的效益：確認在區域服務無法使用時，您可以在次要區域或帳戶中處理目前的負載。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風險暴露等級：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 特有的。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)。

由於實作使用主動/主動、主動/被動 – 熱、主動/被動 - 冷和主動/被動 - 指示燈等方法的設計，產生了更多跨區域的工作負載，請務必了解所有區域和帳戶的配額層級。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劃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，這是指某個指定配額的服務配額限制在某個區域中已變更，但未在所有區域變更的情況，對於追蹤和評估而言非常重要。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 自動配額管理
- REL01-BP06 確保目前配額與最大使用量之間存在足夠差距以適應容錯移轉
- REL03-BP01 選擇如何劃分工作負載
- 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-BP05 使用混沌工程測試彈性

相關文件：

- AWS Well-Architected Framework 的可靠性要素：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 一節)
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 Service Limits
- 什麼是 Service Quotas？
- 如何要求增加配額
- 服務端點和配額
- Service Quotas 使用者指南
- AWS 的配額監視器
- AWS 故障隔離界限
- 可用性與備援性
- 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
- 在 AWS 上管理每租用戶一帳戶 SaaS 環境中的帳戶生命週期
- 管理和監控工作負載中的 API 調節
- 使用 AWS Organizations 大規模檢視 AWS Trusted Advisor 建議
- 使用 AWS Control Tower 自動執行服務限制增加和企業支援

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

相關服務：

- Amazon CodeGuru Reviewer
- AWS CodeDeploy
- AWS CloudTrail
- Amazon CloudWatch
- Amazon EventBridge
- Amazon DevOps Guru
- 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：應用程式或服務會在正常或高流量條件之下如預期般執行。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臨界值

建立此最佳實務的優勢：確認應用程式會在所有預估服務載入層級之下運作，沒有中斷或降級。

未建立此最佳實務時的風險暴露等級：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。如果欄顯示 ### = #服務有硬式限制。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

- 檢閱可用於應用程式設計階段的 AWS 服務完整清單。
- 檢閱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。一些服務在替代位置中說明這些限制。

- 隨著您設計您的應用程式，檢閱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 自動配額管理
- REL01-BP06 確保目前配額與最大使用量之間存在足夠差距以適應容錯移轉
- REL03-BP01 選擇如何劃分工作負載
- 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-BP05 使用混沌工程測試彈性

相關文件：

- AWS Well-Architected Framework 的可靠性要素：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 一節)
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 Service Limits
- 什麼是 Service Quotas？
- 如何要求增加配額
- 服務端點和配額

- Service Quotas 使用者指南
- AWS 的配額監視器
- AWS 故障隔離界限
- 可用性與備援性
- 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
- 在 AWS 上管理每租用戶一帳戶 SaaS 環境中的帳戶生命週期
- 管理和監控工作負載中的 API 調節
- 使用 AWS Organizations 大規模檢視 AWS Trusted Advisor 建議
- 使用 AWS Control Tower 自動執行服務限制增加和企業支援
- 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臨界值。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臨界值
- 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。
- 只設定 AWS Service Quotas 支援的服務警示，但未監控其他 AWS 服務。
- 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。
- 未利用多區域配額管理的範本。

建立此最佳實務的優勢：自動追蹤 AWS Service Quotas 並根據這些配額監控您的使用量，可讓您查看何時會接近配額限制。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風險暴露等級：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、Amazon CloudWatch 或從 AWS Trusted Advisor 觸發。您也可以使用 CloudWatch 日誌上的指標篩選條件，搜尋與擷取日誌中的模式，以判斷用量是否正在接近配額臨界值。

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，用以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，這是一項 AWS 服務，有助於您從單一位置管理超過 250 種 AWS 服務的配額。
- 使用 Trusted Advisor 服務限制以不同臨界值來監控您目前的服務限制。
- 使用服務配額歷史 (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 自訂規則來掃描區域之間的服務配額，並且比較是否有差異。
- 自動化：設定排定 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，掃描區域之間的服務配額，並且比較是否有差異。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 自動配額管理
- REL01-BP06 確保目前配額與最大使用量之間存在足夠差距以適應容錯移轉
- REL03-BP01 選擇如何劃分工作負載
- 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-BP05 使用混沌工程測試彈性

相關文件：

- AWS Well-Architected Framework 的可靠性要素：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 一節)
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 Service Limits
- 什麼是 Service Quotas？
- 如何要求增加配額
- 服務端點和配額
- Service Quotas 使用者指南
- AWS 的配額監視器
- AWS 故障隔離界限
- 可用性與備援性
- 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
- 在 AWS 上管理每租用戶一帳戶 SaaS 環境中的帳戶生命週期
- 管理和監控工作負載中的 API 調節
- 使用 AWS Organizations 大規模檢視 AWS Trusted Advisor 建議
- 使用 AWS Control Tower 自動執行服務限制增加和企業支援

- 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-BP05 自動配額管理

實作工具以在接近閾值時獲得提醒。您可以使用 AWS Service Quotas API，自動化配額增加請求。您可以自動化配額增加請求。

如果您將組態管理資料庫 (CMDB) 或票務系統與 Service Quotas 整合，則可以自動追蹤配額增加請求和目前的配額。除了 AWS 開發套件外，Service Quotas 也會使用 AWS Command Line Interface (AWS CLI) 提供自動化。

常用的反模式：

- 以試算表追蹤配額和使用量。

- 每日、每週或每月執行使用量報告，然後比較使用量與配額。

建立此最佳實務的優勢： 自動追蹤 AWS 服務配額並根據該配額監控您的使用量，可讓您查看何時會接近配額限制。您可以設定自動化，協助您在需要時請求增加配額。當您的使用量與實現風險降低 (登入資料遭危害時) 和成本節省的優勢背道而馳時，您可能會考慮降低部分配額。

若未建立此最佳實務，暴露的風險等級為： 中

## 實作指引

- 設定自動監控：使用開發套件實作工具，以在接近閾值時獲得提醒。
  - 使用 Service Quotas，並以如 AWS Limit Monitor 或 AWS Marketplace 中的產品等自動配額監控解決方案擴大此項服務。
    - 什麼是 Service Quotas？
    - AWS 上的配額監視器 - AWS 解決方案
  - 使用 Amazon SNS 和 AWS Service Quotas API 設定由配額閾值觸發的回應。
  - 測試自動化。
    - 設定限制閾值。
    - 與來自 AWS Config、部署管道、Amazon EventBridge 或第三方的變更事件整合。
    - 人工設定較低配額閾值以測試回應。
    - 設定觸發程序以在收到通知時採取適當的措施，以及在必要時讓人員聯絡 AWS Support。
    - 手動觸發變更事件。
    - 執行演練日以測試配額增長變更程序。

## 資源

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
- AWSService Quotas (先前稱為 Service Limits)
- AWS Trusted Advisor 最佳實務檢查 (請參閱 Service Limits 一節)
- AWS 上的配額監視器 - AWS 解決方案
- Amazon EC2 Service Limits

- 什麼是 Service Quotas？

相關影片：

- AWS Live re:Inforce 2019 - Service Quotas

## REL01-BP06 確保目前配額與最大使用量之間存在足夠差距以適應容錯移轉

資源失敗或無法存取時，在該資源成功終止之前，可能仍會被計入配額。確認您的配額涵蓋失敗或無法存取資源及其替換項目的重疊。計算此差距時，您應該考慮使用像是網路失敗、可用網路失敗或區域失敗的使用案例。

預期成果：資源或資源可存取性中的小型或大型失敗可以涵蓋在目前的服務臨界值內。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。
- 未針對某些服務及其潛在異常用量模式考慮 AWS 服務故障隔離界限。

建立此最佳實務的優勢：服務中斷事件影響應用程式可用性時，雲端可讓您實作策略來緩解或從這些事件中復原。這類策略通常包括建立額外資源以取代失敗或無法存取的資源。您的配額策略適用於這些容錯移轉條件，不會由於服務限制耗盡而導致額外降級。

未建立此最佳實務時的風險暴露等級：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。應該考慮下列類型的容錯移轉案例：

- 中斷或無法存取的 VPC。
- 無法存取的子網路。
- 可用區域的降級程度已足夠影響許多資源的可存取性。
- 各個網路路由或輸入和輸出點遭到封鎖或變更。

- 區域的降級程度已足夠影響許多資源的可存取性。
- 有多個資源，但是並非所有資源都受到區域或可用區域中的失敗影響。

如上所列的失敗會觸發以啟動容錯移轉事件。對每個情境和客戶進行容錯移轉的決策都是唯一的，因為業務影響差距甚大。不過，在操作方面決定容錯移轉應用程式或服務時，容錯移轉位置中資源的容量規劃及其相關配額都必須在事件之前解決。

檢閱每個服務的服務配額，考慮高於可能發生的正常尖峰。由於網路或許可，這些尖峰可能與可以連線的資源相關，但是仍然是作用中。未終止的作用中資源仍然會計入服務配額限制。

實作步驟

- 確認您的服務配額和最大用量之間存在足夠的差距以適應容錯移轉若遺失可存取性。
- 確定服務限制，並在此過程中考慮您的部署模式、可用性要求和使用量增長。
- 視需要請求增加配額。規劃必要的時間來滿足增加配額的請求。
- 確定您的可靠性方案 (也稱為「幾個 9」)。
- 建立故障案例 (例如，元件、可用區域或區域遺失)。
- 建立您的部署方法 (例如，Canary、藍/綠、紅/黑或滾動)。
- 為當前限制新增適當的緩衝 (例如 15%)。
- 適當時包含靜態穩定性的計算 (區域 (Zonal) 和區域 (Regional))。
- 為使用量增長制定計畫 (例如，監控使用量趨勢)。
- 考慮您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。這在最關鍵業務排程期間是有用的策略，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 自動配額管理

- REL03-BP01 選擇如何劃分工作負載
- 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-BP05 使用混沌工程測試彈性

相關文件：

- AWS Well-Architected Framework 的可靠性要素：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 一節)
- AWS Answers 上的 AWS Limit Monitor
- Amazon EC2 Service Limits
- 什麼是 Service Quotas？
- 如何要求增加配額
- 服務端點和配額
- Service Quotas 使用者指南
- AWS 的配額監視器
- AWS 故障隔離界限
- 可用性與備援性
- 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
- 在 AWS 上管理每租用戶一帳戶 SaaS 環境中的帳戶生命週期
- 管理和監控工作負載中的 API 調節
- 使用 AWS Organizations 大規模檢視 AWS Trusted Advisor 建議
- 使用 AWS Control Tower 自動執行服務限制增加和企業支援
- 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

## 規劃您的網路拓撲

工作負載經常存在於多個環境中。這些環境包括多個 (可公開存取和私有的) 雲端環境，且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 地址管理、私有 IP 地址管理和網域名稱解析等網路考量因素。

使用基於 IP 地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，以便在虛擬網路中啟動 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 在雲端和內部部署環境中的私人網路之間佈建備援連線
- REL02-BP03 確保 IP 子網路分配帳戶具有擴展性和可用性

- REL02-BP04 偏好軸輻式拓撲而非多對多網狀拓撲
- REL02-BP05 在連線的所有私有地址空間中強制使用不重疊的私有 IP 位址範圍

## 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：為您的公有端點規劃、建置和操作高度可用網路連線相當重要。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 函數 URL、AWS AppSync API 和 Elastic Load Balancing (ELB) 都提供高度可用公有端點。Amazon Route 53 針對網域名稱解析提供高度可用 DNS 服務，確認可以解析您的公有端點地址。

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 地址，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：設計高度可用且具彈性的網路連線，可確保您的工作負載可存取並且可供您的使用者使用。

未建立此最佳實務時的風險暴露等級：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 地址。使用高度可用和可擴展網域名稱系統 (DNS)，例如 Amazon Route 53，來管理您的網域的 DNS 記錄。您也可以使用 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是高度可用，請使用 Elastic Load Balancing (ELB)。Amazon Route 53 可以用來將流量目標設定為 ELB，這會將流量分發到目標運算執行個體。您也可以使用 Amazon API Gateway 以及 AWS Lambda 做為無伺服器解決方案。客戶也可以在多個 AWS 區域 中執行工作負載。使用多站點主動/主動模式，工作負載可以為來自多個區域的流量提供服務。使用多站點主動/被動模式，工作負載可以為來自主動區域的流量提供服務，而資料會複寫到次要區域並且在主要區域失敗的事件中變成作用中。Route 53 運作狀態檢查接著可以用來控制 DNS 備援從主要區域中的任何端點容錯移轉到次要區域中的端點，確認您的工作負載可連線且可供您的使用者使用。

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。這也會改善您的應用程式的可用性，因為內容的負載從您的伺服器轉移到 CloudFront 的邊緣節點。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 地址，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會觸發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。

為了協助客戶防範 DDoS 攻擊，AWS 提供 AWS Shield Standard。Shield Standard 會自動啟用並且防禦常見基礎設施 (第 3 層和第 4 層) 攻擊，例如 SYN/UDP 泛洪和反射攻擊，在 AWS 上支援您的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，並且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 和 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 地址、HTTP 標題、HTTP 本文、URI 字串、SQL 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. 設定高度可用 DNS：Amazon Route 53 是可度可用且可擴展網域名稱系統 (DNS) Web 服務。Route 53 會將使用者請求連線到 AWS 上或內部部署執行的網際網路應用程式。如需詳細資訊，請參閱將 Amazon Route 53 設定為您的 DNS 服務。
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。從建立 Route 53 運作狀態檢查和設定 DNS 備援開始。以下是設定運作狀態檢查時要考慮的重要層面：
   a. Amazon Route 53 如何判斷運作狀態檢查是運作狀態良好
   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 服務連線到您的端點。
   a. 當使用 Elastic Load Balancing 做為您流量的目標時，使用指向您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. 針對使用 API Gateway 時的無伺服器工作負載或私有 API，使用 Route 53 將流量指向 API Gateway。
4. 決定內容交付網路。
   a. 針對使用更靠近使用者的邊緣節點交付內容，從了解 CloudFront 如何交付內容開始。
   b. 從簡易 CloudFront 分發開始。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. 增加直接從 CloudFront 快取 (快取命中率) 提供服務的請求比例
      iii. 使用 Amazon CloudFront Origin Shield
      iv. 使用 CloudFront 來源容錯移轉最佳化高可用性
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。若要更深入了解，請參閱 AWS WAF 如何運作，當您準備好實作應用程式層 HTTP POST AND GET 泛洪的保護，請參閱開始使用 AWS WAF。您也可以搭配使用 AWS WAF 與 CloudFront，請參閱 AWS WAF 如何使用 Amazon CloudFront 功能上的文件。
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。針

對在 Amazon EC2、Elastic Load Balancing、Amazon CloudFront、AWS Global Accelerator 和 Amazon Route 53 上執行，面向網際網路應用程式的額外保護，您可以考慮 AWS Shield Advanced 和檢閱 DDoS 彈性架構的範例。若要保護您的工作負載和公有端點免於 DDoS 攻擊，請參閱開始使用 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
- REL10-BP02 為您的多位置部署選取適當位置
- REL11-BP04 復原期間需使用資料平面，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？
- 網路連線功能 - 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 Route 53 應用程式復原控制器？
- 為 DNS 備援設定自訂運作狀態檢查

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 網路基礎設施
- AWS re:Invent 2022 - 建置彈性網路

相關範例：

- 使用 Amazon Route 53 應用程式復原控制器 (ARC) 進行災難復原
- 可靠性研討會
- AWS Global Accelerator 研討會

## REL02-BP02 在雲端和內部部署環境中的私人網路之間佈建備援連線

在雲端和內部部署環境中的私人網路之間執行備援連線，以強化連線恢復能力。這目標可以藉由部署兩個或多個連結和流量路徑來實現，並可在網路故障時保持連線通暢。

常見的反模式：

- 您只依賴一種網路連線，這麼做會產生單一故障點。
- 您只使用一個 VPN 通道，或多個以相同可用區域結束的通道。
- 您依賴一個 ISP 進行 VPN 連線，如此可能導致 ISP 中斷期間全面性故障。
- 未實作 BGP 這類的動態路由通訊協定，而該協定對於網路中斷期間重新路由流量卻極為重要。
- 您忽略 VPN 通道的頻寬限制，且高估了該通道的備份功能。

建立此最佳實務的好處：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風險暴露等級：高

### 實作指引

使用 AWS Direct Connect 將內部部署網路連線至 AWS 時，您可以使用在一個以上的內部部署位置，以及在一個以上的 AWS Direct Connect 位置中的不同裝置結束的獨立連線，以藉此達到最大的網路恢復能力 (SLA 99.99%)。此拓樸提供了針對裝置故障、連線問題和完整位置中斷的恢復能力。或者，您也可以透過使用兩個個別連線到多個位置 (每個內部部署位置都連接單一 Direct Connect 位置) 獲得強大的恢復能力 (SLA 99.9%)。這種方法可防止由光纖切斷或裝置故障引起的連線中斷，並有助於減輕全面性的位置故障。AWS Direct Connect 恢復能力工具組可協助您設計 AWS Direct Connect 拓樸。

您也可以考慮將 AWS Site-to-Site VPN 在 AWS Transit Gateway 結束當成主要 AWS Direct Connect 連線的具成本效益備份方式。此設定可跨多個 VPN 通道啟用同等成本多重路徑 (ECMP) 路由，即使每個 VPN 通道的上限為 1.25 Gbps，也能達到最高 50Gbps 的輸送量。然而，請務必留意 AWS Direct Connect 仍然是最有效的選擇，因為它可以將網絡中斷機率降到最低，並提供穩定的連線能力。

透過網際網路使用 VPN 將雲端環境連接到內部部署資料中心時，請將兩個 VPN 通道設定為單一站點對站點 VPN 連線的一部分。每個通道應在不同的可用區域終止，如此才能獲得高可用性，並使用備援硬體以防止內部部署裝置故障。此外，請考慮從內部部署位置的不同網際網路服務供應商 (ISP) 提供多個網際網路連線，以避免因單一 ISP 中斷而導致 VPN 連線完全中斷。選擇具有多樣性路由和基礎設施的 ISP，尤其是具有獨立實體路徑到 AWS 端點的 ISP，這樣即能獲得高連線可用性。

除了具有多個 AWS Direct Connect 連線和多個 VPN 通道 (或兩者的組合) 的實體備援外，實作邊界閘道協定 (BGP) 動態路由也很重要。動態 BGP 會根據即時網路條件和設定的政策，從一個路徑的流量自動重新路由到另一個路徑。在發生連結或網路故障事件時，這種動態行為對於維持網路可用性和服務連續性特別有幫助。此動態行為能快速選擇替代路徑，提高網路的恢復能力和可靠性。

實作步驟

- 在 AWS 和您的內部部署環境之間，獲得高度可用的連線。
  - 在單獨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 位置以實現高可用性。
  - 如果使用多個 AWS 區域，請至少在其中兩個區域中確立備援。
- 在可能的情況下使用 AWS Transit Gateway 終止您的 VPN 連線。
- 評估 AWS Marketplace 設備以結束 VPN，或將您的 SD-WAN 擴展至 AWS。如果您使用 AWS Marketplace 設備，可在不同的可用區域中部署冗餘執行個體以實現高可用性。
- 在您的內部部署環境提供備援連線。
  - 您可能需要與多個 AWS 區域 進行備援連線才能滿足可用性需求。
  - 使用AWS Direct Connect 恢復能力工具組以開始使用。

## 資源

相關文件：

- AWS Direct Connect 恢復能力建議
- 使用備援 Site-to-Site VPN 連線以提供容錯移轉
- 路由政策和 BGP 社群

- AWS Direct Connect 中的主動/主動和主動/被動組態
- 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
- Amazon Virtual Private Cloud 連線能力選項白皮書
- 建置可擴展且安全的多 VPC AWS 網路基礎設施
- 使用備援 Site-to-Site VPN 連線以提供容錯移轉
- 使用 AWS Direct Connect 恢復能力工具組以開始使用
- VPC 端點和 VPC 端點服務 (AWS PrivateLink)
- 什麼是 Amazon VPC？
- 什麼是 Transit Gateway？
- 什麼是 AWS Site-to-Site VPN？
- 使用 Direct Connect 閘道

相關影片：

- AWS re:Invent 2018：進階 VPC 設計和 Amazon VPC 的新功能
- AWS re:Invent 2019：適用於許多 VPC 的 AWS Transit Gateway 參考架構

## REL02-BP03 確保 IP 子網路分配帳戶具有擴展性和可用性

Amazon VPC IP 位址範圍必須足夠大，以適應工作負載的要求，包括考慮將來擴展 IP 位址以及跨可用區域將 IP 位址分配給子網路。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 地址空間。私有 IP 地址範圍 (依循 RFC 1918 指導方針) 應分配給各 VPC。在此流程中請滿足下列要求：

- 允許每個區域為多於一個 VPC 準備 IP 位址空間。
- 在 VPC 內，允許多個子網路的空間，讓您可以跨越多個可用區域。
- 在 VPC 內留下未用 CIDR 區塊空間，以供未來擴展。
- 確保有 IP 位址空間可以滿足您可能會用到之 Amazon EC2 執行個體的任何臨時機群需求，例如，用於機器學習的 Spot Fleets、Amazon EMR 叢集或 Amazon Redshift 叢集。Kubernetes 叢集 (例如 Amazon Elastic Kubernetes Service (Amazon EKS)) 應該考慮類似條件，因為每個 Kubernetes Pod 都會依照預設獲得 VPC CIDR 區塊指派的可路由位址。

- 請注意，在各子網路 CIDR 區塊中，前四個 IP 位址和最後一個 IP 位址均預留起來，無法供您使用。
- 請注意，雖然無法變更或刪除分配給 VPC 的最初 VPC CIDR 區塊，但您可以將不重疊的其他 CIDR 區塊新增至 VPC。無法變更子網路 IPv4 CIDR，但可以變更 IPv6 CIDR。
- 最大可能的 VPC CIDR 區塊是 /16，最小的是 /28。
- 考慮其他連線的網路 (VPC、內部部署或其他雲端供應商)，並確保 IP 位址空間未重疊。如需詳細資訊，請參閱 REL02-BP05 在連線的所有私有地址空間中強制使用不重疊的私有 IP 位址範圍。

預期成果： 您可以使用可擴展的 IP 子網路來適應未來成長趨勢，避免不必要的浪費。

常見的反模式：

- 無法考慮未來成長，導致 CIDR 區塊太小且需要重新配置，最後可能導致停機。
- 錯誤預估 Elastic Load Balancer 可以使用的 IP 位址數量。
- 部署過多高流量負載平衡器於相同的子網路中。
- 使用自動擴展機制，同時無法監控 IP 位址取用。
- 定義的 CIDR 範圍過大，並遠超出未來成長預期，可能導致難以與其他位址範圍發生重疊的網路進行對等互連。

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。

- 根據您的服務要求、延遲、法規和災難復原 (DR) 要求，選擇相關的 AWS 帳戶和區域。
- 確定您對區域 VPC 部署的需求。
- 確定 VPC 的大小。
  - 確定是否要部署多 VPC 連線。
    - 什麼是 Transit Gateway？
    - 單區域多 VPC 連線
  - 確定您是否需要區隔聯網以滿足法規要求
  - 製作包含大小適當 CIDR 區塊的 VPC，以適應目前和未來的需求。

    - 如果成長預測不明，您可能希望發生錯誤的是 CIDR 區塊過大，減少未來要重新配置的情況
- 考慮使用 IPv6 定址 做為雙堆疊 VPC 當中的子網路。非常適合使用 IPv6 的私有子網路中可能包含暫時性執行個體的機群或另外需要大量 IPv4 位址的容器。

### 資源

相關 Well-Architected 的最佳實務：

- REL02-BP05 在連線的所有私有地址空間中強制使用不重疊的私有 IP 位址範圍

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
- Amazon Virtual Private Cloud 連線能力選項白皮書
- 多個資料中心 HA 網路連線
- 單區域多 VPC 連線
- 什麼是 Amazon VPC？
- AWS 上的 IPv6
- 參考架構上的 IPv6
- Amazon Elastic Kubernetes Service 會啟動 IPv6 支援

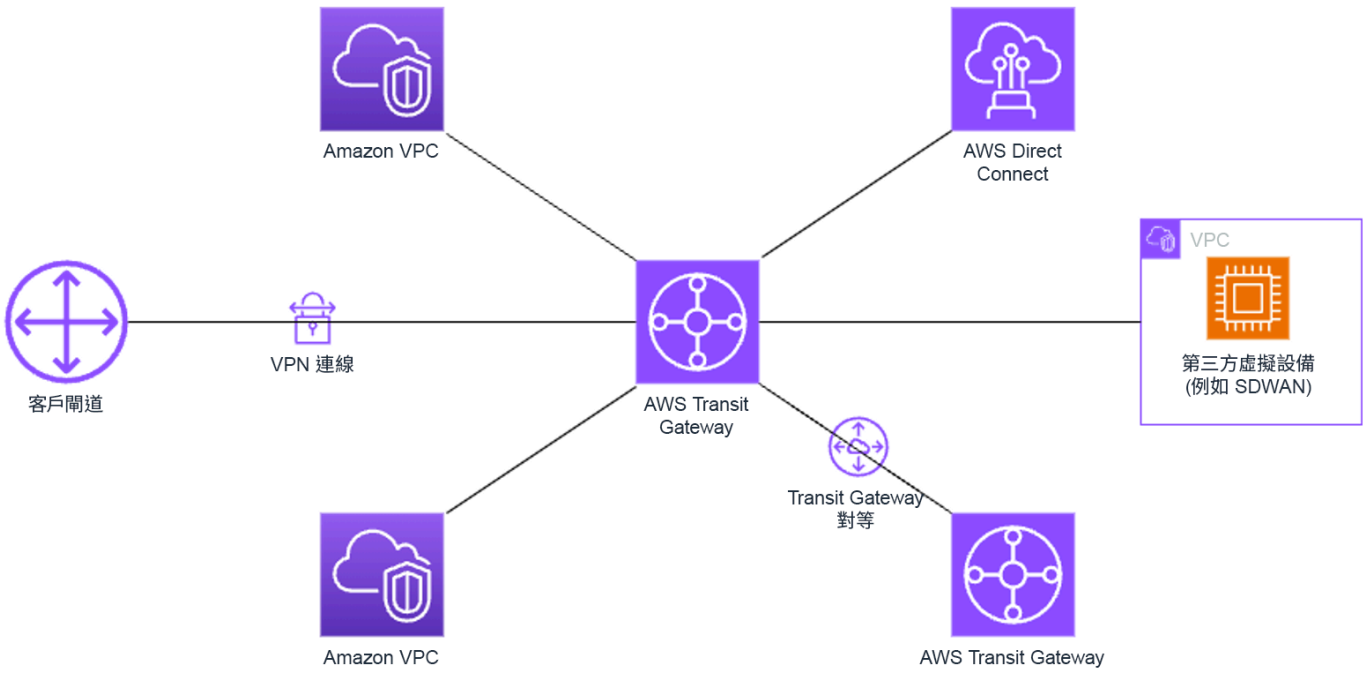
相關影片：

- AWS re:Invent 2018：進階 VPC 設計和 Amazon VPC 的新功能 (NET303)
- AWS re:Invent 2019：適用於許多 VPC 的 AWS Transit Gateway 參考架構 (NET406-R1)
- AWS re:Invent 2023：AWS 準備好踏出下一步了嗎？ 設計可促進成長與靈活性的網路 (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，請選擇軸輻式拓樸而不是網狀拓樸。在網狀拓撲中的每個網路都直接連線到其他網路，因而增加複雜性和管理開銷，但軸輻式架構與之不同，是透過單一集線器集中連線的。這種集中連線可簡化網路結構，並增強其可操作性、可擴展性和控制性。

AWS Transit Gateway 是一種受管、可擴展且高度可用的服務，專為建構在 AWS 上的軸輻式網路而設計。此服務可做為網路的中央樞紐，提供網路分割、集中路由，以及簡化對雲端和內部部署環境的連線。下圖說明如何使用 AWS Transit Gateway 建置軸輻式拓樸。




常見的反模式：

- 您的軸輻式架構中的路由政策過於複雜，致使網路效率降低，並使故障排除和主動管理變得複雜。
- 集線器內以路由為基礎的分段不足，可能會形成漏洞，進而使網路遭受未經授權的存取。
- 如果沒有經過仔細的最佳化處理，通過集線器路由的流量可能會導致更高的資料傳輸成本，特別是對於跨可用區域和區域的流量。有效的交通管理策略對控制費用至關重要。

建立此最佳實務的優勢：隨著連線網路的數量增加，網格連線的管理和擴展變得越來越具挑戰性。AWS Transit Gateway 提供可擴展且可靠的受管集線器，用於建構和操作軸輻式拓撲。使用 AWS Transit Gateway 時，您可以建立連線並集中跨多個網路的流量路由。

未建立此最佳實務時的風險暴露等級：中

## 實作指引

- 規劃您的網路。
- 建立您的 AWS Transit Gateway。

- 連接您的 VPC。
- 如有需要，建立 VPN 連線或 Direct Connect 閘道，並將它們與 Transit Gateway 相連接。
- 透過組態您的 Transit Gateway 路由表，定義如何在連接的 VPC 和其他連線之間轉送流量。
- 視需要使用 Amazon CloudWatch 監控和調整組態，以實現效能和成本最佳化。

### 資源

相關文件：

- 什麼是 Transit Gateway？
- 建置可擴展且安全的多 VPC AWS 網路基礎設施
- 使用 AWS Transit Gateway 區域間對等建置全球網路
- 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

## REL02-BP05 在連線的所有私有地址空間中強制使用不重疊的私有 IP 位址範圍

在對等、透過傳輸閘道連接或透過 VPN 連線時，每個 VPC 的 IP 位址範圍不得重疊。避免 VPC 與內部部署環境或您所使用之其他雲端供應商之間出現 IP 位址衝突。您也須有一種在需要時分配私有 IP 位址範圍的方法。IP 位址管理 (IPAM) 系統可以協助實現此自動化。

預期成果：

- VPC、內部部署環境或其他雲端供應商之間沒有 IP 位址範圍衝突。
- 適當的 IP 位址管理方便更輕鬆地擴展網路基礎設施，以適應網路需求的增長和變化。

常見的反模式：

- 在 VPC 中使用與內部部署、您的企業網路或其他雲端供應商相同的 IP 範圍
- 不追蹤用來部署工作負載之 VPC 的 IP 範圍。
- 依靠手動 IP 位址管理流程，例如試算表。
- CIDR 區塊大小過大或過小，會導致 IP 位址浪費或位址空間不足以供您的工作負載使用。

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。這可防止在使用不同應用程式的工作負載部分中發生路由問題。

未建立此最佳實務時的風險暴露等級：中

## 實作指引

使用 IPAM (例如 Amazon VPC IP Address Manager) 監控和管理您對 CIDR 的使用。AWS Marketplace 也提供數套 IPAM。評估您在 AWS 上的潛在使用情況，將 CIDR 範圍新增到現有 VPC，並建立 VPC 以便用量依規劃增長。

實作步驟

- 擷取目前的 CIDR 消耗 (例如 VPC 和子網路)。
  - 使用服務 API 作業收集目前的 CIDR 消耗。
  - 使用 Amazon VPC IP Address Manager 探索資源。
- 記錄您目前的子網路用量。
  - 使用服務 API 作業收集每個區域中每個 VPC 的子網路。
  - 使用 Amazon VPC IP Address Manager 探索資源。
- 記錄目前用量。
- 判斷您是否已建立任何重疊的 IP 範圍。
- 計算備用容量。
- 識別重疊的 IP 範圍。如果需要連接重疊範圍，您可以遷移到新的位址範圍，或考慮使用私有 NAT 閘道或 AWS PrivateLink 等技術。

## 資源

相關的最佳實務：

- 保護網路

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
- Amazon Virtual Private Cloud 連線能力選項白皮書
- 多個資料中心 HA 網路連線
- 連接具有重疊 IP 範圍的網路
- 什麼是 Amazon VPC？
- 什麼是 IPAM？

相關影片：

- AWS re:Invent 2023 - 進階 VPC 設計及新功能
- AWS re:Invent 2019：適用於許多 VPC 的 AWS Transit Gateway 參考架構
- AWS re:Invent 2023：準備好進行下一步了嗎？ 設計可促進成長與靈活性的網路
- AWS re:Invent 2021 - {新推出} 在 AWS 上大規模管理您的 IP 位址

# 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

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」），我們將叫用數百種微型服務，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 選擇如何劃分工作負載
- REL03-BP02 建置專注於特定業務領域和功能的服務
- REL03-BP03 每個 API 都提供服務合約

# REL03-BP01 選擇如何劃分工作負載

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。根據您的應用程式要求，這最終會盡可能由服務導向架構 (SOA) 與微型服務組合而成。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：

- AWS Well-Architected 微型服務 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。

建立此實務準則的優勢：

- 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 或微型服務架構 (或在少數情況下選擇整合型架構)。即使您選擇從整合型架構開始，仍須確保該架構為模組化，且隨著使用者採用，產品擴展時，該架構最終可以演進成 SOA 或微型服務。SOA 和微型服務各自提供較小的劃分，這些劃分同時也是偏好使用的現代可擴展且可靠的架構；但在部署微型服務架構時，特別要考慮做一些取捨。

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 和微型服務各自提供較小的分隔，而這是偏好使用的現代可擴展和可靠架構。SOA 會是達成較小分隔的良好折衷方案，同時能避免微型服務的部分複雜性。如需詳細資訊，請參閱 微型服務權衡。
- 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。如需詳細資訊，請參閱 實作 AWS 上的微型服務。
- 考慮遵循 Strangler Fig 模式， 將整合型重構為較小的元件。為此，必須逐步將特定的應用程式元件取代為新的應用程式和服務。AWS Migration Hub Refactor Spaces 可作為增量重構的起點。如需詳細資訊，請參閱 「使用扼制模式順暢地遷移內部部署的工作負載」。
- 實作微型服務時可能需要服務探索機制，讓這些分散式服務能夠彼此通訊。AWS App Mesh 可以搭配服務導向架構使用，以提供可靠的服務探索和存取。 AWS Cloud Map 也可用於動態、使用 DNS 的服務探索。
- 如果您要從整合型遷移至 SOA，Amazon MQ 可在您於雲端重新設計舊版應用程式時，以服務匯流排的形式消弭差距。
- 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。在重構程序的這個時間點，您應選擇以關聯式或非關聯式 (NoSQL) 類型的資料庫繼續操作。如需詳細資訊，請參閱 「從 SQL 到 NoSQL」。

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領域和功能的服務

相關文件：

- Amazon API Gateway：使用 OpenAPI 設定 REST API
- 什麼是服務導向架構？
- 有界限的環境 (領域驅動設計的集中模式)
- 實作 AWS 上的微型服務
- 微型服務權衡
- 微型服務 - 此新架構術語的定義
- AWS 上的微型服務
- 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：

- 透過 AWS 上的微型服務提供卓越品質

## REL03-BP02 建置專注於特定業務領域和功能的服務

服務導向架構 (SOA) 會定義服務，具有依商業需求定義的明訂功能。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： 工程師和業務利害關係人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。現有的整合型服務分解為 有界限的環境 ，系統設計改採 SOA 或微型服務架構。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：

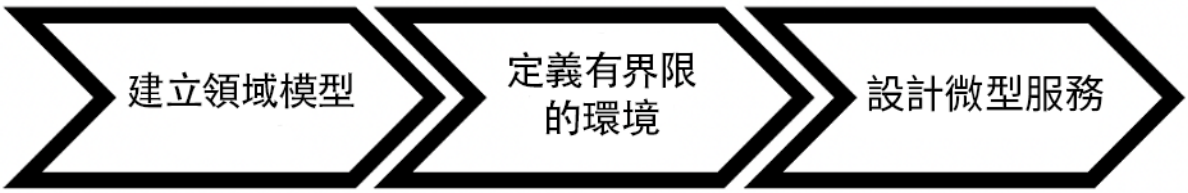
- 團隊是依據特定技術領域 (例如 UI 和 UX、中介軟體或資料庫) 組成的，而不是特定的業務領域。
- 應用程式跨多個領域責任。跨有界限環境的服務可能更難以維護，需要較大量的測試工作，且需要多個領域團隊參與軟體更新。
- 領域相依性 (例如領域實體程式庫) 會跨服務共用，因此一個服務領域出現變更時，需要變更其他服務領域
- 服務合約和商業邏輯無法以通用且一致的領域語言來表達實體，因此會導致翻譯層級使系統複雜化，並增加偵錯工作。

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

領域驅動決策 (DDD) 是依據業務領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。




領域驅動的決策

## 實作步驟

- 團隊可舉辦 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。
- 在領域環境中形成領域實體和函數之後，您可以使用 有界限的環境將領域分成服務，其中具有相似功能和特性的實體會歸類成一組。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。
- 戰術 (如 Bubble 環境 ) 可讓您在現有或舊版應用程式中導入 DDD，而無須預先重寫和對 DDD 完整承諾。在 Bubble 環境方法中，使用服務對應和協調來建立小型的有界限環境 (或 抗損毀層)，以保護新定義的領域模型免受外部影響。

在團隊執行領域分析並定義實體和服務合約之後，他們可以利用 AWS 服務將其領域導向設計實作為雲端架構服務。

- 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) 可協助團隊將服務著重於解決業務問題上。
- 選取 AWS 服務 (最符合您的業務領域要求和 微型服務架構)：
  - AWS 無伺服器 讓您的團隊專注於特定的領域邏輯，而不是管理伺服器和基礎設施。
  - AWS 上的容器 簡化基礎設施的管理，讓您得以專注在您的領域要求上。
  - 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 概述了一個架構，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商業邏輯分開的模式，可協助團隊專注於領域功能及改善軟體品質。

## 資源

相關的最佳實務：

- REL03-BP01 選擇如何劃分工作負載
- REL03-BP03 每個 API 都提供服務合約

相關文件：

- AWS 微型服務
- 實作 AWS 上的微型服務
- 如何將整合型服務分成微型服務
- 在置身於舊式系統時開始使用 DDD

- 領域驅動設計：解決軟體核心的複雜性
- 在 AWS 上建置六邊形架構
- 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：

- 企業雲端原生研討會
- 在 AWS 上設計雲端原生微型服務 (從 DDD/EventStormingWorkshop)

相關工具：

- AWS 雲端 資料庫
- AWS 上的無伺服器
- AWS 上的容器

## REL03-BP03 每個 API 都提供服務合約

服務合約是 API 生產者與取用者之間的記錄協議，載明於機器可讀取的 API 定義中。合約版本控制策略可讓取用者繼續使用現有的 API，並在準備好時將應用程式遷移至更新的 API。只要遵守合約，就隨時可執行生產者部署。服務團隊可以使用自己選擇的技術堆疊，以滿足 API 合約要求。

預期成果：

常見的反模式： 以服務導向或微型服務架構建置的應用程式能夠獨立運作，同時具有整合的執行期相依性。當雙方遵循共同的 API 合約時，部署至 API 取用者或生產者的變更不會中斷整體系統的穩定性。透過服務 API 進行通訊的元件可以執行獨立運作的版本、升級至執行期相依性，或容錯移轉至災難復原 (DR) 站台，且彼此幾乎沒有影響或完全沒有影響。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。

- 建立不具強型別結構描述的服務 API。這會產生無法用來產生 API 繫結的 API，以及無法以程式設計方式驗證的承載。

- 未採用版本控制策略，會迫使 API 取用者更新和發行，或在服務合約發展時失敗。
- 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。
- 未使用 API 合約來開發測試案例和模擬 API 實作，以允許單獨測試服務元件。

建立此最佳實務的優勢： 由透過 API 服務合約進行通訊的元件組成的分散式系統可提升可靠性。開發人員可在開發過程中及早發現潛在問題，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，且必要欄位存在。API 合約為 API 提供了清晰的自我記錄介面，並在不同的系統和程式設計語言之間提供了更好的互通性。

未建立此最佳實務時的曝險等級： 中

## 實作指引

在識別商業領域並確認工作負載區隔後，即可開發服務 API。首先，請定義機器可讀取的 API 服務合約，然後實作 API 版本控制策略。準備好透過 REST、GraphQL 等一般通訊協定或非同步事件來整合服務後，您可以將 AWS 服務併入您的架構中，以便將元件與強型別 API 合約整合。

服務 API 合約的 AWS 服務

將 AWS 服務 (包括 Amazon API Gateway、AWS AppSync和 Amazon EventBridge ) 併入您的架構中，以在您的應用程式中使用 API 服務合約。Amazon API Gateway 可協助您直接與原生 AWS 服務和其他 Web 服務整合。API Gateway 支援 OpenAPI 規格 和版本控制。AWS AppSync 是一個受管 GraphQL 端點，可藉由定義 GraphQL 結構描述來設定，用以定義查詢、變動和訂閱的服務介面。Amazon EventBridge 使用事件結構描述來定義事件及產生事件的程式碼繫結。

## 實作步驟

- 首先，為您的 API 定義合約。合約會說明 API 的功能，並定義強型別的資料物件和欄位，以用於 API 輸入和輸出。
- 在 API Gateway 中設定 API 時，您可以為端點匯入和匯出 OpenAPI 規格。
  - 匯入 OpenAPI 定義 可簡化 API 的建立，並且可以與 AWS 基礎設施即程式碼工具整合，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，並為服務取用者提供整合規格。
- 您可以透過 AWS AppSync 來定義和管理 GraphQL API，方法是 定義 GraphQL 結構描述 檔案以產生合約介面，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 專案 (與 AWS AppSync 整合) 會產生強型別 JavaScript 查詢檔案以供您的應用程式使用，並產生 AWS AppSync GraphQL 用戶端程式庫用於 Amazon DynamoDB 資料表。

- 當您從 Amazon EventBridge 中取用服務事件時，事件會遵循結構描述登錄中已存在的結構描述，或您使用 OpenAPI 規格定義的結構描述。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。
- 擴充您的 API 或對其進行版本控制。在新增可使用選用欄位或必要欄位的預設值來設定的欄位時，擴充 API 是較簡單的選項。
  - REST 和 GraphQL 等通訊協定的 JSON 型合約可能非常適合進行合約展延。
  - SOAP 等通訊協定的 XML 型合約應進行服務取用者的測試，以確認合約展延的可行性。
- 在對 API 進行版本控制時，請考慮實作 Proxy 版本控制，使用 Facade 來支援版本，以便在單一程式碼基底中維護邏輯。
  - 透過 API Gateway，您可以使用 請求和回應對應 建立 Facade 以提供新欄位的預設值，或從請求或回應中去除已移除的欄位，以簡化因應合約變更的工作。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 選擇如何劃分工作負載
- REL03-BP02 建置專注於特定業務領域和功能的服務
- REL04-BP02 實作鬆耦合相依性
- 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 (應用程式設計介面)？
- 實作 AWS 上的微型服務
- 微型服務權衡
- 微型服務 - 此新架構術語的定義
- AWS 上的微型服務
- 使用 OpenAPI 的 API Gateway 延伸
- OpenAPI 規格
- GraphQL：結構描述和類型

- Amazon EventBridge 程式碼繫結

相關範例：

- Amazon API Gateway：使用 OpenAPI 設定 REST API
- 使用 OpenAPI 設定 Amazon DynamoDB CRUD 應用程式的 Amazon API Gateway
- 無伺服器時代的現代化應用程式整合模式：API Gateway 服務整合
- 使用 Amazon CloudFront 實作標題型 API Gateway 版本控制
- AWS AppSync：建置用戶端應用程式

相關影片：

- 在 AWS SAM 中使用 OpenAPI 管理 API Gateway

相關工具：

- Amazon API Gateway
- AWS AppSync
- Amazon EventBridge

## 在分散式系統中設計防止故障的互動

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。這些最佳實務可防止失敗，並延長平均失敗間隔時間 (MTBF)。

最佳實務

- REL04-BP01 識別您依賴的分散式系統類型
- REL04-BP02 實作鬆耦合相依性
- REL04-BP03 持續執行工作
- REL04-BP04 將所有回應設為等冪

# REL04-BP01 識別您依賴的分散式系統類型

分散式系統可以是同步、非同步或批次。同步系統必須盡快處理請求，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來互相通訊。非同步系統透過中介服務以非同步方式交換資料相互通訊，而無需結合個別系統。批次系統接收大量的輸入資料，在無人為干預的情況下執行自動化資料處理，並產生輸出資料。

期望的結果：設計與同步、非同步和批次相依項有效互動的工作負載。

常見的反模式：

- 工作負載會無限期等待其相依項的回應，這可能導致工作負載用戶端逾時，卻不知道對方是否已收到它們的請求。
- 工作負載使用相互同步呼叫的相依系統鏈。這需要每個系統都可用，並在整個鏈成功之前成功處理請求，因而導致潛在的脆弱行為和整體可用性。
- 工作負載以非同步方式與其相依項進行通訊，並依賴保證訊息一次性傳送的概念，而通常仍有可能接收到重複的訊息。
- 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：特定的工作負載通常會在同步、非同步和批次模式之間選擇實作一或多種通訊模式。這個最佳實務可協助您識別與每種通訊模式相關聯的不同權衡，讓您的工作負載能夠承受其任何相依項遭遇中斷的情況。

未建立此最佳實務時的風險暴露等級：高

## 實作指引

以下各節包含針對每種相依項類型的一般和特定實作指引。

一般指引

- 確保相依項提供的效能和可靠性服務層級目標 (SLO) 符合工作負載的效能和可靠性要求。
- 使用 AWS 可觀測性服務監控回應時間和錯誤率，確保您的相依項能在工作負載所需的層級提供服務。
- 識別工作負載與其相依項通訊時可能會面臨的潛在挑戰。分散式系統面臨的各種挑戰，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記錄流程，以在您的相依項遇到問題時協助進行疑難排解。

同步相依

在同步通訊中，您的工作負載會傳送要求給其相依項，並封鎖等待回應的運作。當其相依項收到請求時，會嘗試盡快處理，並將回應傳回至您的工作負載。同步通訊的最大挑戰是它會導致時間耦合，這需要您的工作負載和其相依項同時可用。當您的工作負載需要與其相依項同步通訊時，請考量以下指引：

- 您的工作負載不應依賴多個同步相依項來執行單一函數。這個相依鏈增加了整體脆性，因為路徑中的所有相依項都必須可用才能成功完成請求。
- 當相依項不健全或無法使用時，請判斷如何處理錯誤和重試策略。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。如需有關雙模態行為的詳細資訊，請參閱 REL11-BP05 使用靜態穩定性來預防雙模態行為。
- 請記住，快速檢錯比讓工作負載等待更好。例如，AWS Lambda 開發人員指南說明如何在調用函數時處理重試和失敗。Lambda
- 在工作負載呼叫其相依項時設定逾時。這種技術可避免等待太久或無限期等待回應的現象。如需有關此問題的實用討論，請參閱為延遲感知型 Amazon DynamoDB 應用程式調整 AWS Java SDK HTTP 請求設定。
- 將工作負載僅為滿足單一請求而呼叫其相依項的次數減到最少。它們兩者之間的冗長通話會增加耦合及延遲。

非同步相依項

若要暫時為您的工作負載與其相依項解耦，它們應該以非同步方式通訊。使用非同步方法，您的工作負載可以繼續進行其他任何處理，而無需等待其相依項或相依項鏈傳送回應。

當您的工作負載需要與其相依項非同步通訊時，請考慮下列指引：

- 根據您的使用案例和需求，決定是否使用訊息傳遞或事件串流。傳訊允許您的工作負載與其相依項通訊時，透過訊息代理程式傳送和接收郵件。事件串流允許您的工作負載及其相依項使用串流服務發佈和訂閱事件，而這些作為連續資料串流來傳遞的事件需要盡快受處理。

- 傳訊和事件串流以不同方式處理訊息，因此您需要根據以下方式做出取捨：
  - 訊息優先順序：訊息代理程式可以在處理一般訊息之前處理高優先順序的訊息。在事件串流中，所有訊息都具有相同的優先順序。
  - 訊息取用：訊息代理程式確保取用者接收到訊息。活動串流取用者必須追蹤他們最後一次閱讀的訊息。

  - 訊息排序：傳訊時，除非您採用先進先出 (FIFO) 方式，否則無法保證按照傳送的確切順序接收訊息。事件串流始終會保留資料的產生順序。
  - 訊息刪除：傳訊時，取用者必須在處理訊息後刪除該訊息。事件串流服務會將訊息附加到串流中，並保留在該串流中，直到訊息的保留期限到期為止。此刪除政策使事件串流適合重播訊息。
- 定義您的工作負載如何知道其相依項何時完成其工作。例如，當您的工作負載以非同步方式調用 Lambda 函數時，Lambda 會將事件放置在佇列中，並傳回不含其他資訊的成功回應。處理完後，Lambda 函數可以將結果傳送到目的地，根據成功或失敗進行設定。
- 建置您的工作負載時，利用冪等性處理重複的訊息。冪等性的意思是即使為相同訊息產生您的工作負載一次以上，工作負載的結果也不會改變。重要的是，如果發生網路失敗或未收到確認，傳訊或串流服務將會重新傳遞訊息。
- 如果您的工作負載沒有收到其相依項的回應，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、記憶體和網路資源以處理其他請求。AWS Lambda 文件顯示如何處理非同步調用的錯誤。
- 利用適合的可觀測性、除錯和追蹤工具來管理和操作工作負載與其相依項的非同步通訊。您可以使用 Amazon CloudWatch 監視傳訊和事件串流服務。您也可以利用 AWS X-Ray 檢測工作負載，以快速針對疑難排解問題取得洞見。

批次相依項

批次系統接收輸入資料，啟動一系列工作來處理該資料，並生產一些輸出資料，完全無需手動介入。視資料大小而定，工作的執行可能需要幾分鐘 (在某些情況下) 到幾天時間。當您的工作負載與其批次相依項通訊時，請考慮下列指引：

- 定義工作負載應執行批次工作的時段。您的工作負載可以設定重複模式來調用批次系統，例如每小時或每個月尾。
- 確定資料輸入和受處理資料輸出的位置。選擇一種儲存服務，例如 Amazon Simple Storage Services (Amazon S3)、Amazon Elastic File System (Amazon EFS) 和 Amazon FSx for Lustre，讓您的工作負載能夠大規模讀取和寫入檔案。
- 如果您的工作負載需要呼叫多個批次工作，您可以利用 AWS Step Functions 簡化在 AWS 或內部部署執行的批次工作的協同運作。此範例專案示範了使用 Step 函數、AWS Batch 和 Lambda 的批次工作協同運作。
- 監控批次工作以尋找異常情況，例如完成工作需要比預期更長的時間。您可以使用 CloudWatch Container Insights 等工具來監控 AWS Batch 環境和工作。在這種情況下，您的工作負載會從頭開始停止下一個工作，並通知相關員工有關例外情況。

## 資源

相關文件：

- AWS 雲端 Operations：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
- 為延遲感知型 Amazon DynamoDB 應用程式調整 AWS Java SDK HTTP 請求設定。
- AWS 傳訊
- 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：Amazon SQS 可用的 CloudWatch 指標
- Amazon Kinesis Data Streams 開發人員指南：利用 Amazon CloudWatch 監控 Amazon Kinesis Data Streams 服務
- AWS X-Ray 開發人員指南：AWS X-Ray 概念
- AWS GitHub 上的範例：AWS Step Functions 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 的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。

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佈的界面。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，您可以橫向擴展，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點對點直接互動來整合，但通常會透過中繼耐用儲存層來整合，例如 Amazon SQS 佇列，或如 Amazon Kinesis 或 AWS Step Functions 等串流資料平台。

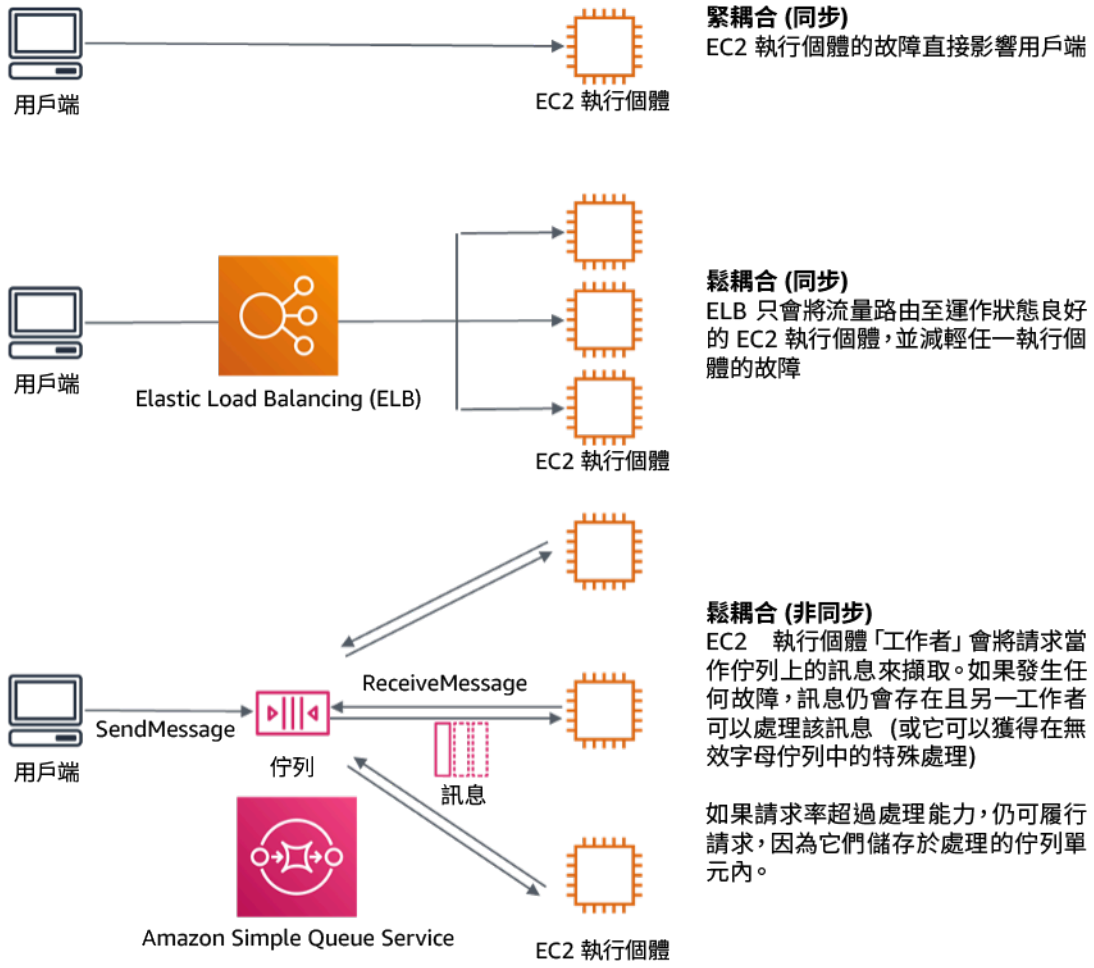



圖 4：佇列系統和負載平衡器之間具有鬆散耦合的相依性。

Amazon SQS 佇列和 Elastic Load Balancer 只是為鬆耦合新增中繼層的兩種方式。事件驅動架構也可以使用 Amazon EventBridge 在 AWS 雲端 建置。其可從用戶端依賴的服務 (事件取用者) 中抽取用戶端 (事件生產者)。當您需要高輸送量、推送架構的多對多傳訊時，Amazon 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發給大量訂閱者端點，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讓您將失敗的影響範圍最小化到元件層級，從而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展。

常見的反模式：

- 部署整合型工作負載。
- 在工作負載層之間直接叫用 API，沒有容錯移轉或非同步處理請求的功能。
- 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。

建立此最佳實務的好處：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，進而提高彈性和靈活性。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風險暴露等級：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。這些解決方案包括用於實作全受管佇列的服務、自動化工作流程、對事件的回應以及 API 等，其有助於將元件的行為與其他元件隔離，從而提高彈性和靈活性。

- 建置事件驅動架構：Amazon EventBridge 可協助您建置鬆耦合和分散式的事件驅動架構。
- 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。
- 將元件容器化為微型服務：微型服務可讓團隊建置由小型獨立元件組成的應用程式，這些元件會通過明確定義的 API 進行通訊。Amazon Elastic Container Service (Amazon ECS) 和 Amazon Elastic Kubernetes Service (Amazon EKS) 可協助您更快地開始使用容器。
- 使用 Step Functions 管理工作流程：Step Functions 可協助您將多個 AWS 服務協調為彈性工作流程。
- 利用發布-訂閱 (pub/sub) 傳訊架構：Amazon Simple Notification Service (Amazon SNS) 可提供從發布者到訂閱用戶 (也稱為生產者和取用者) 的訊息傳遞功能。

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 EventBridge 建置事件驅動應用程式
  - AWS re:Invent 2022 - 使用 Amazon EventBridge 設計事件驅動整合

- 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。這些部分包括分散式系統的元件、用於解耦的佇列 (分散在 Amazon SQS 伺服器上)，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。為了備援，佇列會在多個 Amazon SQS 伺服器儲存訊息。
  - 基本的 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 AWS 上實作微型服務
  - 開始建構吧！ 使用容器建構微型服務
- AWS Step Functions 可讓您建置分散式應用程式、將程序自動化、協調微型服務等。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 AWS Lambda 建立無伺服器工作流程
  - AWS Step Functions 入門

## 資源

相關文件：

- Amazon EC2：確保等冪性
- Amazon Builders' Library：分散式系統的挑戰
- Amazon Builders' Library：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？
- 與整合型分手
- 使用 AWS Step Functions 和 Amazon SQS 協調佇列型微型服務
- 基本的 Amazon SQS 架構
- 佇列式架構

相關影片：

- 2019 年 AWS 紐約高峰會：事件驅動架構和 Amazon EventBridge 簡介 (MAD205)

- AWS re:Invent 2018：閉環與開放思維：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：移至事件驅動架構 (SVS308)
- AWS re:Invent 2019：使用 Amazon SQS 和 Lambda 的可擴展無伺服器事件驅動應用程式 (API304)
- AWS re:Invent 2019：使用 Amazon SQS 和 Lambda 的可擴展無伺服器事件驅動應用程式
- AWS re:Invent 2022 - 使用 Amazon EventBridge 設計事件驅動整合
- AWS re:Invent 2017：Elastic Load Balancing 深入剖析與最佳實務

## REL04-BP03 持續執行工作

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。這實際上是 Amazon Route 53 處理端點 (例如 IP 地址) 的運作狀態檢查，以判斷最終使用者如何路由到其中的方式。

若未建立此最佳實務，暴露的風險等級： 低

### 實作指引

- 執行持續工作，以便負載大量快速變更時，系統不會失敗。
- 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。
  - Amazon Builders' Library：可靠性、持續工作，以及咖啡時刻
  - AWS re:Invent 2018：閉環與開放思維：如何取得大小型系統的控制權 (ARC337) (包括持續工作)
    - 針對運作狀態檢查系統監控 100,000 部伺服器的範例，將工作負載設計為無論成功或失敗的數量為何，承載大小都保持不變。

### 資源

相關文件：

- Amazon EC2：確保等冪性
- Amazon Builders' Library：分散式系統的挑戰
- Amazon Builders' Library：可靠性、持續工作，以及咖啡時刻

相關影片：

- 2019 年 AWS 紐約高峰會：事件驅動架構和 Amazon EventBridge 簡介 (MAD205)
- AWS re:Invent 2018：閉環與開放思維：如何取得大小型系統的控制權 (ARC337) (包括持續工作)
- AWS re:Invent 2018：閉環與開放思維：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：移至事件驅動架構 (SVS308)

## REL04-BP04 將所有回應設為等冪

等冪服務承諾每個請求只完成一次，使得發出多個相同請求與發出單一請求具有相同的效果。等冪服務可讓用戶端更輕鬆地實作重試，而不用擔心錯誤地多次處理請求。為此，用戶端可以使用等冪權杖發出 API 請求，即每次重複請求時，都會使用相同的權杖。等冪服務 API 會使用權杖來傳回與第一次完成請求時傳回之回應相同的回應。

在分散式系統中，執行最多一次動作 (用戶端只發出一個請求) 或至少一次動作 (持續發出請求，直到用戶端確認成功) 很容易。但很難保證動作是等冪的，這表示它 只執行一次， 使得發出多個相同的請求與發出單一請求具有相同效果。透過在 API 中使用等冪性權杖，服務可以收到一次或多次變異請求，而不會產生重複的記錄或副作用。

若未建立此最佳實務，暴露的風險等級： 中

### 實作指引

- 將所有回應設為等冪。等冪服務承諾每個請求只完成一次，使得發出多個相同請求與發出單一請求具有相同的效果。
  - 用戶端可以使用等冪權杖發出 API 請求，即每次重複請求時，都會使用相同的權杖。等冪服務 API 會使用權杖來傳回與第一次完成請求時傳回之回應相同的回應。

- Amazon EC2：確保等冪性

### 資源

相關文件：

- Amazon EC2：確保等冪性
- Amazon Builders' Library：分散式系統的挑戰
- Amazon Builders' Library：可靠性、持續工作，以及咖啡時刻

相關影片：

- 2019 年 AWS 紐約高峰會：事件驅動架構和 Amazon EventBridge 簡介 (MAD205)
- AWS re:Invent 2018：閉環與開放思維：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：移至事件驅動架構 (SVS308)

## 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。這些最佳實務讓工作負載能夠承受壓力或故障，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失敗，並延長平均失敗間隔時間 (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 快速檢錯和限制佇列
- REL05-BP05 設定用戶端逾時
- REL05-BP06 盡可能讓系統變成無狀態
- REL05-BP07 實作緊急控制桿

# 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持續失敗，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 就像絕大多數的關聯式資料庫一樣，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 無伺服器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：限流 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 中的錯誤重試和指數退避
- Michael Nygard「Release It! 設計和部署生產就緒型軟體」
- 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)

相關範例：

- Well-Architected 實驗室：第 300 級：實作運作狀態檢查和管理相依性以提升可靠性

## 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 地址的速率限制。

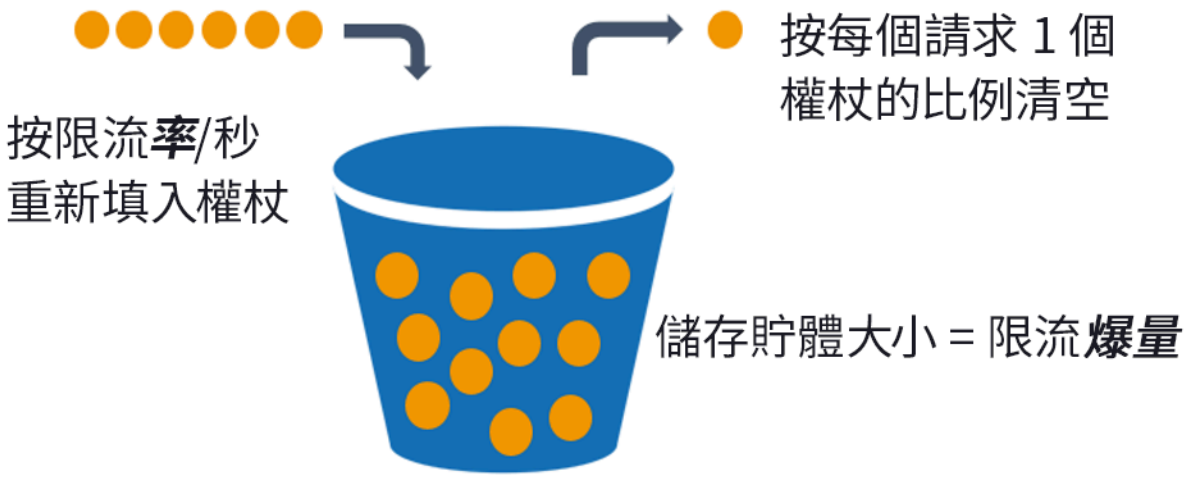
建立此最佳實務的優勢： 設定限流限制的工作負載能夠在非預期的數量尖峰情況下正常運作，並成功處理已接受的請求負載。API 和佇列的請求若突然或持續激增將會受到限制，且不會耗盡請求處理資源。速率限制會限流個別請求者，使來自單一 IP 地址或 API 取用者的大量流量不會耗盡資源而影響到其他取用者。

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 可以緩衝處理請求以緩解請求率，並對於可處理的請求允許提高限流率。最後，您可以透過 AWS WAF 實作速率限制，以限流會產生異常高負載的特定 API 取用者。

## 實作步驟

您可以為 API 設定具有限流限制的 API Gateway，並在超出限制時傳回 429 #### 錯誤。您可以將 AWS WAF 用於 AWS AppSync 和 API Gateway 端點，以就個別 IP 地址啟用速率限制。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。

若使用非同步處理，當您將 Amazon SQS 設定為 AWS Lambda 的事件來源時，您可以 設定並行上限，以避免高事件速率耗用您的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 API Gateway 限流限制 (在個別區域的帳戶層級、個別階段的 API，以及個別使用計畫層級的 API 金鑰)。
- 套用 AWS WAF 速率限制規則 於 API Gateway 和 AWS AppSync 端點，以防止洪水及阻止惡意 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 設定為適當值，正好足以符合您的服務水準目標，但不會耗用並行限制而影響到其他 Lambda 函數。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 進行 Amazon SQS 或 Kinesis 的原生服務整合以緩衝處理請求。
- 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 持續執行工作
- REL05-BP03 控制和限制重試呼叫

相關文件：

- Amazon API Gateway：限流 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 速率型規則
- 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

# 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。

預期成果： 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。
- 在不測試影響的情況下實作重試，或者假設重試已直接內建到 SDK 中，而未測試重試情境。
- 未能了解從相依性發佈的錯誤代碼，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 SDK 依預設會實作重試和指數退避。若情況允許，在工作負載中使用這些內建 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 SDK 是否會透過指數退避和抖動為您選擇的語言實作經過驗證的重試策略，如果會請優先予以採用，而不是撰寫自己的重試實作。
- 請確認 服務是等冪的 ，然後再實作重試。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。
- 在呼叫 AWS 服務 API 時，使用 AWS SDK 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 將所有回應設為等冪
- REL05-BP02 限流請求
- REL05-BP04 快速檢錯和限制佇列
- 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：

- AWS 中的錯誤重試和指數退避
- Amazon 建置者資料中心：逾時、重試、退避與抖動
- 指數退避和抖動
- 使用等冪 API 確保重試安全性

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：Amazon 建置者資料中心簡介 (DOP328)

相關工具：

- AWS SDK 和工具：重試行為
- AWS Command Line Interface：AWS CLI 重試

## REL05-BP04 快速檢錯和限制佇列

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：

- 在 DLQ 磁碟區上實作訊息佇列，但不設定無效字母佇列 (DLQ) 或警示，以偵測系統失敗。
- 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。
- 在後進先出 (LIFO) 佇列可更妥善地滿足用戶端需求時設定了先進先出 (FIFO) 佇列，例如，當不需要嚴格排序，以及積存處理延遲了所有新的和時間敏感的請求，導致所有用戶端都經歷違反服務水準的狀況。
- 將內部佇列公開給用戶端，而不是公開管理工作接受以及將請求放入內部佇列的 API。

- 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，並且在發佈至生產環境之前，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 提供監控失敗和發出相關警示的功能。已知系統失敗時，可以叫用緩解策略，包括背離受損的資源。當系統實作佇列與 Amazon SQS 和其他佇列技術來平滑負載，必須考量如何管理佇列積存，以及訊息取用失敗。

## 實作步驟

- 在軟體中實作程式化判斷提示或特定指標，並使用它們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 檢測來建立指標及提醒。
- 使用 CloudWatch 指標和警示背離受損的資源，這些資源會增加處理的延遲，或持續無法處理請求。
- 藉由設計 API 使用非同步處理來接受請求，並使用 Amazon SQS 將請求附加到內部佇列，然後透過成功訊息回應產生訊息的用戶端，讓用戶端可以釋出資源並繼續進行其他工作，同時讓後端佇列取用者處理請求。
- 藉由比較現在時間與訊息時間戳記，在每次從佇列中取出訊息時產生 CloudWatch 指標，來測量和監控佇列處理延遲。
- 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。這種技術類似於 LIFO 處理，方便對所有新工作進行正常的系統處理。
- 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 透過您的系統監控請求的端對端追蹤

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 佇列中不斷增加的訊息積存？
- Elastic Load Balancing：區域轉移
- Amazon Route 53 應用程式復原控制器：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、目標服務受損或網路分割失敗，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。

利用完善的設計模式 (例如重試和斷路器)，按正常程序處理逾時並支援快速檢錯方法。AWS SDK 和 AWS CLI 允許設定連線和請求逾時，以及具有指數退避和抖動的重試。AWS Lambda 函數支援設定逾時，且透過 AWS Step Functions，您可以建置低程式碼斷路器，以利用預先建置的 AWS 服務和 SDK 整合。AWS App Mesh Envoy 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 AWS SDK 進行呼叫時，請檢閱文件以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 AWS SDK 或 AWS CLI 命令時，請設定預設逾時值，方法是設定 AWS 組態預設值 (針對 `connectTimeoutInMillis` 和 `tlsNegotiationTimeoutInMillis`)。
- 套用 命令列選項 `cli-connect-timeout` 和 `cli-read-timeout` 以控制 AWS 服務的一次性 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。

- 實作 CloudWatch 指標 和 CloudWatch 異常偵測 於呼叫錯誤率、延遲的服務水準目標，以及延遲離群值，讓您能夠深入了解如何管理過於積極或寬鬆的逾時。
- 設定逾時於 Lambda 函數。
- API Gateway 用戶端在處理逾時期間必須實作本身的重試。API Gateway 支援 50 毫秒至 29 秒的整合逾時 (用於下游整合)，且整合請求若逾時將不會重試。
- 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。
- 對於容器型工作負載，請檢閱 App Mesh Envoy 功能，以利用內建的逾時和斷路器。
- 使用 AWS Step Functions 為遠端服務呼叫建置低程式碼斷路器，尤其是在呼叫 AWS 原生 SDK 和支援的 Step Functions 整合以簡化工作負載的情況下。

## 資源

相關的最佳實務：

- REL05-BP03 控制和限制重試呼叫
- REL05-BP04 快速檢錯和限制佇列
- REL06-BP07 透過您的系統監控請求的端對端追蹤

相關文件：

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

相關範例：

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

相關工具：

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

## REL05-BP06 盡可能讓系統變成無狀態

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，不依賴磁碟上和記憶體中本機儲存的資料。這可讓伺服器任意置換，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 AWS Lambda 或 AWS Fargate 等無伺服器運算服務。

除了伺服器替換，無狀態應用程式的另一個好處是它們可以水平擴展，因為任何可用的運算資源 (例如，EC2 執行個體和 AWS Lambda 函數) 都可以處理所有請求。

建立此最佳實務的優勢：設計為無狀態的系統更適應水平擴展，因此可以根據波動的流量和需求增加或移除容量。此系統本質上也具有抵禦失敗的能力，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風險暴露等級：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，並且可以容忍個別節點的故障。分析並了解應用程式中維持架構內狀態的元件。這可協助您評估轉換至無狀態設計的潛在影響。無狀態架構會解除配對使用者資料，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優化資源使用率。

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分隔及管理來解除配對資料。

  - 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解除配對。
  - 您可以使用 AWS Secrets Manager 將機密儲存在安全、集中的位置來解偶使用者資料。這意味著應用程式的程式碼不需要儲存密碼，因此本身更加安全。
  - 考慮使用 Amazon S3 來儲存大型非結構化資料，例如影像和文件。您的應用程式可以在需要時檢索這些資料，進而不需要將其儲存在記憶體中。
  - 使用 Amazon DynamoDB 儲存使用者設定檔等資訊。您的應用程式可以近即時查詢此資訊。
- 將工作階段資料卸載到資料庫、快取或外部檔案。
  - Amazon ElastiCache、Amazon DynamoDB、Amazon Elastic File System (Amazon EFS) 以及 Amazon MemoryDB for Redis 是可以用來卸載工作階段資料的 AWS 服務範例。
- 在確定需要使用所選儲存解決方案保留哪些狀態和使用者資料後，設計無狀態架構。

### 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：

- 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
- 在 AWS 上的無狀態 Web 層的最佳實務

## REL05-BP07 實作緊急控制桿

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，您可以建立已知良好的程序，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。如需適度降級的詳細資訊，請參閱 REL05-BP01 實作適度降級，以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：實作緊急控制桿可以藉由為解析器提供已建立的程序來回應意外的需求突增或非關鍵相依性的失敗，以改善工作負載中關鍵元件的可用性。

未建立此最佳實務時的風險暴露等級：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 的關鍵和非關鍵功能範例，請參閱任何一天都可以是 Prime Day：Amazon.com 搜尋如何使用混沌工程處理每秒超過 84000 個請求。
  - 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。如需詳細資訊，請參閱 REL11-BP05 使用靜態穩定性來防止雙模態行為。
- 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)。

- 這可能包括卸載、限流請求或實作適度降級。

## 資源

相關的最佳實務：

- 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
- REL11-BP05 使用靜態穩定性來防止雙模態行為

相關文件：

- 自動化安全、無人為介入的部署
- 任何一天都可以是 Prime Day：Amazon.com 搜尋如何使用混沌工程處理每秒超過 84,000 個請求

相關影片：

- AWS re:Invent 2020：透過不可變實現可靠性、一致性和可信度

# 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

日誌和指標是深入了解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。

監控是關鍵步驟，可確保您滿足可用性要求。需高效監控，以偵測故障。最糟糕的故障模式是「沉默」故障，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
- REL06-BP06 定期進行審查
- REL06-BP07 透過您的系統監控請求的端對端追蹤

## 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，描述如何從日誌擷取指標 (如果需要)，以及設定觸發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預警訊號。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目) 可以比 CPU 使用率這類的技術指標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 AWS 服務的效能和可用性，取得個人化檢視。

雲端監控提供新機遇。大部分雲端供應商都開發了可自訂的掛鉤，並且可以提供洞察力來協助您監控多層的工作負載。AWS 服務 (例如 Amazon CloudWatch) 會套用統計和機器學習演算法，以持續分析系統和應用程式的指標、決定正常基準，以及顯現使用者介入最少的異常。異常偵測演算法會考慮指標的季節性和趨勢變更。

AWS 提供大量可用於消費的監控和日誌資訊，這些資訊可以用來定義工作負載特有的指標、按需變更流程，以及採用機器學習技術，而不管 ML 專業知識為何。

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，後者會定期執行應用程式消費者執行的一些常見任務。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 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：

收集和使用來自工作負載所有元件的關鍵指標，以確保工作負載可靠性和最佳使用者體驗。偵測到工作負載未實現業務成果可讓您快速宣佈災難並從事故中復原。

常用的反模式：

- 僅監控工作負載的外部界面。
- 不產生任何工作負載特有的指標，而且僅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，而且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： 工作負載中的所有層級監控，可讓您更快速地預測和解決構成工作負載之元件中的問題。

若未建立此最佳實務，暴露的風險等級： 高

## 實作指引

1. 在可用的地方啟用記錄。 應該從工作負載的所有元件中取得監控資料。開啟額外記錄 (例如 S3 存取日誌)，並讓您的工作負載可以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。請參閱 發佈 CloudWatch 指標的 AWS 服務 取得將指標發佈至 CloudWatch 的 AWS 服務清單。
2. 審查所有預設指標並探索任何資料收集差距。 每個服務都會產生預設指標。收集預設指標可讓您更好地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。您也可以建立 自己的指標並將其 發佈至 CloudWatch，方法為使用 AWS CLI 或 API。此
3. 評估所有指標，以判斷哪些指標要對工作負載中的每個 AWS 發出提醒。 您可以選擇要選取對工作負載可靠性有重大影響的指標子集。專注於關鍵指標和閾值可讓您微調 提醒 數目，並可以協助將誤判的情形減至最少。
4. 定義提醒以及在觸發提醒之後工作負載的復原流程。 定義提醒可讓您快速通知、呈報並遵循必要的步驟，從事故中復原並符合您指定的復原時間點目標 (RTO)。您可以使用 Amazon CloudWatch 警示，叫用自動化工作流程，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。 綜合監控會遵循相同的路由並執行與客戶相同的動作，這可讓您持續驗證您的客戶體驗，即使您的工作負載上沒有任何客戶流量也一樣。使用 綜合交易，您可以在客戶探索問題之前先行探索。

## 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：

- AWS Health 儀表板入門 – 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 Amazon CloudWatch Logs 的 AWS Lambda
- Amazon S3 伺服器存取記錄
- 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 代理程式
- 發布自訂指標
- 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：

- 建立軌跡
- 監控 Amazon EC2 Linux 執行個體的記憶體和磁碟指標
- 搭配容器執行個體使用 CloudWatch Logs
- VPC Flow Logs
- 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 偵錯

相關範例和研討會：

- AWS Well-Architected 實驗室：卓越營運 - 相依性監控
- Amazon Builders' Library：偵測分散式系統，以瞭解運作狀態
- 可觀測性研討會

## REL06-BP02 定義和計算指標 (彙總)

視需要儲存日誌資料並套用篩選條件以計算指標，例如特定日誌事件的計數，或是從日誌事件時間戳記計算的延遲。

Amazon CloudWatch 和 Amazon S3 可作為主要的彙總和儲存層。對於某些服務 (例如 AWS Auto Scaling 和 Elastic Load Balancing)，預設會為跨叢集或執行個體的 CPU 負載或平均請求延遲提供預設指標。對於 VPC Flow Logs 及 AWS CloudTrail 等串流服務，事件資料將轉寄到 CloudWatch Logs，且您需要定義和套用指標篩選條件以從事件資料中擷取指標。這為您提供時間序列資料，而此資料可作為您定義用於觸發提醒之 CloudWatch 警示的輸入。

若未建立此最佳實務，暴露的風險等級： 高

### 實作指引

- 定義和計算指標 (彙總)。視需要儲存日誌資料並套用篩選條件以計算指標，例如特定日誌事件的計數，或是從日誌事件時間戳記計算的延遲
  - 指標篩選條件會定義術語與模式，以在傳送到 CloudWatch Logs 的日誌資料中尋找資料。CloudWatch Logs 使用這些指標篩選條件，將日誌資料轉成數值 CloudWatch 指標，讓您可以對其繪製圖表或設定警示。
    - 搜尋和篩選日誌資料
  - 使用受信任的第三方來彙總日誌。
    - 請遵循第三方的指示。大部分第三方產品可與 CloudWatch 和 Amazon S3 整合。
  - 有些 AWS 服務可以直接將日誌發佈到 Amazon S3。如果您的日誌主要需求是儲存在 Amazon S3 中，則可以輕鬆讓產生日誌的服務直接將它們傳送到 Amazon S3，無須設定其他基礎設施。
    - 直接將日誌傳送至 Amazon S3

### 資源

相關文件：

- Amazon CloudWatch Logs Insights 範例查詢
- 使用 Amazon CloudWatch Synthetics 和 AWS X-Ray 偵錯
- 一個觀察工作坊
- 搜尋和篩選日誌資料
- 直接將日誌傳送至 Amazon S3
- Amazon Builders' Library：偵測分散式系統，以瞭解運作狀態

## 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 可讓您建立 指標 和複合警示，過程中根據靜態閾值、異常偵測和其他條件使用 CloudWatch 警示。如需有關可使用 CloudWatch 設定的警示類型詳細資訊，請參閱 CloudWatch 文件的警示一節。

您可以為團隊建構 AWS 資源的指標和警示自訂檢視，過程中使用 CloudWatch 儀表板。您可以透過 CloudWatch 主控台中的可自訂首頁，在單一檢視中監控多個區域的資源。

警示可以執行一或多個動作，例如將通知傳送給 或向 Amazon SNS 主題、執行 Amazon EC2 動作或 Amazon EC2 Auto Scaling 動作，或 建立 OpsItem 或 事故 (AWS Systems Manager)。

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)。如需設定 Amazon SNS 通知的詳細資訊，請參閱 設定 Amazon SNS。

CloudWatch 傳送 EventBridge 的安全 事件，尤其是每當在 CloudWatch 警示進行建立、更新、刪除，或是狀態變更時。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，或自動觸發在您帳戶中的事件，過程是使用 Systems Manager 自動化功能。

何時應該使用 EventBridge 和 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。

在建置應用程式以回應您自己應用程式、SaaS 應用程式和 AWS 服務中的事件時，建議使用 Amazon EventBridge。EventBridge 是唯一直接與第三方 SaaS 合作夥伴整合的事件型服務。EventBridge 還會自動從 200 多個 AWS 服務中擷取事件，而不需開發人員在帳戶中建立任何資源。

EventBridge 會將已定義的 JSON 架構用在事件，並協助您建立在整個事件內文套用的規則，以選取要轉寄至 目標的事件。EventBridge目前支援 20 多項 AWS 服務做為目標，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，且可以是任何格式。Amazon SNS 支援將訊息轉寄至六種不同的目標，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 AWS 訊息，各種 Amazon SNS 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. 建立警示，過程中使用 Amazon CloudWatch 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。此動作可能包括將通知傳送給 或向 Amazon SNS 主題、執行 Amazon EC2 動作或 Amazon EC2 Auto Scaling 動作，或 建立 OpsItem 或 事故 (AWS Systems Manager)。

   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。複合警示可在狀態變更時傳送 Amazon SNS 通知，並可建立 Systems Manager 建立和追蹤這些改善 或 事故 (在進入警示狀態時)，但其無法執行 Amazon EC2 或 Auto Scaling 動作。

2. 設定 Amazon SNS 通知。您可以在建立 CloudWatch 警示時，包含在警示變更狀態時傳送通知的 Amazon SNS 主題。
3. 在 EventBridge 中建立規則 且此規則會比對指定的 CloudWatch 警示。每個規則都支援包括 Lambda 函數的多個目標。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 EventBridge 目標的詳細資訊，請參閱 EventBridge 目標。

## 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 洞見
- 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：

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

相關範例：

- One Observability 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，以調用適當動作，採取這些動作以維持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：自動化警示處理可以提高系統復原能力。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風險暴露等級：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，您可以使用 AWS CLI (使用 Amazon CloudWatch 命令 `describe-alarms`)。根據您設定的警示數量而定，您可能需要使用分頁來擷取每個呼叫的警示子集，或者，您也可以使用 AWS SDK，使用 API 呼叫取得警示。
2. 記錄所有警示動作：不論是手動還是自動的，請更新執行手冊與所有警示及其動作。AWS Systems Manager 會提供預先定義的執行手冊。如需執行手冊的詳細資訊，請參閱使用執行手冊。如需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，請指定使用 CloudWatch SDK 的自動化動作。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。

   您也可以使用 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更。您可以建立規則以指出您感興趣的事件，以及當事件符合規則時要採取的動作。可以自動啟動的動作包括調用 AWS Lambda 函數、叫用 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. 標準作業程序 (SOP)：根據您的應用程式元件，AWS Resilience Hub 建議使用多個 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。您也可以根據 Resilience Hub 建議來建構 SOP，但您需要具有相關彈性政策的 Resilience Hub 應用程式，以及對該應用程式進行歷史彈性評估。SOP 的建議會由彈性評估產生。

   Resilience Hub 會與 Systems Manager 搭配運作，透過提供一些可以作為這些 SOP 基礎的 SSM 文件來自動執行 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：您可以使用 Amazon DevOps Guru 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 時，您可以近乎即時地監控多個來源的營運資料串流 (包括 Amazon CloudWatch 指標、AWS Config、AWS CloudFormation 和 AWS X-Ray)。您也可以使用 DevOps Guru 來自動地在 OpsCenter 中建立 OpsItems，並將事件傳送至 EventBridge 以進行其他自動化。

### 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 自動化
- 建立 EventBridge 規則，以透過 AWS 資源觸發事件
- One Observability 研討會
- Amazon Builders' Library：偵測分散式系統，以瞭解運作狀態
- 什麼是 Amazon DevOps Guru？
- 與自動化文件搭配使用 (程序手冊)

**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 將任何作業自動化
- AWS Resilience Hub 簡介
- 為 Amazon DevOps Guru 通知建立自訂票證系統
- 使用 Amazon DevOps Guru 啟用多帳戶洞見彙總功能

**相關範例：**

- 可靠性研討會
- Amazon CloudWatch 和 Systems Manager 研討會

## REL06-BP05 分析

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，您可使用此語言來分析日誌資料。Amazon CloudWatch Logs 還支援訂閱，而這些訂閱允許資料無縫流至 Amazon S3，您可使用 Amazon S3 或 Amazon Athena 來查詢資料。其也支援對大量格式的查詢。請參閱 請參閱 (位於 Amazon Athena 使用者指南中)，以取得詳細資訊。若要分析大型日誌檔集，您可以執行 Amazon EMR 叢集來執行 PB 級分析。

AWS 合作夥伴和第三方提供了許多工具，可用於彙總、處理、儲存和分析。這些工具包含 New Relic、Splunk、Loggly、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。

若未建立此最佳實務，暴露的風險等級為： 中

## 實作指引

- CloudWatch Logs Insights 可讓您以互動方式在 Amazon CloudWatch Logs 中搜尋和分析日誌資料。
  - 使用 CloudWatch Logs Insights 分析日誌資料
  - Amazon CloudWatch Logs Insights 範例查詢
- 使用 Amazon CloudWatch Logs 將日誌傳送至您可以在其中使用的 Amazon S3，或使用 Amazon Athena 來查詢資料。
  - 我要如何使用 Athena 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。這樣做可減少 Athena 針對每個查詢所分析的資料量。
      - 我要如何為 S3 儲存貯體建立生命週期政策？

## 資源

相關文件：

- Amazon CloudWatch Logs Insights 範例查詢
- 使用 CloudWatch Logs Insights 分析日誌資料

- 使用 Amazon CloudWatch Synthetics 和 AWS X-Ray 偵錯
- 我要如何為 S3 儲存貯體建立生命週期政策？
- 我要如何使用 Athena 分析 Amazon S3 伺服器存取日誌？
- 一個觀察工作坊
- Amazon Builders' Library：偵測分散式系統，以瞭解運作狀態

## REL06-BP06 定期進行審查

經常審查工作負載監控的實作方式，並根據重大事件和變更進行更新。

有效的監控是由關鍵業務指標推動。當業務優先事項變更時，確保您的工作負載中會包含這些指標。

稽核您的監控有助於您知道應用程式何時達到其可用性目標。根本原因分析需要能夠發現發生故障時的具體情況。AWS 提供的服務可讓您在事件發生時追蹤服務狀態：

- Amazon CloudWatch Logs： 您可以將日誌儲存在此服務中並檢查其內容。
- Amazon CloudWatch Logs Insights：是一項全受管服務，讓您可以在數秒內分析大量日誌。其可為您提供快速且互動式的查詢和視覺化。
- AWS Config： 您可以查看在不同時間點使用的 AWS 基礎設施。
- AWS CloudTrail： 您可以查看在什麼時間及透過什麼主體叫用了哪些 AWS API。

在 AWS，我們每週舉行一次會議， 以審查營運效能 及在團隊之間分享經驗。由於 AWS 旗下有太多團隊，我們建立了 The Wheel 以隨機挑選要審查的工作負載。建立定期執行營運效能審查和知識共享的機制，可增強您從營運團隊獲得更高效能的能力。

常用的反模式：

- 僅收集預設指標。
- 設定監控策略，但絕不檢閱。
- 部署重大變更時不討論監控。

建立此最佳實務的優勢： 定期檢閱監控可預期潛在問題，而不是在預期問題實際發生時對通知作出反應。

若未建立此最佳實務，暴露的風險等級為： 中

## 實作指引

- 為工作負載建立多個儀表板。您必須擁有最上層儀表板，其中包含關鍵業務指標，以及經您確認與工作負載預估運作狀態最相關的 (因為用量不同) 技術指標。您也應該有可以檢查各種應用程式層和相依性的儀表板。
  - 使用 Amazon CloudWatch 儀表板
- 排程及定期檢閱工作負載儀表板。定期執行儀表板檢查。您對於檢查深度可能有不同規律。
  - 檢查指標中的趨勢。比較指標值與歷史值，以查看是否有可能指出某項需要調查的趨勢。這些範例包括：增加延遲、減少主要業務功能，以及增加失敗回應。
  - 檢查指標中的異常值/異常。平均值或中位數可以遮罩異常值。查看時間範圍內的最高和最低值，並調查極端分數的原因。隨著您持續消除這些原因，降低極端的定義可讓您持續改善工作負載效能的一致性。
  - 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。

## 資源

相關文件：

- Amazon CloudWatch Logs Insights 範例查詢
- 使用 Amazon CloudWatch Synthetics 和 AWS X-Ray 偵錯
- 一個觀察工作坊
- Amazon Builders' Library：偵測分散式系統，以瞭解運作狀態
- 使用 Amazon CloudWatch 儀表板

# REL06-BP07 透過您的系統監控請求的端對端追蹤

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。

預期成果： 對所有元件進行全面追蹤的工作負載可輕易進行偵錯，藉由簡化根本原因探索來改善錯誤和延遲的 平均解決時間 (MTTR)。端對端追蹤可縮短探索受影響的元件時間，並詳細剖析錯誤或延遲的根本原因。

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，在未追蹤 AWS Lambda 的情況下，團隊可能無法清楚了解尖峰工作負載中的冷啟動所造成的延遲。
- 未對追蹤設定綜合金絲雀或實際使用者監控 (RUM)。若沒有金絲雀或 RUM，追蹤分析就會省略用戶端互動遙測，而產生不完整的效能設定檔。
- 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，應使用雲端原生追蹤 SDK 來檢測不是雲端原生的元件，或使用第三方工具來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。諸如何時應叫用災難復原 (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)。在端對端追蹤組態中包含綜合金絲雀和實際使用者監控，以測量遠端用戶端互動和延遲，以便您根據服務水準協議和目標正確評估系統效能。

您可以使用 AWS X-Ray 和 Amazon CloudWatch 應用程式監控 檢測服務，在請求通過您的應用程式時提供請求的完整檢視。X-Ray 會收集應用程式遙測，並可讓您在承載、函數、追蹤、服務、API 間將其視覺化和加以篩選，並且可針對無程式碼或低程式碼的系統元件開啟。CloudWatch 應用程式監控包含 ServiceLens，可將您的追蹤與指標、日誌和警示整合在一起。CloudWatch 應用程式監控也包含用來監控端點和 API 的綜合功能，以及用來檢測 Web 應用程式用戶端的實際使用者監控。

## 實作步驟

- 對所有受支援的原生服務使用 AWS X-Ray，例如 Amazon S3、AWS Lambda 和 Amazon API Gateway。這些 AWS 服務可使用基礎設施即程式碼、AWS SDK 或 AWS Management Console 來啟用具有組態切換的 X-Ray。
- 檢測應用 適用於 Open Telemetry 的 AWS Distro 和 X-Ray 或第三方收集代理程式。
- 檢閱 AWS X-Ray 開發人員指南， 以了解程式設計語言特定的實作方式。這些文件章節會詳細說明如何檢測 HTTP 請求、SQL 查詢，以及應用程式設計語言特有的其他程序。

- 將 X-Ray 追蹤用於 Amazon CloudWatch 綜合金絲雀 和 Amazon CloudWatch RUM 以分析最終使用者用戶端通過您下游 AWS 基礎設施的請求路徑。
- 根據資源運作狀態和金絲雀遙測來設定 CloudWatch 指標和提醒，以便團隊快速收到問題的提醒，然後可使用 ServiceLens 深入探討追蹤和服務圖。
- 啟用第三方追蹤工具的 X-Ray 整合，例如 Datadog、New Relic或 Dynatrace (如果您將第三方工具用於主要追蹤解決方案)。

## 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 AWS X-Ray 偵錯
- Amazon 建置者資料中心：偵測分散式系統，以了解運作狀態
- 整合 AWS X-Ray 與其他 AWS 服務
- 適用於 OpenTelemetry 的 AWS Distro 和 AWS X-Ray
- Amazon CloudWatch：使用綜合監控
- Amazon CloudWatch：使用 CloudWatch RUM
- 設定 Amazon CloudWatch 綜合金絲雀和 Amazon CloudWatch 警示
- 可用性和超越各種可能：了解和改善 AWS 上分散式系統的恢復能力

相關範例：

- One Observability 工作坊

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 工作負載 提供自動新增或移除資源的彈性，以便隨時盡可能符合目前需求。

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 對工作負載執行負載測試

## REL07-BP01 取得或擴展資源時使用自動化

替換受損的資源或擴展工作負載時，請使用 Amazon S3 和 AWS Auto Scaling 等受管的 AWS 服務進行自動化程序。您還可以使用第三方工具和 AWS 開發套件來自動調整規模。

受管 AWS 服務包括 Amazon S3、Amazon CloudFront、AWS Auto Scaling、AWS Lambda、Amazon DynamoDB、AWS Fargate 和 Amazon Route 53。

AWS Auto Scaling 讓您可以偵測和取代受損的執行個體。這也讓您可以為資源建立擴展計畫，包括 Amazon EC2 執行個體和 Spot 機群叢集、 Amazon ECS 任務、 Amazon DynamoDB 資料表和索引，以及 Amazon Aurora 複本。

擴展 EC2 執行個體時，請確保您使用多個可用區域 (最好至少有三個) 並新增或移除容量，以便在這些可用區域之間維持平衡。ECS 任務或 Kubernetes Pod (使用 Amazon Elastic Kubernetes Service 時) 也應該分散到多個可用區域。

使用 AWS Lambda 時，執行個體會自動擴展。每次收到函數的事件通知時，AWS Lambda 會在其運算叢集內快速找到可用容量，然後執行您的程式碼，直到達到配置的並行為止。您需要確保已在特定 Lambda 和 Service Quotas 中設定必要的並行。

Amazon S3 會自動調整規模以處理高請求率。例如，您的應用程式可以在儲存貯體的每個字首達到每秒至少 3,500 個 PUT/COPY/POST/DELETE 或 5,500 個 GET/HEAD 請求。儲存貯體中的字首數量沒有限制。您可以透過平行化讀取來提升讀取或寫入效能。例如，如果您在 Amazon S3 儲存貯體中建立 10 個字首來平行讀取，則可以將讀取效能擴展為每秒 55,000 個讀取請求。

設定和使用 Amazon CloudFront 或受信任的內容交付網路 (CDN)。CDN 可以提供更快的最終使用者回應時間，而且可以為快取中的內容請求提供服務，因此可減少擴展工作負載的需求。

常用的反模式：

- 實作 Auto Scaling 群組以進行自動修復，但不實作彈性。
- 使用自動調整規模來回應大幅增加的流量。
- 部署高度狀態應用程式，免除彈性選項。

建立此最佳實務的優勢： 自動化會移除在部署和除役資源時可能出現的手動錯誤。自動化可免除因部署或除役需求回應緩慢而造成成本超支和拒絕服務的風險。

若未建立此最佳實務，暴露的風險等級： 高

## 實作指引

- 設定和使用 AWS Auto Scaling。這會監控您的應用程式並自動調整容量，以盡可能低的成本維持穩定、可預測的效能。您可以使用 AWS Auto Scaling 為多個服務的多個資源設定應用程式擴展。
  - 什麼是 AWS Auto Scaling？
    - 在 Amazon EC2 執行個體和 Spot 機群、Amazon ECS 任務、Amazon DynamoDB 表格和索引、Amazon Aurora 複本和 AWS Marketplace 設備上設定 Auto Scaling (如適用)。
      - 使用 DynamoDB Auto Scaling 自動管理輸送量
        - 使用服務 API 操作來指定警示、擴展原則、準備時間和冷卻時間。
- 使用 Elastic Load Balancing。負載平衡器可以按路徑或網路連線來分配負載。
  - 什麼是 Elastic Load Balancing？
    - Application Load Balancers 可以按路徑分配負載。
      - 什麼是 Application Load Balancer？
        - 設定 Application Load Balancer，以根據網域名稱下的路徑將流量分配到不同的工作負載。
        - Application Load Balancers 可用於以與 AWS Auto Scaling 整合的方式分配負載，以管理需求。
          - 搭配 Auto Scaling 群組使用負載平衡器

    - Network Load Balancer 可以透過連線分配負載。
        - 什麼是 Network Load Balancer？
            - 設定 Network Load Balancer，以使用 TCP 將流量分配到不同的工作負載，或為您的工作負載分配固定的 IP 地址集。
            - Network Load Balancer 可用於以與 AWS Auto Scaling 整合的方式分配負載，以管理需求。
- 使用高度可用的 DNS 供應商。DNS 名稱讓您的使用者可以輸入名稱 (而不是 IP 地址) 來存取您的工作負載，並將此資訊分發到已定義的範圍 (通常是工作負載的所有使用者)。
    - 使用 Amazon Route 53 或信任的 DNS 供應商。
        - 什麼是 Amazon Route 53？
    - 使用 Route 53 來管理您的 CloudFront 分發和負載平衡器。
        - 確定要管理的網域和子網域。
        - 使用 ALIAS 或 CNAME 紀錄建立適當的紀錄集。
            - 處理記錄
- 使用 AWS 全球網路，優化從使用者到應用程式的路徑。AWS Global Accelerator 可持續監控應用程式端點的運作狀態，並在 30 秒內將流量重新導向到運作狀態良好的端點。
    - AWS Global Accelerator 是一種可改善具備當地或全球使用的應用程式可用性和效能的服務。它提供靜態 IP 地址，做為單一或多個 AWS 區域 (例如 Application Load Balancers、Network Load Balancers 或 Amazon EC2 執行個體) 應用程式端點的固定進入點。
        - 什麼是 AWS Global Accelerator？
- 設定和使用 Amazon CloudFront 或受信任的內容交付網路 (CDN)。內容交付網路可以提供更快的最終使用者回應時間，並且可以處理可能導致不必要的工作負載擴展的內容請求。
    - 什麼是 Amazon CloudFront？
        - 為您的工作負載設定 Amazon CloudFront 分發，或使用第三方 CDN。
            - 您可以限制對工作負載的存取，使其只能透過在端點安全群組或存取政策中使用 CloudFront 的 IP 範圍從 CloudFront 存取。

## 資源

相關文件：

- APN 合作夥伴：可以幫助您建立自動化運算解決方案的合作夥伴
- 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 自動管理輸送量
- 搭配 Auto Scaling 群組使用負載平衡器
- 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？
- 處理記錄

## 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 AWS CLI 修改 DynamoDB 資料表的輸送量)。但是，應該盡可能使用自動化 (請參閱取得或擴展資源時使用自動化)。

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

未建立此最佳實務時的風險暴露等級：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。定義會擴展所需資源的手動或自動程序。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故障。

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 AWS Auto Scaling 的元件可以使用擴展計劃來設定用於擴展資源的一組指示。如果您使用 AWS CloudFormation 或將標籤新增至 AWS 資源，則可以針對每個應用程式的不同資源集設定擴展計畫。Auto Scaling 為針對每個資源自訂擴展的策略提供建議。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。如需詳細資訊，請參閱擴展計畫的運作方式。
  - Amazon EC2 Auto Scaling 可確認您擁有正確數量的 Amazon EC2 執行個體可處理應用程式的負載。您可以建立稱為 Auto Scaling 群組的 EC2 執行個體集合。您可以在每個 Auto Scaling 群組中指定執行個體的最小和最大數量，而 Amazon EC2 Auto Scaling 可確保您的群組大小永遠不會低於或高於這些限制。如需詳細資訊，請參閱什麼是 Amazon EC2 Auto Scaling？
  - Amazon DynamoDB 自動擴展使用 Application Auto Scaling 服務代替您動態調整佈建的輸送容量，以回應實際的流量模式。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，以在不需限流的情況下處理突然增加的流量。如需詳細資訊，請參閱使用 DynamoDB 自動擴展自動管理輸送容量。

### 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

相關文件：

- AWS Auto Scaling：擴展計畫的運作方式
- 使用 DynamoDB 自動擴展自動管理輸送容量
- 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

主動擴展資源以滿足需求並避免可用性影響。

許多 AWS 服務會自動調整規模以滿足需求。如果使用 Amazon EC2 執行個體或 Amazon ECS 叢集，您可以將這些叢集的自動調整規模功能設定為根據與工作負載需求對應之用量指標來執行。對於 Amazon EC2，平均 CPU 使用率、負載平衡器請求計數或網路頻寬可用於擴展 (或縮減) EC2 執行個體。對於 Amazon ECS，平均 CPU 使用率、負載平衡器請求計數和記憶體使用率可用於橫向擴展 (或

縮減) ECS 任務。透過在 AWS 上使用 Target Auto Scaling，自動調整規模裝置的作用就像家用恆溫器一樣，可新增或移除資源以維持您指定的目標值 (例如，70% 的 CPU 使用率)。

AWS Auto Scaling 也可以執行 Predictive Auto Scaling，其會使用機器學習分析每個資源的歷史工作負載，並定期預測未來兩天的未來負載。

「利特爾法則」有助於計算您需要的運算執行個體 (EC2 執行個體、並行 Lambda 函數等) 的數量。

L = λW

L = 執行個體數量 (或系統中的平均並行)

λ = 請求到達時的平均速率 (請求/秒)

W = 每個請求在系統中花費的平均時間 (秒)

例如，在 100 rps 時，如果每個請求需要 0.5 秒才能處理，您就需要 50 個執行個體才能因應需求。

若未建立此最佳實務，暴露的風險等級為： 中

## 實作指引

- 偵測到工作負載需要更多資源時取得資源。主動擴展資源以滿足需求並避免可用性影響。
  - 計算處理指定請求率所需的運算資源 (運算並行)。
    - 說說「利特爾法則」的故事
  - 當您有使用的歷史模式時，請設定 Amazon EC2 Auto Scaling 的排程擴展。
    - Amazon EC2 Auto Scaling 的排程擴展
  - 使用 AWS 預測擴展。
    - EC2 的預測擴展，採用機器學習技術

## 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 自動管理輸送量
- EC2 的預測擴展，採用機器學習技術
- Amazon EC2 Auto Scaling 的排程擴展

- 說說「利特爾法則」的故事
- 什麼是 Amazon EC2 Auto Scaling？

## REL07-BP04 對工作負載執行負載測試

採用負載測試方法，衡量擴展活動是否能達到工作負載要求。

重要的是執行持續的負載測試。負載測試應探索中斷點並和測試工作負載的效能。AWS 讓您可以輕鬆設定臨時測試環境，以塑造生產工作負載的規模。在雲端中，您可隨需建立生產規模的測試環境、完成測試，再將資源除役。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段，應將生產中的負載測試納入考慮，並動員所有在場人員共同分析結果並處理可能出現的問題。

常見的反模式：

- 在與生產組態不同的部署上執行負載測試。
- 只對工作負載的個別部分而非整個工作負載執行負載測試。
- 使用請求的子集而非代表的實際請求集合來執行負載測試。
- 依據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效，並能夠識別要監看哪些指標，以便及時識別接近該負載的跡象，從而解決問題並避免由此失效造成的影響。

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該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必須新增或移除資源。
  - 在 AWS 上執行分散式負載測試：模擬數千名連線的使用者
    - 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。
    - 實作負載驅動程式。您可以使用自訂程式碼、開放原始碼或商業軟體實作負載驅動程式。
    - 一開始用小容量執行負載測試。您在負載驅動到較小容量 (可能和單一執行個體或容器一樣小) 之後立刻發現一些影響。

- 對較大容量執行負載測試。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：

- 在 AWS 上執行分散式負載測試：模擬數千名連線的使用者
- 對應用程式執行負載測試

相關影片：

- AWS Summit ANZ 2023：透過 AWS 分散式負載測試讓您放心加快腳步

# 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，則難以預測這些變更的效果，或是解決肇因於這些變更的問題。

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 將恢復能力測試整合為部署的一部分
- 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成果的預定義程序。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，從有效的手動流程開始，以程式碼實作並在適當時將其觸發為自動執行。

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。

常用的反模式：

- 在生產環境中對組態執行非計畫中的變更。
- 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提高您的可信度。

若未建立此最佳實務，暴露的風險等級為： 高

## 實作指引

- 透過在執行手冊中記錄程序，對熟知的事件做出一致且迅速的回應。
  - 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、解偶的範本。
    - 確定實作的許可、範本和負責方。
      - 使用 AWS Identity and Access Management 控制存取
    - 使用原始檔控制 (例如 AWS CodeCommit 或受信任的第三方工具) 進行版本控制。
      - 什麼是 AWS CodeCommit？

## 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？
- 什麼是 AWS CodeCommit？

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

功能測試會作為自動化部署的一部分執行。如果未符合成功條件，則會終止或回復管道。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。理想情況下，這是做為部署管道的一部分來完成。

期望的結果：您可以使用自動化來執行功能測試，而相關的測試資料可縮短測試持續時間和費用，並提高測試結果的準確性。您可以將功能測試整合為部署流程的一部分，這可協助您自動執行發布管道，以實現快速且可靠的應用程式和基礎設施更新。

常見的反模式：

- 您在部署管道以外手動執行測試。
- 您利用手動緊急工作流程跳過自動化作業中的測試步驟。
- 您為了加快時間進程，而沒有遵循既定的計畫和流程。

建立此最佳實務的優勢：功能測試會驗證系統是否根據指定的要求運作。該測試用於一致地驗證元件的預期工作順序，例如使用者介面、API、資料庫和原始程式碼。當您檢查系統的這些元件時，功能測試會驗證每個功能是否符合預期的行為，進而保護使用者期望和軟體的完整性。將功能測試整合成定期部署的一部分，並使用自動化部署所有變更，進而降低導入人為錯誤的可能性。

未建立此最佳實務時的風險暴露等級：高

### 實作指引

將功能測試整合為部署的一部分。功能測試會作為自動化部署的一部分執行。如果不符合成功條件，則管道會停止或復原。AWS CodePipeline 會為自動化測試提供連續交付管道，允許測試者自動化整個測試和部署流程。它會與 AWS CodeBuild 和 AWS CodeDeploy 等 AWS 服務整合，以將軟體開發生命週期的組建、測試和部署階段自動化。

實作步驟

- 設定管道：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。
  - 定義來源：您可以使用 AWS CodePipeline，自動從 GitHub、AWS CodeCommit 或 Bitbucket 等版本控制系統中檢索原始程式碼，該系統會確認始終使用最新程式碼進行測試。
  - 自動化組建和測試：AWS CodeBuild 可以自動建置和測試您的程式碼，並產生測試報告。該服務支援受歡迎的測試架構，如 JUnit、NUnit 和 TestNG。
  - 部署程式碼：建置並測試程式碼後，AWS CodeDeploy 可以將其部署到您的測試環境，包括 Amazon EC2 執行個體、AWS Lambda 函數或內部部署伺服器。
  - 監控管道：AWS CodePipeline 可以追蹤管道的進度和每個階段的狀態。您也可以根據測試執行狀態使用品質檢查機制來封鎖管道。您也可以接收任何關於管道階段失敗或管道完成的通知。

## 資源

相關文件：

- 搭配 AWS CodeBuild 使用 AWS CodePipeline 測試程式碼和執行組建版本
- 在 AWS CodeBuild 中記錄和監控
- 功能測試的指標

# REL08-BP03 將恢復能力測試整合為部署的一部分

透過特意導入系統故障來整合恢復能力測試，以衡量其在發生中斷情況時的能力。恢復能力測試與通常在部署週期中整合的單元和功能測試不同，恢復測試僅專注於識別系統中非預期的故障。在試生產期間安全地開始使用恢復能力測試整合的同時，設定目標並在生產中實作這些測試，以做為演練日作業的一部分。

期望的結果：恢復能力測試有助於建立對系統承受生產降級能力的信心。實驗能識別可能導致故障的弱點，進而幫助您改善系統，以自動且有效率地減輕故障和降級的衝擊。

常見的反模式：

- 部署流程中缺乏可觀測性和監控性
- 依賴人力解決系統故障
- 品質分析機制不佳

- 專注於系統中已知問題，以及缺乏識別任何未知問題的實驗
- 找到故障點，但沒有將其解決
- 沒有調查結果和執行手冊的文件

建立最佳實務的優勢：整合在您的部署中的恢復能力測試有助於識別系統中的未知問題，否則這些問題會被忽視，進而導致生產中斷。在系統中識別這些未知問題可幫助您記錄調查結果、將測試整合到 CI/CD 流程中，以及製作執行手冊，藉由高效率的可重複機制來簡化緩解措施。

未建立此最佳實務時的風險暴露等級：中

## 實作指引

可在系統部署中整合的最常見的恢復能力測試形式，是災難復原和混沌工程。

- 在任何重大部署中包括災難復原計畫和標準作業程序 (SOP) 的更新。
- 將可靠性測試整合到您的自動化部署管道中。如 AWS Resilience Hub 等的這類服務可以整合到 CI/CD 管道中，以建立持續的恢復能力評估，這些評估作業會自動作為每個部署的一部分。
- 在 AWS Resilience Hub 中定義您的應用程式。恢復能力評估會產生程式碼片段，可協助您建立復原程序做為應用程式的 AWSSystems Manager 文件，並提供建議的 Amazon CloudWatch 監視器和警示清單。
- 一旦更新了您的 DR 計畫和 SOP，請完成災難復原測試，以確認它們是否有效。災難復原測試可協助您判斷是否可以在事件後還原系統，並恢復正常運作。您可以模擬各種災難復原策略，並識別計畫是否足以滿足您的正常運作時間需求。常見的災難復原策略包括備份和還原、指示燈、冷待機、熱待機、熱待命和主動-主動，而且這些策略在成本和複雜性方面均不相同。在災難復原測試之前，建議您定義復原時間點目標 (RTO) 和復原點目標 (RPO)，以簡化模擬策略的選擇。AWS 提供災難復原工具 (例如 AWS Elastic Disaster Recovery)，協助您開始規劃和測試。
- 混沌工程實驗會導致系統中斷，例如網路中斷和服務故障。透過模擬受控故障，可以發現系統的漏洞，同時遏止注入故障的影響。像其他策略一樣，在非生產環境中使用服務 (例如 AWS Fault Injection Service) 執行受控故障模擬，可在部署至生產之前先得到十足信心。

## 資源

相關文件：

- 使用恢復能力測試實驗失敗以先做好復原準備
- 利用 AWS Resilience Hub 和 AWS CodePipeline 持續評估應用程式恢復能力

- AWS 上的災難復原 (DR) 架構，第 1 部分：在雲端中復原的策略
- 使用混沌工程驗證工作負載的恢復能力
- 混沌工程的原則
- 混沌工程研討會

相關影片：

- AWS re:Invent 2020：使用混沌工程測試恢復能力
- 利用 AWS 故障注入服務提高應用程式恢復能力
- 利用 AWS Resilience Hub 準備並保護您的應用程式免受中斷

## 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時，就地修改不得在工作負載內執行基礎設施資源。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：由於不同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更換為已知且具有版本控制的組態，可將基礎設施設定為已知、經過測試且可信的狀態，以避免組態偏移。
- 不可部分完成的可靠部署：部署不是會成功完成，就是完全沒有變更，以提高部署程序的一致性和可靠性。
- 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：透過不允許對基礎設施進行變更，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風險暴露等級：中

## 實作指引

### 自動化

在定義不可變基礎設施的部署策略時，建議您盡可能使用自動化，以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 使用自動化部署變更和自動化安全、無人為介入的部署。

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化、描述性和宣告式的方式加以定義，並儲存在原始程式碼控制系統中。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：

金絲雀部署：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 (金絲雀) 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。您可以使用允許進行金絲雀部署的部署組態來設定 AWS CodeDeploy。

藍/綠部署：與金絲雀部署類似，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。您可以使用允許進行藍/綠部署的部署組態來設定 AWS CodeDeploy 及 AWS Elastic Beanstalk。

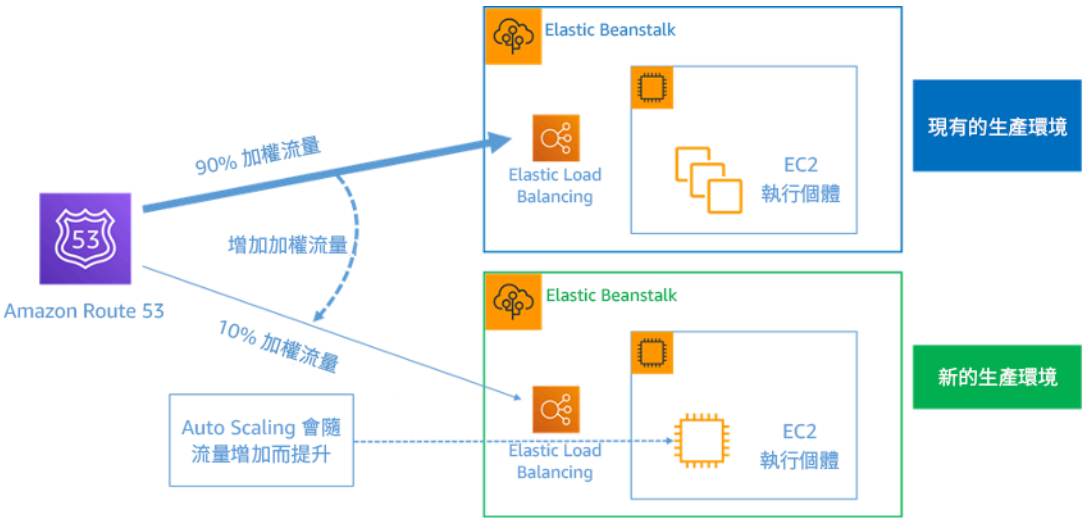



圖 8：使用 AWS Elastic Beanstalk 和 Amazon Route 53 進行藍/綠部署

偏移偵測

偏移是指會導致基礎設施資源具有與預期不同的狀態或組態的任何變更。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) 指定什麼人或項目可以在 AWS 中存取服務和資源、集中管理精細許可，以及分析存取動作以完善 AWS 內的許可。
- 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 簡介白皮書所述，自動化是 AWS 服務的基石，且所有服務、功能和產品內部都支援自動化。
  - 預先封裝 Amazon Machine Image (AMI) 可以加快其啟動時間。EC2 Image Builder 是全受管的 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 服務可在熟悉的伺服器 (例如 Apache、NGINX、Passenger 和 IIS) 上，快速部署和擴展使用 Java、.NET、PHP、Node.js、Python、Ruby、Go 和 Docker 所開發的 Web 應用程式。
    - AWS Proton 可協助平台團隊連接並協調開發團隊為了進行基礎設施佈建、程式碼部署、監控和更新所需的所有不同工具。AWS Proton 可讓您以基礎設施即程式碼的方式，自動佈建和部署無伺服器和容器型的應用程式。

- 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 可協助開發人員以有序且可預測的方式建立 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 (例如 AWS Cloud9) 以 JSON 或 YAML 撰寫資源、將其簽入版本控制系統，然後 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) 是開放原始碼架構，您可以用它在 AWS 上建置無伺服器應用程式。AWS SAM 會與其他 AWS 服務整合，並且是 AWS CloudFormation 的延伸。
  - AWS Cloud Development Kit (AWS CDK) 是開放原始碼的軟體開發架構，可讓您使用熟悉的程式設計語言來建模和佈建雲端應用程式資源。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 Cloud Control API 引入了一組常見的建立、讀取、更新、刪除和列出 (CRUDL) API，以協助開發人員以簡單且一致的方式管理其雲端基礎設施。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。
  - 金絲雀部署：
    - 設定 API Gateway 金絲雀版本部署
    - 使用 AWS App Mesh 為 Amazon ECS 建立具有金絲雀部署的管道
  - 藍/綠部署：AWS 上的藍/綠部署白皮書會描述用來實作藍/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，請參閱偵測堆疊和資源的未受管組態變更。

## 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

- 實作警示以自動偵測 AWS CloudFormation 堆疊中的偏移

相關影片：

- AWS re:Invent 2020：透過不可變實現可靠性、一致性和可信度

## 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。

期望的結果：您可以透過廣泛的試生產測試、自動回復和分期生產部署，將自動化部署安全性建置在發布流程中。此自動化作業可將部署失敗造成對生產的潛在影響降到最低，而開發人員不再需要主動監視生產的部署。

常見的反模式：

- 您執行手動變更。
- 您利用手動緊急工作流程跳過自動化作業的步驟。
- 您為了加快時間進程，而沒有遵循既定的計畫和流程。
- 您快速執行後續部署，卻不允留封裝時間。

建立此最佳實務的優勢：您使用自動化作業部署所有變更時，可以免除導入人為錯誤的可能性，並提供在變更生產前進行測試的能力。在生產推送之前執行此流程，確認您的計畫是否已完成。此外，自動回復到您的發布流程有利於找出生產問題，並將工作負載回復到先前工作的操作狀態。

未建立此最佳實務時的風險暴露等級：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。採用持續整合及持續交付/部署 (CI/CD) 的文化，其中提交或程式碼變更會經過各種自動化階段關卡 (從建置和測試階段到生產環境部署)。

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，但是同樣出於這個原因，我們建議您將最困難的程序自動化。

### 實作步驟

您可以依照下列步驟自動執行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 AWS CodeCommit，建立安全的 Git 型儲存庫。
- 設定持續整合服務以編譯原始程式碼、執行測試以及建立部署成品：若要為此目的設定建置專案，請參閱使用主控台開始使用 AWS CodeBuild。
- 設定部署服務以自動執行應用程式部署，並處理應用程式更新的複雜度，而不需依賴容易出錯的手動部署：AWS CodeDeploy 將軟體自動部署到各種運算服務，例如 Amazon EC2、AWS Fargate、AWS Lambda 和您的內部部署伺服器。若要設定這些步驟，請參閱開始使用 CodeDeploy。
- 設定持續交付服務，將您發布的管道自動化，以實現更快、更可靠的應用程式和基礎設施更新：考慮使用 AWS CodePipeline 來協助您自動發布管道。如需詳細資訊，請參閱CodePipeline 教學課程。

## 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
- 利用 AWS CodeCommit、AWS CodeBuild、AWS CodeDeploy 和 AWS CodePipeline 完成 CI/CD
- 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：

- AWS Summit 2019：AWS 上的 CI/CD

# 失敗管理

> 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可靠，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：已根據關鍵性識別和分類資料來源。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：識別需要備份的位置並實作機制來建立備份，或者能夠從外部源重現資料，可以改善在中斷期間還原和復原資料的能力。

未建立此最佳實務時的風險暴露等級：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 服務提供將備份複製到另一個 AWS 區域 的能力。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 可讓您從內部部署、跨可用區域或跨區域複製完整伺服器工作負載並且維護持續資料保護，使用以秒數測量的復原點目標 (RPO)。

Amazon S3 可以用作自行受管和 AWS 受管資料來源的備份目的地。Amazon EBS、Amazon RDS 和 Amazon DynamoDB 等 AWS 服務具有內建功能來建立備份。也可以使用第三方備份軟體。

內部部署資料可以使用 AWS Storage Gateway 或 AWS DataSync 備份到 AWS 雲端。Amazon S3 儲存貯體可以用來在 AWS 上存放此資料。Amazon S3 提供多個儲存層，例如 Amazon S3 Glacier 或 S3 Glacier Deep Archive，來減少資料儲存的成本。

您能夠從其他資源重現資料來符合資料復原需求。例如，Amazon ElastiCache 複本節點或 Amazon RDS 讀取複本可以用來重現資料，如果主要節點遺失的話。如果像這樣的來源可以用來符合您的復原時間目標 (RTO) 和復原點目標 (RPO)，您可能不需要備份。另一個範例，如果使用 Amazon EMR，可能不需要備份 HDFS 資料存放區，只要您可以從 Amazon S3 將資料重現到 Amazon EMR。

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。資料可以存放在多個資源上，例如資料庫、磁碟區、檔案系統、記錄系統和物件儲存。請參閱資源區段以尋找存放資料所在之不同 AWS 服務的相關文件，以及這些服務提供的備份功能。
2. 根據關鍵性將資料來源分類。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 是受管服務，可以在 AWS 上建立各種資料來源的備份。AWS Elastic Disaster Recovery 會處理對 AWS 區域 的自動化次秒級資料複寫。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。請參閱以下所列的資源，以取得如何從各種 AWS 服務建立資料備份的相關資訊。
4. 對於未備份的資料，請建立資料重現機制。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 載入至資料倉儲 (如 Amazon Redshift) 或 MapReduce 叢集 (如 Amazon EMR)，對該資料執行分析，則這可能是可從其他來源重現的資料範例。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. 建立備份資料的規律。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？
- 什麼是磁碟區閘道？
- 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 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
- 跨區域複寫，使用 Amazon S3
- EFS-to-EFS AWS Backup
- 將日誌資料匯出至 Amazon S3
- 物件生命週期管理
- DynamoDB 的隨需備份和還原
- 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防護
- AWS Backup 示範：跨帳戶和跨區域備份
- AWS re:Invent 2019：深入探討 AWS Backup，ft.Rackspace (STG341)

相關範例：

- Well-Architected 實驗室：實作 Amazon S3 雙向跨區域複寫 (CRR)
- Well-Architected 實驗室：測試備份並還原資料
- Well-Architected 實驗室：透過適用於分析工作負載的容錯恢復進行備份和還原
- Well-Architected 實驗室：災難復原 - 備份和還原

## 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風險暴露等級：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 磁碟區上使用加密。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，但金鑰受到區域限定。您可以在設定複寫時指定目的地金鑰。
- 選擇要使用預設或自訂適用於 Elastic Disaster Recovery 的 Amazon EBS 加密。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 中存放的加密金鑰，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
- 管理加密表格
- 安全支柱 – AWS Well Architected Framework
- 什麼是 AWS Elastic Disaster Recovery？

相關範例：

- Well-Architected 實驗室：實作 Amazon S3 雙向跨區域複寫 (CRR)

## 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：以建立的規律建立資料來源備份的自動化流程。

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：自動化備份可確保它們根據您的 RPO 定期進行備份，如果未進行備份則會提醒您。

未建立此最佳實務時的風險暴露等級：中

## 實作指引

AWS Backup 可以用來建立各種 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。這些服務也會提供原生備份功能。提供自動備份與時間點復原的 AWS 服務包括 Amazon DynamoDB、Amazon RDS 和 Amazon Keyspaces (適用於 Apache Cassandra) – 這些可以還原至備份歷史記錄內的特定時間點。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 會提供連續備份與時間點復原。一旦啟用了 Amazon S3 版本控制，就會自動執行。Amazon Data Lifecycle Manager 可用於自動化建立、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. 識別資料來源，這是目前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. 針對工作負載判斷 RPO。如需詳細資訊，請參閱 REL13-BP01 定義停機和資料遺失的復原目標。
3. 使用自動化備份解決方案或受管服務。AWS Backup 是一種全受管服務，可讓您在雲端和內部部署環境輕鬆集中化和自動化 AWS 服務的資料保護。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。如需如何使用 AWS Backup 建立自動化備份的實作指引，請參閱測試備份並還原資料。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. 針對自動化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)，請考慮使用信任的第三方解決方案建立自動化備份。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，以及使用 Amazon EventBridge，以根據 RPO 的頻率執行它。

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：深入探討 AWS Backup，ft.Rackspace (STG341)

相關範例：

- Well-Architected 實驗室：測試備份並還原資料

## 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目標 (RTO) 和復原點目標 (RPO)。

預期成果：使用妥善定義的機制定期復原來自備份的資料，以確認可在工作負載的既定復原時間點目標 (RTO) 內復原。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：測試備份的復原確認可在需要時還原資料，而不必擔心資料可能丟失或損壞，也可確保還原和復原可在工作負載的 RTO 內進行，而且任何資料遺失都會落在工作負載的 RPO 內。

未建立此最佳實務時的風險暴露等級：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。隨著來源伺服器進行複寫，時間點狀態會根據設定的政策隨著時間進行編製。Elastic Disaster Recovery 可藉由針對測試和練習目的啟動執行個體，而不重新導向流量，協助您確認這些快照的完整性。

實作步驟

1. 識別資料來源，這些資料來源目前正在備份，以及這些備份的存放位置。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準則。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，根據資料關鍵性還原資料。如需實作指引，請參閱 REL13-BP01 定義停機和資料遺失的復原目標。
4. 評估您的復原功能。檢閱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，您可以執行工作負載的評定。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. 執行測試還原，使用在生產環境進行資料還原的目前建立程序。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，如果您是使用像是 AWS Backup 的受管服務，這可能就像是將備份還原到新資源一樣簡單。如果您使用 AWS Elastic Disaster Recovery，您可以啟動復原練習。
6. 從還原的資源驗證資料復原，根據您先前為資料驗證建立的準則。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. 測量還原和復原以還原和復原，並且與您的已建立 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 都會加上時間戳記，而且此資訊可用於 AWS CloudTrail。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。如果使用自動程序，則 Amazon DynamoDB 之類的服務可以用來存放此資訊。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 內，備份和還原動作稱為工作，而且這些工作包含時間戳記資訊做為其中繼資料的一部分，而此中繼資料可以用來測量還原和復原所需的時間。
8. 通知利害關係人，如果資料驗證失敗，或如果還原和復原所需的時間超出針對工作負載建立的 RTO。實作自動化來執行此動作時，例如在此實驗室中，像是 Amazon Simple Notification Service (Amazon SNS) 之類的服務可以用來將推送通知 (例如電子郵件或簡訊) 傳送給利害關係人。這些訊息也可以推送至傳訊應用程式，例如 Amazon Chime、Slack 或 Microsoft Teams，或用來使用 AWS Systems Manager OpsCenter 建立例如 OpsItems 的任務。
9. 將此程序自動化為定期執行。例如，服務 (例如 AWS Lambda 或 AWS Step Functions 中的狀態機器) 可以用來將還原和復原程序自動化，而且 Amazon EventBridge 可以用來定期觸發此自動化工作流程，如下面架構圖所示。進一步了解如何使用 AWS Backup 將資料復原驗證自動化。此外，這個 Well-Architected 實驗室會提供實作體驗，有關在這裡為數個步驟執行自動化的方式。

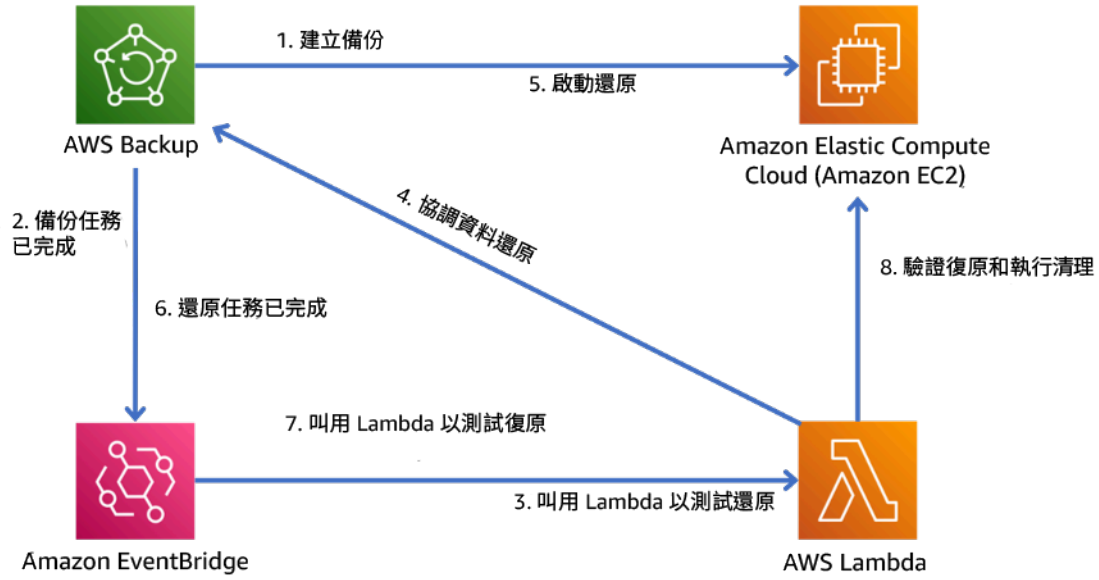



圖 9.自動的備份和還原程序

實作計劃的工作量：中到高，取決於驗證準則的複雜性。

## 資源

相關文件：

- 使用 AWS Backup 將資料復原驗證自動化
- 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 的隨需備份和還原
- 什麼是 AWS Backup？
- 什麼是 AWS Step Functions？
- 什麼是 AWS Elastic Disaster Recovery
- AWS Elastic Disaster Recovery

相關範例：

- Well-Architected 實驗室：測試備份並還原資料

# 使用故障隔離來保護您的工作負載

故障隔離界限會在工作負載內將失敗影響限制至有限數量的元件。界限外的元件不受失敗影響。使用多個故障隔離界限時，您可以限制對工作負載的影響。

最佳實務

- REL10-BP01 將工作負載部署至多個位置
- REL10-BP02 為您的多位置部署選取適當位置
- REL10-BP03 針對限制在單一位置的元件將復原自動化
- REL10-BP04 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。這些位置可以根據需要多樣化。

AWS 服務設計的基本原則之一是避免底層實體基礎設施中出現單點故障。這樣一來，我們將能建置可使用多個可用區域且能應對單一區故障的軟體和系統。同樣地，可將系統建置為能應對單一運算節點、單一儲存磁碟區或資料庫的單一執行個體的故障。建置依賴冗餘元件的系統時，務必要確保元件能獨立運行，而對於 AWS 區域而言，應能自主運行。具有冗餘元件的理論可用性計算，其優點只有在符合此條件時才有效。

可用區域 (AZ)

AWS 區域由多個可用區域組成，它們設計為彼此獨立作業。每個可用區域與其他可用區域是以有意義的實體距離隔開，從而可避免因火災、洪水和龍捲風等環境危害導致相關的失敗情境。每個可用區域也都具有獨立的實體基礎設施：可用區域內部和外部的公用電源專用連接、獨立的備用電源、獨立的機械服務以及獨立的網路連線。這種設計會將任何這些系統中的錯誤僅限制在受影響的可用區域。儘管在地理位置上是分開的，但可用區域位於啟用高輸送量、低延遲聯網的同一區域。整個 AWS 區域 (跨所有可用區域，由多個實體上獨立的資料中心組成) 可以視為工作負載的單一邏輯部署目標，包括同步複寫資料的能力 (例如，在資料庫之間)。這可讓您在主動/主動或主動/待命組態中使用可用區域。

可用區域是各自獨立的，因此當工作負載架構為使用多個區域時，工作負載的可用性也會隨之提高。一些 AWS 服務 (包括 Amazon EC2 執行個體資料平面) 會部署為嚴格的區域服務，其中它們與其所在的可用區域共享命運。不過，其他 AZ 中的 Amazon EC2 執行個體將不受影響並繼續運作。同樣地，如果可用區域中的失敗導致 Amazon Aurora 資料庫失敗，則未受影響 AZ 中的讀取副本 Aurora 執行個體可以自動提升為主要執行個體。另一方面，區域 AWS 服務 (例如 Amazon DynamoDB) 可內部使用主動/主動組態中的多個可用區域，以實現該服務的可用性設計目標，無需您設定 AZ 置放。

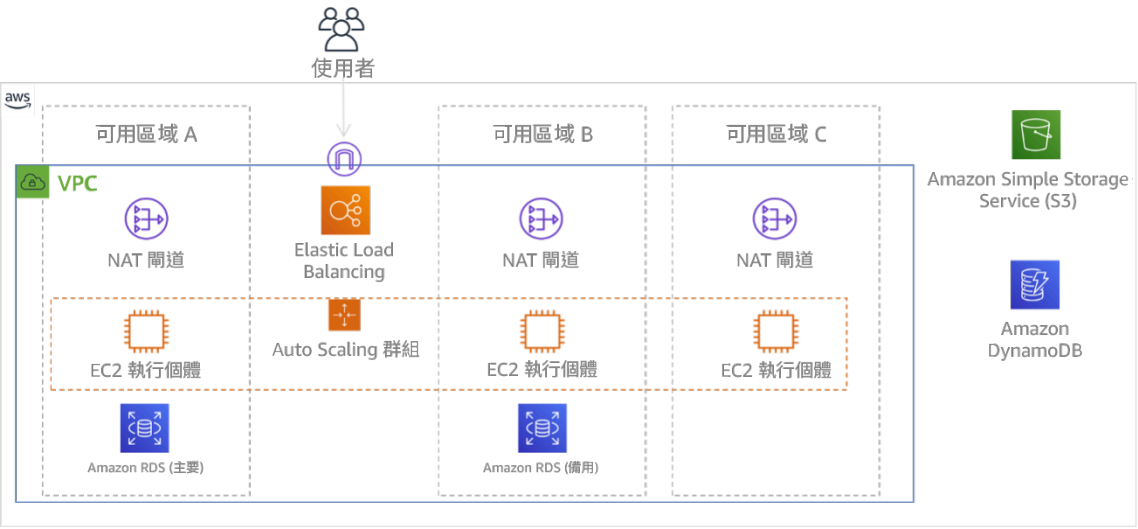



圖 9：跨三個可用區域部署的多層架構。請注意，Amazon S3 和 Amazon DynamoDB 一律自動採用異地同步備份策略。ELB 也會部署至全部三個區域。

儘管 AWS 控制平面通常有能力管理整個區域 (多個可用區域) 內的資源，但是某些控制平面 (包括 Amazon EC2 和 Amazon EBS) 能夠將結果篩選至單一可用區域。完成此操作後，僅在指定的可用區域中處理該請求，從而減少其他可用區域中的中斷風險。此 AWS CLI 範例說明僅從 us-east-2c 可用區域取得 Amazon EC2 執行個體資訊：

```
AWS ec2 describe-instances --filters Name=availability-zone,Values=us-east-2c
```

AWS Local Zones

AWS Local Zones 的作用與各自 AWS 區域內的可用區域類似，它們可在其中被選取為區域 AWS 資源 (如子網路和 EC2 執行個體) 的置放位置。特別之處在於它們不是位於相關聯的 AWS 區域，而是鄰近目前沒有 AWS 區域的大型人口、產業和 IT 中心。然而，它們仍可在本機區域的本機工作負載與在 AWS 區域中執行的本機工作負載之間保持高頻寬、安全的連線。您應該使用 AWS Local Zones，針對低延遲要求部署離使用者更近的工作負載。

Amazon Global Edge Network

Amazon Global Edge Network 由分布在全球各城市的節點組成。Amazon CloudFront 使用此網路以較低的延遲將內容交付給最終使用者。AWS Global Accelerator 讓您可以在這些節點建立工作負載端點，以便在靠近使用者的 AWS 全球網路提供引導服務。Amazon API Gateway 使用 CloudFront 分配啟用邊緣最佳化的 API 端點，以透過最接近的節點加快用戶端存取。

AWS 區域

AWS 區域都設計為自主的，因此，若要使用多區域方法，您要部署專用的服務副本至每個區域。

多區域方法常用於 災難復原 策略，以在一次性大規模事件發生時符合復原目標。請參閱 災難復原 (DR) 計畫 以取得這些策略的詳細資訊。然而在此，我們反而專注於 可用性，尋求隨時間交付平均運行時間目標。對於高可用性目標，多區域架構通常會設計為主動/主動，其中每個服務副本 (在其各自的區域中) 都是主動的 (服務請求)。

> **建議**
>
> 您可以在單一 AWS 區域 內使用異地同步備份策略，滿足大部分工作負載的可靠性目標。僅在工作負載具有極端的可用性要求或其他需要多區域架構的業務目標時，才考慮多區域架構。

AWS 可讓您跨區域操作服務。例如，AWS 使用 Amazon Simple Storage Service (Amazon S3) 複寫、Amazon RDS 讀取複本 (包括 Aurora 讀取複本) 和 Amazon DynamoDB 全域表提供資料的連續、非同步資料複寫。透過持續複寫，您的資料版本幾乎可以立即在您的每個作用中區域中使用。

使用 AWS CloudFormation，您可以定義基礎設施，並以一致方式跨 AWS 帳戶 和跨 AWS 區域 進行部署。為了擴充此功能，AWS CloudFormation StackSets 會讓您可以使用單一作業跨多個帳戶和區域建立、更新或刪除 AWS CloudFormation 堆疊。對於 Amazon EC2 部署執行個體，AMI (Amazon Machine Image) 用來提供資訊，例如硬體組態和安裝的軟體。您可以實作 Amazon EC2 Image Builder 管道，建立您需要的 AMI，並將這些 AMI 複製到作用中區域。這可確保這些 黃金 AMI 具備您在每個新區域中部署和橫向擴展工作負載所需的一切。

若要路由流量，Amazon Route 53 和 AWS Global Accelerator 會啟用政策的定義，而這些政策可決定哪些使用者前往哪個作用中區域端點。透過 Global Accelerator，您可以設定流量刻度盤，來控制導向到每個應用程式端點的流量百分比。Route 53 支援這種百分比方法，也支援多種其他可用政策，包括地理位置臨近性和延遲型政策。Global Accelerator 自動利用廣泛的 AWS 邊緣伺服器網路，盡快將流量上線至 AWS 網路主幹，這會導致降低請求延遲。

所有這些功能都會運作，以保留每個區域的自主權。這種方法幾乎不存在例外情況，包括我們可提供全域交付的服務 (例如 Amazon CloudFront 和 Amazon Route 53) 以及 AWS Identity and Access Management (IAM) 服務的控制平面。大部分服務完全在單一區域內運行。

內部部署資料中心

對於在內部部署資料中心執行的工作負載，請盡可能架構混合式體驗。AWS Direct Connect 提供從內部設施連接至 AWS 的專用網路連線，讓您可以在兩種環境中執行。

另一個選項是使用 AWS Outposts 在內部設施執行 AWS 基礎設施和服務。AWS Outposts 是一種全受管服務，可將 AWS 基礎設施、AWS 服務、API 和工具延伸到您的資料中心。AWS 雲端中使用的硬體基礎設施與資料中心安裝的硬體基礎設施相同。AWS Outposts 會接著連接至最近的 AWS 區域 區域。然後，您可以使用 AWS Outposts 來支援低延遲或有本機資料處理要求的工作負載。

若未建立此最佳實務，暴露的風險等級為： 高

## 實作指引

- 使用多個可用區域和 AWS 區域。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。這些位置可以根據需要多樣化。
  - 區域服務固有地跨可用區域部署。
    - 這包括 Amazon S3、Amazon DynamoDB 和 AWS Lambda (未連線至 VPC 時)
  - 將容器、執行個體和函數中的工作負載部署到多個可用區域中。使用多區域資料存放區，包括快取。使用 EC2 Auto Scaling 的功能、ECS 任務放置、AWS Lambda 函數組態 (在 VPC 中執行時) 和 ElastiCache 叢集。
    - 部署 Auto Scaling 群組時，使用單獨的可用區域中的子網路。
      - 範例：將執行個體分散到多個可用區域
      - Amazon ECS 任務置放策略
      - 設定 AWS Lambda 函數以存取 Amazon VPC 中的資源
      - 選擇區域和可用區域
    - 部署 Auto Scaling 群組時，使用單獨的可用區域中的子網路。
      - 範例：將執行個體分散到多個可用區域
    - 使用 ECS 任務置放參數，指定資料庫子網路群組。
      - Amazon ECS 任務置放策略
    - 將函數設定為在 VPC 中執行時，在多個可用區域中使用子網路。
      - 設定 AWS Lambda 函數以存取 Amazon VPC 中的資源
    - 將多個可用區域與 ElastiCache 叢集一起使用。
      - 選擇區域和可用區域
- 如果您的工作負載必須部署至多個區域，請選擇多區域策略。大多數的可靠性需求都可透過多個可用區域策略，在單一 AWS 區域內滿足。視需要使用多區域策略，以符合您的業務需求。
  - 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)
    - 在另一個 AWS 區域的備份可以進一步確保資料在需要時可用。
    - 有些工作負載會有法規要求，規定要使用多區域策略。

- 針對您的工作負載評估 AWS Outposts。如果您的工作負載需要內部部署資料中心達到低延遲要求，或有本機資料處理要求。然後使用 AWS Outposts 在內部部署執行 AWS 基礎設施和服務
  - 什麼是 AWS Outposts？
- 判斷 AWS Local Zones 是否協助您為使用者提供服務。如果您有低延遲要求，請查看 AWS Local Zones 是否靠近您的使用者。如果是如此，則使用它來部署更靠近這些使用者的工作負載。
  - AWS Local Zones 常見問答集

### 資源

相關文件：

- AWS 全球基礎設施
- AWS Local Zones 常見問答集
- Amazon ECS 任務置放策略
- 選擇區域和可用區域
- 範例：將執行個體分散到多個可用區域
- 全域資料表：使用 DynamoDB 進行多區域複寫
- 使用 Amazon Aurora 全球資料庫
- 使用 AWS Services 部落格系列建立多區域應用程式
- 什麼是 AWS Outposts？

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 2019：AWS 全球網路基礎設施的創新和營運 (NET339)

## REL10-BP02 為您的多位置部署選取適當位置

### 預期成果

如需高可用性，請一律 (如果可能) 將工作負載元件部署到多個可用區域 (AZ)，如圖 10 所示。對於具有極端彈性要求的工作負載，請仔細評估多區域架構的選項。

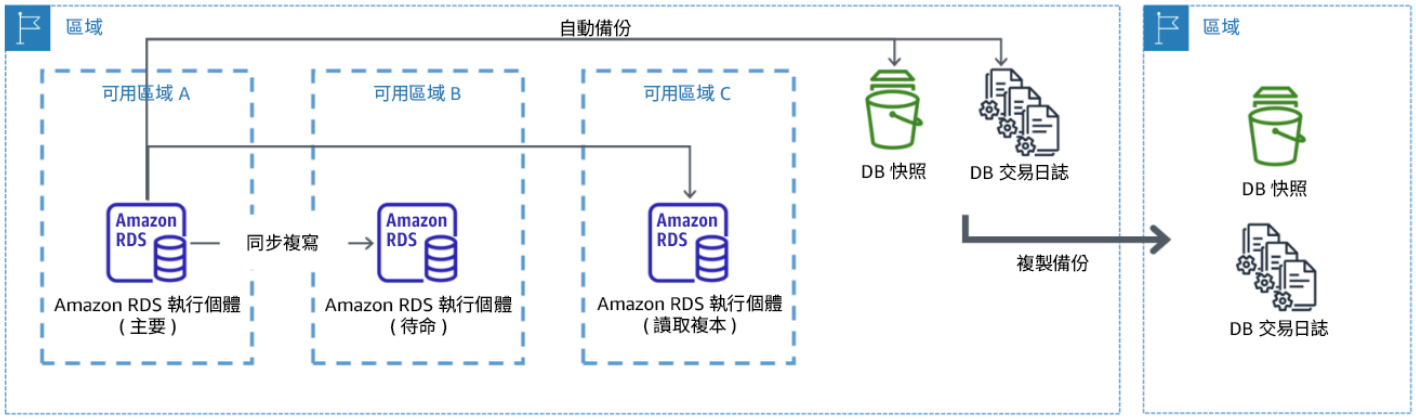



圖 10：備份至另一個 AWS 區域的彈性異地同步備份資料庫部署

## 常用的反模式

- 當異地同步備份架構滿足要求時，選擇設計多區域架構。
- 如果這些元件之間的彈性和多位置要求不同，則不考慮應用程式元件之間的相依性。

## 建立此最佳實務的優勢

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。另一防御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序以跨越多個 AWS 區域。

- 99.5% 可用性和 99.99% 可用性之間的差異每月超過 3.5 小時。如果工作負載位於多個可用區域中，則工作負載的預期可用性只能達到「四個九」。
- 透過在多個可用區域中執行您的工作負載，您可以隔離電源、冷卻和聯網中的故障，以及火災和洪水等大多數自然災害。
- 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。

若未建立此最佳實務，暴露的風險等級： 高

## 實作指引

若是基於一個可用區域之中斷或局部損失的災難事件，在單一 AWS 區域內的多個可用區域中實作高可用工作負載，可緩解自然發生的災難和技術性災難。每個 AWS 區域都是由多個可用區域構成，每個可

用區域都會與其他區域中的錯誤隔離開來，而且會隔開有意義的距離。不過，災難事件若包括失去多個可用區域元件的風險，而這些元件彼此相距甚遠，您應該實作災難復原選項，以緩解整個區域範圍的失敗。對於需要極端彈性的工作負載 (關鍵基礎設施、健康相關應用程式、金融系統基礎設施等)，可能需要多區域策略。

## 實作步驟

1. 評估您的工作負載並判斷異地同步備份方法 (單一 AWS 區域) 是否可以滿足彈性需求，或者它們是否需要多區域方法。實作多區域架構來滿足這些要求將引進額外的複雜性，因此請仔細考慮您的使用案例及其要求。使用單一 AWS 區域，幾乎可以一律符合彈性要求。在判斷是否需要使用多個區域時，請考慮以下可能的要求：
   a. 災難復原 (DR)：若是基於一個可用區域之中斷或局部損失的災難事件，在單一 AWS 區域內的多個可用區域中實作高可用工作負載，可緩解自然發生的災難和技術性災難。災難事件若包括失去多個可用區域元件的風險，而這些元件彼此相距甚遠，您應該跨多個區域實作災難復原，以緩解整個區域範圍的自然災難或技術失敗。
   b. 高可用性 (HA)：多區域架構 (在每個區域中使用多個可用區域) 可以用來實現大於四個 9 (> 99.99%) 的可用性。
   c. 堆疊本地化：將工作負載部署到全球對象時，您可以在不同的 AWS 區域 中部署本地化的堆疊，為這些區域中的對象提供服務。本地化可以包括語言、貨幣及存放的資料類型。
   d. 接近使用者： 將工作負載部署到全球對象時，您可以在接近最終使用者所在位置的 AWS 區域中部署堆疊來減少延遲。
   e. 資料落地：某些工作負載受制於資料落地要求，其中來自特定使用者的資料必須保留在特定國家/地區的邊界內。根據討論中的法規，您可以選擇將整個堆疊或只將資料部署到這些邊界內的 AWS 區域。
2. 以下是 AWS 服務提供的異地同步備份功能的一些範例：
   a. 若要使用 EC2 或 ECS 保護工作負載，請在運算資源前面部署 Elastic Load Balancer。然後，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。
      i. Application Load Balancers 入門
      ii. Network Load Balancer 入門
   b. 如果執行商務現成軟體的 EC2 執行個體不支援負載平衡，您可以透過實作異地同步備份災難復原方法來實現某種形式的容錯。
      i. the section called “REL13-BP02 使用定義的復原策略來滿足復原目標”
   c. 對於 Amazon ECS 任務，將您的服務平均地部署在三個可用區域之中，以實現可用性與成本的平衡。

i. Amazon ECS 可用性最佳實務 | 容器

d. 對於非 Aurora Amazon RDS，您可以選擇異地同步備份做為組態選項。在主資料庫執行個體失敗時，Amazon RDS 會自動提升備用資料庫，以接收另一個可用區域中的流量。也可以建立多區域讀取複本來改善彈性。

i. Amazon RDS 異地同步備份部署

ii. 在不同的 AWS 區域 中建立讀取複本

3. 以下是 AWS 服務提供的多區域功能的一些範例：

a. 對於服務自動提供異地同步備份可用性的 Amazon S3 工作負載，如果需要多區域部署，請考慮使用多區域存取點。

i. Amazon S3 中的多區域存取點

b. 對於服務自動提供異地同步備份可用性的 DynamoDB 資料表，您可以輕鬆地將現有的資料表轉換為全域表，以利用多個區域。

i. 將您的單一區域 Amazon DynamoDB 資料表轉換為全域表

c. 如果您的工作負載面臨 Application Load Balancers 或 Network Load Balancer，請使用 AWS Global Accelerator，透過將流量導向到多個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。

i. AWS Global Accelerator - AWS Global Accelerator 中標準加速器的端點 (amazon.com)

d. 對於利用 AWS EventBridge 的應用程式，請考慮跨區域匯流排，將事件轉送到您選取的其他區域。

i. 在 AWS 區域 之間傳送和接收 Amazon EventBridge 事件

e. 對於 Amazon Aurora 資料庫，請考慮跨越多個 AWS 區域的 Aurora 全球資料庫。您也可以修改現有的叢集來新增區域。

i. Amazon Aurora 全球資料庫入門

f. 如果您的工作負載包括 AWS Key Management Service (AWS KMS ) 加密金鑰，請考慮多區域金鑰是否適合您的應用程式。

i. AWS KMS 中的多區域金鑰

g. 如需其他 AWS 服務功能，請在下列一文參閱此部落格系列： 使用 AWS Services 系列建立多區域應用程式

實作計劃的工作量： 中到高

### 資源

相關文件：

- 使用 AWS Services 系列建立多區域應用程式
- AWS 上的災難復原 (DR) 架構，第 IV 部分：多站點主動/主動
- AWS 全球基礎設施
- AWS Local Zones 常見問答集
- AWS 上的災難復原 (DR) 架構，第 I 部分：在雲端中復原的策略
- 災難復原在雲端中有所不同
- 全域資料表：使用 DynamoDB 進行多區域複寫

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)
- Auth0：多區域高可用架構，可擴展至 1.5B+搭配自動容錯移轉的一個月登入

相關範例：

- AWS 上的災難復原 (DR) 架構，第 I 部分：在雲端中復原的策略
- DTCC 所達成的彈性程度遠超乎其在內部部署所能達到的
- Expedia Group 使用多區域、多可用區域架構，搭配專有的 DNS 服務，為應用程式提高彈性
- 使用者：多區域 Kafka 的災難復原
- Netflix：多區域彈性的主動-主動
- 我們如何為 Atlassian Cloud 建置資料彈性
- Intuit TurboTax 在兩個區域上執行

## REL10-BP03 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風險暴露等級：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。您可以佈建多個節點。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。所有叢集節點都佈建在相同區域中。

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。如果您已在 AWS 中託管，您可以使用 Elastic Disaster Recovery 將工作負載保護到替代可用區域或區域。Elastic Disaster Recovery 會使用持續區塊層級複寫到輕量型模擬區域，提供內部部署和雲端式應用程式的快速、可靠復原。

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，則對 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 地址、私有 IP 地址、彈性 IP 地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 地址、私有 IP 地址、彈性 IP 地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。
        - 使用事件來叫用自動化，以根據您所需的過程邏輯來修復您的元件。
    - 保護使用 AWS Elastic Disaster Recovery 限制為單一位置的有狀態的工作負載。

### 資源

相關文件：

- Amazon ECS 事件
- Amazon EC2 Auto Scaling lifecycle hook
- 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-BP04 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：小組型架構會使用工作負載的隔離執行個體，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，許多常見類型的失敗會包含在小組本身，提供額外的故障隔離。這些故障界限可以提供對於難以包含之失敗類型的彈性，例如失敗的程式碼部署或已損毀或觸發特定失敗模式的請求 (也稱為毒藥請求)。

### 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啟用故障隔離。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。界限外的元件不受失

敗影響。使用多個故障隔離界限時，您可以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。這個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。

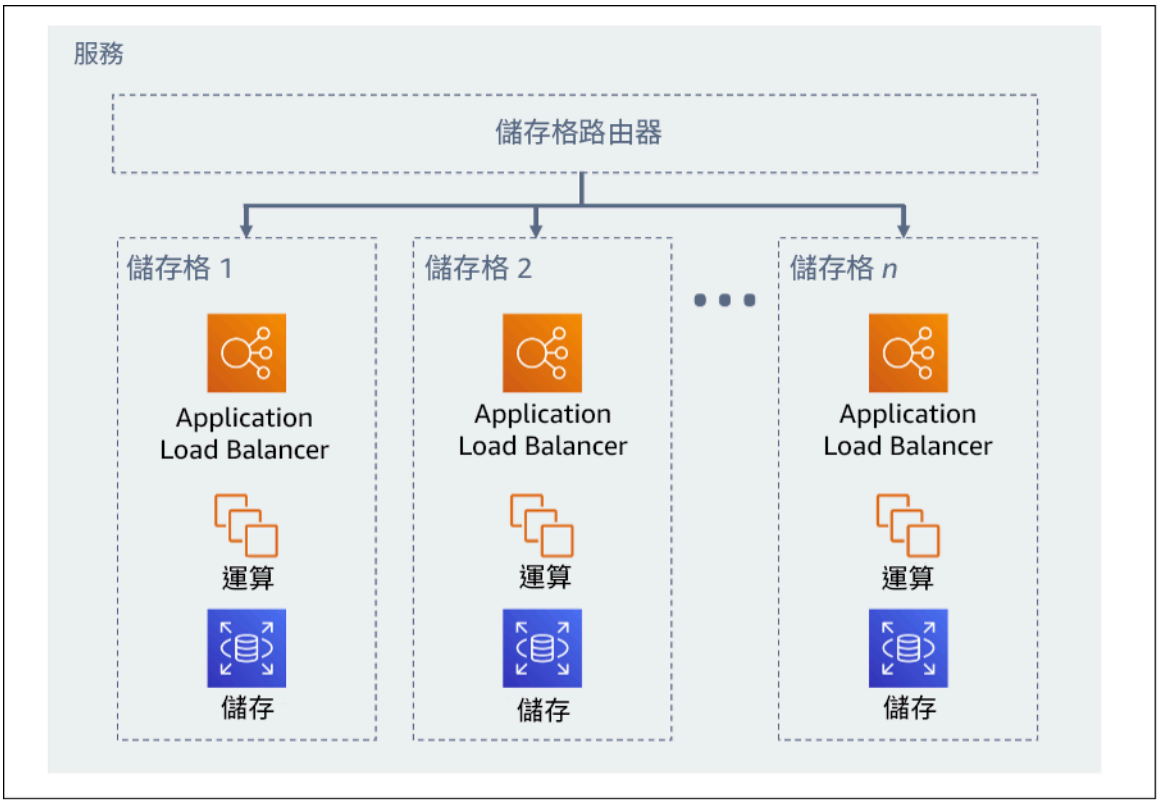



圖 11：小組型架構

實作步驟

設計小組型架構時，有數個設計考慮要考慮：

1. 分割區索引鍵：選擇分割區索引鍵時應該考慮的特殊考慮。
   - 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。範例為 `## ID` 或 `## 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. 持續性小組對應：上游服務在其資源的生命週期中應該只與單一小組互動。

- 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：路由器層會在小組之間共用元件，因此無法遵循與小組相同的區隔策略。
   - 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。如同 Colm MacCárthaigh 在可靠性、持續工作，以及咖啡時刻中所說明，簡單設計和持續工作模式可產生可靠的系統並且降低抗脆弱性。
4. 小組大小：小組應該有最大大小，而且不應該允許成長超出此大小。
   - 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 對工作負載執行負載測試
   - 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：應該利用彈性的多個層次來保護不同的失敗網域。
   - 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。另一防御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序以跨越多個 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-BP02 為您的多位置部署選取適當位置。
6. 程式碼部署：應該偏向交錯程式碼部署策略，而不是將程式碼變更同時部署到所有小組。
   - 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，請參閱自動化安全、無人為介入的部署。

未建立此最佳實務時的風險暴露等級：高

## 資源

相關的最佳實務：

- REL07-BP04 對工作負載執行負載測試

- REL10-BP02 為您的多位置部署選取適當位置

相關文件：

- 可靠性、持續工作，以及咖啡時刻
- AWS 和區隔
- 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 Builders' Library 簡介 (DOP328)
- AWS Summit ANZ 2021 - 所有事情都一直失敗：設計彈性

相關範例：

- Well-Architected 實驗室 - 搭配隨機分片的故障隔離

## 設計工作負載以承受元件失敗

需要高可用性和低平均復原時間 (MTTR) 的工作負載必須建立彈性架構。

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 復原期間需使用資料平面，而非控制平面
- REL11-BP05 使用靜態穩定性來防止雙模態行為
- 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，但在此臨界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

確定將要檢閱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 微小故障 (於《進階多可用區域彈性模式》白皮書中)。

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。
  - 確定是否需要對 EC2 執行個體 和 Auto Scaling 進行詳細監控。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。
  - 確定對 RDS 的 增強型監控 是否必要。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。
  - 判斷以下各項的關鍵無伺服器元件的監控需求： Lambda、 API Gateway、 Amazon EKS、 Amazon ECS，以及所有類型的 負載平衡器。
  - 判斷以下各項的儲存元件的監控需求： Amazon S3、 Amazon FSx、 Amazon EFS和 Amazon EBS。
- 建立 自訂指標 來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 綜合交易測試 (也稱為 Canary 測試，但請別與金絲雀部署混淆)，可執行和模擬客戶行為，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 自訂指標 來追蹤使用者體驗。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。
- 設定警報 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。警報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 向上擴展或縮減工作負載資源。
- 建立 儀表板 以視覺化呈現您的指標。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。
- 建立 分散式追蹤監控 來監控您的服務。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。
- 建立監控系統 (使用 CloudWatch 或者 X-Ray) 儀表板，以及在個別區域和帳戶中進行資料收集。
- 建立 Amazon Health Aware 監控的整合，以監控可能降級的 AWS 資源。針對商務基本工作負載，此解決方案可讓您存取 AWS 服務的主動式即時警示。

## 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
- 增強型監控
- 使用 Amazon CloudWatch 監控您的 Auto Scaling 群組和執行個體
- 發佈自訂指標
- 使用 Amazon CloudWatch 警報
- 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
- 實作 Amazon Health Aware (AHA)

相關影片：

- 減少微小故障

相關範例：

- Well-Architected 實驗室：第 300 級：實作運作狀態檢查和管理相依性以提升可靠性
- One Observability 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

AWS 服務，例如 Elastic Load Balancing 和 Amazon EC2 Auto Scaling，會協助各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、Route 53 Route 53 ARC、CloudFront 和 AWS Global Accelerator 可協助路由 AWS 區域的各流量。

AWS 服務 (例如 Amazon S3、Lambda、API Gateway、Amazon SQS、Amazon SNS、Amazon SES、Amazon Pinpoint、Amazon ECR、AWS Certificate Manager、EventBridge 或 Amazon DynamoDB) 會由 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，多可用區域是組態選項。如果啟動容錯移轉，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、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、Route 53 ARC、AWS Global Accelerator、Route 53 Private DNS for VPCs 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 地址，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
- 使用應用程式負載平衡器設定 Route 53 ARC
- 使用 Route 53 加權路由進行容錯移轉
- 透過 Route 53 ARC 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 Route 53 ARC 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
- 啟用跨區域複寫以進行 EFS 和容錯移轉
- 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
- 容錯移轉區域 API Gateway 與 Route 53 ARC
- 使用多區域 Global Accelerator 進行容錯移轉
- 透過 DRS 進行容錯移轉
- 使用 Amazon Route 53 建立災難復原機制

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，復原的設計和自動修復程序可從 現有的最佳實務取得。

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： 自動執行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。服務範例包括 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 可用來監控並篩選事件，例如 CloudWatch 警報或其他 AWS 服務的狀態變更。根據事件資訊，它可以接著調用 AWS Lambda、Systems Manager 自動化或其他目標，以便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，使用 負載平衡 並選擇適當的負載平衡器類型。
- 考慮 Amazon RDS 的修復。使用待命執行個體，設定 自動容錯移轉 至待命執行個體。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。
- 對於 如果無法將 EC2 執行個體上 已部署的應用程式部署到多個位置，且可以容忍失敗後重新開機，則進行自動復原。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 地址，以及 EBS 磁碟區 和下列掛載點皆會保留：Amazon Elastic File System 或 Lustre 和 Windows的檔案系統。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 進行自動復原。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 可用來監控並篩選事件，例如 CloudWatch 警報 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 警報
- Amazon RDS 容錯移轉
- SSM - Systems Manager 自動化
- 彈性架構最佳實務

相關影片：

- 自動佈建和擴展 OpenSearch Service
- 自動 Amazon RDS 容錯移轉

相關範例：

- Auto Scaling 研討會
- Amazon RDS 容錯移轉研討會

相關工具：

- CloudWatch
- CloudWatch X-Ray

## REL11-BP04 復原期間需使用資料平面，而非控制平面

控制平面提供的管理 API 適用於建立、讀取和描述、更新、刪除和列出 (CRUDL) 資源，而資料平面則處理日常服務流量。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：

- 依賴變更 DNS 記錄來重新路由流量。
- 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。
- 依靠大量、多服務、多 API 的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曝險等級： 中：對於某些類型的服務降級，控制平原會受到影響。若倚賴大量使用控制平面來進行修復，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 Route 53 Application Recovery Controller 轉移 DNS 流量。這些功能會持續監控應用程式從失敗中復原的功能，讓您在多個 AWS 區域、可用區域和內部部署上控管應用程式復原。

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 資料平面會答覆 DNS 查詢，以及執行並評估運作狀態檢查。它們遍布全球，而且針對 100% 可用性服務水準協議 (SLA) 所設計。

您在其中建立、更新和刪除 Route 53 資源的 Route 53 管理 API 和主控台，是在控制平面上執行，這些控制平面的設計旨在優先考慮您在管理 DNS 時所需的強大一致性和耐久性。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)。儘管這兩個系統都建置得非常可靠，但控制平面未包含在 SLA 中。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。

對於 Amazon EC2，請使用靜態穩定性設計來限制控制平面動作。控制平面動作包括個別或使用 Auto Scaling 群組 (ASG) 縱向擴展資源。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，請對整體端對端系統設定節流，以安全地限制總流量達到所限制的資源集。

對於像是 Amazon DynamoDB、Amazon API Gateway、負載平衡 和 AWS Lambda 無伺服器等服務，使用這些服務會利用資料平面。不過，建立新功能、負載平衡器、API 閘道或 DynamoDB 資料表是控制平面動作，應在降級前完成，以準備進行事件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 AWS 如何建置服務以符合高可用性目標的詳細資訊，請參閱 使用可用區域實現靜態穩定性。

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) 與預先擴展 Amazon EC2 資源 (資料平面) 的比較
- 遷移至 Lambda 及其擴展方法 (資料平面) 或 Amazon EC2 ASG (控制平面)
- 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 Route 53 ARC (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。

- 主要區域中的 Amazon EC2 Auto Scaling 或 Amazon EKS 與次要區域中的 Amazon EC2 Auto Scaling 或 Amazon EKS 比較，並將流量路由到次要區域 (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：可以幫助您實現容錯自動化的合作夥伴
- AWS Marketplace：可用於容錯的產品
- Amazon Builders' Library：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 (分割成控制平面和資料平面)
- AWS Elemental MediaStore 資料平面
- 使用 Amazon Route 53 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
- 使用 Amazon Route 53 應用程式復原控制器建置高彈性應用程式，第 2 部分：單一區域堆疊
- 使用 Amazon Route 53 建立災難復原機制
- 什麼是 Route 53 應用程式復原控制器
- 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點工作負載

相關範例：

- 簡介 Amazon Route 53 應用程式復原控制器
- Amazon Builders' Library：控管較小服務，避免分散式系統過載
- 使用 Amazon Route 53 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
- 使用 Amazon Route 53 應用程式復原控制器建置高彈性應用程式，第 2 部分：單一區域堆疊
- 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態行為

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：

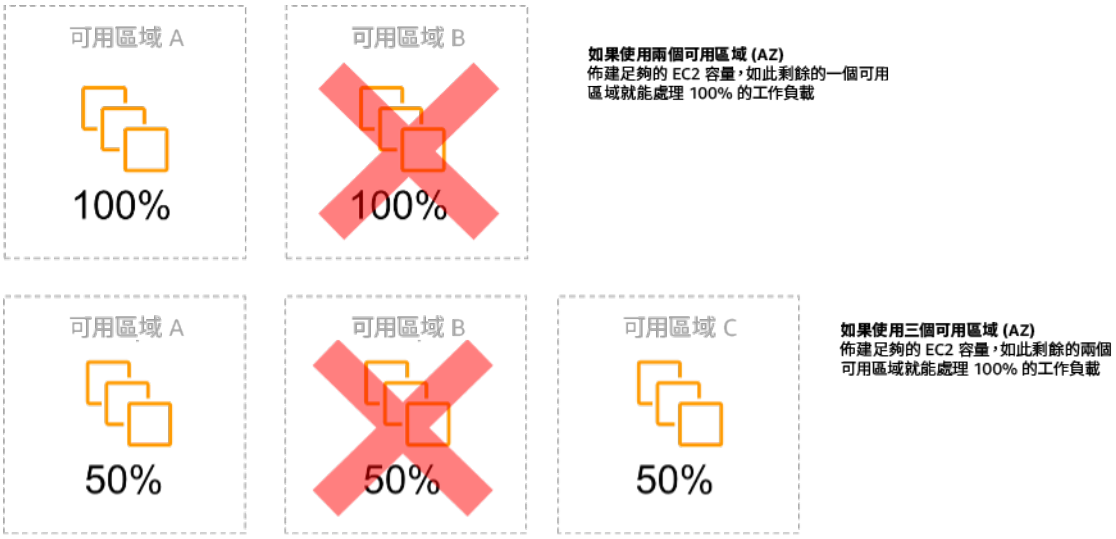
- 假設無論故障範圍，一律可以佈建資源。
- 嘗試在故障期間動態取得資源。
- 在發生故障之前，請勿在多個區域佈建適度的資源。
- 僅考慮運算資源的靜態穩定設計。

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。例如在雙模態行為下，如果移除一個可用區域，則當靜態 Amazon EC2 設計在每個可用區域佈建足夠的執行個體來處理工作負載時，系統會執行 Elastic Load Balancing 或 Amazon Route 53 運作狀態檢查，將負載從受損的執行個體移出。流量轉移後，使用 AWS Auto Scaling 以非同步方式取代故障區域的執行個體，並在運作良好的區域中啟動這些執行個體。運算部署 (例如 EC2 執行個體或容器) 的靜態穩定性可提供最高的可靠性。




在多個可用區域之 EC2 執行個體的靜態穩定性

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。靜態穩定架構適用於多種架構，包括在多個可用區域的運算執行個體、資料庫僅供讀取複本設計、Kubernetes (Amazon EKS) 叢集設計，以及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。叫用失敗時，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：Amazon S3 (單一區域)、Amazon EFS (掛載)、Amazon FSx (掛載)
- 負載平衡器： 特定設計之下

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 多區域路由
  - AWS Global Accelerator
  - Amazon Route 53 Application Recovery Controller
- 設定 資料庫讀取複本 以說明遺失單一主要執行個體或僅供讀取複本。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。若 Amazon S3 單區域 – 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。
- 負載平衡器 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，靜態穩定設計可能是在更複雜的設計中將工作負載分散到多個可用區域。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 復原期間需使用資料平面，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 Builders' Library：使用可用區域實現靜態穩定性
- 故障隔離界線
- 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 多區域路由
- AWS Global Accelerator
- Route 53 ARC
- 單一區域 Amazon S3
- 跨區域負載平衡

相關影片：

- AWS 中的靜態穩定性：AWS re:Invent 2019：Amazon 建置者資料中心簡介 (DOP328)

相關範例：

- Amazon Builders' Library：使用可用區域實現靜態穩定性

## 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。
- 不考慮 微小故障的影響 (在設計監控和警示時)。
- 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曝險等級： 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 ### 或 ##)，以及觸發警示的詳細資訊。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，無需人為介入。

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。
- 基礎架構警示 指示何時擴展資源。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 水平擴展或縮減工作負載資源。
- 簡單 靜態警示 經過建立之後，將會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。您可以直接調用 Amazon SNS API 來傳送通知，並連結任何自動化程序以進行補救或通訊。
- 整合 Amazon Health Aware 監控的整合，以監控可能降級的 AWS 資源。針對商務基本工作負載，此解決方案可讓您存取 AWS 服務的主動式即時警示。

## 資源

相關 Well-Architected 的最佳實務：

- 可用性定義

相關文件：

- 根據靜態臨界值建立 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 警報
- Amazon Health Aware (AHA)
- 設定 CloudWatch 複合警示
- re:Invent 2022 中 AWS 可觀測性的最新消息

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：每個應用程式都有針對可用性的已定義目標和針對效能指標的 SLA，可加以監控和維護以符合業務成果。

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，可協助您符合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。

若未建立此最佳實務，暴露的風險等級： 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，檢閱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？
- 與利害關係人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。
- 一旦部署，監控和報告 SLA。

## 資源

相關的最佳實務：

- REL03-BP01 選擇如何劃分工作負載
- 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-BP05 使用混沌工程測試彈性
- REL13-BP01 定義停機和資料遺失的復原目標
- 了解工作負載運作狀態

相關文件：

- 可用性與備援性

- 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 上小組型架構的指引
- 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幫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 測試功能要求
- REL12-BP04 測試擴展和效能需求
- REL12-BP05 使用混沌工程測試彈性
- REL12-BP06 定期執行演練日

# REL12-BP01 使用程序手冊調查失敗

透過在程序手冊中記錄調查程序，實現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。在確定或向上呈報問題之前，任何程序步驟的結果都用於確定要採取的後續步驟。

程序手冊是您必須進行的主動規劃，然後才能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。

常用的反模式：

- 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一致地遵循程序。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。

若未建立此最佳實務，暴露的風險等級為： 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)。
  - 將程序手冊實作為程式碼。透過編寫程序手冊指令碼，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。執行手冊活動可以作為程序手冊活動的一部分被觸發或執行，或者在程序手冊中提示執行，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 Run Command
    - 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 Run Command
- 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。其中一個機制是錯誤糾正 (COE) 程序。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可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風險暴露等級：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。使用五個為什麼之類的技術深入了解基礎問題。

維護事件分析所學教訓的儲存庫。此機構知識可以作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。這些業務決定可主導您工程設計的優先順序。您可以選擇在開發環境中以可靠性作為代價最佳化成本，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：

- 為什麼您應該開發錯誤糾正 (COE)

相關影片：

- Amazon 對於成功故障的方法
- AWS re:Invent 2021 - Amazon 建置者資料中心：在 Amazon 卓越營運

## REL12-BP03 測試功能要求

使用驗證所需功能的單位測試和整合測試等技術。

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。系統會建置這些變更，並執行測試。測試完成後，會將內建的程式碼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。

此外，經驗顯示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。Amazon CloudWatch Synthetics 讓您能夠 建立 Canary， 以監控您的端點和 API。

若未建立此最佳實務，暴露的風險等級： 高

### 實作指引

- 測試功能要求。這包括驗證所需功能的單位測試和整合測試。
  - 搭配使用 CodePipeline 與 AWS CodeBuild 以測試程式碼和執行建置
  - AWS CodePipeline 新增對於單位的支援，以及透過 AWS CodeBuild 自訂整合測試
  - 持續交付與持續整合
  - 使用 Canary (Amazon CloudWatch Synthetics)
  - 軟體和測試自動化

### 資源

相關文件：

- APN 合作夥伴：可以幫助實作持續整合管道的合作夥伴
- AWS CodePipeline 新增對於單位的支援，以及透過 AWS CodeBuild 自訂整合測試
- AWS Marketplace：可用於持續整合的產品
- 持續交付與持續整合
- 軟體和測試自動化
- 搭配使用 CodePipeline 與 AWS CodeBuild 以測試程式碼和執行建置
- 使用 Canary (Amazon CloudWatch Synthetics)

## REL12-BP04 測試擴展和效能需求

使用負載測試等技術，以驗證工作負載是否滿足擴展和效能需求。

在雲端，您可以隨需建立工作負載的生產規模測試環境。如果您在縮減的基礎設施上執行這些測試，您必須將觀察到的結果擴展到您認為在生產環境中會發生的情況。如果您很謹慎，力求不影響實際使用者，也可以在生產環境中執行負載和效能測試，並將您的測試資料加上標籤，以免與實際使用者資料混淆並損毀使用統計資料或生產報告。

透過測試，確保您的基本資源、擴展設定、服務配額和彈性設計能夠在負載下如預期運作。

若未建立此最佳實務，暴露的風險等級： 高

### 實作指引

- 測試擴展和效能需求。進行負載測試，以驗證工作負載是否滿足擴展和效能需求。
  - AWS 上的分散式負載測試：模擬數千名連線的使用者
  - Apache JMeter
    - 在與生產環境相同的環境中部署應用程式並執行負載測試。
      - 使用基礎設施即程式碼概念來建立與您的生產環境盡可能相似的環境。

### 資源

相關文件：

- AWS 上的分散式負載測試：模擬數千名連線的使用者
- Apache JMeter

## REL12-BP05 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，還可以藉由在錯誤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。

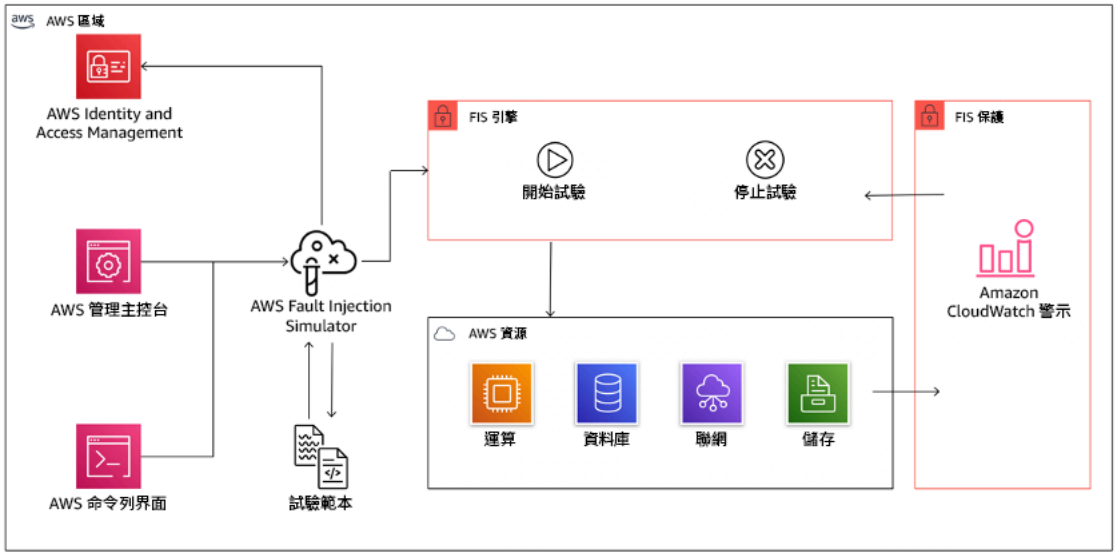
若要對因間歇性網路中斷而產生的外部相依性驗證其 備用或容錯移轉機制 ，您的元件應藉由封鎖對第三方供應商的存取達指定的持續期間 (可延續數秒到數小時)，來模擬這類事件。

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) 是用來執行錯誤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。它支援同時在不同類型的資源間導入錯誤，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錯誤注入試驗。

錯誤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 Gremlin 之類的商業選項。為了擴展可在 AWS 上注入的錯誤範圍，AWS FIS 與 Chaos Mesh 和 Litmus Chaos 整合，讓您能夠在多項工具間協調錯誤注入工作流程。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

- 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的最佳實務建立的) 可根據重大相依性、過去的事件、已知問題和合規要求來衡量風險。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。如需關於建立這類清單的詳細資訊，請參閱 「營運準備度審查」白皮書， 此文件會說明如何建立相關程序來防止過去的事故再次發生。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 中提供了 FMEA 的詳細說明。

- 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錯誤注入試驗。

- 對於您所執行的每個試驗，均應依循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。

- 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。

- 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**
  >
  > 若 #### 發生， ###### 工作負載將 ###### 以維護 #########。

  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)。

- 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 是指您知道，但未能完全了解的事物， 未知的未知 則是指您不知道也未能完全了解的事物。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 會提供許多預先定義的錯誤模擬，名為 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。若要進一步了解使用 AWS FIS 的各種試驗，另請參閱「將彈性和 Well-Architected 應用程式用於混沌工程」實驗室。此外，AWS Resilience Hub 也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。

試驗應在真實的負載下執行於生產環境中，且應使用 金絲雀部署 同時推動控制和試驗系統部署 (在情況允許時)。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。路由層 綜合監控 也稱為使用者金絲雀，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。

執行錯誤注入試驗時，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。

- 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 (也稱為使用者金絲雀)。

- 改善工作負載設計的彈性。

  如果未維持穩定狀態，請調查工作負載設計可經由哪些改進來減輕錯誤，運用 AWS Well Architected 可靠性支柱的最佳實務。如需其他指引和資源，請前往 AWS Builder's Library，內含相關文章說明如何 改善您的運作狀態檢查 或 在應用程式碼中使用退避重試，以及其他主題。

  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。

- 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。若要了解其執行方式，請參閱此

部落格： 如何使用 AWS CodePipeline 執行 AWS FIS 試驗。此實驗室涉及 CI/CD 管道中的 AWS FIS 試驗， 可讓您進行實際操作。

錯誤注入試驗也是演練日的一部分 (請參閱 REL12-BP06 定期執行演練日) 建立持續整合/持續部署 (CI/CD) 管道。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。

- 擷取並儲存試驗結果。

錯誤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。AWS FIS 的試驗記錄 可作為此資料擷取的一部分。

## 資源

相關的最佳實務：

- REL08-BP03 將恢復能力測試整合為部署的一部分
- 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
- 混沌試驗的金絲雀部署

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)

相關範例：

- Well-Architected 實驗室：第 300 級：測試 Amazon EC2、Amazon RDS 和 Amazon S3 的彈性
- 「AWS 上的混沌工程」實驗室
- 「將彈性和 Well-Architected 應用程式用於混沌工程」實驗室
- 「無伺服器混沌」實驗室
- 「使用 AWS Resilience Hub 測量及改善您的應用程式彈性」實驗室

相關工具：

- AWS Fault Injection Service
- AWS Marketplace： Gremlin 混沌工程平台
- Chaos Toolkit
- Chaos Mesh
- Litmus

## REL12-BP06 定期執行演練日

使用演練日定期執行回應事件和失敗的程序，盡可能接近生產環境 (包括在生產環境中)，並與實際參與失敗情境的人員共同演練。在演練日當天強制執行措施，以確保生產事件不會影響使用者。

演練日模擬失敗或事件，以測試系統、流程和團隊的回應。目的是實際執行在異常事件發生時團隊將要執行的動作。如此可協助您了解何處有改善空間，並能協助發展組織處理活動的經驗。這些應該定期進行，以便您的團隊建置 有關如何回應的 肌肉記憶。

在彈性設計就緒，並已在非生產環境中進行測試之後，演練日就是確保生產中的一切按照計畫運作。演練日，特別是第一個演練日，是一個「全員參與」活動，工程師和操作人員會被告知何時發生，以及會發生什麼情況。執行手冊已就緒。以規定的方式在生產系統中執行模擬事件 (包括可能的失敗事件)，並評估影響。如果所有系統都如設計運作，偵測和自我修復將幾乎不會產生影響。不過，如果觀察到負面影響，測試會回復並視需要手動修復工作負載問題 (使用執行手冊)。由於演練日通常會在生產環境中進行，因此應採取所有預防措施，以確保不會對客戶的可用性造成影響。

常用的反模式：

- 記載您的程序，但絕不練習程序。
- 未在測試練習中納入業務決策者。

建立此最佳實務的優勢： 定期進行演練日可確保所有員工在發生實際事件時遵守政策和程序，並驗證這些政策和程序是否適當。

若未建立此最佳實務，暴露的風險等級： 中

### 實作指引

- 安排演練日以定期練習您的執行手冊和程序手冊。演練日應納入生產事件發生時參與的每個人：企業擁有者、開發人員、營運人員和事件反應團隊。
  - 執行負載或效能測試，然後執行錯誤注入。
  - 尋找執行手冊上的異常情況，並尋找練習程序手冊的機會。
    - 如果您偏離了執行手冊，應優化執行手冊或更正該行為。如果您練習程序手冊，確定應使用的執行手冊，或建立一個新的執行手冊。

### 資源

相關文件：

- 什麼是 AWS GameDay？

相關影片：

- AWS re:Invent 2019：透過混沌工程提升彈性 (DOP309-R1)

相關範例：

- AWS Well-Architected 實驗室 - 測試彈性

## 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，例如監控失敗、部署到多個位置，以及自動容錯移轉。不過，可用性著重於工作負載的元件，而災難復原著重於整個工作負載的離散副本。災難復原具有不同於可用性的目標，著重於災難後的復原時間。

最佳實務

- REL13-BP01 定義停機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 自動化復原

## REL13-BP01 定義停機和資料遺失的復原目標

工作負載具有復原時間目標 (RTO) 和復原點目標 (RPO)。

復原時間目標 (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) 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。

在為您的工作負載選取適用的災難復原 (DR) 策略時，RTO 和 RPO 值是重要的考慮因素。這些目標是由企業決定，然後由技術團隊用來選取和實作 DR 策略。

預期成果：

每個工作負載都獲指派一個 RTO 和 RPO，其是根據業務影響來定義的。工作負載會指派給預先定義的層級，定義服務可用性和可接受的資料遺失，以及相關聯的 RTO 和 RPO。如果這類分層不可行，則可以根據工作負載以定制方式指派此分層，旨在稍後建立層級。RTO 和 RPO 會在選取工作負載的災難復原策略實作時的主要考量之一。挑選 DR 策略的其他考量是成本限制、工作負載相依性和營運要求。

對於 RTO，了解基於中斷持續時間的影響。它是線性的，還是有非線性的影響？(例如，四個小時後，您關閉了一條生產線，直到下一個輪班開始)。

如下的災難復原方法可以協助您了解工作負載關鍵性與復原目標之間的關係。(請注意，X 軸和 Y 軸的實際值應根據您的組織需求加以自訂)。

| 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 |

圖 16：災難復原方法

常用的反模式：

- 沒有已定義的復原目標。
- 選擇任意復原目標。
- 選擇過於寬鬆且不符合業務目標的復原目標。
- 不了解關機時間和資料遺失的影響。
- 選取不切實際的復原目標，例如零時間復原和零資料遺失，這對於您的工作負載組態可能無法實現。
- 選擇比實際業務目標更嚴格的復原目標。這會被迫進行比工作負載所需更昂貴和更複雜的 DR 實作。
- 選取的復原目標與工作負載的復原目標不相容。
- 您的復原目標未考慮法規合規性要求。
- 已定義工作負載的 RTO 和 RPO，但從未進行測試。

建立此最佳實務的優勢： 需以時間和資料損失的復原目標來引導 DR 實作。

若未建立此最佳實務，暴露的風險等級： 高

## 實作指引

對於給定的工作負載，您必須了解停機時間和資料遺失對您業務的影響。隨著停機時間或資料遺失的增加，影響會大幅地增長，但這種增長形式可能會根據工作負載類型而有所不同。例如，您可以容忍長達一小時的停機時間而影響不大，但在此之後影響會迅速加大。對業務的影響會以多種形式顯現，包括貨幣成本 (例如收益損失)、客戶信任 (以及對信譽的影響)、營運問題 (例如發不出薪資或生產力下降)，以及監管風險。使用下列步驟來了解這些影響，並為您的工作負載設定 RTO 和 RPO。

實作步驟

1. 確定此工作負載的業務利害關係人，並與他們一起實作這些步驟。工作負載的復原目標是業務決策。然後，技術團隊與業務利害關係人合作，使用這些目標來選取 DR 策略。

> **Note**
>
> 針對步驟 2 和 3，您可以使用 the section called “實作工作表”。

2. 收集必要資訊，藉由回答下列問題來做出決策。
3. 對於組織中的工作負載影響，您是否具有關鍵性的類別或層級？
   a. 若是，請將此工作負載指派給類別。
   b. 若否，請建立這些類別。建立五個或更少的類別，然後縮小每個類別的復原時間目標範圍。範例類別包括：重大、高、中、低。若要了解工作負載如何對應至類別，請考慮工作負載是任務為主、業務為主，還是非業務推動。
   c. 根據類別設定工作負載 RTO 和 RPO。一律選擇比進入此步驟所計算的原始值更嚴格的類別 (更低的 RTO 和 RPO)。如果這導致值發生不適當的大變更，則考慮建立一個新類別。
4. 根據這些答案，將 RTO 和 RPO 指派給工作負載。這可以直接完成，也可以透過將工作負載指派給預先定義的服務層來完成。
5. 在工作負載團隊和利害關係人可存取的位置記錄此工作負載的 災難復原計劃 (DRP)，這是貴組織業務持續性計劃 (BCP) 的一部分。
   a. 記錄 RTO 和 RPO，以及用來決定這些值的資訊。包括用於評估對業務之工作負載影響的策略。
   b. 記錄除 RTO 和 RPO 之外的其他指標，您是否正在追蹤或規劃追蹤災難復原目標。
   c. 建立 DR 策略和執行手冊的詳細資訊時，會將這些資訊新增至此計劃。
6. 藉由在如圖 15 所示的矩陣中查看工作負載的關鍵性，您可以開始建立針對組織定義的預先定義服務層。
7. 在您根據 實作了 DR 策略 (或 DR 策略的概念證明) 之後the section called “REL13-BP02 使用定義的復原策略來滿足復原目標”，請測試此策略以判定工作負載的實際 RTC (復原時間能力) 和 RPC (復原點能力)。如果這些不符合目標復原目標，則可與您的業務利害關係人合作，一起調整這些目標，或可對 DR 策略進行變更以符合目標。

主要問題

1. 在對業務產生嚴重影響之前，工作負載可以關閉的時間上限
   a. 如果工作負載中斷，請判定每分鐘對業務造成的貨幣成本 (直接財務影響)。

b. 考慮到影響並非總是線性的。一開始影響可能會受到限制，然後在超過關鍵時間點後迅速增加。

2. 在對業務產生嚴重影響之前，可以遺失的資料量上限
   a. 考慮將此值用於您最關鍵的資料存放區。識別其他資料存放區的各自關鍵性。
   b. 如果遺失工作負載資料，可以重建嗎？ 如果在操作上這樣做比備份和還原更容易，則根據用來重建工作負載資料之來源資料的關鍵性來選擇 RPO。
3. 依賴下游游相依性的工作負載或依賴上游相依性的工作負載，其復原目標和可用性期望是什麼？
   a. 選擇可讓此工作負載符合上游相依性要求的復原目標
   b. 鑑於下游相依性的復原能力，選擇可實現的復原目標。可以執行非關鍵的下游相依性 (您可以「解決」的相依性)。或者，使用關鍵的下游相依性，在必要時改善其復原能力。

其他問題

考慮這些問題，以及它們如何套用至這個工作負載：

4. 您是否有不同的 RTO 和 RPO，取決於中斷的類型 (區域與可用區域等)？
5. 您的 RTO/RPO 是否會在特定時間 (季節性、銷售活動、產品發佈) 發生變化？ 若是，有什麼不同的測量和時間界限？
6. 如果工作負載中斷，有多少客戶會受到影響？
7. 如果工作負載中斷，對信譽有何影響？
8. 如果工作負載中斷，可能會發生哪些其他營運影響？ 例如，如果電子郵件系統無法使用，或如果薪資系統無法提交交易，則會影響員工的生產力。
9. 工作負載 RTO 和 RPO 如何與業務線和組織 DR 策略保持一致？
10. 是否有提供服務的內部合約義務？ 未符合它們時會受到處罰嗎？
11. 資料的法規或合規限制是什麼？

## 實作工作表

您可以將此工作表用於實作步驟 2 和 3。您可以調整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。<br>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

實作計劃的工作量： 低

## 資源

相關的最佳實務：

- the section called “REL09-BP04 定期執行資料復原以驗證備份的完整性和程序”
- the section called “REL13-BP02 使用定義的復原策略來滿足復原目標”
- the section called “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 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 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，都有一個已定義和實作的 DR 策略，讓該工作負載能夠實現 DR 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，其討論在 REL13-BP01 定義停機和資料遺失的復原目標。

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。這些策略按成本和複雜度遞增的順序列出，以及按 RTO 和 RPO 的遞減順序列出。 復原區域稱為 AWS 區域，而不是用於工作負載的主要區域。

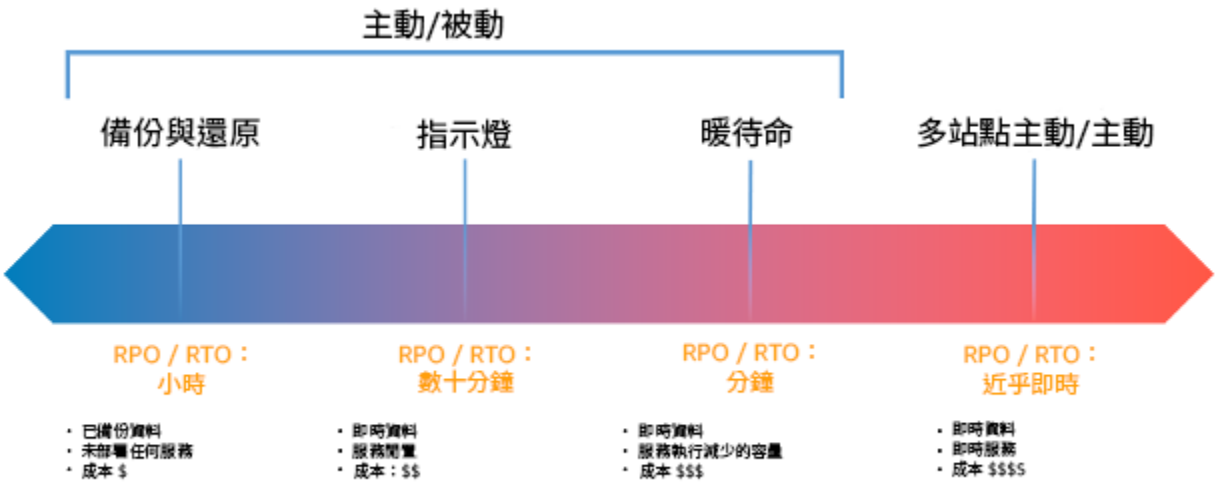



圖 17：災難復原 (DR) 策略

- 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啟用時間點復原，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 幾分鐘，RTO 幾十分鐘)：在復原區域中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 幾秒鐘，RTO 幾分鐘)：維持工作負載的縮減但完整功能版本，該工作負載始終在復原區域中執行。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。完全擴展時，稱之為熱待命。
- 多區域 (多站點) 主動-主動 (RPO 近乎零，RTO 可能為零) 您的工作負載會部署至多個 AWS 區域，並主動處理來自多個 AWS 區域 的流量。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) 基礎設施並縱向擴展，而暖待命只需要您縱向擴展 (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. 確定將滿足此工作負載之復原要求的 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。

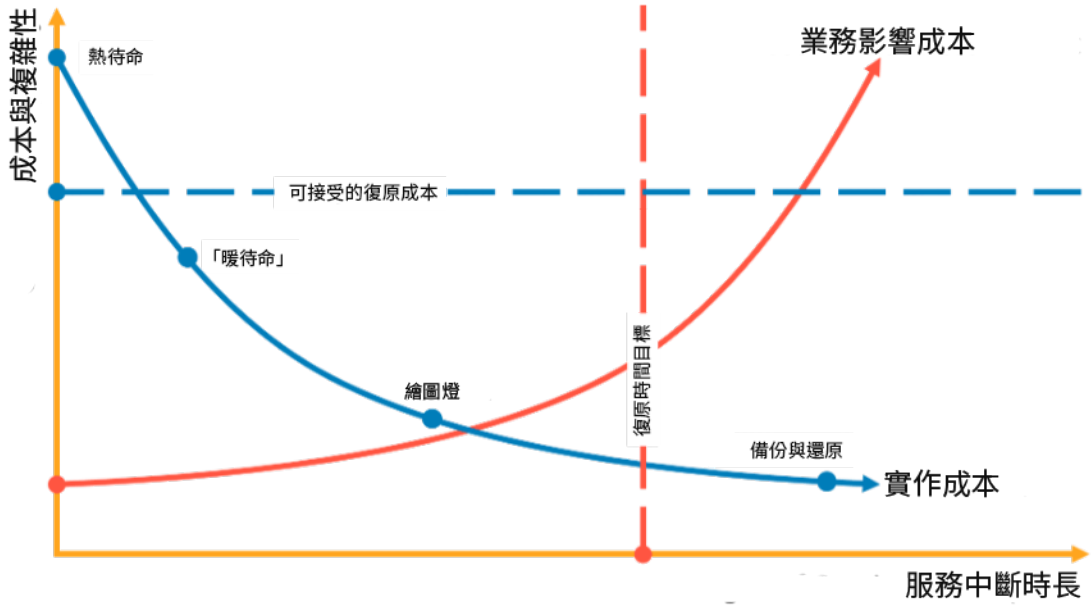



圖 18：根據 RTO 和成本選擇 DR 策略

若要進一步了解，請參閱業務持續性計劃 (BCP)。

2. 檢閱如何實作所選 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。

備份與恢復

備份與還原是最不複雜的實作策略，但需要更多時間和精力來還原工作負載，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。

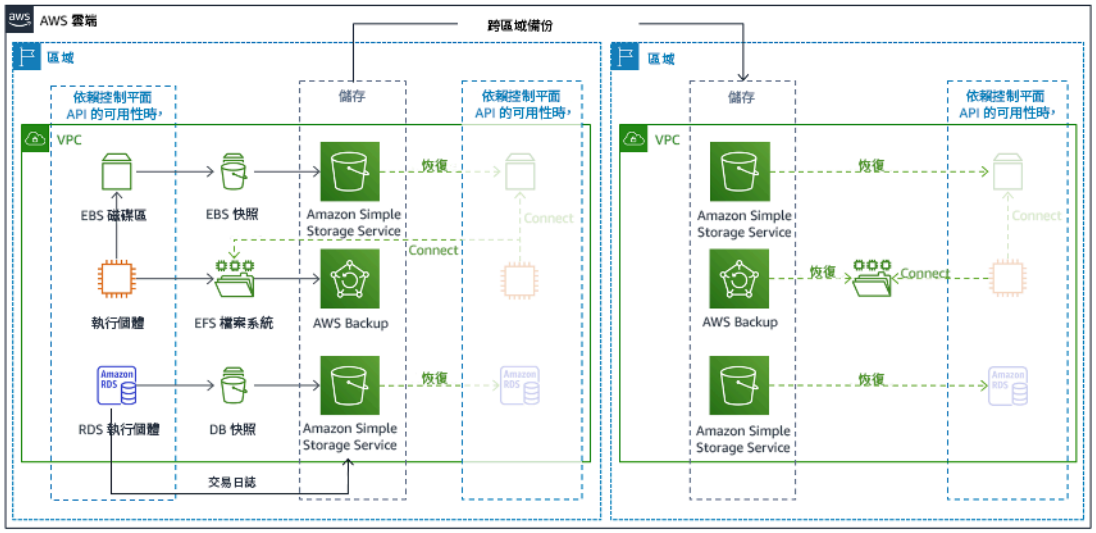



圖 19：備份和還原架構

如需此策略的詳細資訊，請參閱 AWS 上的災難復原 (DR) 架構，第 II 部分：具有快速復原的備份和還原。

指示燈

使用指示燈方法，您可以將資料從主要區域複寫至復原區域。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。

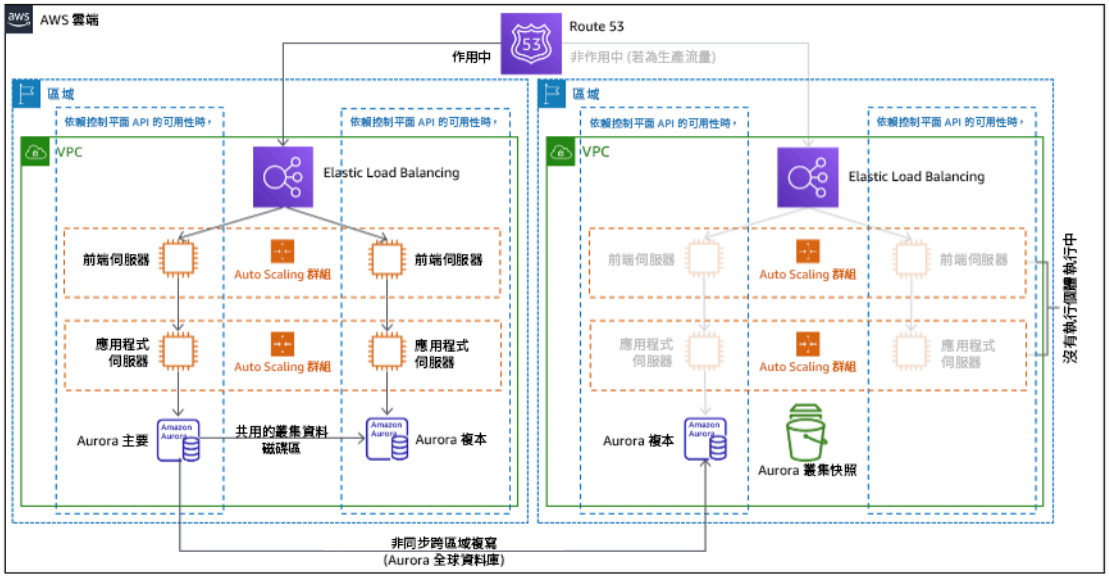



圖 20：指示燈架構

如需此策略的詳細資訊，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

暖待命方法涉及確保在另一個區域中有一個縮減規模，但功能完全的生產環境副本。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。如果部署完整容量的復原區域，這稱為熱待命。

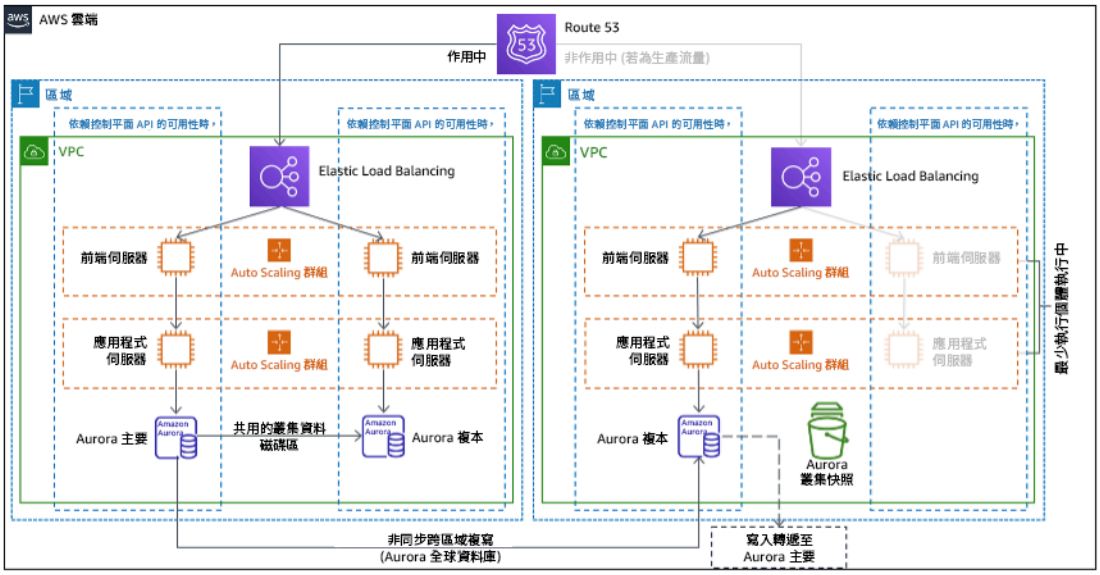



圖 21：暖待命架構

使用暖待命或指示燈需要縱向擴展復原區域中的資源。若要在需要時確認容量可用，請考慮針對 EC2 執行個體使用容量保留。如果使用 AWS Lambda，則佈建的並行可以提供執行環境，以便它們準備好立即回應您的函數叫用。

如需此策略的詳細資訊，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

您可以同時在多個區域中執行工作負載，做為多站點主動/主動策略。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。基於 DR 以外的理由，客戶可能會選取此策略。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。

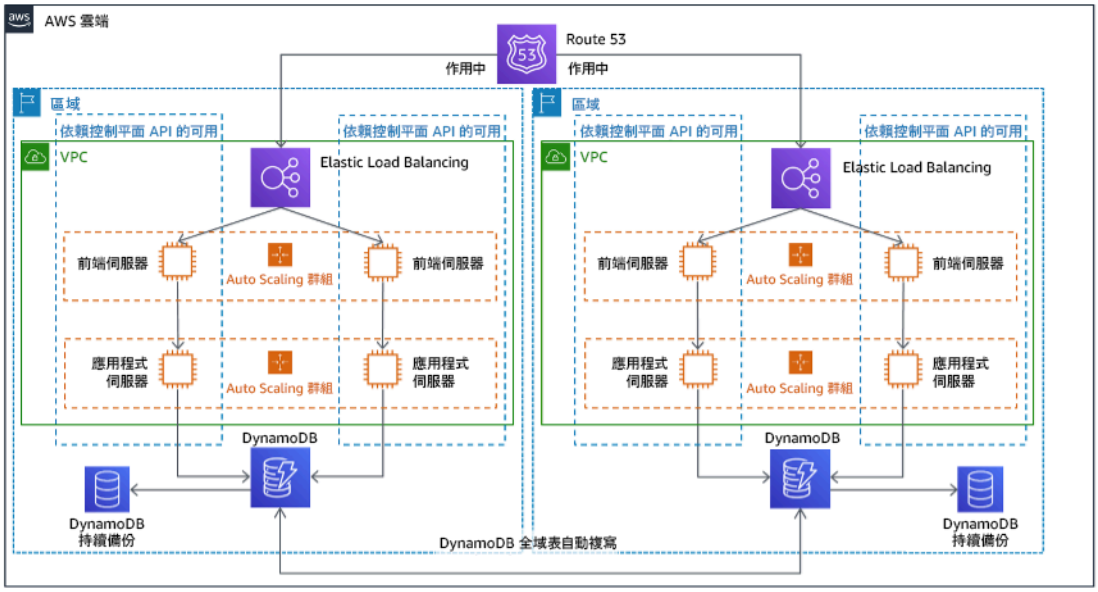



圖 22：多站點主動/主動架構

如需此策略的詳細資訊，請參閱 AWS 上的災難復原 (DR) 架構，第 IV 部分：多站點主動/主動。

AWS Elastic Disaster Recovery

如果您針對災難復原考慮指示燈或暖待命策略，AWS Elastic Disaster Recovery 可以提供具有改進優點的替代方法。Elastic Disaster Recovery 可以提供類似於暖待命的 RPO 和 RTO 目標，但是維持指示燈的低成本方法。Elastic Disaster Recovery 會將您的資料從主要區域複寫到您的復原區域，使用持續資料保護來達成以秒數測量的 RPO 和可以分鐘數測量的 RTO。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。

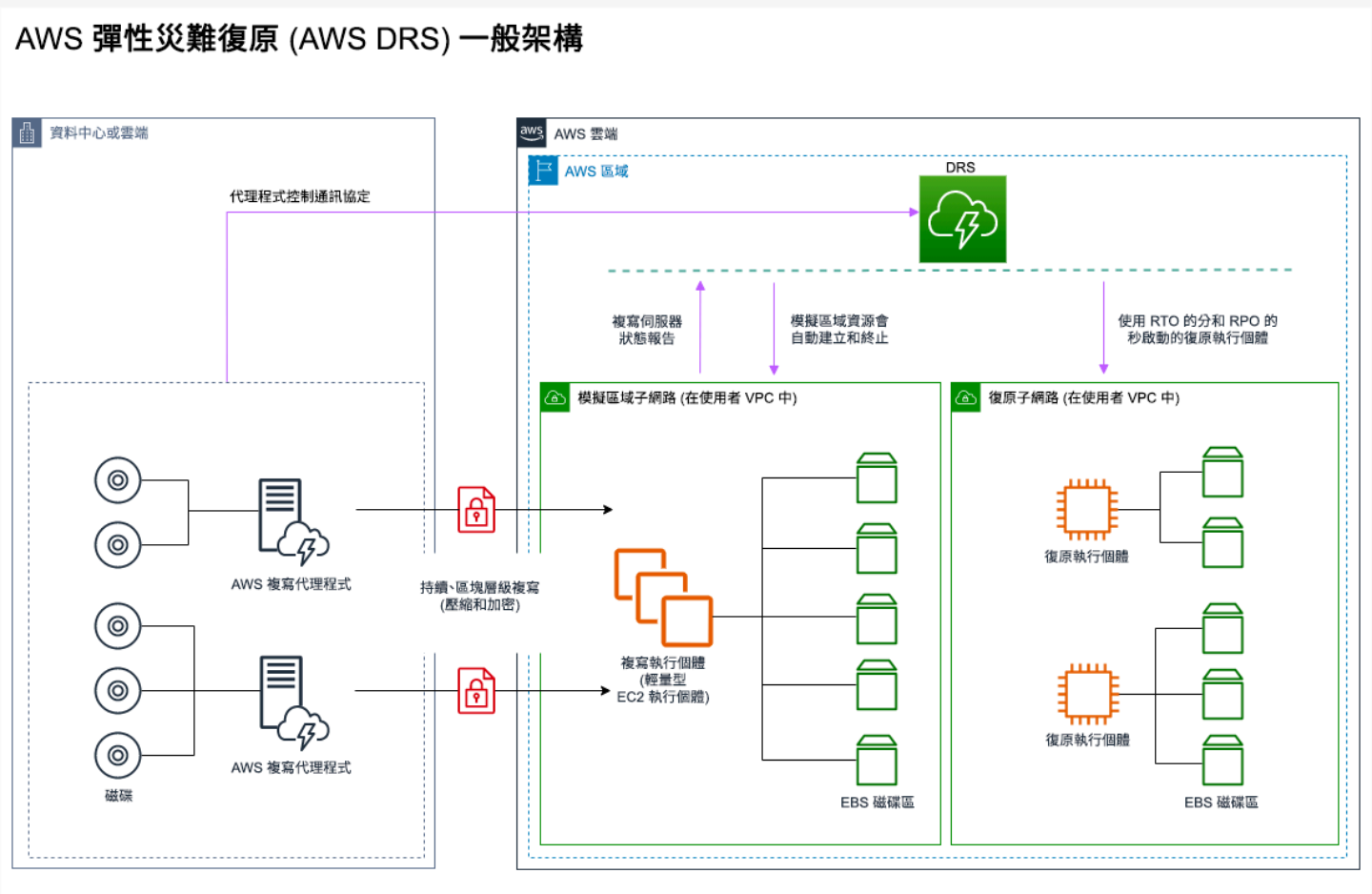



圖 23：AWS Elastic Disaster Recovery 架構

其他保護資料的做法

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。此架構會利用多站點主動/主動方法，因為 Amazon EC2 執行個體和 Elastic Load Balancer 已在多個 AZ 中部署資源，主動處理請求。架構也會示範熱待命，其中如果主要 Amazon RDS 執行個體失敗 (或 AZ 本身失敗)，則待命執行個體會提升至主要執行個體。

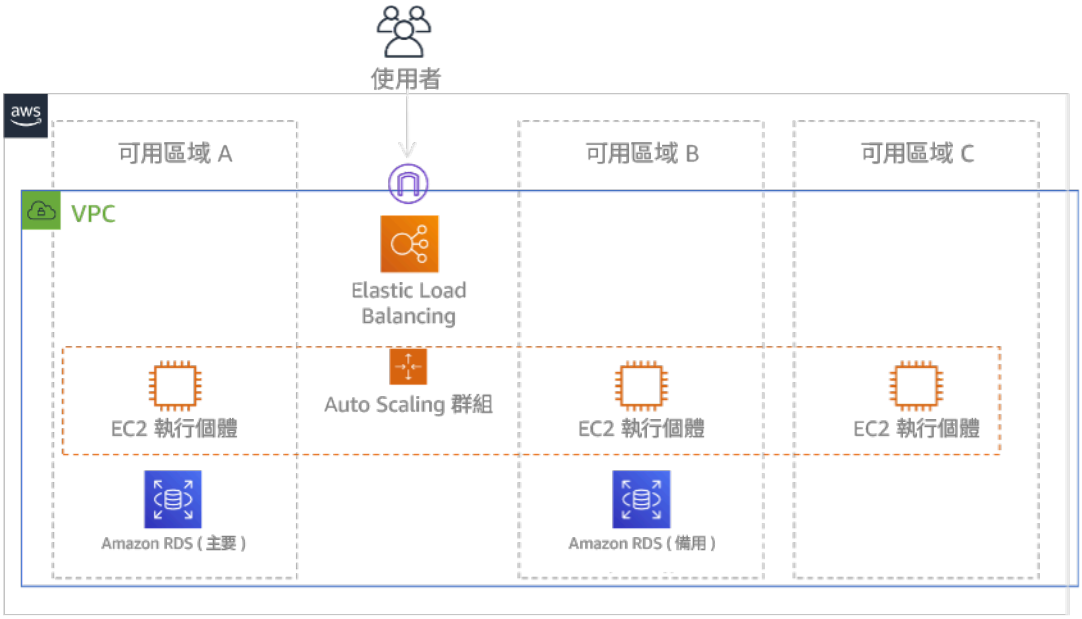



圖 24：多可用區域架構

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。這對於受制於單一區域的資料尤其重要，例如 Amazon EBS 磁碟區或 Amazon Redshift 叢集。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。

下列部落格文章中描述了一種不太常見的單一區域替代方法 (多可用區域 DR)：使用 Amazon Route 53 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。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. 評估工作負載的資源，以及在容錯移轉之前 (在正常操作期間) 其哪個組態將位於復原區域中。

針對基礎設施和 AWS 資源使用基礎設施即程式碼，例如 AWS CloudFormation 像是 Hashicorp Terraform 的第三方工具。若要使用單一作業跨多個帳戶和區域進行部署，您可以使用 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 參數和條件式邏輯，您可以使用單一範本控制已部署堆疊是主動或待命。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 策略都要求在 AWS 區域內備份資料來源，然後將這些備份複製到復原區域。AWS Backup 提供了一個集中檢視，您可以在其中設定、排定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，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) DB 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 服務如何跨區域操作，請參閱使用 AWS 服務建立多區域應用程式這個部落格系列。

4. 確定並實作如何在需要時 (在災難事件期間) 使您的復原區域為容錯移轉做好準備。

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。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 以取得詳細資訊。重建基礎設施包括建立 EC2 執行個體之類的資源，還有 Amazon Virtual Private Cloud (Amazon VPC)、子網路及所需的安全群組。您可以將大部分還原程序自動化。若要了解做法，請參閱這篇部落格文章。

5. 確定並實作如何在需要時 (在災難事件期間) 將流量路由至容錯移轉。

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。其中一個選項是使用 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 Route 53 應用程式復原控制器，其會提供一個高度可用的資料平面 API，將流量重新路由到復原區域。實作容錯移轉時，使用資料平面作業並避免控制平面作業，其描述在 REL11-BP04 復原期間需使用資料平面，而非控制平面。

若要深入了解這個和其他選項，請參閱災難復原白皮書的這一節。

6. 設計您的工作負載將如何復原的計劃。

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 全域資料表，即使主要區域中的資料表變成無法可用，則當它重新上線時，DynamoDB 仍會繼續傳播任何擱置的寫入。如果使用 Amazon Aurora 全球資料庫和使用受管規劃容錯移轉，則會維持 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。例如，如需如何在假設非計劃容錯移轉的情況下使用 Amazon Aurora 完成此操作，請參閱此實驗室：復原全球資料庫。

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，服務會協助協調和自動化容錯恢復程序。如需詳細資訊，請參閱執行容錯恢復。

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 復原期間需使用資料平面，而非控制平面”
- the section called “REL13-BP01 定義停機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 Route 53 應用程式復原控制器？
- AWS 彈性災難復原
- HashiCorp Terraform：開始使用 - 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)
- 開始使用 AWS 彈性災難復原 | Amazon Web Services

相關範例：

- Well-Architected 實驗室 - 災難復原 - 說明 DR 策略的研討會系列

## 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劃，可確保該計劃能在需要時運作，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風險暴露等級：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. 針對 Amazon EC2 型工作負載，使用 AWS Elastic Disaster Recovery 來實作和啟動您的 DR 策略的練習執行個體。AWS Elastic Disaster Recovery 提供有效率地執行練習的能力，可協助您為容錯移轉事件做準備。您也可以針對測試和練習目的使用 Elastic Disaster Recovery 頻繁地啟動您的執行個體，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

相關範例：

- Well-Architected 實驗室 - 測試彈性

## REL13-BP04 管理 DR 站點或區域的組態偏移

確保根據需要在 DR 站點或區域提供基礎設施、資料和組態。例如，檢查 AMI 和服務配額是否為最新版本。

AWS Config 會持續監控和記錄 AWS 資源組態。它可以偵測偏移，並觸發 AWS Systems Manager Automation 修正並引發警示。AWS CloudFormation 可額外偵測您已部署之堆疊中的偏移。

常用的反模式：

- 當您在主要位置進行組態或基礎設施變更時，無法在復原位置進行更新。
- 未考量主要和復原位置中潛在的限制 (例如服務差異)。

建立此最佳實務的優勢： 確保 DR 環境與現有環境一致，便可保證完整復原。

若未建立此最佳實務，暴露的風險等級： 中

### 實作指引

- 確保您的交付管道同時交付到主要站點和備份站點。用於將應用程式部署到生產中的交付管道，應分發到所有指定的災難復原策略位置，包括開發和測試環境。
- 啟用 AWS Config 追蹤潛在的偏移位置。使用 AWS Config 規則建立系統，以執行災難復原策略，並在發現偏移時產生提醒。
  - 依 AWS Config 規則 修補不合規的 AWS 資源
  - AWS Systems Manager Automation

- 使用 AWS CloudFormation 偵測您的基礎設施。AWS CloudFormation 可以偵測 CloudFormation 範本指定項目與實際部署項目之間的偏移。
  - AWS CloudFormation：在整個 CloudFormation 堆疊上偵測偏移

### 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 CloudFormation：在整個 CloudFormation 堆疊上偵測偏移
- AWS Marketplace：可用於災難復原的產品
- 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)

## REL13-BP05 自動化復原

使用 AWS 或第三方工具自動化系統復原，並將流量路由到 DR 站點或區域。

根據設定的運作狀態檢查，Elastic Load Balancing 和 AWS Auto Scaling 等 AWS 服務可將負載分散到運作狀態良好的可用區域，而 Amazon Route 5、AWS 和 Global Accelerator 等服務則可將負載路由到運作狀態良好的 AWS 區域。Amazon Route 53 應用程式復原控制器可協助您使用準備度檢查和路由控制功能，來管理和協調容錯移轉。這些功能會持續監控應用程式從失敗中復原的功能，以便您跨多個 AWS 區域、可用區域和內部部署來控制應用程式復原。

對於現有實體或虛擬資料中心或私有雲端上的工作負載， AWS 彈性災難復原(可透過 AWS Marketplace 取得) 可讓組織設定 AWS 的自動化災難復原策略。CloudEndure 也支援 AWS 中的跨區域/跨可用區域災難復原。

常用的反模式：

- 實作相同的自動化容錯移轉和容錯回復會在失敗發生時導致翻動。

建立此最佳實務的優勢： 自動化復原可以消除手動錯誤的機會，減少您的復原時間。

若未建立此最佳實務，暴露的風險等級為： 中

## 實作指引

- 自動化復原路徑。若復原時間較短，則人為判斷和行動無法用於可用性高的方案。系統應在每種情況下都能自動復原。
  - 使用 CloudEndure Disaster Recovery 進行自動化容錯移轉和容錯回復：CloudEndure Disaster Recovery 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低成本階段區域。發生災難時，您可以指示 CloudEndure Disaster Recovery 在數分鐘內自動啟動處於完全佈建狀態的數千部機器。
    - 執行災難復原容錯移轉和退回
    - CloudEndure Disaster Recovery

## 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 Systems Manager Automation
- AWS 的 CloudEndure Disaster Recovery
- AWS 上工作負載的災難復原：在雲端中復原 (AWS 白皮書)

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)

# 可用性目標的實作範例

在本節中，我們將使用由反向代理、Amazon S3 上的靜態內容、應用程式伺服器，以及用於持久性儲存資料的 SQL 資料庫組成的典型 Web 應用程式的部署，來審查工作負載設計。對於每個可用性目標，我們提供一個範例實作。此工作負載可改為使用容器或 AWS Lambda 進行運算，並針對資料庫使用 NoSQL (例如 Amazon DynamoDB)，但方法類似。在每個案例中，我們示範如何透過針對下列主題的工作負載設計來達成可用性目標：

| 主題 | 如需詳細資訊，請參閱此節 |
| --- | --- |
| 監控資源 | 監控工作負載資源 |
| 適應需求變更 | 設計工作負載以適應需求變更 |
| 實作變更 | 實作變更 |
| 備份資料 | 備份資料 |
| 彈性架構師 | 使用故障隔離來保護您的工作負載<br>設計工作負載以承受元件失敗 |
| 測試彈性 | 測試可靠性 |
| 災難復原 (DR) 計畫 | 災難復原 (DR) 計畫 |

# 相依性選擇

我們已選擇將 Amazon EC2 用於我們的應用程式。我們將展示如何使用 Amazon RDS 和多個可用區域來提高應用程式的可用性。我們將使用 Amazon Route 53 執行 DNS。使用多個可用區域時，我們會使用 Elastic Load Balancing。Amazon S3 可用於備份和靜態內容。當我們設計更高的可靠性時，我們必須採用具有更高可用性的服務。

# 單一區域情境

主題

- 99% 情境

## 99% 情境

這些工作負載對業務有幫助，但只有在無法使用時才會造成不便。這類工作負載可以是內部工具、內部知識管理或專案追蹤。或者，這些可以是實際面向客戶的工作負載，但由實驗性服務提供，且內含可視需要隱藏服務的功能切換。

可以在一個區域和一個可用區域中部署這些工作負載。

### 監控資源

我們將進行簡單的監控，以指示服務主頁是否返回 HTTP 200 OK 狀態。當出現問題時，我們的程序手冊將指出可使用來自執行個體的記錄來確定根本原因。

### 適應需求變更

我們將提供有關常見硬體故障、緊急軟體更新和其他破壞性更改的程序手冊。

### 實作變更

我們將使用 AWS CloudFormation 定義我們的 Infrastructure as Code，特別是在發生故障時加快重建速度。

我們將使用執行手冊手動執行軟體更新，並需要停機才能安裝和重啟服務。如果部署期間發生問題，請參閱執行手冊，了解如何回復到以前的版本。

錯誤的任何糾正措施都是透過營運和開發團隊的日誌分析來完成，而且會在排定修正的優先順序並完成修正之後部署糾正措施。

### 備份資料

我們將使用供應商或特定目的的備份解決方案，以使用執行手冊將加密的備份資料傳送到 Amazon S3。我們將透過使用執行手冊還原資料，並確保能夠定期使用資料來測試備份是否能正常工作。我們在 Amazon S3 物件上設定版本控制，並移除備份的刪除許可。我們將根據要求使用 Amazon S3 儲存貯體生命週期政策，來存檔或永久刪除。

### 彈性架構師

在一個區域和一個可用區域中部署工作負載。我們將應用程式 (包括資料庫) 部署到一個執行個體中。

## 測試彈性

新軟體的部署管道已排程好，並測試了一些單位，但大多數是對組裝工作負載的白箱/黑箱測試。

## 災難復原 (DR) 計畫

在故障期間，我們將等待故障完成，可以選擇透過執行手冊來使用 DNS 修改，進而將請求路由到靜態網站。此操作的復原時間取決於基礎設施的部署速度，以及資料庫還原到最新備份的速度。如果使用執行手冊的可用區域發生故障，則此部署可以位於相同的可用區域中，也可以位於不同的可用區域中。

## 可用性設計目標

假設我們部署到新的可用區域，並假設可以在 30 分鐘內還原資料庫，我們需要 30 分鐘的時間來理解並決定執行復原，在 10 分鐘內將整個堆疊部署到 AWS CloudFormation 中。這意味著從故障中復原大約需要 70 分鐘。假設每季度發生一次故障，我們估計全年的影響時間為 280 分鐘，即四小時 40 分鐘。

這表示可用性的上限為 99.9%。實際可用性還取決於實際故障率、故障持續時間以及每個故障實際復原的速度。對於這種架構，我們要求應用程式離線處理更新 (估計每年 24 小時：每次變更四小時，每年六次) 以及實際事件。因此，根據本白皮書前文有關應用程式可用性的資料表，我們發現我們的 可用性設計目標 是 99%。

## 總結

| 主題 | 實作 |
| --- | --- |
| 監控資源 | 僅現場運行狀態；沒有提醒。 |
| 適應需求變更 | 透過重新部署垂直擴展。 |
| 實作變更 | 用於部署和還原的執行手冊。 |
| 備份資料 | 用於備份和還原的執行手冊。 |
| 彈性架構師 | 完成重建；從備份還原。 |
| 測試彈性 | 完成重建；從備份還原。 |
| 災難復原 (DR) 計畫 | 加密的備份，如果需要，還原到其他可用區域。 |

# 99.9% 情境

下一個可用性目標是針對那些必須具有高度可用性的應用程式，但它們可以容忍短期的不可用性。這種類型的工作負載通常用於內部操作，而這些操作會在員工宕機時對他們產生影響。這種類型的工作負載還可以面向客戶，但對於企業而言並不會產生很高的收益，且可以承受更長的復原時間或復原點。此類工作負載包括用於帳戶或訊息管理的管理應用程式。

我們可以透過使用兩個可用區域來進行部署，並將應用程式劃分為不同的層級，從而提高工作負載的可用性。

## 監控資源

透過檢查主頁上的 HTTP 200 OK 狀態，監控範圍將擴大以提醒整個網站的可用性。此外，每次更換 Web 伺服器以及資料庫庫進行容錯移轉時，都會發出提醒。我們還將監控 Amazon S3 上的靜態內容的可用性，並在其不可用時發出提醒。系統將彙總日誌記錄，以簡化管理並幫助進行根本原因分析。

## 適應需求變更

已將自動調整規模功能設定為監控 EC2 執行個體上的 CPU 使用率，以及新增或移除執行個體，使 CPU 目標維持在 70%，但每個可用區域不得少於一個 EC2 執行個體。如果 RDS 執行個體的載入模式指出需要擴展，我們會在維護時段變更執行個體類型。

## 實作變更

基礎設施部署技術保持與先前情境相同的狀態。

新軟體的交付按固定的時間表 (每兩到四週一次) 進行。軟體更新將自動進行，且並不使用 Canary 或藍/綠部署模式，而是使用就地取代。回復的決定將透過執行手冊決定。

我們擁有的程序手冊可用於確定問題的根本原因。在確定根本原因之後，營運和開發團隊將攜手合作，以確定錯誤的糾正措施。在開發修補程序後部署糾正措施。

## 備份資料

可以使用 Amazon RDS 完成備份和還原。我們將使用執行手冊定期執行，以確保我們能夠滿足復原要求。

## 彈性架構師

我們可以透過使用兩個可用區域來進行部署，並將應用程式劃分為不同的層級，從而提高應用程式的可用性。我們將使用可跨多個可用區域工作的服務，例如 Elastic Load Balancing、Auto Scaling 和

Amazon RDS 異地同步備份，並透過 AWS Key Management Service 加密儲存。如此一來，將能確保在資源層級和可用區域層級上對故障的容忍度。

負載平衡器只會將流量路由到運行狀態良好的應用程式執行個體。運行狀態檢查需要在資料平面/應用程式層執行，以指示執行個體上應用程式的功能。此檢查不應與控制平面相悖。隨即將顯示 Web 應用程式的運行狀態檢查 URL，並會將其設定為由負載平衡器和 Auto Scaling 使用，以便刪除及取代失敗的執行個體。如果執行個體在主要可用區域中失敗，則 Amazon RDS 將管理第二可用區域中提供的活躍資料庫引擎，然後進行修復以還原到相同的彈性。

分離各層之後，我們可以使用分散式系統彈性模式來提高應用程式的可靠性，以便即使在可用區域容錯移轉期間資料庫暫時不可用時，該應用程式仍然可用。

### 測試彈性

我們進行功能測試，與之前情境相同。我們不會測試 ELB、自動調整規模或 RDS 容錯移轉的自我修復功能。

我們將提供程序手冊，其中說明了常見資料庫問題、與安全相關的事故和部署失敗。

### 災難復原 (DR) 計畫

我們擁有可用於完整工作負載復原和通用報告的執行手冊。復原使用的備份會存放在與工作負載相同的區域。

### 可用性設計目標

我們假設至少某些故障將需要人工決策才能執行復原。但是，在這種情境下，如果自動化程度更高，我們會假設每年僅需要兩次事件就可以作出此決定。我們需要 30 分鐘的時間來決定執行復原，並假設會在 30 分鐘內完成復原。這意味著需要 60 分鐘才能從故障中復原。假設每年兩次事件，則我們估計全年的影響時間為 120 分鐘。

這表示可用性的上限為 99.95%。實際可用性將還取決於實際故障率、故障持續時間以及每個故障實際復原的速度。對於這種架構，我們要求應用程式短暫離線處理更新，但是這些更新是自動進行的。我們估計為此每年需要 150 分鐘：每次變更 15 分鐘，每年 10 次。當服務不可用時，每年總計 270 分鐘，因此我們的可用性設計目標 是 99.9%。

### 總結

| 主題 | 實作 |
| --- | --- |
| 監控資源 | 僅現場運作狀態檢查；停機時傳送提醒。 |

| 主題 | 實作 |
| --- | --- |
| 適應需求變更 | 適用於 Web 和自動調整規模應用程式層的 ELB；調整異地同步備份 RDS。 |
| 實作變更 | 自動部署到位並執行手冊以進行還原。 |
| 備份資料 | 透過 RDS 自動備份以滿足 RPO 和執行手冊進行還原。 |
| 彈性架構師 | 執行自動調整規模操作以提供自我修復的 Web 和應用程式層；RDS 是異地同步備份。 |
| 測試彈性 | ELB 和應用程式具有自我修復功能；RDS 是多可用區域；沒有明確的測試。 |
| 災難復原 (DR) 計畫 | 透過 RDS 加密備份到同一 AWS 區域。 |

## 99.99% 情境

應用程式的可用性目標要求應用程式具有高可用性，並能夠承受元件故障。該應用程式必須能夠承受故障，而無需獲取其他資源。此可用性目標適用於關鍵任務應用程式，這些應用程式是公司的主要收入來源或重要收入來源，例如電子商務站點，企業對企業 Web 服務或高流量內容/媒體站點。

我們可以透過使用在區域內 靜態穩定 的架構來進一步提高可用性。此可用性目標不需要控制工作負載行為中的平面變更即可容忍故障。例如，應該有足夠的容量來承受一個可用區域的丟失。我們不應要求更新 Amazon Route 53 DNS。無論是建立或修改 S3 儲存貯體，建立新的 IAM 政策 (或政策的修改) 還是修改 Amazon ECS 任務組態，我們都無需建立任何新的基礎設施。

### 監控資源

監控將包括成功指標，並會在出現問題時發出提醒。此外，在每次更換故障的 Web 伺服器，資料庫進行容錯移轉以及可用區域發生故障時，都會發出提醒。

### 適應需求變更

我們將使用 Amazon Aurora 做為 RDS，以自動調整僅供讀取複本規模。對於這些應用程式，在主要內容的寫入可用性的基礎上設計讀取可用性，也是一個關鍵的架構決策。Aurora 也可以視需要對儲存容量執行 Automatic Scaling 操作，以 10 GB 遞增，最多到 64 TB。

## 實作變更

我們將使用 Canary 或藍/綠色部署，以將更新單獨部署到各個隔離區域中。部署是完全自動化的，如果 KPI 指示存在問題，則包括回復。

我們擁有符合嚴格報告需求及效能追蹤的執行手冊。如果成功營運趨向於無法達到效能或可用性目標，則將使用程序手冊來確定導致這種趨勢的原因。我們擁有針對未發現的故障模式和安全事故的程序手冊。我們還擁有可用於確定失敗的根本原因的程序手冊。我們還將與 AWS Support for Infrastructure Event Management 產品互動。

建立和營運該網站的團隊將識別任何意外故障的錯誤糾正措施，並在實作修補程序後優先考慮部署該修補程序。

## 備份資料

可以使用 Amazon RDS 完成備份和還原。我們將使用執行手冊定期執行，以確保我們能夠滿足復原要求。

## 彈性架構師

我們為此方法提供了三個可用區域的建議。使用三個可用區域部署，每個可用區域的靜態容量為峰值的 50%。可以使用兩個可用區域，但是靜態穩定容量的成本會更高，因為兩個區域均須具有峰值容量的 100%。我們將新增 Amazon CloudFront 以提供地理位置快取，並減少應用程式資料平面上的請求。

我們將使用 Amazon Aurora 做為 RDS，並在這三個區域中部署僅供讀取複本。

將在所有層中使用軟體/應用程式彈性模式來建置應用程式。

## 測試彈性

部署管道將包含完整的測試套件，包括效能、負載和故障注入測試。

我們將在演練日期間持續演練我們的故障復原程序，使用執行手冊可確保我們能執行任務並且不會偏離程序。建立網站的團隊也經營該網站。

## 災難復原 (DR) 計畫

我們擁有可用於完整工作負載復原和通用報告的執行手冊。復原使用的備份會存放在與工作負載相同的區域。還原程序會在演練日期間定期執行。

## 可用性設計目標

我們假設至少某些故障將需要人工決策才能執行復原，但是在這種情況下，如果具有更高的自動化程度，我們會假設每年只需兩次事件就需要執行此決策，並且能快速執行復原動作。我們需要 10 分鐘的時間來決定執行復原，並假設會在五分鐘內完成復原。這意味著需要 15 分鐘才能從故障中復原。假設每年兩次故障，則我們估計全年的影響時間為 30 分鐘。

這表示可用性的上限為 99.99%。實際可用性還取決於實際故障率、故障持續時間以及每個因素實際復原的速度。對於這種架構，我們假設應用程式可透過更新持續聯網。因此，我們的 可用性設計目標 為 99.99%。

## 總結

| 主題 | 實作 |
| --- | --- |
| 監控資源 | 在所有層和 KPI 上執行運行狀態；觸發已設定提醒時傳送的提醒；進而提醒所有故障。營運會議將嚴格偵測趨勢並管理設計目標。 |
| 適應需求變更 | 適用於 Web 和自動調整規模應用程式層的 ELB；針對 Aurora RDS 在多個區域中自動調整儲存空間和僅供讀取複本規模。 |
| 實作變更 | 當 KPI 或提醒揭示應用程式中未檢測到問題時，可透過 Canary 或藍/綠自動部署並自動還原。部署是透過隔離區域進行的。 |
| 備份資料 | 透過 RDS 進行自動備份，以滿足 RPO 和在演練日定期進行的自動復原。 |
| 彈性架構師 | 為應用程式實作了故障隔離區；自動擴展以提供我修復的 Web 和應用程式層；RDS 是多可用區域。 |
| 測試彈性 | 元件和隔離區域的故障測試正在進行中，並會在演練日定期與營運人員進行練習；存在用於診斷未知問題的程序手冊；且存在根本原因分析過程。 |

| 主題 | 實作 |
| --- | --- |
| 災難復原 (DR) 計畫 | 透過 RDS，可將加密的備份錄入至演練日中使用的相同 AWS 區域。 |

# 多區域情境

在多個 AWS 區域中實作我們的應用程式會增加營運成本，部分原因是我們為確保其自主權而隔離了區域。走這條路應是經過深思熟慮的決定。也就是說，區域提供了很強的隔離邊界，而我們會竭盡全力避免跨區域的相關故障。在區域 AWS 服務上發生硬相依性故障時，使用多個區域可讓您更好地控制復原時間。在本節中，我們將討論各種實作模式及其典型的可用性。

主題



## 99.95%，復原時間在 5 到 30 分鐘之間

要實現應用程式的這一可用性目標，需要停機時間極短，並且在特定時間幾乎沒有資料丟失。具有此可用性目標的應用程式包括以下領域的應用程式：銀行、投資、緊急服務和資料捕獲。這些應用程式的復原時間和復原點非常短。

透過在兩個 AWS 區域之間使用 「暖待命」 跨兩個 AWS 區域的方法。我們會將整個工作負載部署到兩個區域，縮小被動站點的規模，並使所有資料最終保持一致。兩個部署在 各自區域內都保持 靜態穩定。應使用分散式系統彈性模式來建置應用程式。我們需要建 立輕量型路由元件， 該元件可監控工作負載的運作狀態，並視需要設定為將流量路由到被動區域。

### 監控資源

在每次更換 Web 伺服器，資料庫和區域進行容錯移轉時，都會發出提醒。我們還將監控 Amazon S3 上的靜態內容的可用性，並在其不可用時發出提醒。系統將彙總日誌記錄，以簡化管理並幫助每個區域進行根本原因分析。

路由元件會同時監控我們的應用程式運作狀態和所擁有的任何區域硬相依性。

### 適應需求變更

與 99.99% 情境相同。

## 實作變更

新軟體的交付按固定的時間表 (每兩到四週一次) 進行。軟體更新將使用 Canary 或藍/綠部署模式自動進行。

執行手冊適用於發生區域故障移轉的時間，事件期間發生的常見客戶問題以及常見報告。

我們將提供程序手冊，其中說明了常見資料庫問題、與安全相關的事故、部署失敗、區域故障移轉上的意外客戶問題以及確定問題根本原因。在確定根本原因之後，營運和開發團隊將攜手合作，以確定錯誤的糾正措施並在開發修補程序時部署這些措施。

我們還將與 AWS Support for Infrastructure Event Management 接洽。

## 備份資料

與 99.99% 情境一樣，我們會使用自動化 RDS 備份並使用 S3 版本控制。系統會自動非同步地將資料從主動區域中的 Aurora RDS 叢集，複寫到被動區域中的跨區域僅供讀取複本。S3 跨區域複寫可用來自動非同步地將資料從主動區域移至被動區域。

## 彈性架構師

與 99.99% 情境相同，且可能發生區域容錯移轉。此為手動管理過程。在容錯移轉期間，我們將使用 DNS 備援將請求路由到靜態網站，直到在第二個區域中進行復原。

## 測試彈性

與 99.99% 情境相同，我們將使用執行手冊在整個演練日驗證架構。此外，RCA 糾正措施應優先於功能發佈，以便即時實作和部署

## 災難復原 (DR) 計畫

區域容錯移轉是手動管理過程。所有資料都會以非同步方式複寫。暖待命中的 基礎架構 會向外擴展。這可以使用 AWS Step Functions 上執行的工作流程來自動化。AWS Systems Manager (SSM) 也可協助進行此自動化，因為您可以建立更新 Auto Scaling 群組和調整執行個體大小的 SSM 文件。

## 可用性設計目標

我們假設至少某些故障需要人工決策才能執行復原，但是在這種情況下，如果具有良好的自動化程度，我們會假設每年只需兩次事件就需要執行此決策。我們需要 20 分鐘的時間來決定執行復原，並假設會

在 10 分鐘內完成復原。這意味著從故障中復原需要 30 分鐘。假設每年兩次故障，則我們估計全年的影響時間為 60 分鐘。

這表示可用性的上限為 99.95%。實際可用性還取決於實際故障率、故障持續時間以及每個因素實際復原的速度。對於這種架構，我們假設應用程式可透過更新持續聯網。因此，我們的 可用性設計目標 為 99.95%。

總結

| 主題 | 實作 |
| --- | --- |
| 監控資源 | 在所有層和 KPI 上執行運行狀態，包括 AWS 區域層級上的 DNS 運行狀態及 KPI；觸發已設定提醒時傳送的提醒；進而提醒所有故障。營運會議將嚴格偵測趨勢並管理設計目標。 |
| 適應需求變更 | 適用於 Web 和自動調整規模應用程式層的 ELB；針對 Aurora RDS 在主動和被動區域的多個區域中自動調整儲存空間和僅供讀取複本規模。在 AWS 區域之間同步資料和基礎設施以實現靜態穩定性。 |
| 實作變更 | 當 KPI 或提醒揭示應用程式中未檢測到問題時，可透過 Canary 或藍/綠自動部署並自動還原，一次在一個 AWS 區域的一個隔離區域進行部署。 |
| 備份資料 | 透過 RDS 在每個 AWS 區域中進行自動備份，以滿足 RPO 和在演練日定期進行的自動復原。Aurora RDS 和 S3 資料會自動非同步地從主動區域複寫到被動區域。 |
| 彈性架構師 | 執行自動調整規模操作以提供自我修復的 Web 和應用程式層；RDS 是異地同步備份；手動管理區域容錯移轉，同時在容錯移轉時顯示靜態站點。 |
| 測試彈性 | 元件和隔離區域的故障測試正在進行中，並會在演練日定期與營運人員進行練習；存在用於診斷未知問題的程序手冊；並且存在根本原因分析流 |

| 主題 | 實作 |
| --- | --- |
| | 程，其中該流程帶有通訊路徑以查找問題的根源以及糾正和預防的方式。RCA 糾正應優先於功能發佈，以便即時實作和部署。 |
| 災難復原 (DR) 計畫 | 暖待命部署在其他區域。使用 AWS Step Functions 或 AWS Systems Manager 文件執行的工作流程擴展基礎設施。透過 RDS 加密備份。兩個 AWS 區域之間的跨區域僅供讀取複本。Amazon S3 中靜態資產的跨區域複寫。還原到目前活躍的 AWS 區域，在演練日實作，並與 AWS 協調。 |

## 99.999% 或更高情境，復原時間低於 1 分鐘

應用程式的可用性目標要求在特定時間幾乎沒有停機時間及資料丟失。可能具有此可用性目標的應用程式包括，例如某些銀行、投資、金融、政府和關鍵業務應用程式，它們是產生巨大收入之業務的核心業務。期望在所有層上具有高度一致的資料存放區和完全冗餘。我們選擇了一個基於 SQL 的資料存放區。但是，在某些情況下，我們發現很難實現非常小的 RPO。如果可以對資料進行分區，則可能不會丟失資料。這可能需要您新增應用程式邏輯和延遲，從而確保在地理位置之間具有一致的資料，並具有在分區之間移動或複製資料的功能。如果您使用 NoSQL 資料庫，執行此分區可能會更容易。

我們可以透過跨多個 AWS 區域使用 主動-主動 跨多個 AWS 區域的方法。工作負載將部署在所有跨區域 靜態穩定的 所需區域中 (因此剩餘區域可以處理遺失一個區域的負載)。路由層 會將流量 定向到運作狀態良好的地理位置，並在位置運作狀態不佳時自動變更目的地，並暫時停止資料複寫層。Amazon Route 53 可提供 10 秒的間隔運作狀態檢查，並且還為您的記錄集提供 TTL (低至一秒)。

### 監控資源

與 99.95% 情境相同，且當某個區域被偵測為運作狀態不佳並從中將流量路由離開時發出提醒。

### 適應需求變更

與 99.95% 情境相同。

## 實作變更

部署管道將包含完整的測試套件，包括效能、負載和故障注入測試。我們將使用 Canary 或藍/綠部署，一次將更新部署到一個區域中的一個隔離區域，然後再開始另一個區域。在部署期間，舊版本仍將在執行個體上繼續執行，以加快回復速度。這些是完全自動化的，如果 KPI 指示存在問題，則包括回復。監控將包括成功指標，並會在出現問題時發出提醒。

我們擁有符合嚴格報告需求及效能追蹤的執行手冊。如果成功營運趨向於無法達到效能或可用性目標，則將使用程序手冊來確定導致這種趨勢的原因。我們擁有針對未發現的故障模式和安全事故的程序手冊。我們還擁有可用於確定失敗的根本原因的程序手冊。

建立網站的團隊也經營該網站。該團隊將識別任何意外故障的錯誤糾正措施，並在實作修補程序後優先考慮部署該修補程序。我們還將與 AWS Support for Infrastructure Event Management 接洽。

## 備份資料

與 99.95% 情境相同。

## 彈性架構師

應使用軟體/應用程式彈性模式來建置應用程式。可能需要許多其他路由層來實現所需的可用性。不應低估此附加實作的複雜性。該應用程式將在部署故障隔離區域中實作，並進行分區和部署，如此一來，即使是區域範圍的事件也不會影響所有客戶。

## 測試彈性

我們將在演練日期間驗證架構，使用執行手冊可確保我們能執行任務並且不會偏離程序。

## 災難復原 (DR) 計畫

主動-主動 多區域部署，可在多個區域提供完整的工作負載基礎設施和資料。透過使用讀取本機、寫入全域策略，一個區域是所有寫入的主要資料庫，且會複寫資料以讀取到其他區域。如果主要資料庫區域失敗，則需要提升新的資料庫。透過讀取本機、寫入全域，將使用者指派給處理資料庫寫入的主區域。這讓使用者可以從任何區域讀取或寫入，但需要複雜的邏輯來管理不同區域中跨寫入的潛在資料衝突。

當某個區域被偵測為運作狀態不佳時，路由層會自動將流量路由到其餘運作狀態良好的區域。不需要手動介入。

資料存放區在區域之間的複寫方式必須能夠解決潛在的衝突。由於延遲的原因，將需要建立工具和自動化程序來在分區之間複製或移動資料，並平衡每個分區中的請求或資料數量。補救資料衝突的方法也將需要其他營運執行手冊。

## 可用性設計目標

我們假設已為自動化所有復原進行了大量投資，並且在一分鐘內完成了復原。我們假設沒有手動觸發的復原，但是每季度最多執行一次自動復原動作。這意味著每年需要四分鐘才能復原。我們假設應用程式可透過更新持續聯網。因此，我們的 可用性設計目標 為 99.999%。

總結

| 主題 | 實作 |
| --- | --- |
| 監控資源 | 在所有層和 KPI 上執行運行狀態，包括 AWS 區域層級上的 DNS 運行狀態及 KPI；觸發已設定提醒時傳送的提醒；進而提醒所有故障。營運會議將嚴格偵測趨勢並管理設計目標。 |
| 適應需求變更 | 適用於 Web 和自動調整規模應用程式層的 ELB；針對 Aurora RDS 在主動和被動區域的多個區域中自動調整儲存空間和僅供讀取複本規模。在 AWS 區域之間同步資料和基礎設施以實現靜態穩定性。 |
| 實作變更 | 當 KPI 或提醒揭示應用程式中未檢測到問題時，可透過 Canary 或藍/綠自動部署並自動還原，一次在一個 AWS 區域的一個隔離區域進行部署。 |
| 備份資料 | 透過 RDS 在每個 AWS 區域中進行自動備份，以滿足 RPO 和在演練日定期進行的自動復原。Aurora RDS 和 S3 資料會自動非同步地從主動區域複寫到被動區域。 |
| 彈性架構師 | 為應用程式實作了故障隔離區；自動擴展以提供我修復的 Web 和應用程式層；RDS 是多可用區域；自動化區域故障移轉。 |
| 測試彈性 | 元件和隔離區域的故障測試正在進行中，並會在演練日定期與營運人員進行練習；存在用於診斷未知問題的程序手冊；並且存在根本原因分析流程，其中該流程帶有通訊路徑以查找問題的根源 |

| 主題 | 實作 |
| --- | --- |
| | 以及糾正和預防的方式。RCA 糾正應優先於功能發佈，以便即時實作和部署。 |
| 災難復原 (DR) 計畫 | 主動-主動部署到至少兩個區域。基礎設施可在區域全面擴展並實現靜態穩定。資料會跨區域分割和同步。透過 RDS 加密備份。區域故障在演練日練習，並與 AWS 協調。在還原期間，可能需要提升新的主要資料庫。 |

# 資源

## 文件

- Amazon Builders' Library – Amazon 如何建置和操作軟體
- AWS 架構中心

## 實驗室

- AWS Well-Architected 可靠性實驗室

## 外部連結

- 調整式佇列模式： 大規模故障
- 可用性和超越：了解和改善 AWS 上分散式系統的彈性

## 書籍

- Robert S. Hammer「容錯軟體的模式」
- Andrew Tanenbaum 和 Marten van Steen「分散式系統：原則與典範」

# 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引發您的思考，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以設計、營運和復原為導向的建議以及我們的 AWS 解決方案架構師的知識和經驗。我們希望收到您的來信，特別是關於您在 AWS 上實現高可用性的成功案例。請聯絡您的客戶團隊或使用 我們網站上的「聯絡我們」。

# 作者群

此文件的作者包括：

- Seth Eliot，Amazon Web Services 首席開發人員律師
- Mahanth Jayadeva，Amazon Web Services Well-Architected 解決方案架構師
- Sajee Mathew，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 App Modernization 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 首席技術程式經理
- 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 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 Well-Architected Framework
- 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 |
| --- | --- | --- |
| 白皮書已更新 | 最佳實務更新了新的實作指引。 | June 27, 2024 |
| 已更新最佳實務指引 | 最佳實務已更新，納入了以下方面的新指引：在分散式系統中設計防止故障的互動、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。 | December 6, 2023 |
| 已更新最佳實務指引 | 最佳實務已更新，納入了以下方面的新指引：監控工作負載資源和設計工作負載以承受元件失敗。 | October 3, 2023 |
| 已更新最佳實務指引 | 最佳實務已更新，納入了以下方面的新指引：設計您的工作負載服務架構、設計分散式系統中的互動以緩解或承受失敗和監控工作負載資源。 | July 13, 2023 |
| 小幅度更新 | 移除非包容性語言。 | April 13, 2023 |
| 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。 | April 10, 2023 |
| 白皮書已更新 | 最佳實務更新了新的實作指引。 | December 15, 2022 |

| | | |
|---|---|---|
| 小幅度更新 | 更正了圖片編號號和整體小幅變更。 | November 17, 2022 |
| 白皮書已更新 | 已擴充最佳實務並新增了改善計劃。 | October 20, 2022 |
| 白皮書已更新 | 已將兩個新的最佳實務加入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以及設計工作負載以承受元件失敗。 | May 5, 2022 |
| 小幅度更新 | 已將永續性支柱新增至簡介。 | December 2, 2021 |
| 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。 | October 26, 2021 |
| 小幅度更新 | 已新增 AWS Fault Injection Service (AWS FIS) 的相關資訊。 | March 15, 2021 |
| 小幅度更新 | 小幅度文字更新。 | January 4, 2021 |

| | | |
|---|---|---|
| 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 | December 7, 2020 |
| 小幅度更新 | 已更新附錄 A 來更新 AWS Lambda 的可用性設計目標 | October 27, 2020 |
| 小幅度更新 | 已更新附錄 A 來新增 AWS Global Accelerator 的可用性設計目標 | July 24, 2020 |

| | | |
|---|---|---|
| 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。 | July 8, 2020 |
| 小幅度更新 | 修正了中斷的連結 | October 1, 2019 |
| 白皮書已更新 | 更新了附錄 A | April 1, 2019 |
| 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 | September 1, 2018 |
| 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。 | June 1, 2018 |

| | | |
|---|---|---|
| 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 | March 1, 2018 |
| 小幅度更新 | 對可用性計算的細微糾正，以包括應用程式可用性 | December 1, 2017 |
| 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。 | November 1, 2017 |
| 初版 | 可靠性支柱 – AWS Well Architected Framework 已發佈。 | November 1, 2016 |