偵測和緩解灰階故障

進階異地同步備份復原模

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

進階異地同步備份復原模: 偵測和緩解灰階故障

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

摘要及介紹 i
簡介 1
灰色故障 2
差分可觀測性 2
灰色故障範例 4
回應灰階故障 5
異地同步備份可觀測 8
使用 CloudWatch 複合式警報偵測故障 11
偵測單一可用區域中的影響 12
確保影響不是區域 13
確保影響不是由單一執行個體造成 13
整合練習 15
使用離群值偵測進行故障偵測 16
單一執行個體區域資源的故障偵測 20
Summary 22
可用區域疏散模式 23
可用區域獨立 23
控制平面和資料平面 27
數據平面控制疏散 28
路線 53 應用程式復原控制器 (ARC) 中的區域移位 29
Route 53 ARC 30
使用自我管理的 HTTP 端點 31
控制平面控制疏散 36
總結 39
結論 41
附錄 A — 取得可用區域識別碼 42
附錄 B-示例卡方計算 44
貢獻者 50
文件修訂 51
注意 52
AWS 詞彙表 53
..... liv

進階異地同步備份復原模

出版日期：2023年7月11日(文件修訂)

許多客戶在高可用性的多重可用區域 (AZ) 組態中執行工作負載。這些架構在二進制故障事件期間表現良好，但經常遇到問題灰色失敗。這種類型的故障的表現可能是微妙的，並且無視快速而明確的檢測。本白皮書提供指引，說明如何檢測工作負載以偵測灰色故障所造成的影響 (隔離至單一可用區域)，然後採取行動以減輕可用區域中的影響。

簡介

本文件的目的是協助您更有效地實作具備彈性的異地同步備份架構。建立彈性系統的最佳做法之一亞馬遜虛擬私有雲（VPC）網絡是將每個工作負載部署到多個可用區。

一個可用區域是一或多個具備備援電源、網路和連線能力的獨立資料中心。使用多個可用區域可讓您操作比單一資料中心更具可用性、容錯能力和可擴充性的工作負載。

許多AWS服務，例如亞馬遜彈性運算雲（EC2）自動擴展或者亞馬遜關聯式資料庫（亞馬遜 RDS），提供異地同步備份配置。這些服務不需要您建立任何額外的可觀測性或容錯移轉工具。它們使工作負載具有彈性，可以輕鬆檢測到的二進制故障模式AWS 區域會影響單一可用區域。這可能是完全實體硬體故障、電源中斷或影響大部分資源的潛在軟體錯誤。

但是還有另一類失敗稱為灰色失敗, 其表現是微妙的，違抗快速和明確的檢測. 這反過來會導致更長的時間，以減輕故障引起的影響。本白皮書著重於灰色故障可能對異地同步備份架構造成的影響、如何偵測它們，以及如何減輕故障。

本白皮書提供的指引主要適用於以下特定類別的工作負載：

- 主要使用區域AWS服務
- 需要提高單一區域的韌性
- 願意進行大量投資以建立所需的可觀察性和彈性模式

在這些工作負載中，您可能不願意做出一些或全部的權衡???，或者沒有使用多個區域的選項。這些類型的工作負載很可能代表整體產品組合的一小部分，因此應在工作負載層級與平台層級考慮此指引。

灰色故障

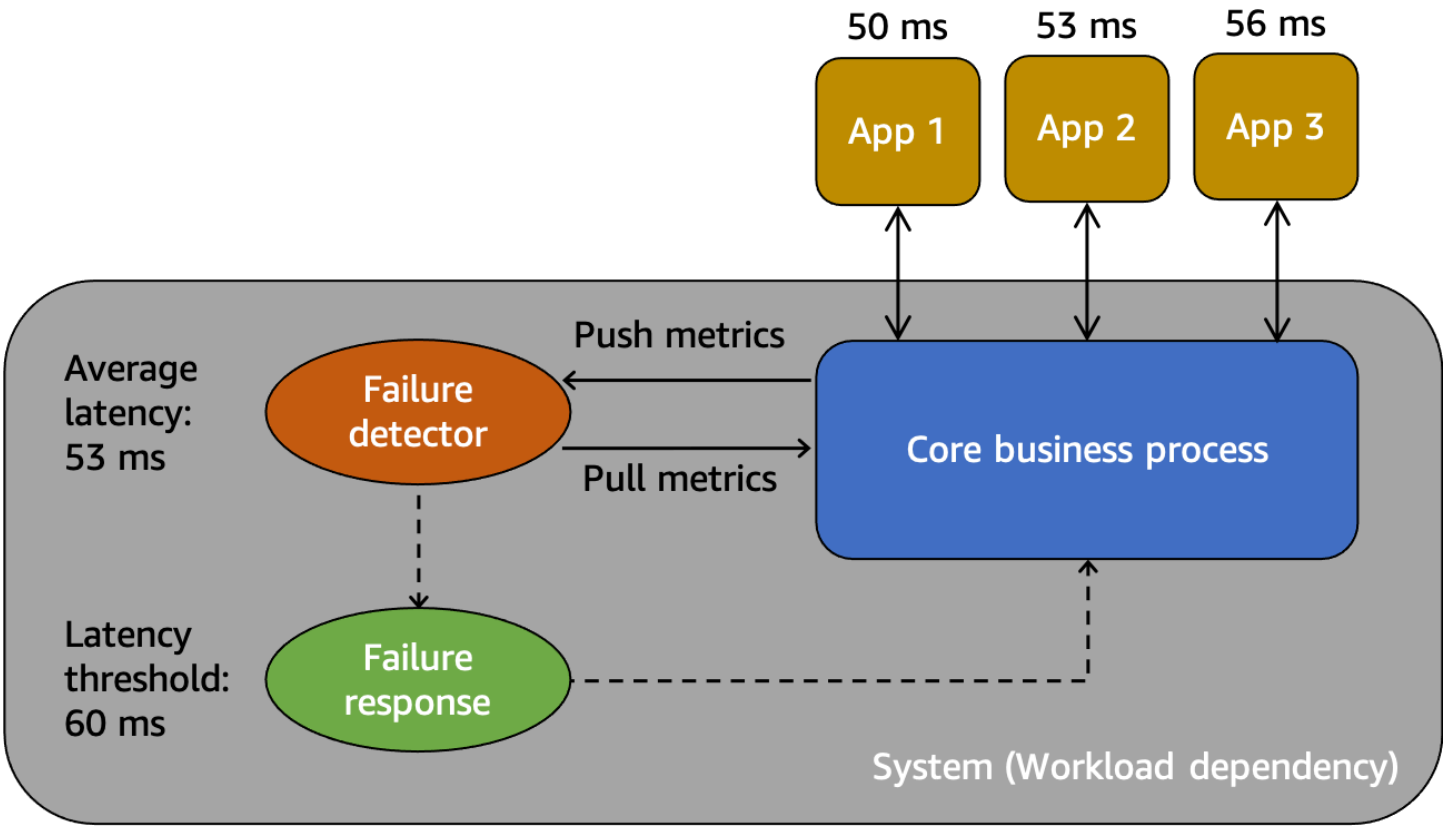
灰色故障由特徵定義差分可觀測性，這意味著不同的實體以不同的方式觀察失敗。讓我們來定義這意味著什麼。

差分可觀測性

您操作的工作負載通常具有相依性。例如，這些可以是AWS用於建置工作負載的雲端服務，或用於聯合的第三方身分識別提供者 (IdP)。這些依賴關係幾乎總是實現自己的可觀察性，記錄有關錯誤，可用性和延遲的指標以及其他由客戶使用情況生成的內容。當超過這些度量之一的臨界值時，相依性通常會採取一些動作來更正它。

這些依賴關係通常有他們的服務的多個消費者。消費者還實現了自己的可觀察性，並記錄有關其與其依賴關係的交互的指標和日誌，記錄諸如磁盤讀取中有多少延遲，API 請求失敗了多少，或者數據庫查詢花了多長時間。

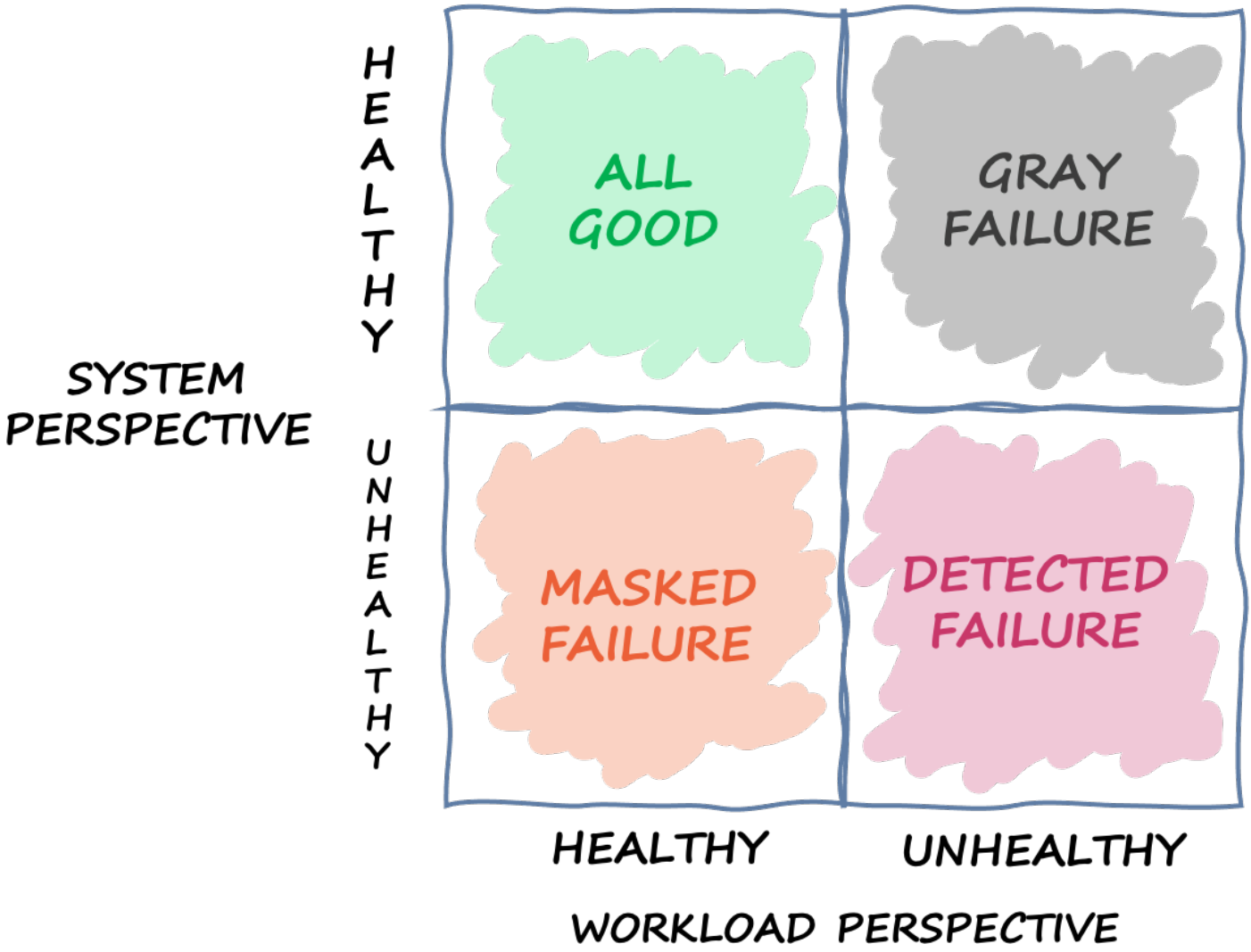
這些相互作用和測量在下圖中的抽象模型中描繪。



理解灰色失敗的抽象模型

首先，我們有系統，這是消費者應用程式 1、應用程式 2 和應用程式 3 在此案例中的相依性。該系統具有故障檢測器，用於檢查從核心業務流程創建的指標。它還具有故障響應機制，以減輕或糾正故障檢測器觀察到的問題。系統會看到 53 ms 的整體平均延遲，並且已設定臨界值，以在平均延遲超過 60 ms 時叫用失敗回應機制。應用程序 1，應用程序 2 和應用程序 3 也對其與系統的交互進行了自己的觀察，分別記錄 50 毫秒，53 毫秒和 56 毫秒的平均延遲。

差異觀測性是指系統消費者偵測到系統運作狀況不佳，但系統本身的監控未偵測到問題，或影響未超過警示臨界值的情況。讓我們想像一下，應用程序 1 開始經歷 70 毫秒而不是 50 毫秒的平均延遲。應用程式 2 和應用程式 3 看不到平均延遲的變化。這會將基礎系統的平均延遲時間增加到 59.66 ms，但這不會跨越延遲閾值來啟動故障回應機制。但是，應用程序 1 的延遲增加了 40%。這可能會超過針對應用程式 1 設定的用戶端逾時來影響其可用性，或者可能會在較長的互動鏈中造成串聯影響。從 App 1 的角度來看，它所依賴的基礎系統是不健康的，但從系統本身以及 App 2 和 App 3 的角度來看，系統是健康的。下圖總結了這些不同的觀點。



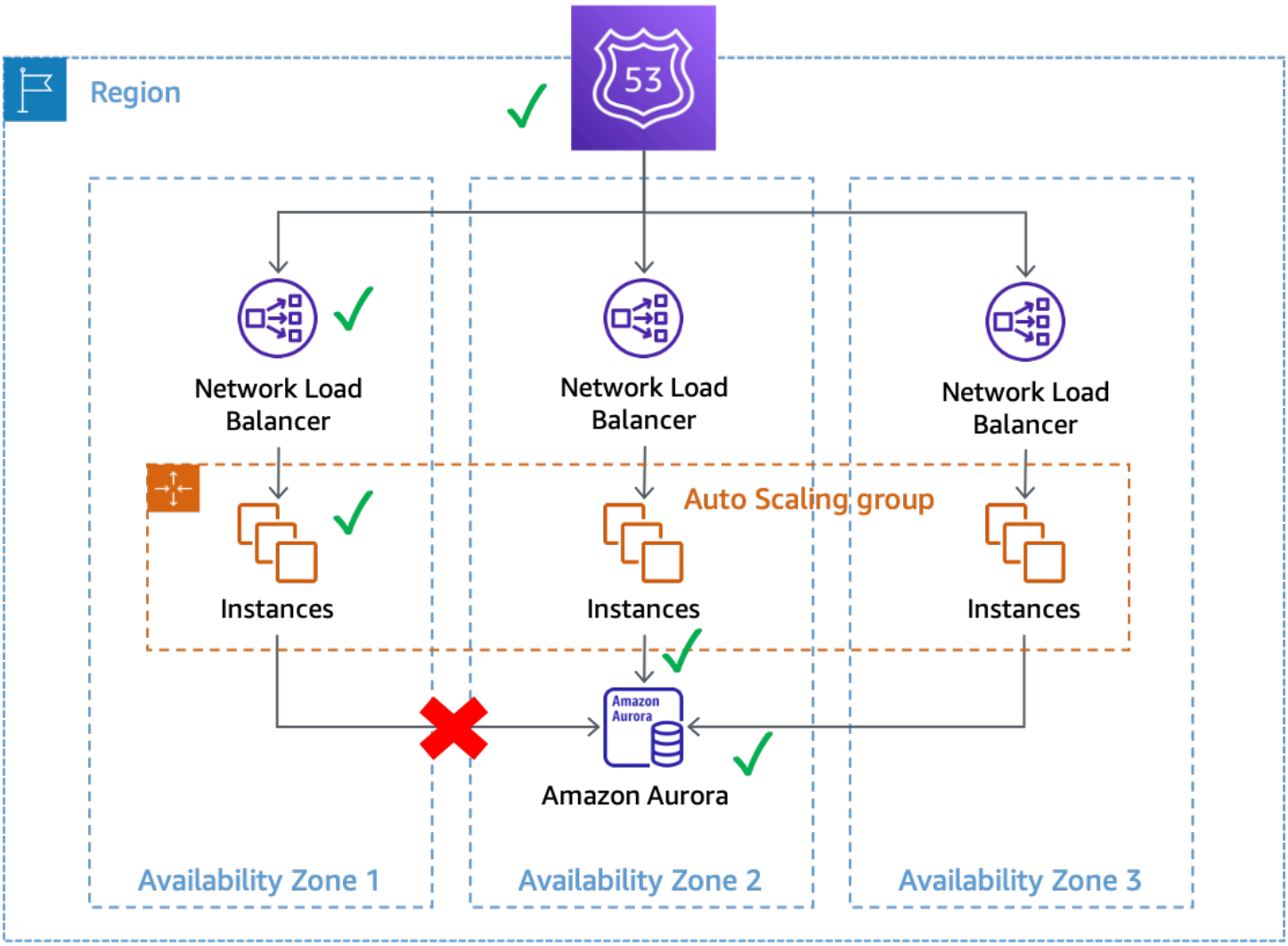
根據不同的觀點，定義系統可處於不同狀態的象限

失敗也可以遍歷此象限。事件可能會以灰色失敗的形式開始，然後變成偵測到的失敗，然後移至遮罩失敗，然後又可能回到灰色失敗。沒有一個定義的循環，並且在解決其根本原因之前，幾乎總是有失敗復發的機會。

我們從中得出的結論是，工作負載不能總是依賴基礎系統來檢測和緩解故障。無論底層系統有多複雜和彈性，總有可能失敗未被發現或停留在反應閾值之下。該系統的消費者（如 App 1）需要配備，以便快速檢測和減輕灰色故障引起的影響。這需要為這些情況構建可觀察性和恢復機制。

灰色故障範例

灰色故障可能會影響異地同步備份系統AWS。例如，採取的艦隊亚马逊在三個可用區域部署的 Auto Scaling 群組中的執行個體。它們都連接到一個可用區域中的 Amazon Aurora 資料庫。然後，會發生灰色故障，影響可用區域 1 和可用區域 2 之間的網路。造成此損害的結果是，可用區域 1 中執行個體的新資料庫和現有資料庫連線的百分比失敗。這種情況如下圖所示。



影響可用區域 1 中執行個體的資料庫連線的灰色失敗

在此範例中，Amazon EC2 會將可用區域 1 中的執行個體視為健康狀態良好，因為它們持續通過系統和執行個體狀態檢查。Amazon EC2 自動擴展也不會偵測到對任何可用區域的直接影響，而且會繼續已設定可用區域中的啟動容量。網路負載平衡器 (NLB) 也會將其後面的執行個體視為健全狀況，就像針對 NLB 端點執行的 Route 53 健康狀態檢查一樣健全。同樣地，亞馬遜關聯式資料庫服務 (Amazon RDS) 會將資料庫叢集視為健康狀況良好，且不會觸發自動容錯移轉。我們有許多不同的服務，都將其服務和資源視為健康狀態，但工作負載會偵測到會影響其可用性的故障。這是一個灰色的失敗。

回應灰階故障

當您遇到灰色故障時AWS環境中，您通常有三個可用選項：

- 什麼都不做，等待減值結束。
- 如果將減值隔離到單一可用區域，請撤除該可用區域。
- 容錯移轉至其他AWS 區域並使用的好處AWS區域隔離以減輕影響。

許多AWS客戶對大多數工作負載都可以使用選項一。他們接受可能擴展復原時間目標 (RTO)他們不必建立額外的可觀察性或彈性解決方案的權衡。其他客戶選擇實施第三個選項，多區域災難復原（DR），作為他們對各種故障模式的緩解計劃。在這些情況下，多區域架構可以很好地運作。但是，使用此方法時有一些權衡（請參閱AWS多區域基礎知識有關多區域考量的完整討論）。

首先，建置和操作多區域架構可能是一項具有挑戰性、複雜且可能昂貴的工作。多區域架構需要仔細考慮哪些DR 策略您選擇。實作多區域主動-主動式 DR 解決方案來處理區域損傷可能不是財務上可行的，而備份和還原策略可能不符合您的彈性需求。此外，必須在生產環境中持續實施多區域容錯移轉，以便您相信它們可以在需要時運作。這一切都需要大量專門的時間和資源來構建，操作和測試。

二、跨資料複製AWS 區域運用AWS今天的服務都是異步完成的。非同步複製可能會導致資料遺失。這表示在區域容錯移轉期間，可能會造成一定程度的資料遺失和不一致。您對數據丟失量的容忍度定義為復原點目標 (RPO)。需要強大資料一致性的客戶，必須建立對帳系統，以便在主要區域再次推出時修正這些一致性問題。或者，他們必須建置自己的同步複寫或雙寫系統，這可能會對回應延遲、成本和複雜性產生重大影響。它們還使次要區域成為每個交易的硬性依賴性，這可能會降低整體系統的可用性。

最後，對於使用主動/待命方法的許多工作負載，執行容錯移轉到另一個區域所需的時間非零。您的工作負載產品組合可能需要以特定順序在主要區域中斷、需要耗盡連線或停止特定程序。然後，可能需要按特定順序恢復服務。也可能需要佈建新資源，或需要時間才能通過所需的健康狀態檢查，然後才能進入服務。此容錯移轉程序可能會經歷一段完全無法使用的時間。這就是 RTO 所關注的。

在一個區域內，許多AWS服務提供高度一致的資料持續性。Amazon RDS 異地同步備份部署使用同步複寫。亞馬遜簡單存儲服務（亞馬遜 S3）優惠強大read-after-write一致性。亞馬遜彈性區塊儲存(亞馬遜 EBS) 優惠多磁碟區當機一致快照。亚马逊可以執行強烈一致的讀取。這些功能可協助您在單一區域中達到較低的 RPO (在大多數情況下為零 RPO)，而不是在多區域架構中所能達到的。

撤除可用區域的 RTO 可能比多區域策略低，因為您的基礎結構和資源已在可用區域佈建。當可用區域受損時，異地同步備份架構可以繼續以靜態方式繼續運作，而不需要小心訂購正在關閉和備份的服務，或排除連線。由於工作轉移到剩餘的可用區域，許多系統可能只會在區域容錯移轉期間發生一段完全無法使用的時間，而不是在可用區域撤離期間發生的完全無法使用時間。如果系統已被設計為靜態穩定可用區域失敗 (在此情況下，表示在其他可用區域中預先佈建容量以吸收負載)，工作負載的客戶可能根本看不到影響。

> 單一可用區域的損害可能會影響一個或多個AWS 區域服務除了您的工作量。如果您觀察到區域影響，則應將事件視為區域服務減損，儘管該影響的來源來自單一可用區域。撤除可用區域不會減輕此類問題。發生這種情況時，請使用您所擁有的回應計劃來回應區域服務損害。

本文件的其餘部分著重於撤除可用區域的第二個選項，以便針對單一可用區域灰色故障達到更低的 RTO 和 RPO。這些模式有助於實現異地同步備份架構的更高價值和效率，對於大多數類別的工作負載，可減少建立多區域架構以處理這些類型的事件的需求。

異地同步備份可觀測

若要能夠在與單一可用區域隔離的事件期間撤除可用區域，您首先必須能夠偵測到故障實際上是隔離到單一可用區域。這需要對系統在每個可用區域中的行為進行高保真度能見度。許多 AWS 服務提供的 out-of-the-box 指標可提供有關您資源的營運見解。例如，Amazon EC2 提供了許多指標，例如CPU使用率、磁碟讀取和寫入以及網路流量進出。

不過，當您建置使用這些服務的工作負載時，您需要的不僅僅是標準指標，還需要更多的可見性。您想要瞭解工作負載所提供的客戶體驗。此外，您還需要將指標與產生指標的可用區域保持一致。這是您需要檢測差異觀察灰色故障的洞察力。該級別的可見性需要儀器。

儀器需要編寫明確的代碼。此代碼應該執行諸如記錄任務需要多長時間，計算成功或失敗的項目數量，收集有關請求的元數據等等。您還需要提前定義閾值，以定義什麼被認為是正常的，什麼是不正常的。您應該概述工作負載中延遲、可用性和錯誤計數的目標和不同嚴重性閾值。Amazon 建置者程式庫文章檢測分散式系統以獲得操作可見性，提供了許多最佳實務。

指標都應該從服務器端以及客戶端生成。產生用戶端指標和瞭解客戶體驗的最佳做法是使用 Can ary，該軟體會定期探測您的工作負載並記錄指標。

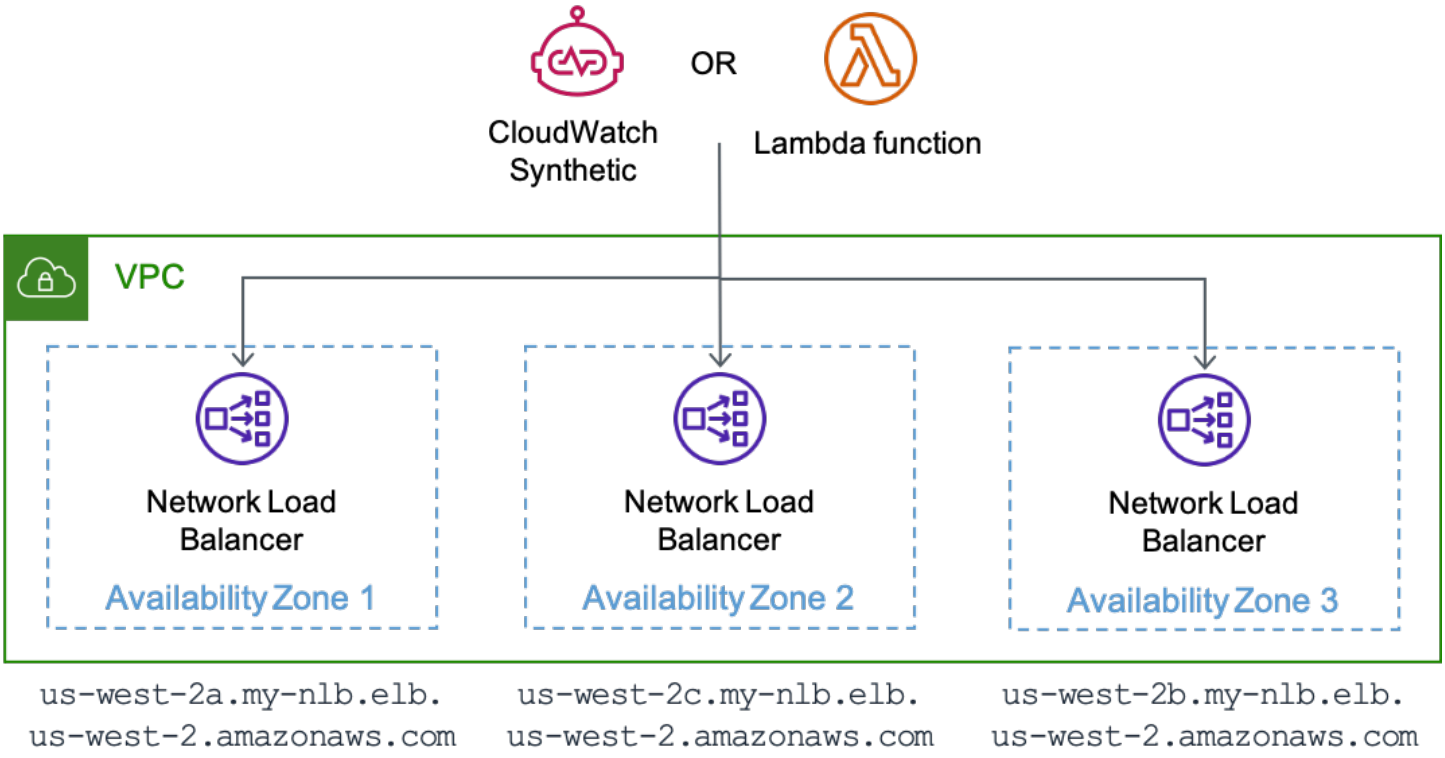
除了生成這些指標之外，您還需要了解它們的上下文。執行此操作的一種方法是使用維度。維度會為量度提供唯一的身分，並協助說明指標告訴您的內容。對於用來識別工作負載失敗的指標 (例如延遲、可用性或錯誤計數)，您需要使用符合錯誤隔離界限的維度。

例如，如果您使用 M odel-view-controller (MVC) Web 架構在一個區域中跨多個可用區域執行 Web 服務，則應使用Region、、Availability Zone IDControllerAction、和作InstanceId為維度集的維度 (如果您使用的是微服務，則可以使用服務名稱和HTTP方法而不是控制器和動作名稱)。這是因為您預期不同類型的失敗會被這些邊界隔離。您不會期望 Web 服務的代碼中存在會影響其列出產品以也影響主頁的能力的錯誤。同樣地，您也不會期望單EC2一執行個體上的完整EBS磁碟區會影響其他EC2執行個體提供您的 Web 內容。「可用區域 ID」維度可讓您一致地識別可用區域相關影響。AWS 帳戶您可以透過多種不同方式在工作負載中找到可用區域 ID。如需一附錄 A — 取得可用區域識別碼些範例，請參閱。

雖然本文檔主要使用 Amazon EC2 作為示例中的運算資源，但InstanceId可以用 Amazon 彈性容器服務（AmazonECS）和亞馬 Amazon Elastic Kubernetes Service（AmazonEKS）運算資源的容器 ID 替換為維度的組件。

如果您的工Region作負載具有區域端點 Controller ActionAZ-ID，您的金絲雀也可以在指標中使用、和作為維度。在這種情況下，請調整您的金絲雀，以便在他們正在測試的可用區域中運行。這樣可確保如果隔離的可用區域事件影響正在執行初期測試的可用區域，則不會記錄使其正在測試的不同可

用區域的指標看起來不健康。例如，您的初期測試可以使用其區域名稱來測試 Network Load Balancer (NLB) 或應用程式負載平衡器 (ALB) 後面的服務的每個區域DNS端點。



在 CloudWatch Synthetics 上運行的金絲雀或測試每個區域端點的 AWS Lambda 函數 NLB

透過使用這些維度產生指標，您可以建立 Amazon CloudWatch 警示，以便在這些邊界內發生可用性或延遲變更時通知您。您也可以使用儀表板快速分析該資料。為了有效地使用指標和日誌，Amazon CloudWatch 提供了嵌入式指標格式 (EMF)，可讓您將自訂指標與日誌資料內嵌。 CloudWatch 自動提取自定義指標，以便您可以對其進行可視化和警報。 AWS 為不同的程式設計語言提供數個用戶端程式庫，讓您輕鬆開始使用EMF。它們可以與 AmazonEC2，AmazonECS，Amazon EKS 和現場部署環境一起使用。AWS Lambda透過將指標嵌入日誌中，您也可以使用 Amazon CloudWatch 貢獻者深入解析來建立顯示參與者資料的時間序列圖表。在這種情況下，我們可以顯示按維度分組的數據 AZ-IDInstanceId，或Controller以及日誌中的任何其他字段類似SuccessLatency或HttpResponseCode。

```
{
  "_aws": {
    "Timestamp": 1634319245221,
    "CloudWatchMetrics": [
      {
        "Namespace": "workloadname/frontend",
        "Metrics": [
```

```
          { "Name": "2xx", "Unit": "Count" },
          { "Name": "3xx", "Unit": "Count" },
          { "Name": "4xx", "Unit": "Count" },
          { "Name": "5xx", "Unit": "Count" },
          { "Name": "SuccessLatency", "Unit": "Milliseconds" }
        ],
        "Dimensions": [
          [ "Controller", "Action", "Region", "AZ-ID", "InstanceId"],
          [ "Controller", "Action", "Region", "AZ-ID"],
          [ "Controller", "Action", "Region"]
        ]
      }
    ],
    "LogGroupName": "/loggroupname"
  },
  "CacheRefresh": false,
  "Host": "use1-az2-name.example.com",
  "SourceIp": "34.230.82.196",
  "TraceId": "|e3628548-42e164ee4d1379bf.",
  "Path": "/home",
  "OneBox": false,
  "Controller": "Home",
  "Action": "Index",
  "Region": "us-east-1",
  "AZ-ID": "use1-az2",
  "InstanceId": "i-01ab0b7241214d494",
  "LogGroupName": "/loggroupname",
  "HttpResponseCode": 200,
  "2xx": 1,
  "3xx": 0,
  "4xx": 0,
  "5xx": 0,
  "SuccessLatency": 20
}
```

此記錄有三組維度。它們會以精細度順序進行，從執行個體到可用區域再到區域 (Controller此範例一律包含在此範例中)。Action它們支援在整個工作負載中建立警示，以指出何時會影響單一執行個體、單一可用區域或整個執行個體中的特定控制器動作 AWS 區域。這些維度用於 2xx、3xx、4xx 和 5xx HTTP 回應量度的計數，以及成功要求量度的延遲 (如果要求失敗，也會記錄失敗要求延遲的量度)。記錄檔也會記錄其他資訊，例如HTTP路徑、要求者的來源 IP，以及此要求是否需要重新整理本機快取。然後，可以使用這些資料點來計算每個API工作負載提供的可用性和延遲時間。

使用可用性測量結果的HTTP回應代碼的注意事項

通常情況下，您可以將 2xx 和 3xx 響應視為成功，5xx 作為故障。4xx 響應代碼落在中間的某個地方。通常，它們是由於客戶端錯誤而產生的。也許一個參數超出範圍，導致 400 響應，或者他們正在請求不存在的東西，導致 404 響應。您不會根據工作負載的可用性來計算這些回應。但是，這也可能是軟件中出現錯誤的結果。

例如，如果您引入了更嚴格的輸入驗證來拒絕之前會成功的要求，則 400 個回應可能會視為可用性下降。或者，也許您正在限制客戶並返回 429 響應。雖然節流客戶可以保護您的服務以維持其可用性，但從客戶的角度來看，該服務無法用於處理其請求。您需要決定 4xx 回應碼是否為可用性計算的一部分。

雖然本節概述了用 CloudWatch 作收集和分析指標的一種方法，但這不是您可以使用的唯一解決方案。您也可以選擇將指標傳送至適用於 Prometheus 和 Amazon 受管的 Grafana (Amazon DynamoDB 表) 的亞馬遜受管服務，或使用第三方監控解決方案。關鍵在於您的工作負載產生的指標必須包含有關工作負載故障隔離界限的內容。

如果工作負載產生維度與錯誤隔離界限一致，您可以建立可檢測可用區域隔離故障的觀察性。下列各節說明三種免費方法，用於偵測單一可用區域的損害所產生的故障。

主題

- 使用 CloudWatch 複合式警報偵測故障
- 使用離群值偵測進行故障偵測
- 單一執行個體區域資源的故障偵測
- Summary

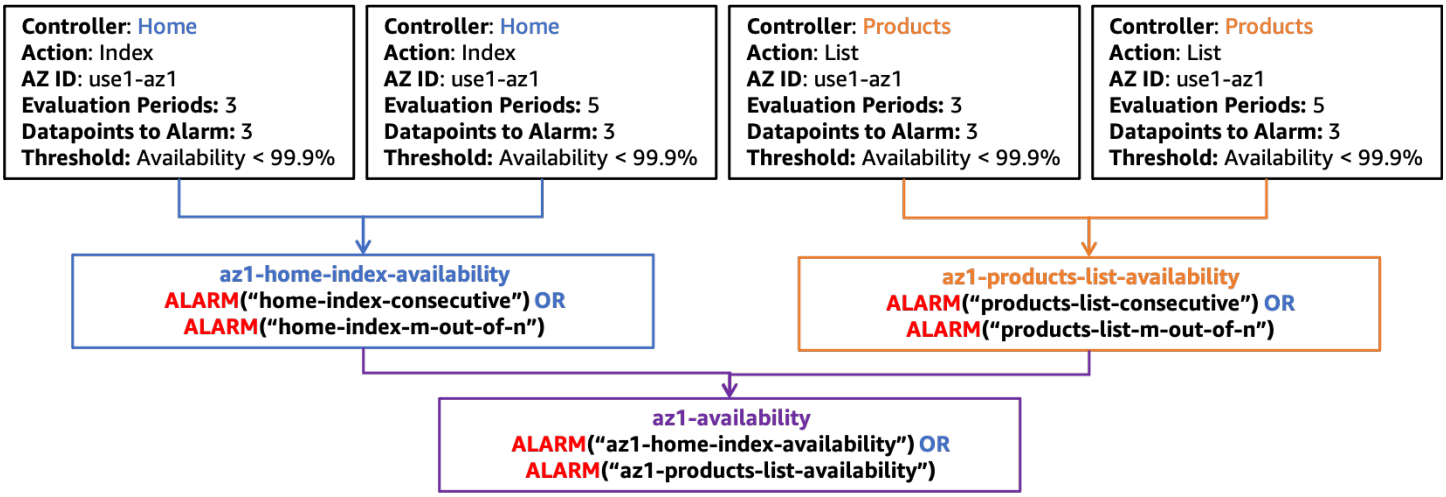
使用 CloudWatch 複合式警報偵測故障

在 CloudWatch 量度中，每個維度集都是唯一的量度，您可以在每個維度集上建立 CloudWatch 警示。然後，您可以建立 Amazon CloudWatch 複合警示來彙總這些指標。

為了準確偵測撞擊，本 paper 中的範例將針對其 CloudWatch 警示的每個尺寸集使用兩種不同的警報結構。每個警報都會使用一分鐘的期間，表示每分鐘評估一次量度。第一種方法是通過將「評估期」和「數據點」設置為「警報」設置為三個來使用連續三個違規數據點，這意味著總共三分鐘的影響。第二種方法將使用「M out N」，當五分鐘視窗中的任何 3 個資料點違反時，方法是將「評估期間」設定為 5，將「資料點設定為警示」設定為 3。這提供了檢測恆定信號以及在短時間內波動的信號的能力。此處包含的時間持續時間和數據點數量是一個建議，請使用對您的工作負載有意義的值。

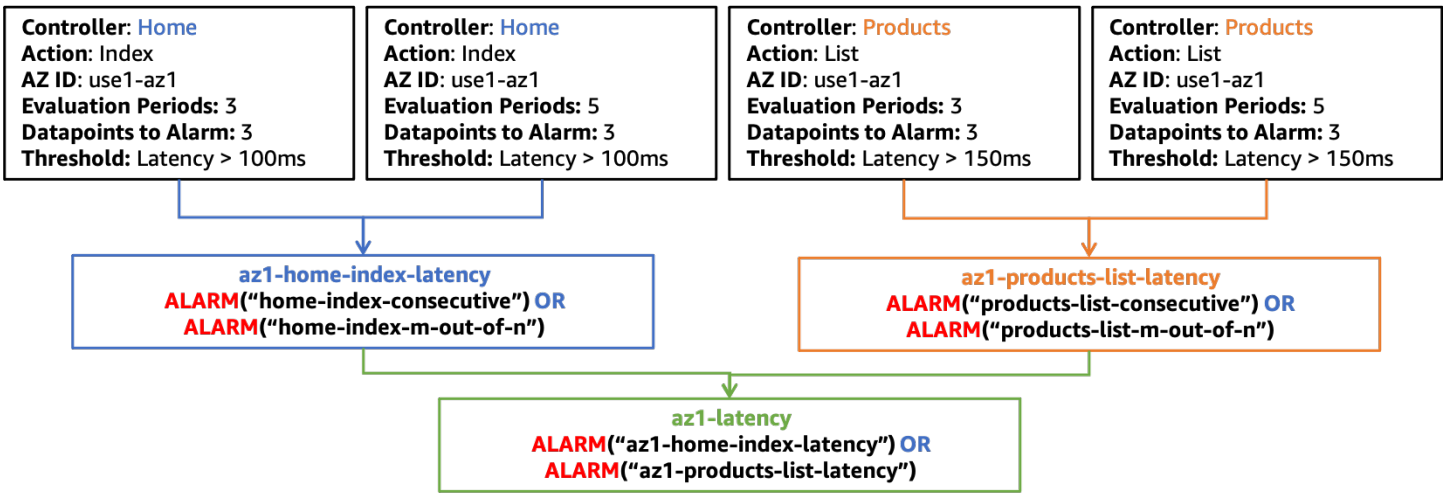
偵測單一可用區域中的影響

使用此建構，考慮使用Controller、ActionInstanceIdAZ-ID、和作Region為維度的工作負載。工作負載有兩個控制器：產品和首頁，以及每個控制器一個動作、清單和索引分別。它在區域中的三個可用區us-east-1域中運作。您可以針對每個可用區域的可用性Controller和Action組合建立兩個警示，以及每個可用區域的兩個延遲警示。然後，您可以選擇為每個警報和組合建立可用性的複ControllerAction合警示。最後，您會建立複合警示，彙總可用區域的所有可用性警示。對於單一可用區域use1-az1，使用每個Controller和Action組合的選用複合警示 (與可用區use1-az3域也會存在類似的警示use1-az2，但不會顯示為簡單起見)，如下圖所示。



複合報警結構，可用於 *use1-az1*

您也可以針對延遲建立類似的警示結構，如下圖所示。

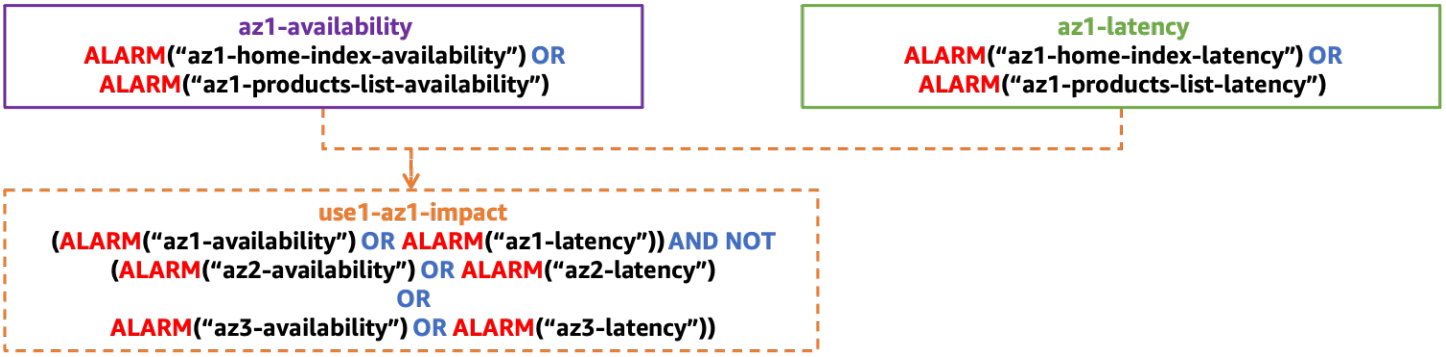


用於延遲的複合警報結構 *use1-az1*

對於本節的其餘數字，只有az1-availability和az1-latency複合警報會顯示在頂層。對於工作負載的任何部分az1-latency，這些複合警示az1-availability和，將告訴您是否可用性低於或延遲超過特定可用區域中定義的閾值。您也可以考慮測量輸送量，以偵測可防止單一可用區域中的工作負載接收工作的影響。您也可以將金絲雀發出的指標產生的警報集成到這些複合警報中。如此一來，如果伺服器端或用戶端看到可用性或延遲的影響，警示就會建立警示。

確保影響不是區域

另一組複合警報可用於確保只有隔離的可用區域事件才會啟動警示。這是透過確保可用區域複合警示處於ALARM狀態，而其他可用區域的複合警示處於OK狀態來執行。這將導致您使用的每個可用區域產生一個複合警報。下圖顯示了一個範例 (請記住，和、、、和中use1-az2有延遲和可用性的警示 use1-az3 az2-latency az2-availability az3-latencyaz3-availability，這些警示並未顯示為簡單起見)。



複合式警報結構，可偵測隔離至單一 AZ 的衝擊

確保影響不是由單一執行個體造成

單一執行個體 (或整體叢集的一小部分) 可能會對可用性和延遲指標造成不成比例的影響，這些指標可能會使整個可用區域看起來受到影響，但事實上並非如此。與撤除可用區域相比，移除單一有問題的執行個體更快速且有效。

執行個體和容器通常會被視為暫時資源，通常會被等服務取代。AWS Auto Scaling每次建立新執行個體時都很難建立新 CloudWatch 警示 (但當然可以使用 Amazon EventBridge 或 Amazon EC2 Auto Scaling 生命週期勾點)。相反，您可以使用CloudWatch 參與者見解來識別可用性和延遲指標的貢獻者數量。

例如，對於 HTTP Web 應用程式，您可以建立規則，以識別每個可用區域中 5xx HTTP 回應的主要貢獻者。這將識別哪些執行個體導致可用性下降 (上面定義的可用性指標是由存在 5xx 錯誤驅動)。使用記EMF錄範例，使用的索引鍵建立規則InstanceId。然後，按HttpResponseCode字段過濾日誌。此範例是use1-az1可用區域的規則。

```
{
    "AggregateOn": "Count",
    "Contribution": {
        "Filters": [
            {
                "Match": "$.InstanceId",
                "IsPresent": true
            },
            {
                "Match": "$.HttpStatusCode",
                "IsPresent": true
            },
            {
                "Match": "$.HttpStatusCode",
                "GreaterThan": 499
            },
            {
                "Match": "$.HttpStatusCode",
                "LessThan": 600
            },
            {
                "Match": "$.AZ-ID",
                "In": ["use1-az1"]
            },
        ],
        "Keys": [
            "$.InstanceId"
        ]
    },
    "LogFormat": "JSON",
    "LogGroupNames": [
        "/loggroupname"
    ],
    "Schema": {
        "Name": "CloudWatchLogRule",
        "Version": 1
    }
}
```

CloudWatch 也可以根據這些規則來創建警報。您可以使用度量數學和具有量度的INSIGHT_RULE_METRIC函數，根據參與者見解規則建立警示。UniqueContributors除了可用

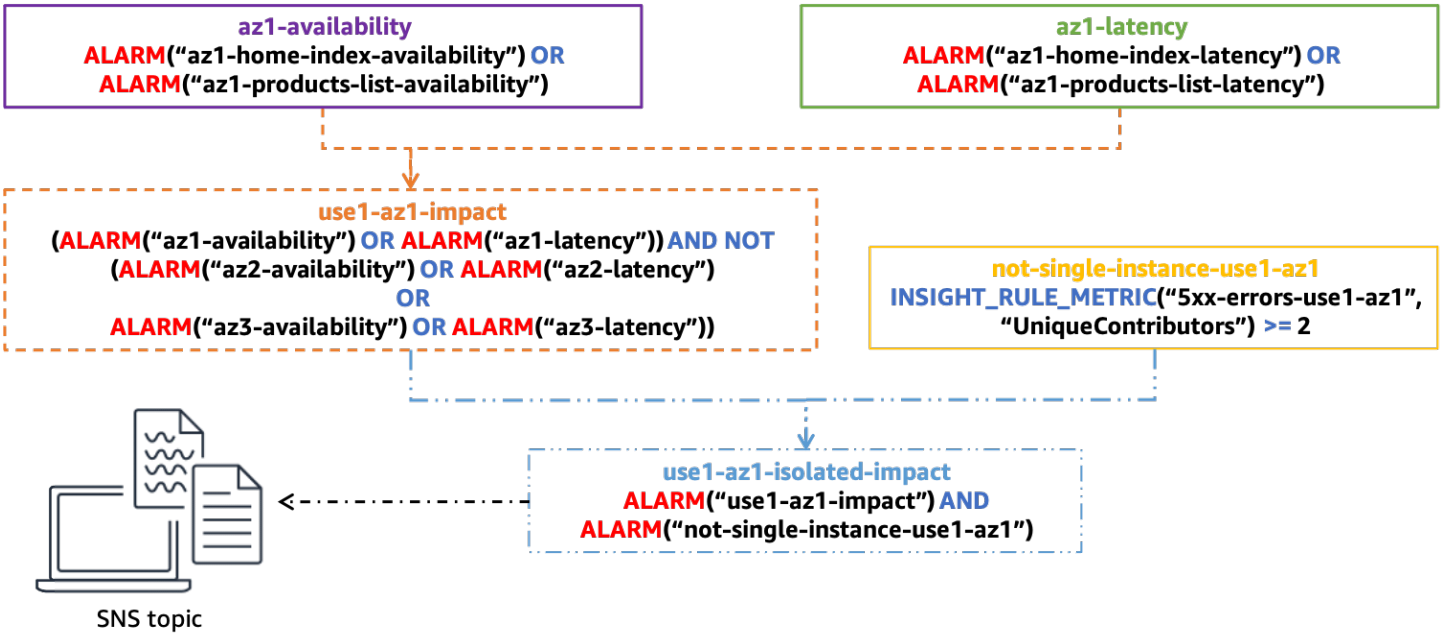
性，您還可以使用延遲或錯誤計數等指標的 CloudWatch 警示來建立其他「參與者見解」規則。這些警示可與隔離的可用區域影響複合警示搭配使用，以確保單一執行個體不會啟動警示。見解規則的量度use1-az1可能如下所示：

```
INSIGHT_RULE_METRIC("5xx-errors-use1-az1", "UniqueContributors")
```

當此量度大於臨界值時，您可以定義警示；例如，本範例為 2。當 5xx 回應的獨特貢獻者超過該閾值時，就會啟動此功能，表示影響來自兩個以上的執行個體。此警報使用大於比較而不是小於的原因是為了確保唯一貢獻者的零值不會引發警報。這告訴您，影響不是來自單一執行個體。針對您的個別工作負載調整此閾值。一般指南是將此數字設定為可用區域中總資源的 5% 或更多。在足夠的樣本數量下，超過 5% 的資源受到影響顯著表現出統計顯著性。

整合練習

下圖顯示單一可用區域的完整複合警示結構：



完整的複合報警結構，用於確定單一可用區

當指示隔離可用區域對延遲或可用性造成的影響的複合警示處於ALARM狀態，use1-az1-aggregate-alarm以及根據該可用區域的 Contributor Insights 規則的警示也處於ALARM狀態 (表示影響超過單一執行個體) 時，會啟動最終複合警示。use1-az1-isolated-impact not-single-instance-use1-az1您可以為工作負載使用的每個可用區域建立此警示堆疊。

您可以在此最終警報上附加 Amazon 簡單通知服務（AmazonSNS）警報。所有先前的警報都是在沒有動作的情況下設定的。該警報可以通過電子郵件通知操作員開始手動調查。它也可以啟動自動化以撤離

可用區域。但是，請注意樓宇自動化以響應這些警報。在可用區域撤離發生之後，結果應該是緩解提高的錯誤率，並且警示會回到OK狀態。如果其他可用區域發生影響，則自動化可能會撤除第二個或第三個可用區域，進而可能會移除工作負載的所有可用容量。在採取任何行動之前，自動化應檢查是否已執行疏散。在疏散成功之前，您可能還需要擴展其他可用區域中的資源。

當您將新的控制器或動作添加到 MVC Web 應用程序，或新的微服務或一般情況下，要單獨監視的任何其他功能時，只需要在此設置中修改一些警報即可。您將為該新功能建立新的可用性和延遲警示，然後將這些警示新增至適當的可用性區域對齊可用性和延遲複合警示，以`az1-latency`及我們`az1-availability`在此處使用的範例中。其餘的複合警報在配置完成後仍保持靜態。這使得使用此方法的新功能成為更簡單的過程。

使用離群值偵測進行故障偵測

當您在多個可用區域中看到由於不相關原因而發生的錯誤率提高時，可能會出現與先前方法的一個差距。假設您有跨三個可用區域部署EC2執行個體且可用性警示閾值為 99% 的案例。然後，會發生單一可用區域損害，隔離許多執行個體，並導致該區域中的可用性降至 55%。同時，但在不同的可用區域中，單一EC2執行個體會耗盡其EBS磁碟區上的所有儲存空間，而且無法再寫入記錄檔。這會導致它開始傳回錯誤，但它仍會通過負載平衡器健康狀態檢查，因為這些檢查不會觸發要寫入的記錄檔。這會導致該可用區域中的可用性降至 98%。在此情況下，您的單一可用區域影響警示不會啟動，因為您看到多個可用區域中的可用性影響。但是，您仍然可以通過撤離受損的可用區域來減輕幾乎所有的影響。

在某些類型的工作負載中，您可能會在所有可用區域中一致地遇到錯誤，而先前的可用性指標可能沒有用處。舉個 AWS Lambda 例子。 AWS 可讓客戶建立自己的程式碼，以便在 Lambda 函數中執行。要使用該服務，您必須將代碼上傳到ZIP文件中，包括依賴關係，並定義該函數的入口點。但是有時候客戶錯了這個部分，例如，他們可能會忘記ZIP檔案中的關鍵相依性，或者在 Lambda 函數定義中輸入錯誤的方法名稱。這會導致函數無法被調用並導致錯誤。 AWS Lambda 一直看到這些錯誤，但它們並不表示任何事情都不一定是不健康的。但是，類似可用區域的損害也可能導致出現這些錯誤。

若要尋找這類雜訊中的訊號，您可以使用離群值偵測來判斷可用區域之間的錯誤數目是否存在統計上顯著的偏斜。雖然我們看到跨多個可用區域的錯誤，但是如果其中一個可用區域確實發生故障，我們預期會在該可用區域中看到與其他可用區域相比較高的錯誤率，或者可能低得多。但是多少更高或更低？

執行此分析的其中一種方法是使用卡方 (^{2}) 測試來偵測可用區域之間錯誤率的統計顯著差異 (執行離群值偵測有許多不同的演算法)。讓我們來看看卡方測試是如何工作的。

卡方測試評估可能發生某些結果分佈的可能性。在這種情況下，我們感興趣的是在一些已定義的集合中的錯誤分佈AZs。在此範例中，若要簡化數學作業，請考慮四個可用區域。

首先，建立 null 假設，它定義了你認為默認結果是什麼。在此測試中，Null 假設是您預期錯誤會平均分配到每個可用區域。然後，產生替代假設，即錯誤未平均分佈，表示可用區域減值。現在，您可以使用指標中的數據來測試這些假設。為了達到這個目的，您將從五分鐘的視窗抽樣您的指標。假設您在該視窗中取得 1000 個已發佈的資料點，其中您會看到 100 個總錯誤。您預期在平均分佈的情況下，錯誤會在四個可用區域中每個區域中發生 25% 的時間。假設下表顯示了與您實際看到的相比，您所期望的內容。

表 1：看到的預期與實際錯誤

AZ	預期	實際
use1-az1	25	20
use1-az2	25	20
use1-az3	25	25
use1-az4	25	35

因此，您會看到現實中的分佈不均勻。但是，您可能認為這是由於您採樣的數據點中某種程度的隨機性而發生的。有一定程度的概率，這種類型的分佈可能發生在樣本集中，並且仍然假設 null 假設是真的。這導致了以下問題：至少如此極端獲得結果的可能性是多少？ 如果該概率低於定義的閾值，則拒絕零假設。在統計上具有顯著意義，此概率應為 5% 或更低。 [1]

[1] 克拉帕羅, 羅伯特 ·M. (2007). 「意義水平」。在薩爾金德，尼爾 ·J· 測量和統計百科全書 3. 加州千橡市：SAGE出版刊物，第 889—891 頁。ISBN1-412-91611-9.

你如何計算這個結果的概率？ 您可以使用 [2] 統計信息，該統計信息提供了非常好的研究分佈，並且可以用於確定使用此公式獲得極端或更極端結果的可能性。

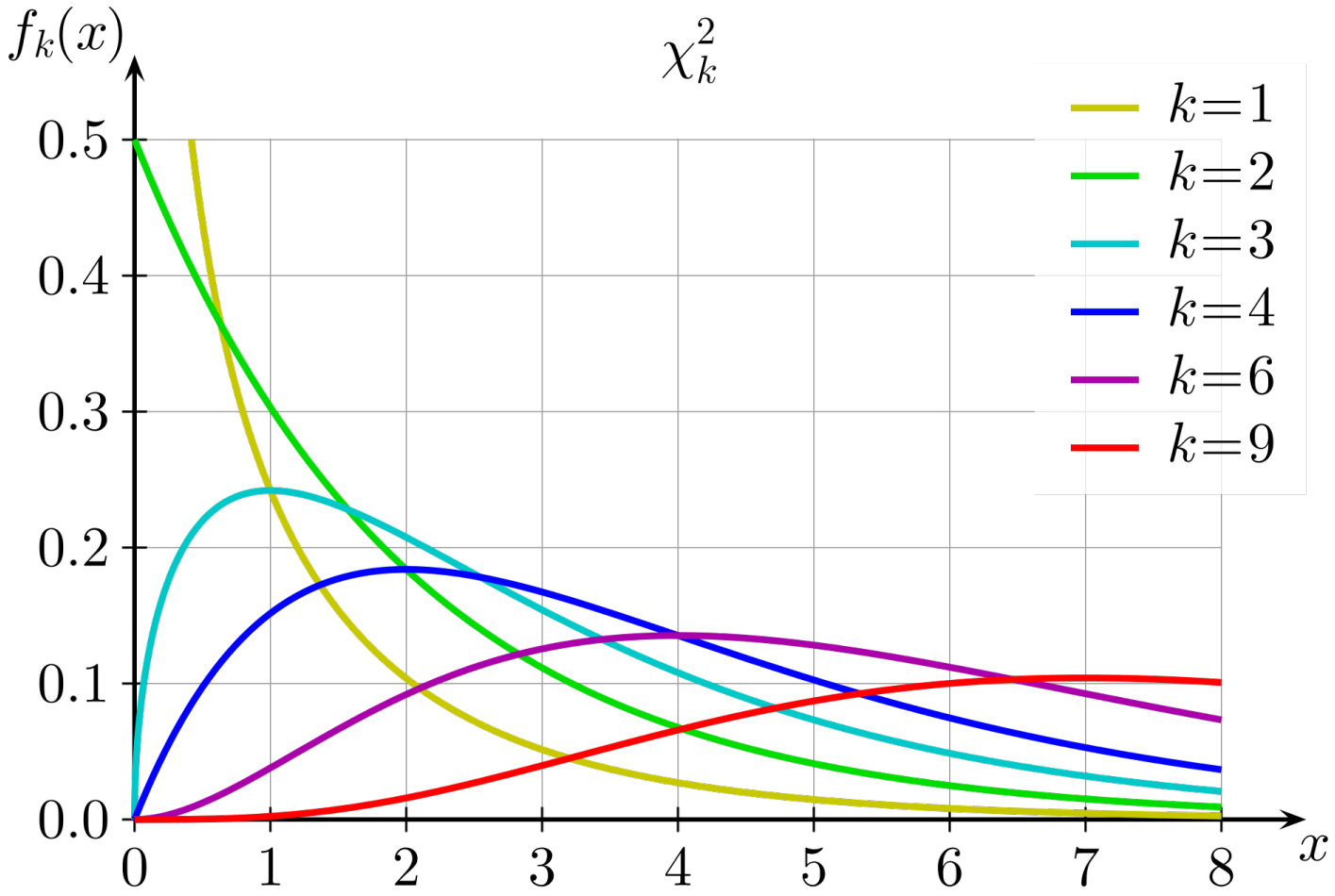
$$E_i = expected\ observations\ of\ type\ i$$
$$O_i = actual\ observations\ of\ type\ i \qquad (1)$$

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

對於我們的例子，這會導致：

$$\chi^2 = \frac{(20-25)^2}{25} + \frac{(20-25)^2}{25} + \frac{(25-25)^2}{25} + \frac{(35-25)^2}{25}$$
$$\chi^2 = \frac{-5^2}{25} + \frac{-5^2}{25} + \frac{0^2}{25} + \frac{10^2}{25} \qquad (2)$$
$$\chi^2 = 1 + 1 + 0 + 4$$
$$\chi^2 = 6$$

那麼，在我們的概率方面6意味著什麼？ 您需要查看具有適當自由度的卡方分佈。下圖顯示了針對不同自由度的數個卡方分佈。



不同自由度的卡方分佈

自由度的計算方式比測試中的選擇數少一。在這種情況下，由於有四個可用區域，所以自由度為三個。然後，您想知道 k = 3 繪圖上 x ≥ 6 的曲線下面積（積分）。您也可以使用具有常用值的預先計算表來近似該值。

表 2：卡方臨界值

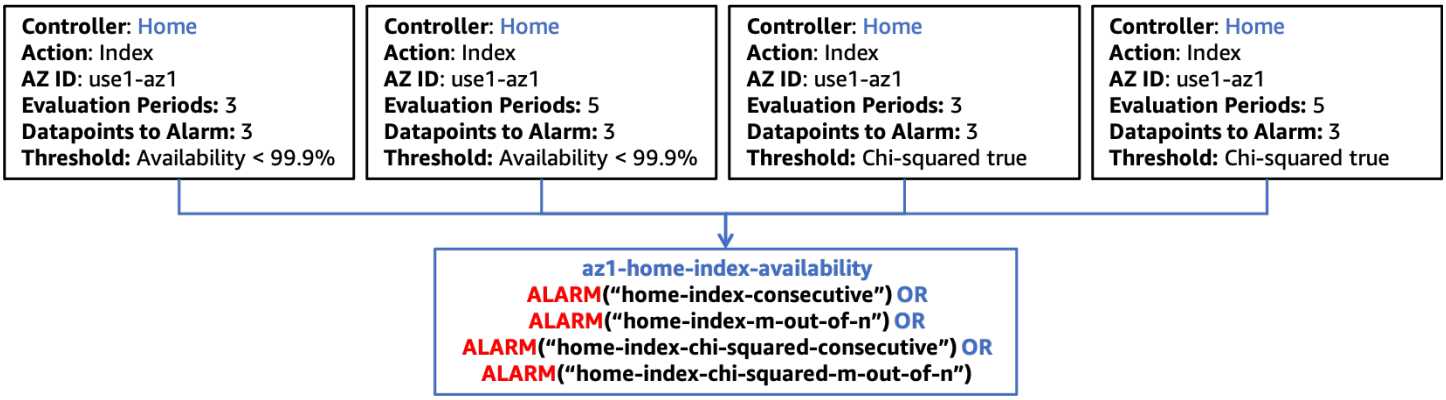
自由度	概率小於臨界值				
	0.75	0.90	0.95	0.99	0.999
1	1.323	2.706	3.841	6.635	10.828
2	2.773	4.605	5.991	9.210	13.816
3	4.108	6.251	7.815	11.345	16.266
4	5.385	7.779	9.488	13.277	18.467

對於三個自由度，6 的卡方值落在 0.75 和 0.9 機率欄之間。這意味著這種分佈發生的機會大於 10%，這不低於 5% 的閾值。因此，您接受 Null 假設，並判斷可用區域之間的錯誤率在統計上沒有顯著差異。

指標數學原生不支援執行卡方統計測試，因此您需要從運算環境 (例如 Lambda) 中 CloudWatch 收集適用的錯誤指標， CloudWatch 然後在運算環境中執行測試。您可以決定在MVC控制器/動作或個別微服務層級，或在可用區域層級執行此測試。您需要考慮可用區域的損害是否會平等地影響每個控制器/動作或微服務，或者類似DNS失敗的事情可能會對低輸送量服務造成影響，而不是在高輸送量服務中造成影響，這可能會在彙總時遮罩影響。無論是哪一種情況，都可以選取適當的維度來建立查詢。細微程度等級也會影響您建立的產生的 CloudWatch 警示。

收集指定時間範圍內每個 AZ 和控制器/動作的錯誤計數量度量。首先，將卡方測試的結果計算為 true（存在統計上顯著的偏斜）或 false（沒有，即空假設成立）。如果結果為 false，請將 0 資料點發佈至您的指標串流，以取得每個可用區域的卡方結果。如果結果為 true，請針對可用區域發佈 1 個資料點，其中的錯誤距離預期值最遠，發佈其他資料點為 0 (如需可在 Lambda 函數中使用的範附錄 B-示例卡方計算例程式碼，請參閱)。您可以遵循與先前可用性警示相同的方法，方法是根據 Lambda 函數產生的資料點，建立連 CloudWatch 續 3 個 CloudWatch 量度警示和 5 個指標警示中的 3 個。與前面的例子一樣，這種方法可以修改為在較短或更長的窗口中使用更多或更少的數據點。

然後，將這些警示新增至控制器與動作組合的現有可用區域可用性警示，如下圖所示。



整合卡方統計測試與複合警報

如前所述，當您在工作負載中啟動新功能時，您只需要建立特定於該新功能的適當 CloudWatch 指標警示，並更新複合警示階層中的下一層以包含這些警示。警報結構的其餘部分保持靜態。

單一執行個體區域資源的故障偵測

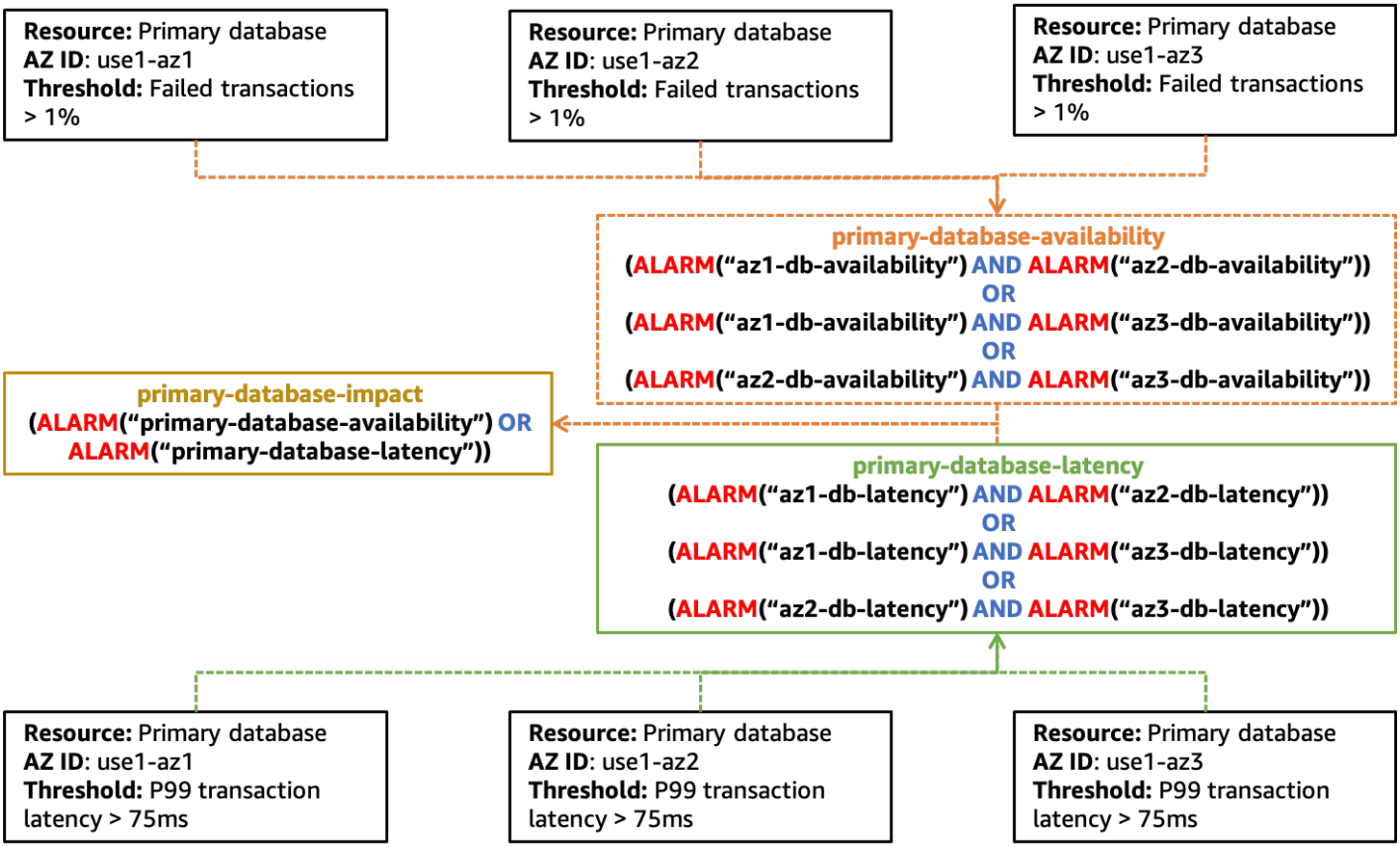
在某些情況下，您可能擁有區域資源的單一作用中執行個體，通常需要單一寫入器元件的系統，例如關聯式資料庫 (例如 AmazonRDS) 或分散式快取 (例如 Amazon ElastiCache (Redis OSS))。如果單一可用區域減損影響主要資源所在的可用區域，則可能會對存取資源的每個可用區域造成影響。這可能會導致每個可用區域超過可用性閾值，這表示第一種方法無法正確識別單一可用區域的影響來源。此外，您可能會在每個可用區域看到類似的錯誤率，這表示異常值分析也不會偵測到問題。這意味著您需要實現其他可觀察性以專門檢測此情況。

您所關心的資源很可能會產生有關其健康狀況的指標，但在可用區域損害期間，資源可能無法提供這些指標。在這種情況下，您應該創建或更新警報以了解您何時失明。如果您已經監控並開啟警示的重要指標，您可以設定警示以將遺失的資料視為違規。這將幫助您了解資源是否停止報告數據，並且可以包含在同一行中，以及先前使用的 n 個警報中的 m 個。

也有可能在某些指示資源健康狀態的指標中，當沒有活動時，它會發佈零值資料點。如果減值阻止了與資源的交互，則不能對這些類型的指標使用缺少的數據方法。您也可能不想在值為零時發出警報，因為可能存在正常閾值之內的合法情況。檢測這種類型的問題的最佳方法是使用此依賴項的資源生成的指標。在這種情況下，我們想要使用複合警報來偵測多個可用區域中的影響。這些警示應使用與資源相關的少數重要指標類別。下面列出了幾個例子：

- 輸送量 — 傳入工作單位的速率。這可能是事務，讀取，寫入等。
- 可用性 — 測量成功與失敗工作單位的數量。
- 延遲 — 測量多個延遲百分位數，以便在關鍵操作中成功執行的工作。

再一次，您可以為每個要測量的度量類別中的每個量度建立連續的 n 個量度警示和 m 個量度警示。和以前一樣，這些可以合併為複合警報，以確定此共用資源是跨可用區域的影響來源。您希望能夠使用複合警示識別對多個可用區域的影響，但影響並不一定需要是所有可用區域。這種方法的高級複合報警結構如下圖所示。



建立警示以偵測單一資源對多個可用區域造成影響的範例

您會注意到，此圖表對於應使用哪種類型的度量警示以及複合警示的階層架構較不具規定性。這是因為發現這種問題可能很困難，並且需要仔細注意共享資源的正確信號。這些信號也可能需要以特定的方式進行評估。

此外，您應該注意到`primary-database-impact`警示與特定可用區域沒有關聯。這是因為主要資料庫執行處理可以位於設定要使用的任何可用區域中，而且沒有 CloudWatch測量結果可指定它所在的位置。當您看到此警示啟動時，您應該使用它作為資源可能有問題的信號，並啟動容錯移轉至另一個可用區域 (如果尚未自動完成)。將資源移至另一個可用區域之後，您可以等待並查看隔離的可用區域警示是否已啟動，或者您可以選擇先發製呼叫可用區域撤離計劃。

Summary

本節說明三種協助識別單一可用區域損害的方法。每種方法都應該一起使用，以提供工作負載健康狀況的全面檢視。

CloudWatch 複合警示方法可讓您找出可用性偏差在統計學上並不顯著的問題，例如 98% (受損的可用區域)、100% 和 99.99% 的可用性，這不是由單一共用資源引起的。

離群值偵測將有助於偵測單一可用區域損傷，如果您在多個可用區域中發生不相關的錯誤，這些錯誤都超過了警示閾值。

最後，識別單一執行個體區域資源的降級有助於探索可用區域損害何時會影響跨可用區域共用的資源。

這些模式中的每一種產生的警示都可以組合到 CloudWatch 複合警示階層中，以探索單一可用區域損壞的發生時間，以及對工作負載的可用性或延遲造成影響。

可用區域疏散模式

偵測到單一可用區域中的影響之後，下一個步驟是撤除該可用區域。疏散需要實現兩個結果。

首先，您想要停止將工作傳送至受影響的可用區域。這可能意味著不同體系結構中的不同事物。在要求/回應工作負載中，這表示停止傳送至負載平衡器或可用區域中其他資源之類的 HTTP 或 GRPC 要求之類的項目。在批次處理或佇列處理系統中，這可能表示停止計算資源在受影響的可用區域中處理工作。您還需要防止未受影響可用區域中的資源與受影響可用區域中的資源互動，例如 EC2 執行個體將流量傳送到VPC 端點介面在受影響的可用區域中，或連線至資料庫的主要執行個體。

第二個結果是防止在受影響的可用區域中佈建新容量。這很重要，因為在受影響的可用區域中佈建 EC2 執行個體或容器等新資源可能會產生與現有資源相同的影響。此外，由於第一個結果會阻止將工作傳送給他們，因此無法吸收佈建要處理的負載。這導致對現有資源的負載增加，這可能最終導致變成褐色或工作負載的完全無法使用。有幾個自動擴展服務可用AWS在適用的情況下：亞馬遜 EC2 自動擴展,應用程式自動縮，以及AWS Auto Scaling。此外，亞馬遜 ECS，亞馬遜 EKS 和服務AWS Batch可以將 VPC 中跨可用區域的主機排程工作，作為其正常操作的一部分。

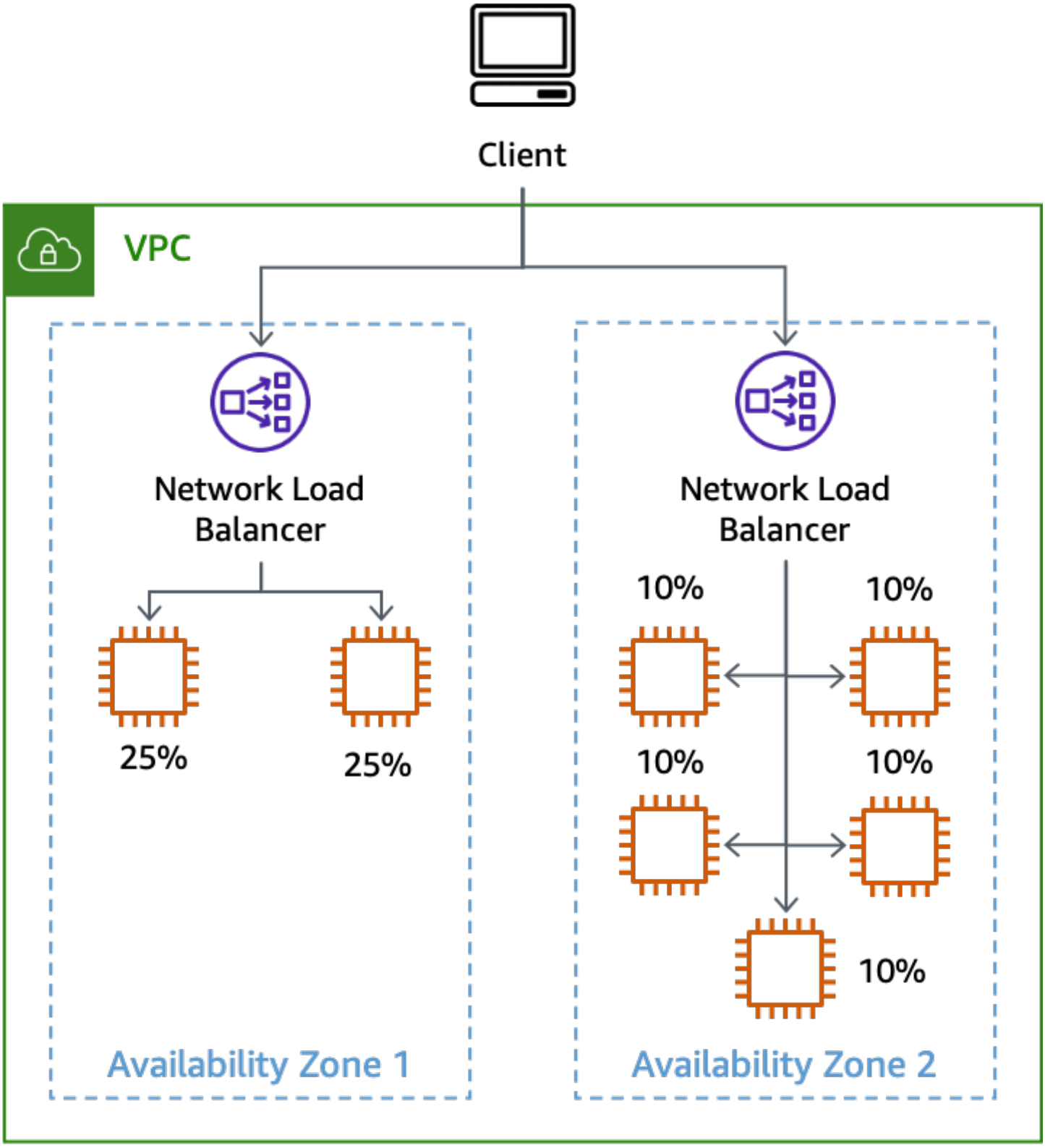
主題

- 可用區域獨立
- 控制平面和資料平面
- 數據平面控制疏散
- 控制平面控制疏散
- 總結

可用區域獨立

若要達成第一個結果，若要停止將工作傳送至受影響的可用區域，疏散需要您實作可用區域獨立性AZI），有時也被稱為可用區域相似性。此架構模式會隔離可用區域內的資源，並防止不同可用區域中資源之間的互動，除非絕對需要，例如連線至不同可用區域中的主要資料庫執行個體。

在要求/回應類型工作負載中，實作 AZI 需要您停用應用程式負載平衡器的跨區域負載平衡(ALB),傳統負載平衡器(CLB)，以及網路負載平衡器(NLB) (NLB 預設會停用跨區域負載平衡)。停用跨區域負載平衡有一些權衡。當您停用跨區域負載平衡時，流量在每個可用區域之間平均分配無論每個實例中有多少個實例。如果您的資源或 Auto Scaling 群組不平衡，這可能會對資源較少的可用區域中資源增加額外負載。如下圖所示，其中可用區域 1 中的兩個執行個體各接收 25% 的負載，而可用區域 2 中的五個執行個體各接收 10% 的負載。



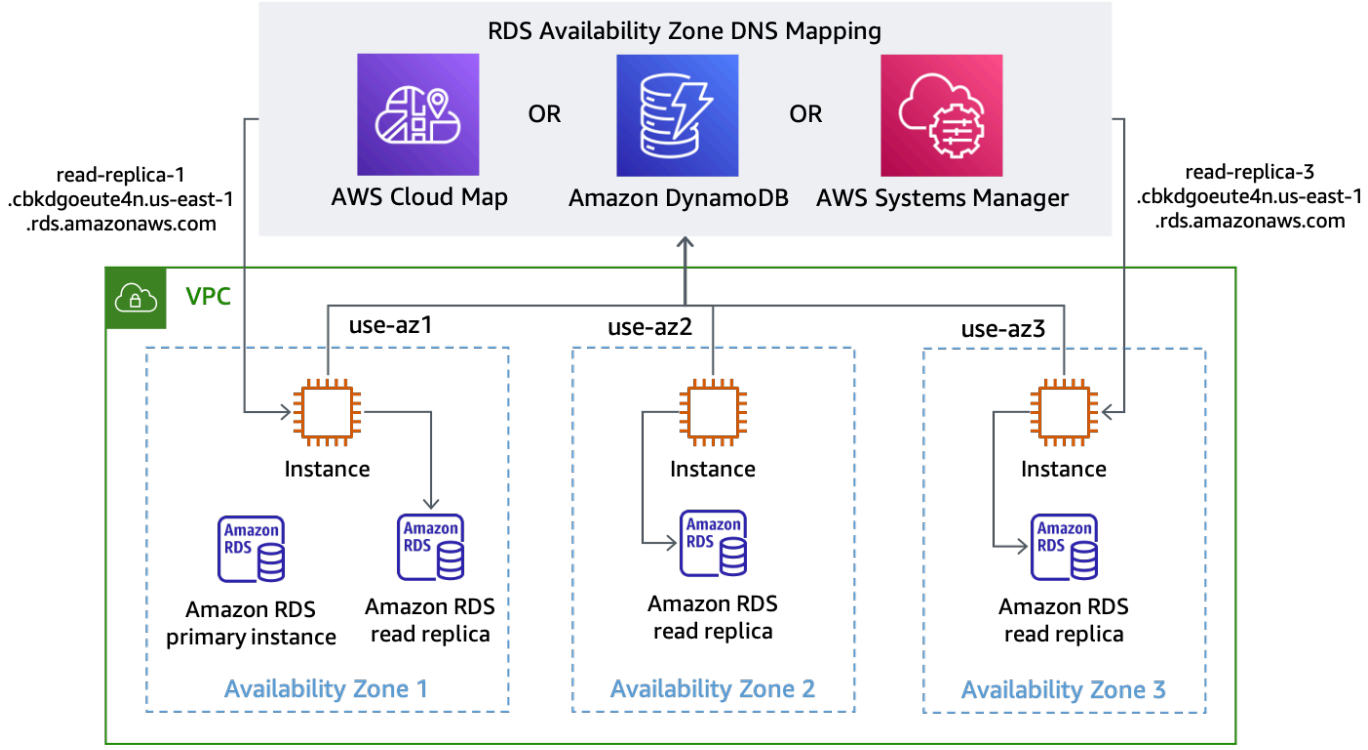
使用非平衡執行個體停用跨區域負載平衡的影響

您使用的其他區域服務也必須使用 AZI 模式來實作，以支援有效的可用區域撤離。例如，介面 VPC 端點提供每個可用區域的特定 DNS 名稱介面端點在中提供。

實作 AZI 的一項挑戰是資料庫，特別是因為大多數關聯式資料庫隨時只支援單一主要寫入器。與主執行個體通訊時，您可能需要跨越可用區域界限。許多AWS資料庫服務支援使用者定義的異地同步備份組態並具有內建的異地同步備份容錯移轉功能亞馬遜 RDS或者亞馬遜極光。在許多失敗案例中，服務可以偵測影響，並在發生問題時自動將資料庫容錯移轉至不同的可用區域。但是，在灰色故障期間，服務可能無法偵測到影響工作負載的影響，或是影響可能與資料庫完全無關。在這些情況下，一旦您偵測到可用區域中的影響，就可以手動叫用容錯移轉來移動主要資料庫。這可讓您有效地對單一可用區域的損害做出反應。

如果您要搭配這些資料庫使用僅供讀取複本，您可能也會想要為這些資料庫實作 AZI，因為您無法像主要資料庫一樣，將僅供讀取複本容錯移轉至不同的可用區域。如果您在可用區域 1 中有單一僅供讀取複本，且三個可用區域的執行個體設定為使用它，則影響可用區域 1 的損害也會影響其他兩個可用區域的作業。這就是您想要防止的影響。

對於 RDS 執行個體，您會收到 DNS 端點，以存取特定可用區域中的複本。若要達到 AZI，您需要每個可用區域的僅供讀取複本，以及讓應用程式知道其所在可用區域使用哪個複本端點的方法。您可以採取的一種方法是使用可用區域 ID 作為數據庫標識符的一部分，類似於`use1-az1-read-replica.cbkdgoeute4n.us-east-1.rds.amazonaws.com`。您也可以使用服務發現來執行此操作（例如AWS Cloud Map）或查找存儲在中的簡單地圖AWS系統管理員參數儲存或一個動態資料表。這個概念如下圖所示。

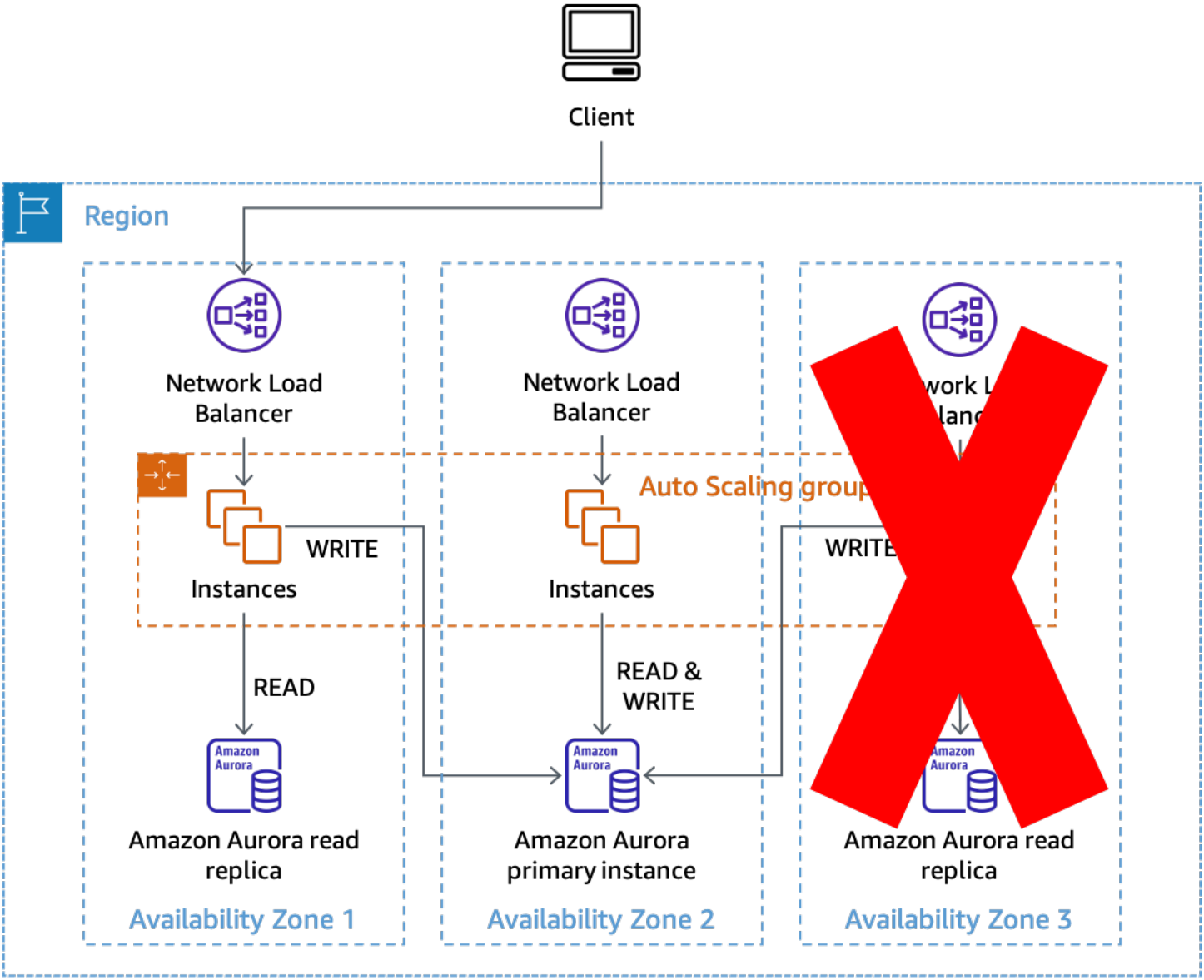


探查每個可用區域的 RDS 端點 DNS 名稱

亞馬遜極光的默認配置是提供單一讀取器端點負載平衡可用僅供讀取複本的請求。為了使用極光實現 AZI，您可以使用自訂端點針對每個僅供讀取複本，使用ANYtype (如此一來，您就可以視需要升級僅供讀取複本)。根據部署複本的可用區域 ID 為自訂端點命名。然後，您可以使用自訂端點提供的 DNS 名稱，連線到特定可用區域中的特定僅供讀取複本，如下圖所示。

針對 Aurora 僅供讀取複本使用自訂端點

當您的系統以這種方式架構時，它會使可用區域疏散成為更簡單的任務。例如，在下圖中，當存在影響可用區域 3 的損害時，可用區域 1 和 2 中的讀取和寫入作業都不會受到影響。



使用 AZI 防止對亞馬遜極光僅供讀取複本造成影響

或者，如果可用區域 2 受到影響，讀取作業仍會在可用區域 1 和 3 中成功。然後，如果 Amazon Aurora 並未自動容錯移轉主資料庫，您可以手動叫用容錯移轉至其他可用區域，以還原處理寫入的能力。當您需要撤除可用區域時，此方法可避免在資料庫連線中進行任何組態變更。最大限度地減少所需的更改並保持過程盡可能簡單將使其更加可靠。

控制平面和資料平面

在我們得到你可以用來執行可用區撤離的實際模式之前，我們需要討論控制平面和數據平面的概念。AWS在我們的服務中區分控制平面和數據平面。控制平面是對系統進行變更 (新增資源、刪除資

源、修改資源) 所涉及的機器，以及讓這些變更傳播到任何需要生效的地方，例如更新 ALB 的網路組態或建立AWS Lambda功能。

資料平面是這些資源的主要功能，例如執行中 EC2 執行個體，或從 Amazon DynamoDB 表格取得項目或將項目放入 Amazon DynamoDB 表格。如需控制平面和資料平面的更詳細討論，請參閱使用可用區域的靜態穩定和AWS故障隔離邊界。

針對本文件的目的，請考慮控制平面比資料平面具有更多的移動零件和相依性。這使得與數據平面相比，在統計學上更有可能控制平面變得受損。這與提供 AZI 的服務尤其相關，例如 Amazon EC2 和 EBS，因為這些服務的某些部分具有也是區域獨立的控制平面，並且可能在單一可用區事件期間受到影響。

雖然控制平面動作可用於執行 AZ 疏散，但根據先前的資訊，它們的成功機率可能較低，尤其是在發生故障事件時。若要增加成功減輕影響的可能性，您可以使用兩種不同的模式。第一個模式僅依賴於資料平面動作，藉由防止工作路由到受影響的可用區域，或停止工作在受影響的可用區域中完成，從而減輕影響。然後，可以嘗試使用控制平面動作來更新資源的組態，以防止在受影響的可用區域中佈建容量，以及停止與該可用區域之間的可用區域通訊。

本節討論的復原模式如下：大, 紅色, 按鈕。它們是您用來快速採取大規模行動的機制，類似於拉動安東電源線在組裝線上。他們假設工作負載已經嘗試過諸如此類的策略重試具有抖動的指數輪詢在他們的代碼中克服暫時性錯誤。這表示偵測到隔離的可用區域影響時，其對可用性或延遲的影響會嚴重到足以需要撤除可用區域才能有效減輕。

數據平面控制疏散

您可以實作多種解決方案，以使用僅限資料計劃的動作執行可用區域疏散。本節將介紹其中的三個以及您可能想要選擇另一個的用例。

使用上述任何解決方案時，您必須確保剩餘可用區域中有足夠的容量，以處理您要轉離的可用區域負載。最有彈性的方法是在每個可用區域中預先佈建所需的容量。如果您使用三個可用區域，您將擁有 50% 的所需容量來處理每個區域中部署的尖峰負載，如此一來，單一可用區域的遺失仍會保留所需容量的 100%，而不必依賴控制平面來佈建更多。

此外，如果您使用 EC2 Auto Scaling，請確保您的 Auto Scaling 群組 (ASG) 在輪班期間不會擴展，以便在班次結束時，群組中仍有足夠的容量來處理客戶流量。您可以確保 ASG 的最低所需容量能夠處理您目前的客戶負載，以達到這個目的。您也可以在指標中使用平均值，而不是 P90 或 P99 等離群值的百分位數量，協助確保 ASG 不會意外擴展。

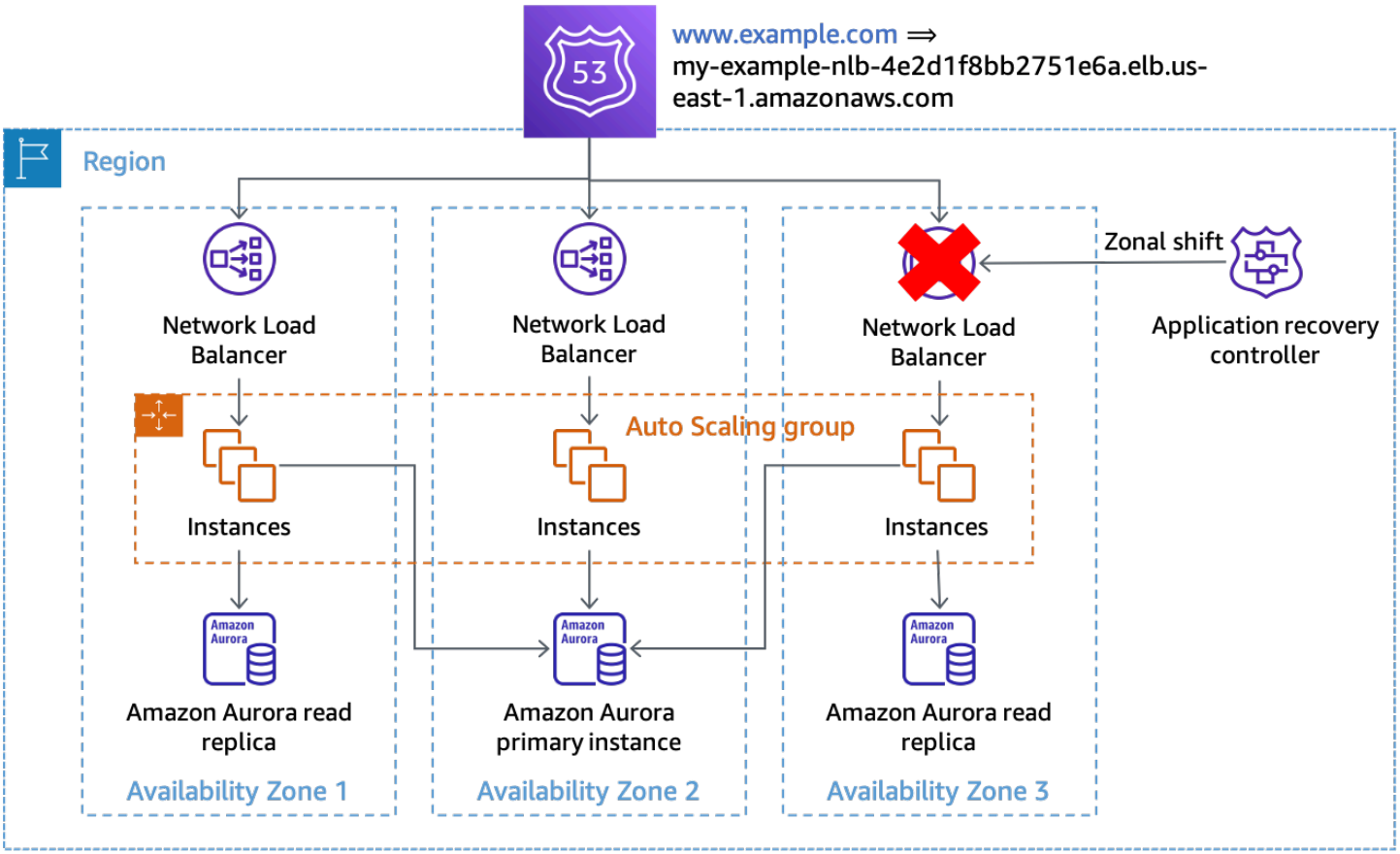
在輪班期間，不再為流量提供服務的資源應該具有非常低的使用率，但其他資源會增加其對新流量的使用率，從而保持平均水平相當一致，這將阻止擴展行動。最後，您也可以使用目標群組健全狀況設

定ALB和NLB，以指定具有健全狀況主機的百分比或計數的 DNS 容錯移轉。這樣可防止流量路由到沒有足夠健全主機的可用區域。

路線 53 應用程式復原控制器 (ARC) 中的區域移位

可用區域疏散使用的第一個解決方案53 號公路弧的區域轉移。此解決方案可用於使用 NLB 或 ALB 作為客戶流量輸入點的請求/回應工作負載。

當您偵測到可用區域已受損時，您可以使用 Route 53 ARC 啟動區域轉移。一旦此作業完成且現有的快取 DNS 回應過期，所有新要求都只會路由至剩餘可用區域中的資源。下圖顯示了區域偏移的工作原理。在下圖中，我們有一個 Route 53 別名記錄`www.example.com`指向`my-example-nlb-4e2d1f8bb2751e6a.elb.us-east-1.amazonaws.com`。會針對可用區域 3 執行區域轉移。



區域移位

在此範例中，如果主要資料庫執行處理不在可用區域 3 中，則執行區域轉移是達到第一個疏散結果所需的唯一動作，以防止在受影響的可用區域中處理工作。如果主節點位於可用區域 3，則如果 Amazon RDS 尚未自動容錯移轉，則您可以與區域轉移協調執行手動啟動的容錯移轉 (確實依賴於 Amazon RDS 控制平面)。本節中所有資料平面控制的解決方案都是如此。

您應該使用 CLI 命令或 API 啟動區域轉移，以最大程度地減少啟動撤離所需的依賴關係。疏散過程越簡單，它就越可靠。特定的命令可以存儲在本地 runbook 中，隨時待命的工程師可以輕鬆訪問。區域轉移是疏散可用區域的最優選和最簡單的解決方案。

Route 53 ARC

第二個解決方案使用 Route 53 ARC 的功能手動指定特定 DNS 記錄的健康狀態。此解決方案具有使用高可用性 Route 53 ARC 群集數據平面的好處，使其能夠恢復多達兩個不同的損害AWS 區域。它具有額外費用的權衡，並且需要一些額外的 DNS 記錄配置。若要實作此模式，您需要建立別名記錄可用性區域特定的 DNS 名稱由負載平衡器 (ALB 或 NLB) 提供。這顯示在下表中。

表 3：為負載平衡器的區域 DNS 名稱設定的路由 53 別名記錄

路由策略: 加權	路由策略：加權	路由策略：加權
名稱: www.example.com	名稱: www.example.com	名稱: www.example.com
类型:A（别名）	類型: A（別名）	類型: A（別名）
Value (值)：us-east-1b.load-balancer-name.elb.us-east-1.amazonaws.com	值： us-east-1a.load-balancer-name.elb.us-east-1.amazonaws.com	值： us-east-1c.load-balancer-name.elb.us-east-1.amazonaws.com
重量:100	重量: 100	重量: 100
評估目標健康: 真	評估目標健康狀況： true	評估目標健康狀況： true

對於這些 DNS 記錄中的每一個，您都需要設定與路由 53 ARC 相關聯的路由 53 健康狀態檢查路由控制。當您要啟動可用區域撤離時，請將路由控制狀態設定為Off。AWS建議您使用 CLI 或 API 執行此操作，以便將啟動可用區域撤離所需的相依性降至最低。作為最佳做法，您應該保留 Route 53 ARC 叢集端點的本機副本，以便在需要執行疏散時不需要從 ARC 控制平面擷取這些端點。

若要將使用此方法時的成本降至最低，您可以在單一建立單一 Route 53 ARC 叢集和健康狀態檢查AWS 帳戶和與其他人共用健康檢查AWS 帳戶在您的組織中。當您採用這種方法時，您應該使用可用區域識別碼(AZ-識別碼) (例如，use1-az1) 而非可用區域名稱 (例如，us-east-1a）用於您的路由控件。因為AWS將實體可用區域隨機對應至每個區域的可用區域名稱AWS 帳戶，使用 AZ-ID 可提供一致的方式來參考相同的實體位置。當您啟動可用區撤離時，請說use1-az2，每個路線 53 記錄集AWS帳戶應確保他們使用 AZ-ID 對應來為每個 NLB 記錄配置正確的健康狀態檢查。

例如，假設我們有一個 Route 53 健康狀態檢查與 Route 53 ARC 路由控制項相關聯use1-az2，其識別碼為0385ed2d-d65c-4f63-a19b-2412a31ef431。如果在不同AWS 帳戶想要使用此健康檢查，us-east-1c已對應至use1-az2，您將需要使用use1-az2健康檢查記錄us-east-1c.load-balancer-name.elb.us-east-1.amazonaws.com。您將使用健康檢查 ID0385ed2d-d65c-4f63-a19b-2412a31ef431設置了該資源記錄。

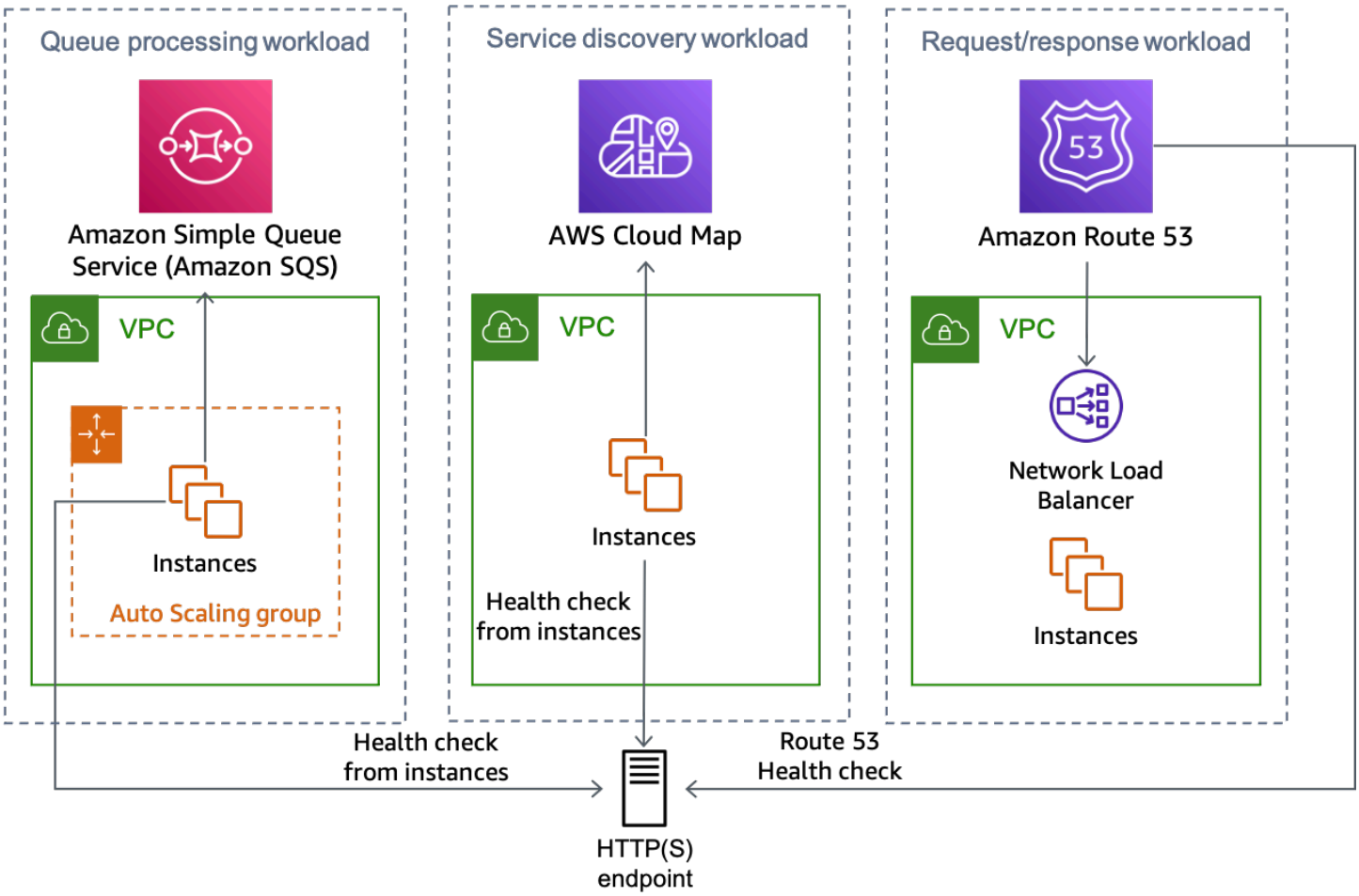
使用自我管理的 HTTP 端點

您也可以透過管理指出特定可用區域狀態的 HTTP 端點來實作此解決方案。它可讓您根據 HTTP 端點的回應，手動指定可用區域何時運作狀況不佳。此解決方案的成本低於使用 Route 53 ARC，但比區域轉移昂貴，並且需要管理額外的基礎架構。它具有針對不同場景更加靈活的好處。

該模式可用於 NLB 或 ALB 架構以及 Route 53 健康狀態檢查。它也可用於非負載平衡架構，例如服務探索或佇列處理系統，其中工作者節點會執行自己的健康狀態檢查。在這些情況下，主機可以使用後台線程，在該線程中定期使用其 AZ-ID 向 HTTP 端點發出請求（請參閱附錄 A — 取得可用區域識別碼有關如何找到這個的詳細信息）並收到有關可用區域運行狀況的響應。

如果可用區域已宣告為健康狀況不佳，則有多個選項可用於如何回應。他們可能會選擇不通過來源 (例如 ELB、Route 53 或服務探索架構中的自訂健康狀態檢查) 進行外部健康狀態檢查，以使這些服務看起來不健康。他們也可以在收到要求時立即回應錯誤，以允許用戶端退回並重試。在事件導向架構中，節點可能會故意無法處理工作，例如故意將 SQS 訊息傳回佇列。在中央服務計劃在特定主機上工作的工作路由器架構中，您也可以使用此模式。在選取 Worker、端點或儲存格之前，路由器可以檢查可用區域的狀態。在使用的服務探索架構中AWS Cloud Map，您可以通過在請求中提供過濾器來發現端點，例如 AZ-識別碼。

下圖顯示如何將此方法用於多種類型的工作負載。



多種工作負載類型都可以使用 HTTP 端點解決方案

有多種方法可以實現 HTTP 端點方法，其中兩種方法接下來概述。

使用 Amazon S3

這種模式最初是在這裡提出的部落格文章適用於多區域災難復原。您可以使用相同的模式進行可用區域撤離。

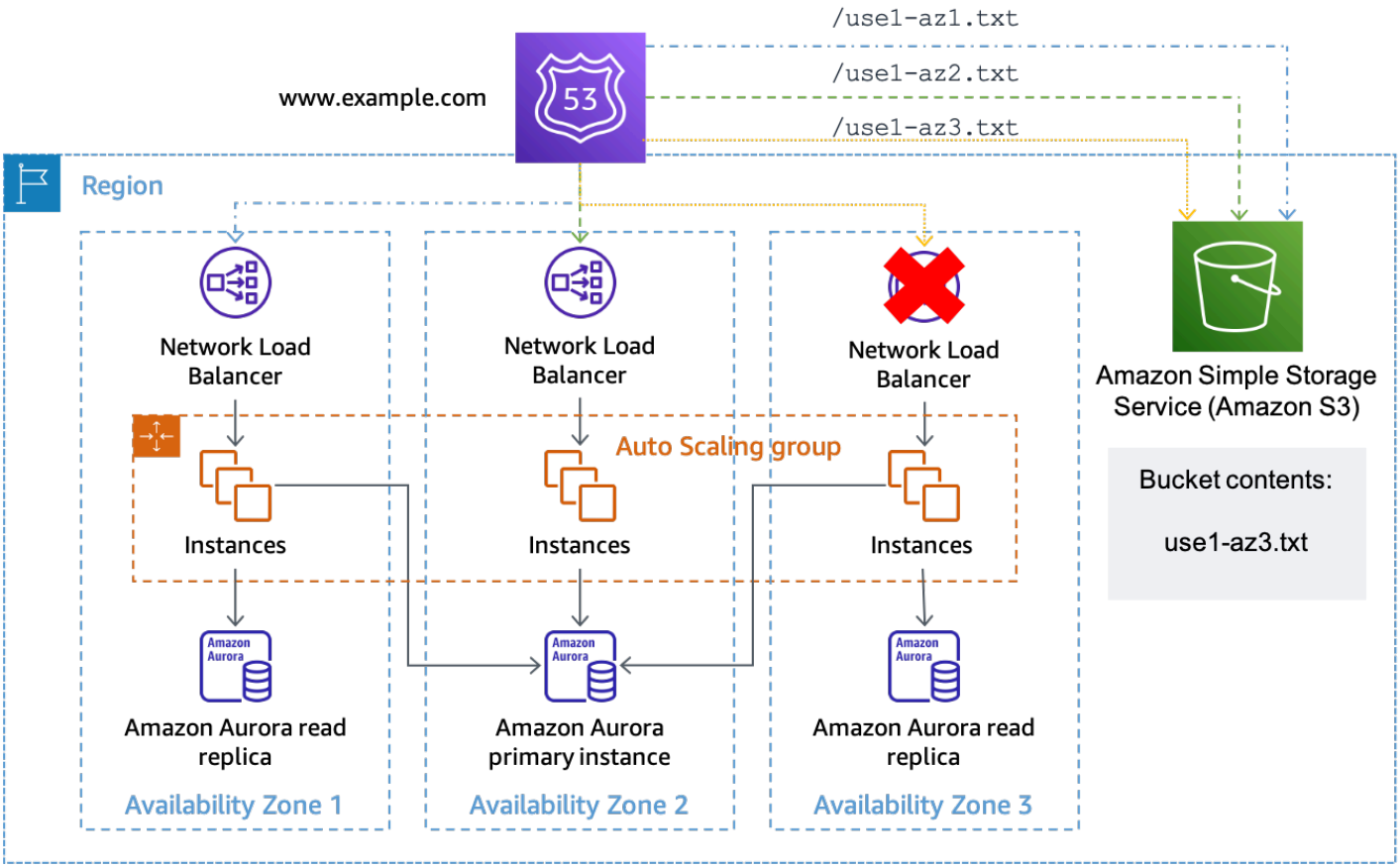
在這個案例中，您會為每個區域 DNS 記錄建立 Route 53 DNS 資源記錄集，就像53 號公路弧上述情況以及相關的健康狀態檢查。不過，對於此實作，而不是將健康狀態檢查與 Route 53 ARC 路由控制項產生關聯，而是將它們設定為使用端點並反轉以防止 Amazon S3 中的損害意外觸發撤離。健康檢查被考慮健康當對象不存在時不健康當對象存在時。此設定如下表所示。

表 4：針對每個可用區域使用 Route 53 健康狀態檢查的 DNS 記錄組態

健康檢查類型:	健康檢查類型:	健康檢查類型:	←	健康檢查

監視端點	監視端點	監視端點		
Protocol (通訊協定)：HTTPS	Protocol (通訊協定)：HTTPS	Protocol (通訊協定)：HTTPS		
識別碼:dddd-4444	識別碼:eeee-5555	識別碼:ffff-6666		
網址:https://*bucketname*.s3.us-east-1.amazonaws.com/use1-az1.txt	網址:https://*bucketname*.s3.us-east-1.amazonaws.com/use1-az3.txt	網址:https://*bucketname*.s3.us-east-1.amazonaws.com/use1-az2.txt		
↑	↑	↑		
路由策略: 加權	路由策略：加權	路由策略：加權		
名稱:www.example.com	名稱:www.example.com	名稱:www.example.com		
类型:A（别名）	類型: A（別名）	類型: A（別名）		
Value (值)：us-east-1b.load-balancer-name.elb.us-east-1.amazonaws.com	Value (值)：us-east-1a.load-balancer-name.elb.us-east-1.amazonaws.com	Value (值)：us-east-1c.load-balancer-name.elb.us-east-1.amazonaws.com	←	頂層、均勻加權別名 A 記錄指向 NLB AZ 特定端點
重量:100	重量: 100	重量: 100		
評估目標健康:true	評估目標健康狀況： true	評估目標健康狀況： true		

讓我們假設可用區us-east-1a已對應至use1-az3在我們有一個工作負載的帳戶中，我們要執行可用區撤離。針對為建立的資源記錄集us-east-1a.load-balancer-name.elb.us-east-1.amazonaws.com會關聯測試 URL 的健康檢查https://*bucket-name*.s3.us-east-1.amazonaws.com/use1-az3.txt。當您想要啟動可用區域撤離use1-az3，上傳名為的檔案use1-az3.txt使用 CLI 或 API 連接到值區。該文件不需要包含任何內容，但它確實需要公開，以便 Route 53 健康檢查可以訪問它。下圖演示了這個實現被用來撤離use1-az3。



使用亞馬遜 S3 作為路線 53 運行狀態檢查的目標

使用 API 閘道和動態支援

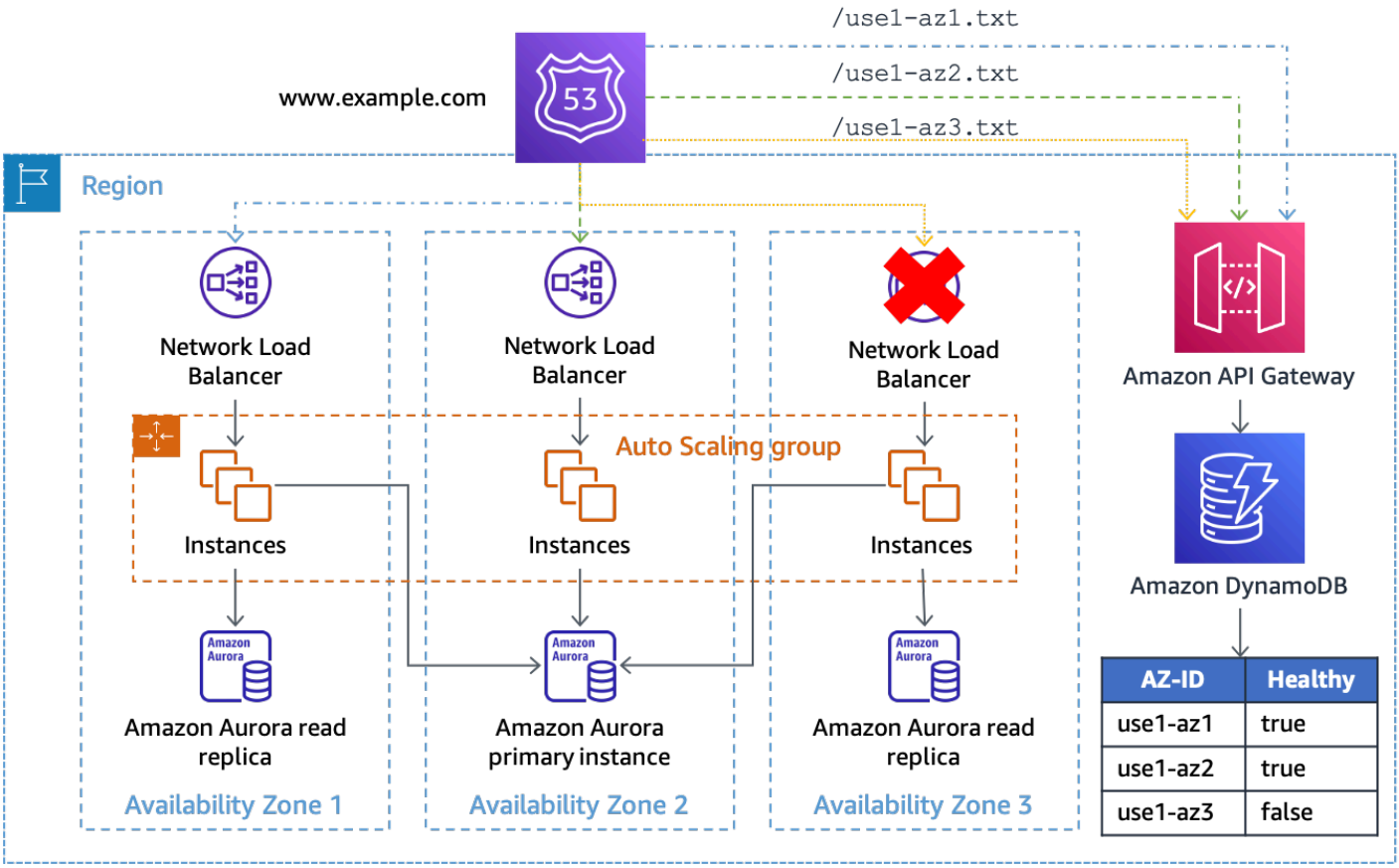
這種模式的第二個實現使用亞馬遜 API 閘道 休息 API。該 API 配置了服務整合傳送至 Amazon DynamoDB，其中會儲存每個使用中可用區域的狀態。此實作比 Amazon S3 方法更靈活，但需要建置、操作和監控更多基礎設施。它也可以用於 Route 53 健康狀態檢查，以及個別主機執行的健康狀態檢查。

如果您將此解決方案與 NLB 或 ALB 架構搭配使用，請以與上述 Amazon S3 範例相同的方式設定 DNS 記錄，但變更運作狀態檢查路徑以使用 API 閘道端點並提供AZ-ID在網址路徑中。例如，如果 API 閘道配置了自訂網域az-status.example.com，完整的請求use1-az1會看起來像https://

az-status.example.com/status/use1-az1。當您想要啟動可用區域撤離時，可以使用 CLI 或 API 建立或更新 DynamoDB 項目。該項目使用AZ-ID作為它的主鍵，然後有一個名為的布爾屬性Healthy用於表示 API 網關如何響應。以下是 API 閘道組態中用來決定此項目的範例程式碼。

```
#set($inputRoot = $input.path('$'))
#if ($inputRoot.Item.Healthy['BOOL'] == (false))
    #set($context.responseOverride.status = 500)
#end
```

如果屬性是true（或不存在），API 網關使用 HTTP 200 響應運行狀態檢查，如果它是假的，它會以 HTTP 500 進行響應。此實作如下圖所示。



使用 API 閘道和 DynamoDB 做為路由 53 運作狀態檢查的目標

在此解決方案中，您需要在 DynamoDB 前面使用 API 閘道，以便可以公開存取端點，並將請求 URL 操作為GetItem動態支援的請求。如果您想要在要求中包含其他資料，此解決方案也會提供彈性。例如，如果您想要建立更精細的狀態 (例如每個應用程式)，您可以設定健全狀況檢查 URL，以在路徑中提供應用程式 ID，或查詢字串也與 DynamoDB 項目相符。

可用區域狀態端點可集中部署，以便跨越多個健康狀態檢查資源AWS 帳戶所有人都可以使用可用區域健康狀態的相同一致性檢視 (確保您的 API 閘道 REST API 和 DynamoDB 表被調整以處理負載)，並且不需要共用 Route 53 運作狀態檢查。

該解決方案也可以擴展到多個AWS 區域使用亞馬遜全球表以及每個區域中 API 閘道 REST API 的副本。這樣可以防止此解決方案依賴於單一區域，並提高其可用性。您可以跨三個或五個區域部署解決方案，並使用大多數端點的結果來確保仲裁，每個區域查詢可用區域健全狀況。這允許在全域資料表中最終一致地複寫更新，以及減輕可能導致端點無法回應的損失。例如，如果您使用五個區域，而三個端點將可用區域報告為狀況不良，則一個端點會將可用區域報告為狀況良好，而一個端點沒有回應，您可以選擇將可用區域視為狀況不良。你也可以創建一個53 號路線計算健康檢查使用n 的米計算以執行此邏輯來判斷可用區域健全狀況。

如果您要為個別主機建置解決方案，以做為判斷其 AZ 健康狀態的機制，作為替代提供運作狀態檢查的提取機制，您可以使用推送通知。一種方法是使用您的消費者訂閱的 SNS 主題。當您想要觸發斷路器時，請將訊息發佈至 SNS 主題，指出哪個可用區域受損。這種方法使得與前者的權衡。它不需要建立和操作 API 閘道基礎結構，以及執行容量管理。它也可能提供可用區域狀態的更快速整合。不過，它會移除執行隨機查詢的能力，並依賴於SNS 傳遞重試政策以確保每個端點都會收到通知。它還需要每個工作負載或服務建立一種接收 SNS 通知並對其採取行動的方式。

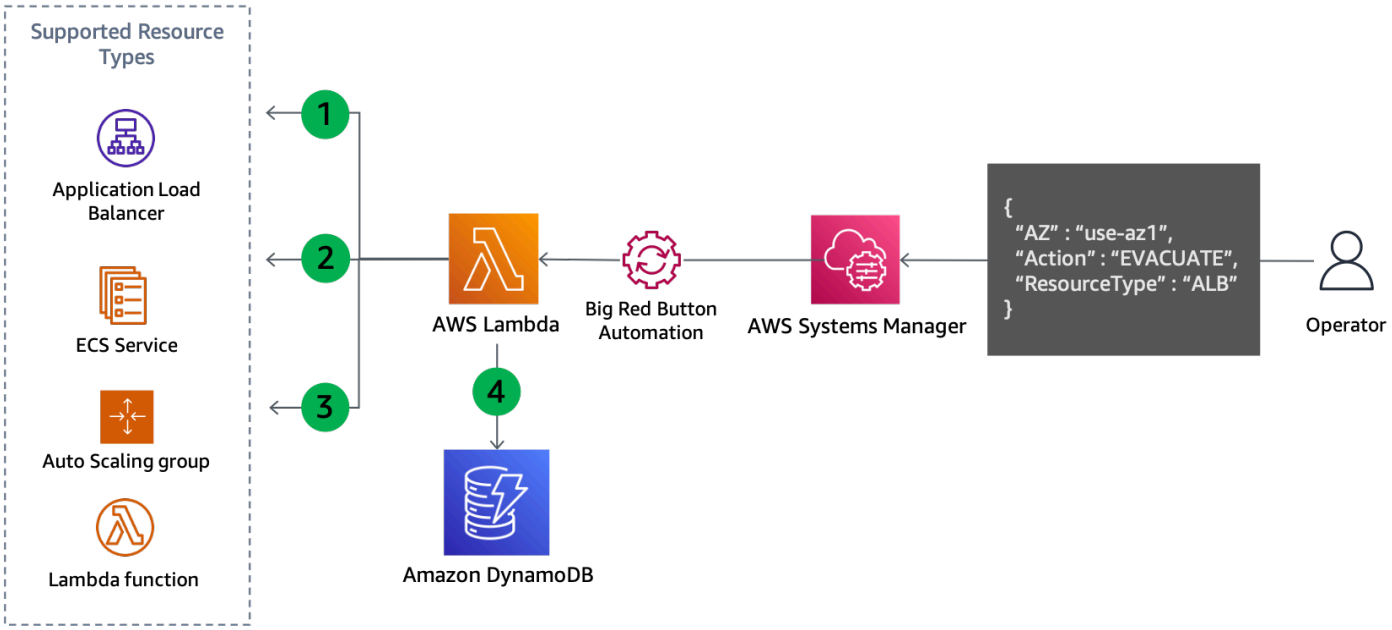
例如，啟動的每個新 EC2 執行個體或容器都需要在啟動期間使用 HTTP 端點訂閱主題。然後，每個執行個體都需要實作在傳送通知的此端點上偵聽的軟體。此外，如果執行個體受到事件影響，則可能不會收到推播通知並繼續執行工作。而透過提取通知，執行個體就會知道其提取要求是否失敗，並且可以選擇要採取的動作來回應。

發送推送通知的第二種方式是長壽WebSocket連接。亞馬遜 API 網關可用於提供WebSocketAPI消費者可以在以下情況下連接並接收消息由後端發送。用一個WebSocket，執行個體都可以執行定期提取，以確保連線狀態良好，並接收低延遲推送通知。

控制平面控制疏散

第一個模式使用資料平面作業來防止在受影響的可用區域中執行工作，以減輕事件的影響。但是，您可能使用的架構不使用負載平衡器，或者設定每台主機健康狀態檢查是不可行的。或者，您可能想要透過 Auto Scaling 或正常工作排程，防止將新容量部署到受影響的可用區域。

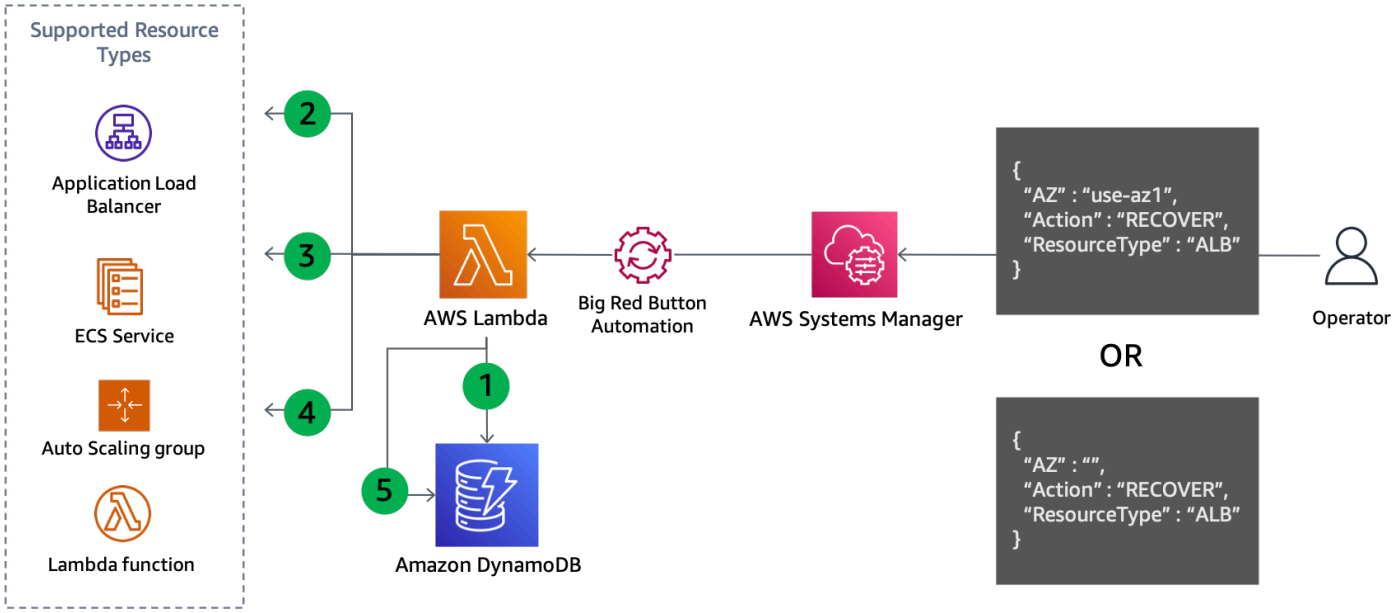
為了解決這兩種情況，需要控制平面動作來更新資源的配置。該模式適用於任何可更新網路組態的服務，例如 EC2 自動擴展、Amazon ECS、Lambda 等。它需要為每個服務編寫代碼，但業務邏輯遵循標準模式。程式碼應由回應事件的運算子在本機執行，以減少所需的相依性。腳本邏輯的基本流程如下圖所示。



控制平面更新以撤除可用區域

1. 指令碼會列出指定類型的所有資源，例如自動擴展群組、ECS 服務或 Lambda 函數，並從資源資訊擷取其子網路。支援的資源取決於指令碼已設定為支援的資源。
2. 它會將每個子網路的可用區域名稱與其作為輸入參數提供的對應可用區域識別碼進行比較，判斷應移除哪些子網路。
3. 資源的網路組態會更新，以移除已識別的子網路。
4. 更新的詳細資訊會記錄在 DynamoDB 資料表中。可用區域 ID 會儲存為分割索引鍵並且資源 ARN 或名稱存儲為排序鍵。已移除的子網路會儲存為字串陣列。最後，資源類型也被存儲並用作哈希鍵全球次要指數(GSI).

因為步驟四會記錄所做的更新，所以當您準備好復原時，這個方法也可以輕鬆復原，如下圖所示。



控制平面更新以從可用區域撤離中恢復

恢復步驟：

1. 查詢 GSI 以取得針對指定可用區域中指定類型的每個資源移除子網路 (如果未指定所有可用區域)。
2. 描述 DynamoDB 查詢中找到的每個資源，以取得其目前的網路組態。
3. 將目前網路組態中的子網路與從 DynamoDB 查詢擷取的子網路結合在一起。
4. 使用新的子網路集更新資源的網路組態。
5. 成功完成更新後，請從 DynamoDB 表格中移除記錄。

此一般化模式可防止將工作路由傳送至受影響的可用區域，並防止在該處部署新容量。以下是如何針對不同服務完成此操作的範例。

- 拉姆達-更新功能虛擬私人雲端組以移除指定可用區域中的子網路。
- 自動縮放群組—從 ASG 組態移除子網路這將取代剩餘可用區域中的該容量。
- 亞馬遜 ECS—更新 ECS 服務虛擬私人雲端組態以移除子網路。
- 亚马逊 EKS— 申請污點至受影響可用區域中的節點，以收回現有的網繭，並防止在該處排程其他網繭。

每個服務都會對組態更新做出不同的反應。例如，亞馬遜 ECS 將遵循更新後的服務部署組態並觸發新任務的滾動部署或藍/綠部署。

對於某些工作負載，這些更新可能會過快地將工作轉移到正常運作的可用區 雖然設定為靜態穩定以防止故障 (在剩餘可用區域中預先佈建足夠的容量以處理受影響的可用區域的工作)，但您也可能想要逐漸從受影響的可用區域逐步淘汰容量。

如果您計劃更新 Auto Scaling 群組的網路組態，該群組是具有跨區域負載平衡功能之負載平衡器的目標群組殘疾人，請遵循此指導。

「自動縮放」會使用其對此變更做出反應可用性區域重新平衡邏輯。它會在其他可用區域中啟動執行個體，以符合您所需的容量，並在您移除的可用區域中終止執行個體。不過，負載平衡器會繼續在每個可用區域中平均分割流量，包括您從 ASG 移除的可用區域，同時終止執行個體。這可能會導致該可用區域的剩餘容量失效，直到所有執行個體在該處成功終止為止。這與中描述的相同問題可用區域獨立關於停用跨區域負載平衡時的可用區域不平衡。為了防止這種情況發生，您可以：

- 始終首先執行可用區域疏散，以便流量僅在剩餘的可用區域中分割
- 指定DNS 容錯移轉的最小健全目標計數以符合該可用區域所需的最低目標計數。

這有助於確保在執行個體開始終止之後，流量不會傳送到您移除的可用區域。

總結

下表總結了所描述的疏散模式的利弊。

表 5：疏散模式的利弊

方法	優點	缺點
數據平面控制疏散	僅依賴資料平面動作 快速防止在受影響的可用區域中完成工作 集中檢視可用區域健全狀況的彈性方法	不會防止容量部署在受影響的可用區域 並非所有工作負載類型都能輕鬆使用此方法
控制平面控制疏散	防止在受影響的可用區域中部署新容量	依賴於每個服務的控制平面

方法	優點	缺點
	從受影響的可用區域移除現有容量	需要為每個服務編寫代碼 必須按服務完成服務 需要注意不要在更新期間壓倒容量

您可能會同時使用這兩種方法作為可用區撤離計劃的一部分。從資料平面控制的疏散動作開始，這些動作更有可能成功地快速停止受影響可用區域中的處理工作。然後，一旦最初的影響被緩解，跟進控制平面控制的疏散行動，如果你認為有必要。

結論

本白皮書提供了灰色故障的概述，以及它們如何顯示，並概述了為什麼您需要構建可觀察性和疏散工具以減輕這些類型的事件發生時。在下一節中，您將檢閱異地同步備份可觀察性，以及可實施的三種方法來偵測單一可用區域的影響。在上一節中，本文介紹了執行可用區域撤離的兩種一般方法。第一種方法使用資料平面動作來防止將工作路由到受影響的可用區域，而第二種方法則使用控制平面動作來防止在受影響的可用區域中佈建容量。這兩種方法共同實現了可用區域撤離意圖的兩個結果。

本白皮書描述的復原模式可能是較大型監控和故障回復解決方案的一部分。這種處理單一可用區域灰色故障的方法需要進行工程工作，以建立偵測它們所需的儀器，以及回應這些故障的工具。但是，對於許多工作負載而言，這種方法可能是建立多區域架構的更簡單且成本更低的替代方案。此外，與多區域 DR 相比，它可以幫助實現更小的 RPO 和 RTO（從而增加工作負載的可用性）。

附錄 A — 取得可用區域識別碼

如果您正在使用AWS.NET SDK (以及其他一些類似的JavaScript) 或在 EC2 執行個體 (包括 Amazon ECS 和 Amazon EKS) 上執行您的系統，您可以直接取得可用區域 ID。

- AWS.NET SDK

```
Amazon.Util.EC2InstanceMetadata.GetData(“/placement/availability-zone-id”)
```

- EC2 實例中繼資料服務

```
curl http://169.254.169.254/latest/meta-data/placement/availability-zone-id
```

在其他平台 (例如 Lambda 和 Fargate) 上，您將需要擷取可用區域名稱，然後尋找與可用區域 ID 的對應。使用可用區域名稱，您可以找到如下所示的可用區域 ID：

```
aws ec2 describe-availability-zones --zone-names $AZ --output json
    --query ‘AvailabilityZones[0].ZoneId’
```

下面的實例找到要在上面的例子中使用的可用區域名稱都寫在 bash 中使用AWS CLI和包JQ。需要將它們轉換為用於工作負載的程式設計語言。

- 亞馬遜 ECS-如果主機封鎖執行個體中繼資料服務 (IMDS)，您可以改用容器中繼資料檔案。

```
AZ=$(cat $ECS_CONTAINER_METADATA_FILE | jq --raw-output
    .AvailabilityZone)
```

- 法蓋特(平台版本 1.4 或更新版本)

```
AZ=$(curl $ECS_CONTAINER_METADATA_URI_V4/task | jq --raw-output
    .AvailabilityZone)
```

- 拉姆達— 可用區域不會直接暴露給功能。要找到它，您需要完成幾個步驟。為此，您將需要構建一個私有 API 閘道 REST 端點，該端點返回請求者的 IP 地址。這將識別分配給該功能正在使用的彈性網絡接口的私有 IP。
  - 呼叫拉姆達`GetFunction`用於尋找函數的虛擬私人雲端識別碼的 API。

- 呼叫 API 閘道服務以取得函數的 IP。
- 使用 IP 和 VPC ID，尋找相關聯的網路介面並擷取可用區域。

```
VPC_ID=$(aws lambda get-function --function-name $ AWS_LAMBDA_FUNCTION_NAME --
region $AWS_REGION --output json --query 'Configuration.VpcConfig.VpcId')

MY_IP=$(curl http://whats-my-private-ip.internal)

AZ=$(aws ec2 describe-network-interfaces --filters Name=private-ip-address,Values=
$MY_IP Name=vpc-id,Values=$VPC_ID --region $AWS_REGION --output json –query
 'NetworkInterfaces[0].AvailabilityZone')
```

附錄 B-示例卡方計算

以下是收集錯誤測量結果和對資料執行卡方測試的範例。程式碼尚未就緒，不會執行必要的錯誤處理，但確實提供了邏輯運作方式的概念證明。您應該更新此範例以符合您的需求。

首先，一個 Lambda 函數是由亞馬遜每分鐘調用EventBridge已排程的事件。事件的內容設定為下列資料：

```
{
  "timestamp": "2023-03-15T15:26:37.527Z",
  "namespace": "multi-az/frontend",
  "metricName": "5xx",
  "dimensions": [
    { "Name": "Region", "Value": "us-east-1" },
    { "Name": "Controller", "Value": "Home" },
    { "Name": "Action", "Value": "Index" }
  ],
  "period": 60,
  "stat": "Sum",
  "unit": "Count",
  "chiSquareMetricName": "multi-az/chi-squared",
  "azs": [ "use1-az2", "use1-az4", "use1-az6" ]
}
```

該數據用於指定檢索適當所需的共同數據CloudWatch量度 (例如命名空間、量度名稱和維度)，然後針對每個可用區域發佈卡方結果。在拉姆達函數中的代碼看起來像下面使用 Python 3.9。在較高的水平，它收集指定的CloudWatch前一分鐘的指標，對該資料執行卡方測試，然後發佈CloudWatch與指定之每個可用區域之測試結果相關的測量結果。

```
import os
import boto3
import datetime
import copy
import json
from datetime import timedelta
from scipy.stats import chisquare
from aws_embedded_metrics import metric_scope

cw_client = boto3.client("cloudwatch", os.environ.get("AWS_REGION", "us-east-1"))
```

```
@metric_scope
def handler(event, context, metrics):
    metrics.set_property("Event", json.loads(json.dumps(event, default = str)))
    time = datetime.datetime.strptime(event["timestamp"], "%Y-%m-%dT%H:%M:%S.%fZ")

    # Round down to the previous minute
    end: datetime = roundTime(time)

    # Subtract a minute for the start
    start: datetime = end - timedelta(minutes = 1)

    # Get all the metrics that match the query
    results = get_all_metrics(event, start, end, metrics)
    metrics.set_property("MetricCounts", results)

    # Calculate the chi squared result
    chi_sq_result = chisquare(list(results.values()))
    expected = sum(list(results.values())) / len(results.values())
    metrics.set_property("ChiSquaredResult", chi_sq_result)

    # Put the chi square metrics into CloudWatch
    put_all_metrics(event, results, chi_sq_result[1], expected, start, metrics)

def get_all_metrics(detail: dict, start: datetime, end: datetime, metrics):
    """
    Gets all of the error metrics for each AZ specified
    """
    metric_query = {
        "MetricDataQueries": [
        ],
        "StartTime": start,
        "EndTime": end
    }

    for az in detail["azs"]:

        dim = copy.deepcopy(detail["dimensions"])
        dim.append({"Name": "AZ-ID", "Value": az})

        query = {
            "Id": az.replace("-", "_"),
            "MetricStat": {
                "Metric": {
                    "Namespace": detail["namespace"],
```

```
                    "MetricName": detail["metricName"],
                    "Dimensions": dim
                },
                "Period": int(detail["period"]),
                "Stat": detail["stat"],
                "Unit": detail["unit"]
            },
            "Label": az,
            "ReturnData": True
        }

        metric_query["MetricDataQueries"].append(query)

    metrics.set_property("GetMetricRequest", json.loads(json.dumps(metric_query,
default=str)))
    next_token: str = None
    results = {}

    while True:
        if next_token is not None:
            metric_query["NextToken"] = next_token

        data = cw_client.get_metric_data(**metric_query)

        if next_token is not None:
            metrics.set_property("GetMetricResult::" + next_token,
json.loads(json.dumps(data, default = str)))
        else:
            metrics.set_property("GetMetricResult", json.loads(json.dumps(data, default
= str)))

        for item in data["MetricDataResults"]:
            key = item["Id"].replace("_", "-")
            if key not in results:
              results[key] = 0

            results[key] += sum(item["Values"])

        if "NextToken" in data:
            next_token = data["NextToken"]

        if next_token is None:
            break
```

```
    return results

def put_all_metrics(detail: dict, results: dict, chi_sq_value: float, expected: float,
 timestamp: datetime, metrics):
    """
    Adds the chi squared metric for all AZs to CloudWatch
    """
    farthest_from_expected = None
    if len(results) > 0:
        keys = list(results.keys())
        farthest_from_expected = keys[0]

        for key in keys:
            if abs(results[key] - expected) > abs(results[farthest_from_expected] -
 expected):
                farthest_from_expected = key

    metric_query = {
        "Namespace": detail["namespace"],
        "MetricData": []
    }

    for az in detail["azs"]:
        dim = copy.deepcopy(detail["dimensions"])
        dim.append({"Name": "AZ-ID", "Value": az})

        query = {
            "MetricName": detail["chiSquareMetricName"],
            "Dimensions": dim,
            "Timestamp": timestamp,
        }

        if chi_sq_value <= 0.05 and az == farthest_from_expected:
            query["Value"] = 1
        else:
            query["Value"] = 0

        metric_query["MetricData"].append(query)

    metrics.set_property("PutMetricRequest", json.loads(json.dumps(metric_query,
 default = str)))

    cw_client.put_metric_data(**metric_query)
```

```
def roundTime(dt=None, roundTo=60):
   """Round a datetime object to any time lapse in seconds
   dt : datetime.datetime object, default now.
   roundTo : Closest number of seconds to round to, default 1 minute.
   """
   if dt == None : dt = datetime.datetime.now()
   seconds = (dt.replace(tzinfo=None) - dt.min).seconds
   rounding = (seconds+roundTo/2) // roundTo * roundTo
   return dt + datetime.timedelta(0,rounding-seconds,-dt.microsecond)
```

然後，您可以為每個 AZ 建立警示。下面的例子是use1-az2以及連續三個 1 分鐘資料點的警示，其上限值等於 1 (1 是卡方測試判斷錯誤率的統計上顯著偏斜時所發佈的量度)。

```
{
    "Type": "AWS::CloudWatch::Alarm",
    "Properties": {
        "AlarmName": "use1-az2-chi-squared",
        "ActionsEnabled": true,
        "OKActions": [],
        "AlarmActions": [],
        "InsufficientDataActions": [],
        "MetricName": "multi-az/chi-squared",
        "Namespace": "multi-az/frontend",
        "Statistic": "Maximum",
        "Dimensions": [
            {
                "Name": "AZ-ID",
                "Value": "use1-az2"
            },
            {
                "Name": "Action",
                "Value": "Index"
            },
            {
                "Name": "Region",
                "Value": "us-east-1"
            },
            {
                "Name": "Controller",
                "Value": "Home"
            }
```

```
        ],
        "Period": 60,
        "EvaluationPeriods": 3,
        "DatapointsToAlarm": 3,
        "Threshold": 1,
        "ComparisonOperator": "GreaterThanOrEqualToThreshold",
        "TreatMissingData": "missing"
    }
}
```

您也可以建立m-of-n報警並將這兩個警報與複合警報結合在一起。您也需要為每個可用區域中的每個控制器/動作組合或微服務建立相同的警示。最後，您可以將卡方複合警報新增至每個控制器/動作組合的可用區域特定警報，如圖所示使用離群值偵測進行故障偵測。

貢獻者

本文件的貢獻者包括：

- 邁克爾·哈肯，首席解決方案架構師，亞馬遜 Web 服務

文件修訂

若要收到有關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	已更新其他可觀測性指引，並使用新的區域偏移功能。	2023年7月11日
初始出版	白皮書首次出版。	2022 年 3 月 2 日

> **Note**
>
> 若要訂閱 RSS 更新，您必須為正在使用的瀏覽器啟用 RSS 外掛程式。

注意

客戶有責任對本文件中的資訊進行獨立評估。本文件：（a）僅供參考，（b）代表當前文件AWS產品供應項目和做法如有變更，恕不另行通知，且 (c) 不會建立任何承諾或保證AWS及其關聯公司、供應商或授權人。AWS產品或服務係依「原狀」提供，不含任何明示或暗示之擔保、陳述或條件。的責任和責任AWS它的客戶控制AWS協定，且此文件不屬於任何協議的一部分，也不會修改兩者之間的協議AWS及其客戶。

© 2023 亞馬遜網絡服務公司或其附屬公司。保留所有權利。

AWS 詞彙表

如需最新的 AWS 術語，請參閱《AWS 詞彙表 參考》中的 AWS 詞彙表。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。