AWS 白皮书

可用性和超越：了解和提高分佈式系統的彈性 AWS

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

可用性和超越：了解和提高分佈式系統的彈性 AWS: AWS 白皮书

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，這些擁有者可能隸屬於 Amazon，或與 Amazon 有合作關係，或由 Amazon 贊助。

Table of Contents

摘要及介紹 i
簡介 1
了解可用性 2
分散式系統可用 3
分散式系統中的故障類型 5
可用性與相依性 6
備援的可用性 7
帽定理 10
容錯和故障隔離 11
測量可用性 13
伺服器端和用戶端要求成功率 13
年度停機時 15
Latency (延遲) 15
設計高可用性的分散式系統 AWS 17
減少 MTTD 17
減少 MTTR 18
故障周圍的路由 18
返回已知良好狀態 19
故障診斷 21
手冊和自動化 21
增加 MTBF 21
增加分散式系統 MTBF 21
增加相依性 MTBF 23
減少常見的影響來源 24
結論 26
附錄 1 — MTTD 和 MTTR 的關鍵指標 28
貢獻者 29
深入閱讀 30
文件歷史紀錄 31
注意 32
AWS 詞彙表 33
..... xxxiv

可用性和超越：了解和提高分佈式系統的彈性 AWS

出版日期：二零二一年十一月十二文件歷史紀錄日

如今，企業在雲端和內部部署中運作複雜的分散式系統。他們希望這些工作負載具有彈性，以便為客戶提供服務並實現其業務成果。本白皮書概述了可用性的共同理解，作為恢復性的衡量方法，建立了建立高可用性工作負載的規則，並提供有關如何提高工作負載可用性的指導。

簡介

建置高可用性工作負載意味著什麼？ 您如何衡量可用性？ 我該怎麼做才能提高工作負載的可用性？ 本文件將幫助您回答這些類型的問題。它分為三個主要部分。第一節，了解可用性在很大程度上是理論上的。它建立了可用性的定義和影響它的因素的共同理解。第二節「測量可用性」提供有關以實證方式衡量工作負載可用性的指導。第三部分，在上設計高可用性的分散式系統AWS是第一節中介紹的想法的實際應用。此外，在這些章節中，本白皮書將識別用於建置彈性工作負載的規則。本文件旨在支援「AWS架構良好的可靠性支柱」中提供的指導和最佳實務。

在本論文中，您將遇到很多代數數學。關鍵要點是這個數學支持的概念，而不是數學本身。也就是說，這也是本文提出挑戰的意圖。當您操作高可用性工作負載時，您需要能夠在數學上證明您建置的內容正在實現您的預期。即使是建立在良好意圖上的最佳設計，也可能無法始終達到預期的結果。這意味著您需要測量解決方案有效性的機制，因此在構建和操作具有彈性且高可用性的分佈式系統時，需要某種程度的數學。

了解可用性

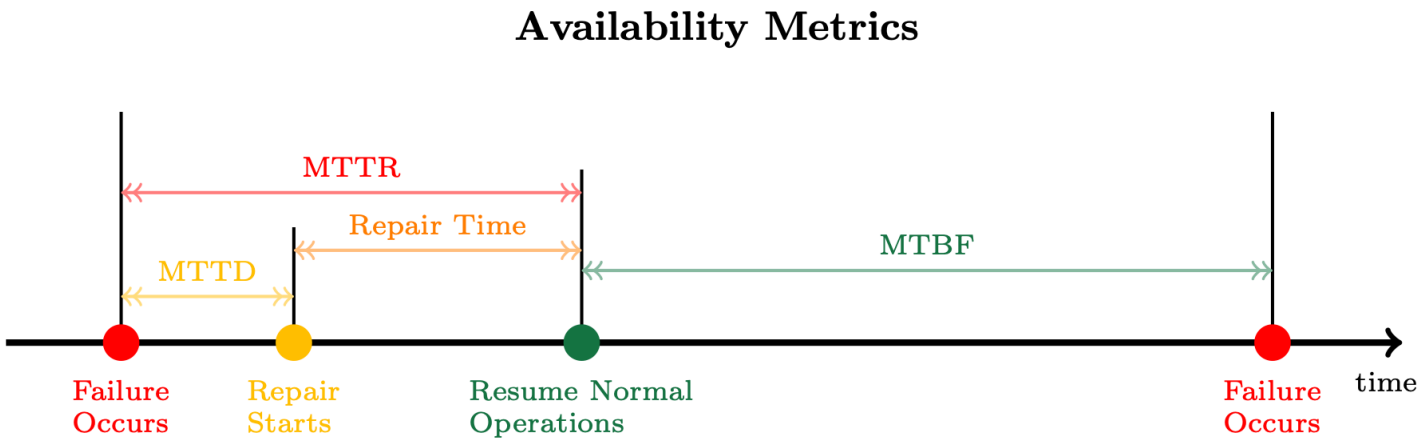
可用性是我們可以量化測量恢復能力的主要方法之一。我們將可用性 A 定義為工作負載可供使用的時間百分比。這是其預期的「正常運行時間」（可用）與測量總時間（預期的「正常運行時間」加上預期的「停機時間」）的比率。

$$A = \frac{uptime}{uptime + downtime}$$

公式 1-可用性

為了更好地理解此公式，我們將研究如何衡量正常運行時間和停機時間。首先，我們想知道工作負載將不會失敗多長時間。我們稱之為平均故障間隔時間 (MTBF)，即工作負載開始正常運作到下一次失敗之間的平均時間。然後，我們想知道失敗後恢復需要多長時間。

我們稱此為修復 (或復原) 的平均時間 (MTTR)，這是在修復失敗的子系統或返回服務時，工作負載無法使用的一段時間。MTTR 中的一個重要時間段是平均偵測時間 (MTTD)、發生故障與修復作業開始之間的時間長度。下圖說明所有這些度量如何相關。



MTTD、MTTR 與 MTBF 之間的關係

因此，我們可以表示可用性，A，使用 MTBF，工作負載啟動的時間和 MTTR，工作負載關閉的時間。

$$A = \frac{MTBF}{MTBF + MTTR}$$

方程式 2-MTBF 和 MTTR 之間的關係

而工作負載「下降」的概率（也就是說，不可用）是失敗的概率，F。

$$F = 1 - A$$

公式 3-失敗的概率

可靠性是指工作負載能夠在要求時在指定的回應時間內完成正確事情的能力。這是什麼可用性措施。工作負載失敗的頻率較低 (MTBF 較長) 或修復時間較短 (縮短 MTTR)，可改善其可用性。

> 規則 1
>
> 較少的故障頻率 (較長的 MTBF)、更短的故障偵測時間 (縮短 MTTD)，以及更短的修復時間 (縮短 MTTR) 是用來改善分散式系統可用性的三個因素。

主題

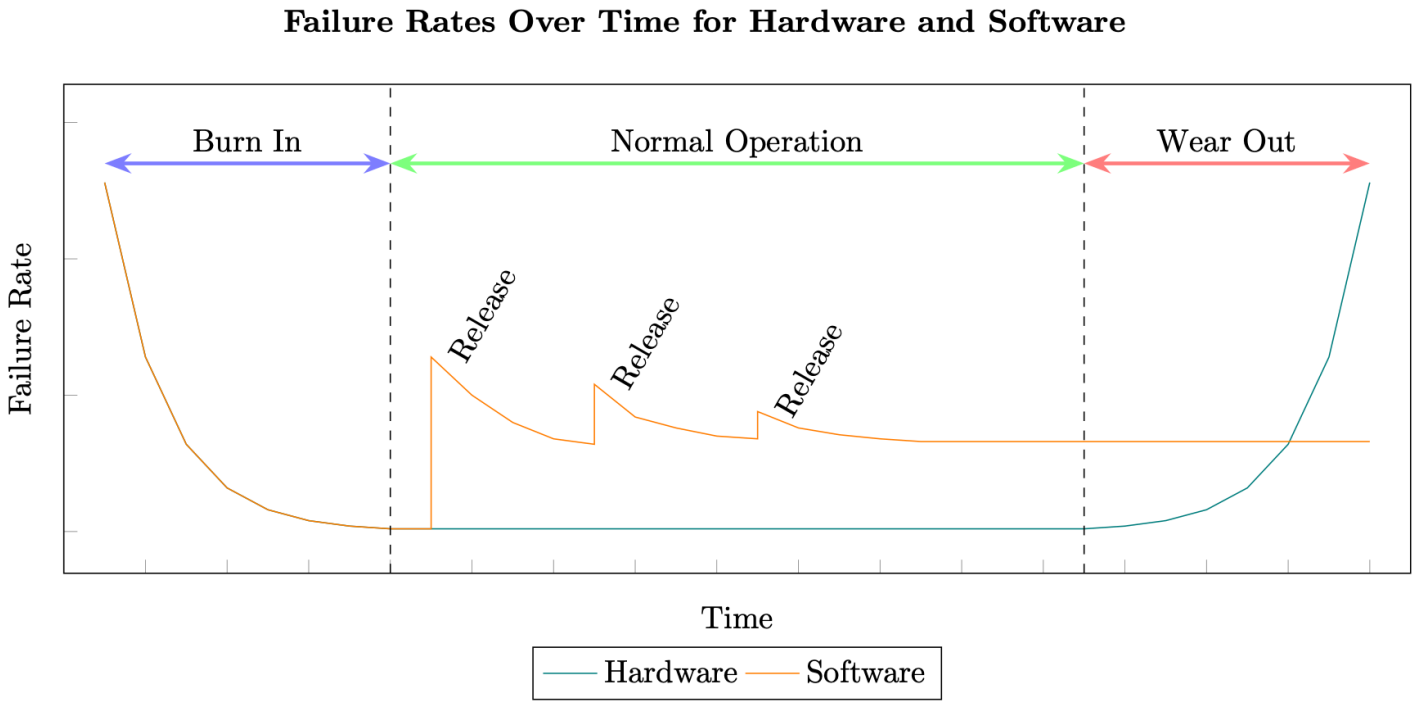
- 分散式系統可用
- 可用性與相依性
- 備援的可用性
- 帽定理
- 容錯和故障隔離

分散式系統可用

分佈式系統由軟件組件和硬件組件組成。某些軟體元件本身可能是另一個分散式系統。基礎硬體和軟體元件的可用性會影響工作負載產生的可用性。

使用 MTBF 和 MTTR 的可用性的計算有其根源於硬件系統。但是，分散式系統失敗的原因與一個硬體完全不同。如果製造商可以持續計算硬體元件耗盡之前的平均時間，則相同的測試無法套用至分散式系

統的軟體元件。硬體通常遵循故障率的「浴缸」曲線，而軟體則遵循由每個新版本引入的額外缺陷所產生的交錯曲線 (請參閱軟體可靠性)。



硬件和軟件故障率

此外，分散式系統中的軟體通常會以比硬體更高的速率變更。例如，標準磁性硬碟的平均年故障率 (AFR) 可能為 0.93%，在實際情況下，硬碟的使用壽命可能至少為 3-5 年，可能會延長 (請參閱 2020 年 Backblaze 硬碟資料與統計資料)。硬碟在該生命週期內不會發生重大變化，例如，在 3 到 5 年內，Amazon 可能會在其軟體系統上部署超過 450 至 7.5 億項變更。請參閱 Amazon Builders' Library — 自動執行安全、免提部署。)

硬件也受到計劃過時的概念，即具有內置的壽命，在一段時間後需要更換。（見偉大的燈泡陰謀。）從理論上講，軟件不受這種限制，它沒有磨損期，可以無限期地操作。

所有這些都意味著用於生成 MTBF 和 MTTR 號碼的硬件的相同測試和預測模型不適用於軟件。自 1970 年代以來，已經有數百次嘗試構建模型來解決此問題，但它們通常分為兩類，即預測模型和估計建模。(請參閱軟體可靠性型號清單。)

因此，計算分佈式系統的前瞻性 MTBF 和 MTTR，從而具有前瞻性的可用性，將始終從某種類型的預測或預測衍生出來。它們可以通過預測建模，隨機模擬，歷史分析或嚴格的測試生成，但這些計算並不保證正常運行時間或停機時間。

分散式系統過去失敗的原因可能永遠不會再次發生。future 失敗的原因很可能會有所不同，並且可能不知道。future 失敗所需的復原機制也可能與過去使用的復原機制不同，因此需要大幅不同的時間。

此外，MTBF 和 MTTR 是平均值。從平均值到看到的實際值會有一些變化（標準差 σ，測量此變化）。因此，在實際生產使用中，工作負載可能會在故障和復原時間之間經歷較短或更長的時間

話雖如此，構成分佈式系統的軟件組件的可用性仍然很重要。軟體失敗的原因有許多 (在下一節中詳細討論)，並會影響工作負載的可用性。因此，對於高可用性的分散式系統而言，應該關注軟體元件的計算、測量和改善軟體元件的可用性等於硬體和外部軟體子系統。

> 規則 2
>
> 工作負載中軟體的可用性是工作負載整體可用性的重要因素，而且應該與其他元件相同。

重要的是要注意，儘管 MTBF 和 MTTR 難以預測分散式系統，但它們仍然提供有關如何提高可用性的關鍵見解。降低故障頻率 (更高的 MTBF) 並減少故障發生後的復原時間 (降低 MTTR)，都會導致更高的實證可用性。

分散式系統中的故障類型

在影響可用性的分佈式系統中通常有兩類錯誤，親切地命名為博爾錯誤和海森堡（見「與布魯斯·林賽的對話」，ACM 隊列第 2 卷，第 8-2004 年 11 月）。

Bohrbug 是一個可重複的功能軟件問題。給定相同的輸入，該錯誤將始終產生相同的不正確輸出（如確定性的 Bohr 原子模型，它是固體且易於檢測的）。當工作負載進入生產環境時，這些類型的錯誤很少見。

海森錯誤是一個暫時性的錯誤，這意味著它只發生在特定和不常見的情況下。這些條件通常與硬體 (例如暫時性裝置錯誤或硬體實作細節 (如暫存器大小)、編譯器最佳化和語言實作、限制條件 (例如暫時不在儲存空間) 或競爭條件 (例如，不使用信號量進行多執行緒作業)。

Heiisenbugs 構成了生產中的大多數錯誤，並且很難找到，因為它們難以捉摸，並且在您嘗試觀察或調試它們時似乎會改變行為或消失。但是，如果您重新啟動程式，失敗的作業可能會成功，因為作業環境略有不同，從而消除了引入 Heiisenbug 的條件。

因此，生產中的大多數故障都是暫時的，當重試操作時，不太可能再次失敗。為了保持彈性，分佈式系統必須對海森蟲具有容錯能力。我們將探討如何實現這一部分增加分佈式系統 MTBF。

可用性與相依性

在上一節中，我們提到硬體、軟體和潛在的其他分散式系統都是您工作負載的元件。我們稱這些元件相依性為您的工作負載所依賴的項目來提供其功能。有一些硬性依賴關係，這是您的工作負載無法在沒有的情況下運行的東西，以及不可用性可能在一段時間內被忽視或容忍的軟相依性。硬式相依性會直接影響工作負載的可用性。

我們可能想要嘗試計算工作負載的理論上最大可用性。這是所有相依性 (包括軟體本身) 的可用性產品 (α_n 是單一子系統的可用性)，因為每個子系統都必須可以運作。

$$A = \alpha_1 \times \alpha_2 \times \ldots \times \alpha_n$$

公式 4-理論上的最大可用性

這些計算中使用的可用性數字通常與 SLA 或服務等級目標 (SLO) 等項目相關聯。SLA 定義了客戶將獲得的預期服務等級、測量服務的指標，以及如果未達到服務水平的補救或罰款（通常是金錢）。

使用上面的公式，我們可以得出結論，純粹的數學上，工作負載不能超過其任何依賴關係。但實際上，我們通常看到的是情況並非如此。使用具有 99.99% 可用性 SLA 的兩個或三個相依性建置的工作負載本身仍可達到 99.99% 或更高的可用性。

這是因為正如我們在上一節中概述的那樣，這些可用性數字是估計值。他們估計或預測故障發生的頻率以及修復的速度。它們不是停機時間的保證。相依性通常超過其規定的可用性 SLA 或 SLO。

相依性也可能具有較高的內部可用性目標，這些目標是針對效能，而不是公開 SLA 中提供的數字。當未知或不可知的情況發生時，這提供了一定程度的降低風險，以滿足 SLA。

最後，您的工作負載可能具有其 SLA 無法知道或不提供 SLA 或 SLO 的相依性。例如，全球網際網路路由是許多工作負載的共同依存性，但很難知道您的全球流量正在使用哪個網際網路服務提供者、它們是否具有 SLA，以及它們在提供商之間的一致性很難。

這一切告訴我們的是，計算最大理論可用性只會產生粗略的數量級計算，但本身可能不準確或提供有意義的見解。數學確實告訴我們的是，您的工作負載所依賴的東西越少會降低整體失敗的可能性。少於一個乘以在一起的數字越少，結果越大。

> **規則 3**
>
> 減少相依性可能會對可用性產生積極影響。

數學還有助於通知依賴關係選擇過程。選取程序會影響您設計工作負載的方式、如何利用相依性中的備援來提高其可用性，以及您是否將這些相依性視為軟體或硬體。應仔細選擇可能對工作負載造成影響的依賴關係。下一條規則提供有關如何執行此操作的指導。

> **規則 4**
>
> 一般而言，請選取可用性目標等於或大於工作負載目標的相依性。

備援的可用性

當工作負載使用多個、獨立且備援的子系統時，與使用單一子系統相比，它可以達到更高層級的理論可用性。例如，考慮由兩個相同子系統組成的工作負載。它可以是完全可操作的，如果無論是子系統一個或子系統二是操作的。要使整個系統關閉，兩個子系統必須同時關閉。

如果一個子系統的故障概率為 1-α，那麼兩個冗餘子系統同時關閉的概率是每個子系統的故障概率的乘積，F = (1− α_1) × (1− α) 。 $_2$對於具有兩個備援子系統的工作負載，使用方程式 (3)，這會提供可用性定義為：

$$A = 1 - F$$
$$F = (1 - \alpha_1) \times (1 - \alpha_2)$$
$$A = 1 - (1 - \alpha)^2$$

方程式 5

因此，對於兩個可用性為 99% 的子系統，其中一個失敗的可能性為 1%，並且它們都失敗的概率為 (1− 99%) × (1− 99%) = .01%。這使得使用兩個備援子系統的可用性達 99.99%。

這可以概括來納入額外的冗餘備件，也是如此。在方程式 (5) 中，我們只假設一個備用，但一個工作負載可能有兩個、三個或更多個備件，以便在多個子系統的同時損失中存活，而不會影響可用性。如果工作負載有三個子系統，而兩個是備用系統，則三個子系統同時失敗的可能性為 (1− α) × (1− α) × (1− α) 或 (1− α) 3。一般而言，只有當 s + 1 個子系統故障時，具有備用的工作負載才會失敗。

對於具有 n 個子系統和 s 備件的工作負載，f 是失敗模式的數量或 s + 1 個子系統可能會故障 n 的方式。

這實際上是二項式定理，即從一組 n 中選擇 k 個元素的組合數學，或者 「n 選擇 k」。在這種情況下，k 是 s + 1。

$$k = s + 1$$

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

$$\binom{n}{s+1} = \frac{n!}{(s+1)!\,\big(n-(s+1)\big)!}$$

$$f = \frac{n!}{(s+1)!\,(n-s-1)!}$$

方程式 6

然後，我們可以產生一個廣義的可用性近似值，其中包含了失敗模式和備用的數量。（要了解為什麼在近似值中這樣做，請參閱海萊曼等的附錄 2。 打破可用性障礙。）

$$s = \ Number\ of\ spares$$
$$\alpha = Availability\ of\ subcomponent$$
$$f = Number\ of\ failure\ modes$$
$$A = 1 - F \approx 1 - f(1-\alpha)^{s+1}$$

方程式 7

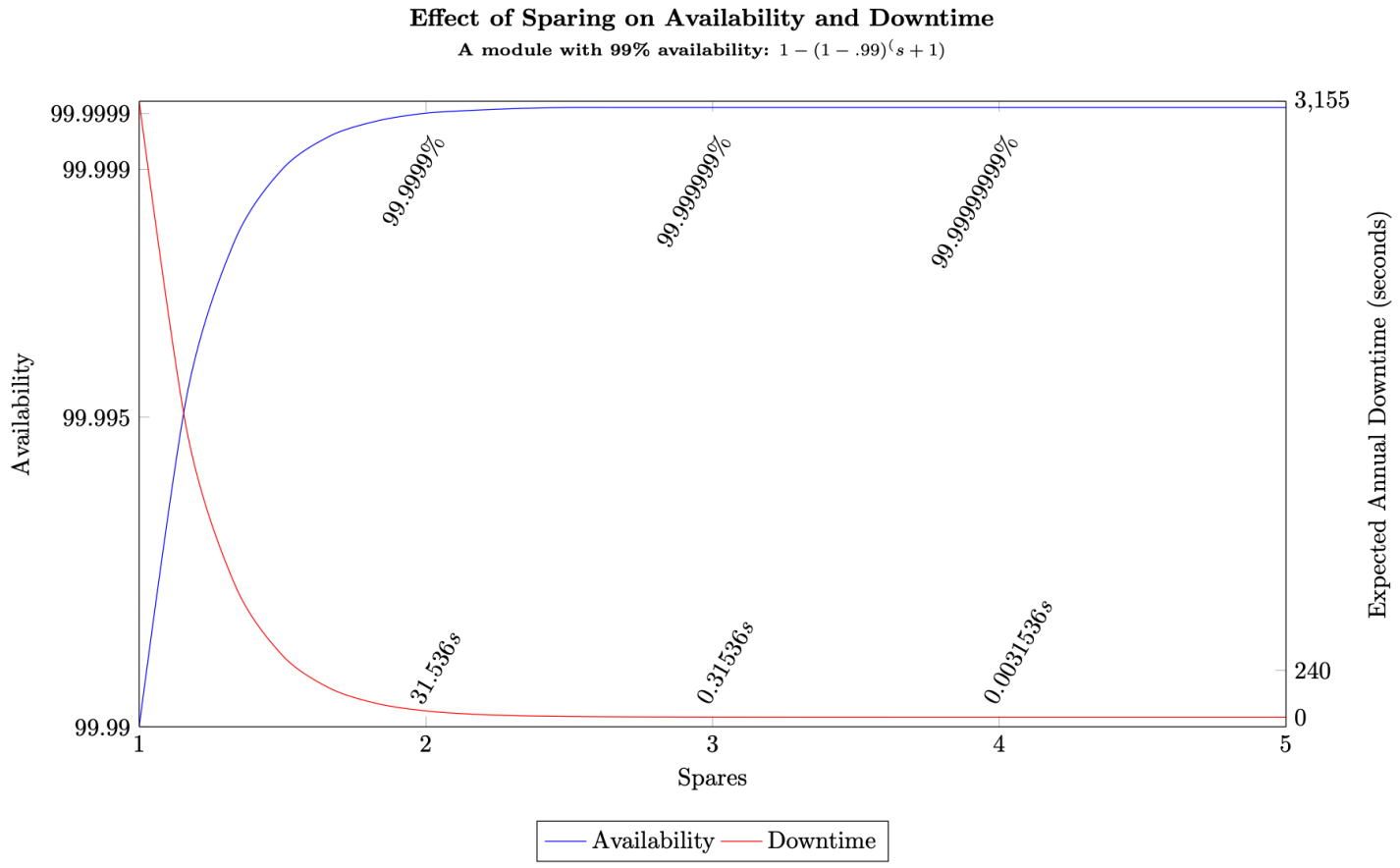
備用可套用至提供獨立失敗資源的任何相依性。不同 AZ 或 Amazon S3 儲存貯體中不同的 Amazon EC2 執行個體就AWS 區域是這個範例。使用備援可協助該相依性達到更高的整體可用性，以支援工作負載的可用性目標。

> **規則五**
>
> 使用備用來增加工作負載中相依性的可用性。

但是，備用是需要付出代價的。每個額外的備用成本與原始模塊相同，至少線性驅動成本。建置可以使用備援的工作負載也會增加其複雜性。它必須知道如何識別相依性失敗、將工作量從它轉移到健康狀態良好的資源，以及管理工作負載的整體容量。

冗餘是一個優化問題。備用裝置太少，而且工作負載的故障頻率可能比預期的要高，備用裝置太多，而且工作負載執行的成本太高。有一個臨界值，即增加更多備件的成本將超過其實現認股權證的額外可用性。

對於具有 99.5% 可用性的子系統，將我們的一般可用性與備用公式 (7) 搭配使用，工作負載的可用性為 A 1 − (1) (1−.995) ^{3} = 99.9999875% (每年停機時間約為 3.94 秒)，而在 10 個備用裝置中，我們會得到一個 1 − (1) (1−995) (1−9 $'$) = 25.5 將是每年 1.26252 × ^{10} 至 15 米，實際上是 0)。在比較這兩個工作負載時，我們的備用成本增加了 5 倍，以實現每年減少 4 秒的停機時間。對於大部分的工作負載而言，成本的增加是不必要的，因為可用性的增加。下圖顯示了這種關係。



備用增加所帶來的回報減少

在三個備件及以後，結果是每年預期停機時間的一秒鐘的幾分之一，這意味著在這一點之後，您將到達收益遞減的區域。可能有衝動「只是添加更多」以實現更高水平的可用性，但實際上，成本效益消失非

常快。當子系統本身至少具有 99% 的可用性時，使用三個以上的備用裝置並不能為幾乎所有工作負載帶來明顯的增益。

> **規則第六條**
>
> 備用的成本效率有一個上限。利用所需的最少備用裝置來達到所需的可用性。

選取正確的備援數目時，您應該考慮失敗單位。例如，讓我們檢查一個需要 10 個 EC2 執行個體來處理尖峰容量的工作負載，並將它們部署在單一 AZ 中。

由於 AZ 設計為故障隔離界限，故障單位不僅是單一 EC2 執行個體，因為整個 AZ 值的 EC2 執行個體可能會一起失敗。在這種情況下，您需要與另一個 AZ 新增冗餘，部署 10 個額外的 EC2 執行個體來處理 AZ 故障時的負載，總共 20 個 EC2 執行個體 (遵循靜態穩定性模式)。

雖然這似乎是 10 個備用 EC2 實例，但它實際上只是一個備用 AZ，因此我們沒有超過收益遞減的點。不過，您可以更具成本效益，同時利用三個 AZ 並在每個 AZ 部署五個 EC2 執行個體來提高可用性。

這為一個備用 AZ 提供總共 15 個 EC2 執行個體 (而兩個 AZ 具有 20 個執行個體)，仍然提供所需的 10 個執行個體總計，以在影響單一 AZ 的事件期間提供尖峰容量。因此，您應該在工作負載 (執行個體、儲存格、AZ 和區域) 使用的所有容錯隔離界限內建備容錯能力。

帽定理

我們可能會考慮可用性的另一種方式是與 CAP 定理有關。該定理指出，分佈式系統，一個由存儲數據的多個節點組成，不能同時提供以下三個保證中的兩個以上：

- C 唯一性：無法保證一致性時，每個讀取請求都會收到最新的寫入或錯誤。
- 有效性：即使節點關閉或無法使用，每個請求都會收到非錯誤響應。
- P 關節容差：即使節點之間遺失任意數量的訊息，系統仍會繼續運作。

（有關更多詳細信息，請參閱塞斯·吉爾伯特和南希·林奇，布魯爾的猜想以及一致，可用的，容忍分區的網絡服務的可行性，ACM SIGACT 新聞，第 33 卷第 2 期（2002），第 51-59 頁。）

大多數分散式系統都必須容忍網路故障，因此必須允許網路磁碟分割。這表示當網路磁碟分割發生時，這些工作負載必須在一致性和可用性之間做出選擇。如果工作負載選擇可用性，則一律會傳回回應，但資料可能不一致。如果選擇一致性，那麼在網絡分區期間，它將返回一個錯誤，因為工作負載無法確定數據的一致性。

對於其目標是提供更高層級的可用性的工作負載，他們可能會選擇可用性和分割區容許度 (AP)，以防止在網路磁碟分割期間傳回錯誤 (無法使用)。這導致需要更寬鬆的一致性模型，例如最終一致性或單調一致性。

容錯和故障隔離

當我們考慮可用性時，這是兩個重要的概念。容錯能力是承受子系統故障並維持可用性的能力（在已建立的 SLA 內做正確的事情）。若要實作容錯，工作負載會使用備用 (或備援) 子系統。當冗餘集合中的其中一個子系統發生故障時，另一個子系統會取得其工作，通常幾乎無縫。在這種情況下，備用零件是真正的備用容量；它們可以承擔故障子系統 100% 的工作。使用真正的備援裝置時，需要多個子系統故障，才能對工作負載產生不利影響。

故障隔離可最大限度地減少發生故障時的影響範圍。這通常是通過模塊化實現的。工作負載會分解成小型子系統，這些子系統會獨立故障，而且可以單獨修復。模塊的故障不會傳播到模塊之外。這個想法橫跨工作負載中的不同功能，以及橫向跨越提供相同功能的多個子系統。這些模組充當錯誤容器，可限制事件期間的影響範圍。

控制平面、資料平面和靜態穩定性的架構模式直接支援實作容錯和容錯隔離。Amazon Builder Library 文章使用可用區域的靜態穩定性為這些術語提供了良好的定義，以及它們如何適用於建置彈性、高可用性的工作負載。本白皮書在 [設計上的高可用性分散式系統] 一節中使用了這些模式AWS，我們也在此總結它們的定義。

- 控制平面 — 進行變更所涉及的工作負載部分：新增資源、刪除資源、修改資源，以及將這些變更傳播到需要的位置。控制平面通常比資料平面更複雜，而且具有更多的移動零件，因此在統計學上更容易發生故障並具有較低的可用性。
- 資料平面 — 提供 day-to-day 業務功能的工作負載部分。數據平面往往比控制面更簡單，並且以更高的體積運行，從而導致更高的可用性。
- 靜態穩定性 — 儘管相依性受損，工作負載仍能繼續正確操作的能力。實現的一種方法是從數據平面中刪除控制平面依賴關係。另一種方法是鬆散耦合工作負載相依性。也許工作負載沒有看到任何更新的信息（例如新事物，刪除的東西或修改的東西），它的依賴項應該已經交付。但是，在依賴關係受損之前它所做的一切都繼續起作用。

當我們考慮工作負載的損害時，我們可以考慮兩種高層次的方法進行恢復。第一種方法是在損害發生後對該損害做出反應，也許使用AWS Auto Scaling來添加新的容量。第二種方法是在這些損壞發生之前做好準備，也許是通過過度佈建工作負載的基礎結構，以便它可以繼續正確運行，而不需要額外的資源。

靜態穩定的系統使用後一種方法。它預先佈建了故障期間可用的備用容量。此方法可避免在工作負載復原路徑中建立控制平面的相依性，以便佈建新容量以從故障中復原。此外，為各種資源佈建新容量需要時間。在等待新容量時，您的工作負載可能會因現有需求而超載，並經歷進一步降級的情況，進而導致「中斷」或完全可用性損失。但是，您也應該考慮將預先佈建的容量用於可用性目標的成本影響。

靜態穩定性為高可用性工作負載提供接下來的兩個規則。

規則第 7 條

請勿依賴資料平面中的控制平面，尤其是在復原期間。

規則第 8 條

鬆散地耦合相依性，因此您的工作負載可以在可能的情況下正確運作，儘管依賴

測量可用性

正如我們之前所看到的，為分散式系統建立前瞻性的可用性模型很難做到，而且可能無法提供所需的見解。可以提供更多實用性的是開發一致的方法來衡量工作負載的可用性。

將可用性定義為正常運行時間和停機時間將失敗表示為二進制選項，無論是工作負載已啟動還是不是。

但是，這種情況很少是這種情況。失敗會有一定程度的影響，而且通常在工作負載的某些子集中經歷，這會影響一定百分比的使用者或要求、位置百分比或延遲百分位數。這些都是部分故障模式。

雖然 MTTR 和 MTBF 對於了解驅動系統產生的可用性的原因很有用，因此，如何改進它，但它們的效用並不是作為可用性的實證衡量標準。此外，工作負載由許多元件組成。例如，像付款處理系統這樣的工作負載由許多應用程式設計介面 (API) 和子系統組成。因此，當我們想要問一個問題時，「整個工作負載的可用性是多少？」，這實際上是一個複雜而細微的問題。

在本節中，我們將研究以實證方式衡量可用性的三種方式：伺服器端要求成功率、用戶端要求成功率和年度停機時間。

伺服器端和用戶端要求成功率

前兩種方法非常相似，只有與測量的角度不同。您可以從服務中的檢測收集伺服器端度量。但是，它們還不完整。如果用戶端無法連線到服務，您就無法收集這些指標。為了瞭解用戶端體驗，而不是仰賴來自用戶端有關失敗要求的遙測，收集用戶端指標的更簡單方法是使用 Canaries 模擬客戶流量，而是定期偵測服務並記錄指標的軟體。

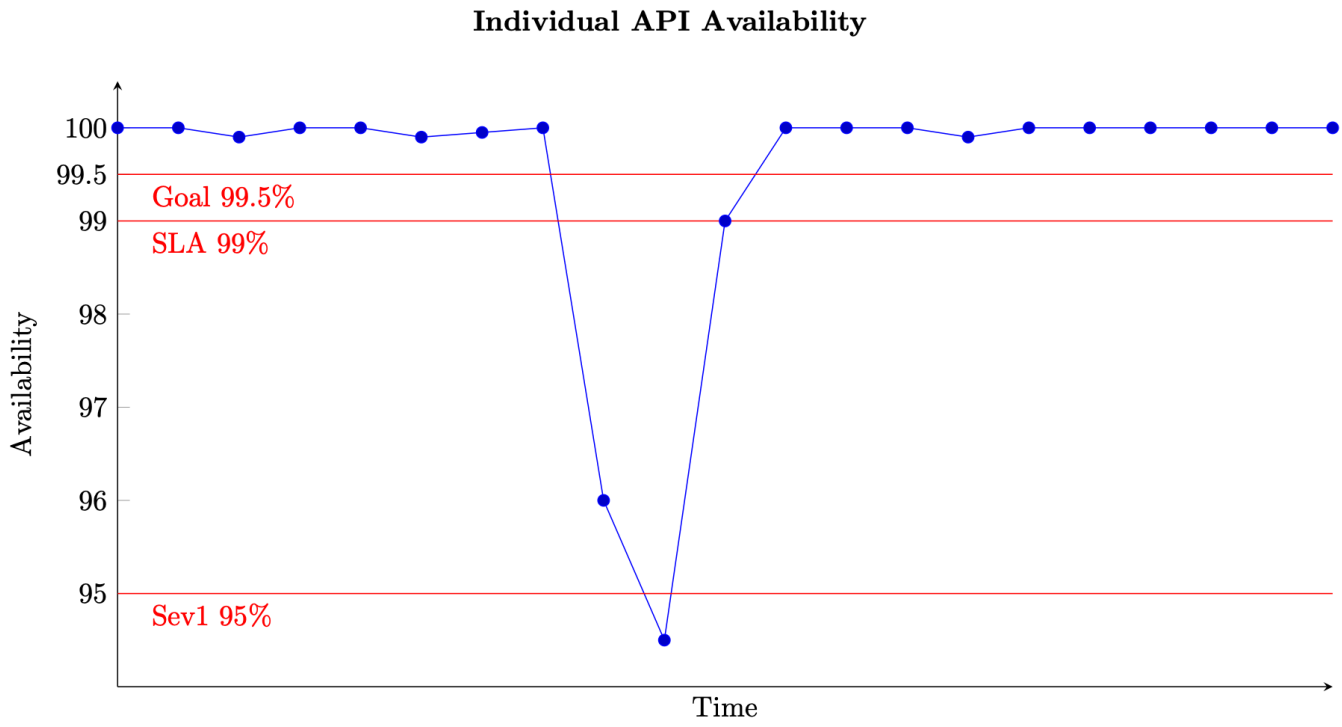
這兩種方法會將可用性計算為服務接收的有效工作單位總計，以及成功處理的工作單位 (這會忽略無效的工作單位，例如導致 404 錯誤的 HTTP 要求)。

$$A = \frac{Successfully\ Processed\ Units\ of\ Work}{Total\ Valid\ Units\ of\ Work\ Received}$$

方程式 8

對於以要求為基礎的服務，工作單位就是要求，就像 HTTP 要求一樣。對於以事件為基礎或以工作為基礎的服務，工作單位是事件或工作，例如從佇列中處理訊息。這種可用性度量在短時間間隔內是有意

義的，例如一分鐘或五分鐘的時間。它也最適合在細微的角度，例如在每個 API 級別為基於請求的服務。下圖提供以這種方式計算時，隨時間推移的可用性可能會是什麼樣子的檢視。圖表上的每個資料點都是在五分鐘時段內使用方程式 (8) 計算的 (您可以選擇其他時間維度，例如一分鐘或十分鐘的間隔)。例如，資料點 10 顯示 94.5% 的可用性。這意味著，如果服務收到 1,000 個請求，則在 T+45 到 +50 的分鐘內，只有 945 個請求被成功處理。



測量單一 API 隨時間變化的可用性範例

此圖表也會顯示 API 的可用性目標、99.5% 可用性、提供給客戶的服務等級協定 (SLA)、99% 的可用性，以及高嚴重性警示的臨界值 95%。如果沒有這些不同臨界值的內容，可用性圖表可能無法提供您服務運作方式的重要見解。

我們也希望能夠追蹤和描述更大型子系統的可用性，例如控制平面或整個服務。其中一種方法是取得每個子系統每五分鐘資料點的平均值。該圖看起來與前一個圖形類似，但將代表一組較大的輸入。它還為構成您的服務的所有子系統提供了相同的權重。另一種方法可能是將從服務中所有 API 接收並成功處理的所有要求加總，以便以五分鐘的間隔計算可用性。

但是，後一種方法可能會隱藏具有低吞吐量和不良可用性的單個 API。作為一個簡單的例子，考慮一個具有兩個 API 的服務。

第一個 API 在五分鐘的時間內收到 1,000,000 個要求，並成功處理其中 999,000 個要求，提供 99.9% 的可用性。第二個 API 會在相同的五分鐘時段內接收 100 個要求，而且只能成功處理其中 50 個要求，提供 50% 的可用性。

如果我們將來自每個 API 的請求加總在一起，則總共有 1,000,100 個有效請求，其中 999,050 個被成功處理，從而為整體服務提供 99.895% 的可用性。但是，如果我們平均兩個 API（前一種方法）的可用性，我們得到 74.95% 的可用性，這可能更多地說明了實際體驗。

這兩種方法都不是錯誤的，但它顯示了了解可用性指標告訴您的重要性。如果您的工作負載在每個子系統上收到類似的請求量，您可能會選擇對所有子系統加總請求。這種方法著重於「請求」及其成功作為可用性和客戶體驗的衡量標準。或者，您也可以選擇平均子系統可用性，以平均代表其重要性，儘管請求數量有所差異。此方法著重於子系統以及每個子系統作為客戶經驗代理的能力。

年度停機時

第三種方法是計算年度停機時間。這種形式的可用性量度比較適合長期目標設定和檢閱。它需要定義停機時間對您的工作負載的意義。然後，您可以根據工作負載不處於「中斷」狀況的分鐘數 (相對於指定期間內的總分鐘數) 來測量可用性。

某些工作負載可能會將停機時間定義為一分鐘或五分鐘的間隔 (發生在先前的可用性圖表中) 的單一 API 或工作負載函數 95% 以下的可用性。您也可能只考慮停機時間，因為它適用於關鍵資料平面作業的子集。例如，適用於 SQS 可用性的 Amazon 簡訊 (SQS、SNS) 服務等級協議適用於 SQS 傳送、接收和刪除 API。

較大、更複雜的工作負載可能需要定義整個系統的可用性指標。對於大型電子商務網站，全系統指標可以類似於客戶訂單率。在這裡，與任何五分鐘期間的預測數量相比，訂單下降 10% 或更多可以定義停機時間。

無論哪種方法，您都可以加總所有中斷期間，以計算年度可用性。例如，如果某個日曆年度有 27 個 5 分鐘的停機時間 (定義為任何資料平面 API 的可用性降至 95% 以下)，則整體停機時間為 135 分鐘 (可能是連續五分鐘的期間，其他時間隔離)，代表年度可用性為 99.97%。

這種額外的測量可用性方法可提供用戶端和伺服器端度量遺失的資料和洞察力。例如，假設受損且錯誤率大幅提升的工作負載。此工作負載的客戶可能會完全停止呼叫其服務。也許他們已經激活了斷路器，或者按照災難恢復計劃在不同地區使用該服務。如果我們只是衡量失敗的響應，那麼在減值期間工作負載的可用性實際上會增加，但不是因為減值改善或消失，而是因為客戶只是停止使用它。

Latency (延遲)

最後，測量工作負載內工作處理單元的延遲也很重要。可用性定義的一部分是在已建立的 SLA 中執行工作。如果傳回回應所花費的時間超過用戶端逾時，則來自用戶端的看法是要求失敗且工作負載不可用。但是，在伺服器端，請求可能似乎已成功處理。

測量延遲提供另一個用於評估可用性的鏡頭。使用百分位數和修剪均值是此測量的良好統計數據。它們通常在第 50 個百分位數（P50 和 TM50）和第 99 個百分位數（P99 和 TM99）進行測量。延遲應使用 Canaries 來衡量，以代表用戶端體驗以及伺服器端指標。每當某些百分位數延遲的平均值（例如 P99 或 TM99.9）高於目標 SLA 時，您可以考慮停機時間，這有助於計算年度停機時間。

設計高可用性的分散式系統 AWS

前面的章節主要是關於工作負載的理論可用性以及它們可以實現的目標。它們是建置分散式系統時要牢記的重要概念集。它們將有助於告知您的依賴關係選擇過程以及如何實現冗餘。

我們也研究了MTTD、MTTR與MTBF可用性之間的關係。本節將根據以前的理論介紹實用指導。簡而言之，高可用性的工程工作負載旨在增加MTBF和減少MTTD. MTTR

儘管消除所有失敗將是理想的，但這並不現實。在具有深層堆疊相依性的大型分散式系統中，將會發生故障。「一切都失敗了所有的時間」(見沃納·沃格爾斯,CTO, Amazon.com, 10 從 Amazon Web Services 多年的經驗教訓。) 和「你不能立法針對失敗 [所以] 專注於快速檢測和響應。」（請參閱 Amazon EC2 團隊創始成員 Chris Pinkham，為故障ARC335設計：在上構建彈性系統） AWS

這意味著您通常無法控制是否發生故障。您可以控制的是檢測故障並對其執行某些操作的速度。因此，雖然增加仍然MTBF是高可用性的重要組成部分，但客戶在其控制範圍內最重要的變化是減少MTTD和MTTR。

主題

- 減少 MTTD
- 減少 MTTR
- 增加 MTBF

減少 MTTD

減少失敗意味著盡快發現故障。MTTD縮短基MTTD於可觀察性，或者您如何檢測工作負載以了解其狀態。客戶應在工作負載的關鍵子系統中監控其「客戶體驗」指標，以便主動識別問題發生時機 (請參閱附錄 1 — 以MTTD及關MTTR鍵指標以取得有關這些指標的詳細資訊)。). 客戶可以使用 Amazon CloudWatch Synthetics 建立監控您APIs和主控台的金絲雀，以主動衡量使用者體驗。還有許多其他健康狀態檢查機制可用於最小化MTTD，例如 E lastic Load Balancing (ELB) 運作狀態檢查、Amazon Route 53 運作狀態檢查等。（請參閱 Amazon Builders' Library-實施運行狀態檢查。）

您的監視也需要能夠偵測整個系統和個別子系統中的部分故障。您的可用性、失敗和延遲指標應使用錯誤隔離界限的維度作為CloudWatch 量度維度。例如，假設單一EC2執行個體屬於儲存格架構的一部分，位於 us-east-1 區域的 use1-az1 AZ 中，該執行個體屬於其控制平面子系統的工作負載更新API的一部分。伺服器推送指標時，可以使用其執行個體 ID、AZ、地區、API名稱和子系統名稱作為維度。這使您可以在每個維度上具有可觀察性並設置警報以檢測故障。

減少 MTTR

在發現故障之後，剩餘的MTTR時間就是實際修復或緩解影響。要修復或減輕故障，您必須知道出了什麼問題。在此階段有兩個關鍵的指標群組可提供深入分析：1/ 影響評估指標和 2/ 作業 Health 度量。第一個群組會告訴您失敗期間的影響範圍，測量受影響的客戶、資源或工作負載的數量或百分比。第二組有助於確定為什麼會產生影響。在發現原因之後，操作員和自動化可以回應並解決故障。如需有關這些測量結果的詳細資訊，請參閱附錄 1 — MTTD 和MTTR重要測量結果。

> **規則第 9 條**
>
> 可觀測性和儀器儀表對於減少MTTD和MTTR.

故障周圍的路由

減輕影響的最快方法是透過快速故障的子系統繞過故障。這種方法會將故障子系統的工作快速轉移至備援，MTTR藉此減少備援。備援範圍包括軟體程序、EC2執行個體、多個區域AZs，以及多個區域。

備用子系統可以減少到MTTR幾乎為零。恢復時間只是將工作重新路由到備用備用所需的時間。這通常以最小的延遲發生，並允許工作在定義的範圍內完成SLA，從而保持系統的可用性。這種產生MTTRs的經歷是輕微的，甚至難以察覺的延遲，而不是長時間的不可用性。

例如，如果您的服務使用 Application Load Balancer (ALB) 後面的EC2執行個體，您可以設定健全狀況檢查的間隔 (最少 5 秒)，而且只需要兩次失敗的健康狀態檢查，才能將目標標示為狀態不良。這表示您可以在 10 秒內偵測到故障並停止將流量傳送到健康狀態不良的主機。在這種情況下MTTR，實際上與相同，MTTD因為一旦檢測到故障，它也會被緩解。

這就是高可用性或持續可用性工作負載試圖實現的目標。我們希望藉由快速偵測失敗的子系統、將其標示為失敗、停止傳送流量至備援子系統，以及將流量傳送至備援子系統，藉此快速繞過工作負載中的故障。

請注意，使用這種快速故障機制會使您的工作負載對暫時性錯誤非常敏感。在提供的範例中，請確定負載平衡器健康狀態檢查只對執行個體執行淺層或生命性以及本機健康狀態檢查，而不是測試相依性或工作流程 (通常稱為深度健康狀態檢查)。這有助於避免在影響工作負載的暫時性錯誤期間不必要的執行個體更換

可觀察性和檢測子系統故障的能力對於成功繞線故障至關重要。您必須知道影響範圍，以便受影響的資源可以標記為狀態不良或失敗並退出服務，以便可以繞過這些資源。例如，如果單一 AZ 遇到部分服務

損害，則您的檢測將需要能夠識別是否存在 AZ 本地化問題，以繞過該 AZ 中的所有資源，直到恢復為止。

能夠繞過故障可能還需要額外的工具，具體取決於環境。使用上一個範例與後面的EC2執行個體一起使用ALB，假設一個 AZ 中的執行個體可能會通過本機運作狀態檢查，但是隔離的 AZ 損害導致它們無法連線至其他 AZ 中的資料庫。在此情況下，負載平衡健康狀態檢查不會將這些執行個體停止服務。需要使用不同的自動化機制，才能從負載平衡器移除 AZ，或強制執行個體的健康狀態檢查失敗，這取決於識別影響範圍是 AZ。對於未使用負載平衡器的工作負載，需要使用類似的方法來防止特定 AZ 中的資源接受工作單位或完全從 AZ 移除容量。

在某些情況下，將工作轉移到冗餘子系統無法自動化，例如主要到次要資料庫的容錯移轉，其中技術不會提供自己的領導者選舉。這是AWS 多區域架構的常見案例。由於這些類型的容錯移轉需要一定程度的停機時間才能完成，無法立即回復，並且將工作負載保留一段時間而不需要冗餘，因此在決策過程中擁有人力是非常重要的。

使用多區域容錯移轉自動化繞過故障的路MTTRs由，可以採用較不嚴格的一致性模型的工作負載縮短。Amazon S3 跨區域複寫或 Amazon DynamoDB 全域表等功能可透過最終一致的複寫提供多區域功能。此外，當我們考慮CAP定理時，使用寬鬆的一致性模型是有益的。在影響到可設定狀態子系統連線的網路故障期間，如果工作負載選擇可用性而不是一致性，它仍然可以提供非錯誤回應，也就是另一種繞過故障路由的方式。

圍繞失敗的繞線可以使用兩種不同的策略來實現。第一個策略是透過預先佈建足夠的資源來處理故障子系統的完整負載，以實作靜態穩定性。這可以是單一EC2執行個體，也可能是整個 AZ 容量。在失敗期間嘗試佈建新資源，會增加復原路徑中控制平面的相依性，MTTR並增加相依性。但是，它需要額外付費。

第二個策略是將某些流量從故障的子系統路由到其他流量，並將剩餘容量無法處理的過量流量負載。在此降級期間，您可以擴展新資源以取代故障的容量。這種方法具有更長的時間，MTTR並在控制平面上產生依賴性，但在待機，備用容量方面的成本更低。

返回已知良好狀態

修復期間緩解的另一種常見方法是將工作負載返回先前已知的良好狀態。如果最近的變更可能導致失敗，則復原該變更是返回先前狀態的一種方法。

在其他情況下，暫時性情況可能導致失敗，在這種情況下，重新啟動工作負載可能會減輕影響。讓我們來看看這兩種情況。

在部署期間，最小化MTTD並MTTR依賴可觀測性和自動化。您的部署程序必須持續觀察工作負載，以提高錯誤率、延遲增加或異常情況。識別這些之後，它應該停止部署程序。

有各種各樣的部署策略，例如就地部署、藍/綠部署和滾動式部署。其中每一個都可能使用不同的機制來返回已知的良好狀態。它可以自動回復到先前的狀態，將流量轉移回藍色環境，或者需要手動介入。

CloudFormation 提供了作為其創建和更新堆棧操作的一部分自動復原的功能，就像一樣AWS CodeDeploy。 CodeDeploy 也支援藍/綠和滾動式部署。

若要充分利用這些功能並將您的功能降到最低MTTR，請考慮透過這些服務自動化所有基礎結構和程式碼部署。在無法使用這些服務的案例中，請考慮使用實作 saga 模式 AWS Step Functions 以復原失敗的部署。

考慮重新啟動時，有幾種不同的方法。這些範圍從重新啟動服務器，最長的任務，重新啟動線程，最短的任務。這是一個表格，概述了一些重新啟動方法和要完成的近似時間（代表數量級差，這些差異並不精確）。

故障恢復機制	估計 MTTR
啟動並設定新的虛擬伺服器	15 分鐘
重新部署軟體	10 分鐘
重啟伺服器	5 分鐘
重新啟動或啟動容器	2 秒
叫用新的無伺服器功能	一百毫秒
重新啟動程	10 毫秒
重啟執行緒	10 微秒

查看表格，使用容器和無伺服器功能（如 AWS Lambda）有一些明顯的好處。MTTR它們MTTR的速度比重新啟動虛擬機器或啟動新的虛擬機器快。不過，透過軟體模組化使用故障隔離也是有益的。如果您可能導致單一處理序或執行緒失敗，則從該失敗中復原會比重新啟動容器或伺服器快得多。

作為恢復的一般方法，您可以從下移動到頂部：1/ 重新啟動，2/ 重新啟動，3/ 重新映像/重新部署，4 / 替換。但是，一旦進入重新啟動步驟，繞過失敗通常是一種更快的方法（通常最多需要 3—4 分鐘）。因此，為了最快地減輕嘗試重新啟動後的影響，請繞過故障，然後在背景繼續復原程序，以將容量返回至您的工作負載。

> **規則第十條**
>
> 專注於緩解影響，而不是解決問題。以最快的路徑回到正常操作。

故障診斷

檢測後修復過程的一部分是診斷期。這是運營商試圖確定什麼是錯誤的時間段。此程序可能涉及查詢記錄檔、檢閱作業 Health 狀態指標，或登入主機以進行疑難排解。所有這些動作都需要時間，因此建立工具和 Runbook 來加速這些動作也有助於減少這些動MTTR作。

手冊和自動化

同樣地，在您判斷錯誤的原因以及要修復工作負載的動作方式之後，操作員通常需要執行一組步驟來執行此操作。例如，在失敗之後，修復工作負載的最快方法可能是重新啟動工作負載，這可能涉及多個有序的步驟。使用可以自動執行這些步驟或向操作員提供特定方向的 runbook 將加快流程並有助於降低無意採取行動的風險。

增加 MTBF

提高可用性的最後一個元件正在增加MTBF. 這可以同時適用於軟件以及用於運行它的 AWS 服務。

增加分散式系統 MTBF

增加的一種方法MTBF是減少軟件中的缺陷。有幾種方式可以執行此作業。客戶可以使用 Amazon CodeGuru 審核者等工具來尋找並修復常見錯誤。您也應該在軟體部署到生產環境之前，對軟體執行全面的對等程式碼檢閱、單元測試、整合測試、迴歸測試和負載測試。增加測試中的代碼覆蓋率將有助於確保即使是不常見的代碼執行路徑也會被測試。

部署較小的變更也可降低變更的複雜性，協助防止意外結果。每個活動都提供了一個機會，以識別和修復缺陷之前，它們可以被調用。

防止故障的另一種方法是定期測試。實作混沌工程計畫可協助測試工作負載失敗的情況、驗證復原程序，以及協助在生產環境中發生之前尋找和修正失敗模式。客戶可以使用AWS 故障注入模擬器作為其混亂工程實驗工具集的一部分。

容錯能力是防止分散式系統故障的另一種方法。快速故障模組、具有指數輪詢和抖動的重試、交易和冪等都是協助讓工作負載容錯的技術。

交易是一組遵守ACID屬性的操作。如下所示：

- 原子性 — 無論是所有的行動發生或沒有一個會發生.
- 一致性 — 每個交易都會使工作負載保持有效狀態。
- 隔離 — 同時執行的交易會使工作負載保持與按順序執行相同的狀態。
- 耐久性 — 一旦交易認可，即使在工作負載失敗的情況下，也會保留其所有效果。

透過指數輪詢和抖動進行重試，可讓您克服由 Heisenbugs、超載或其他情況造成的暫時性故障。當事務是冪等的時候，它們可以重試多次而不會產生副作用。

如果我們考慮一個 Heisenbug 對容錯硬件配置的影響，我們會相當不關心，因為 Hisenbug 出現在主要和冗餘子系統上的概率是無限小的。（見吉姆·格雷，「為什麼計算機停止，可以做些什麼？「，一九八五年六月，串聯技術報告 85.7。）在分佈式系統中，我們希望使用我們的軟件實現相同的結果。

當調用 Hisenbug 時，軟件必須快速檢測到不正確的操作並失敗，以便再次嘗試。這是通過防禦性編程來實現的，並驗證輸入，中間結果和輸出。此外，處理序會被隔離，並且不會與其他程序共用任何狀態。

這種模組化方法可確保故障期間的影響範圍受到限制。程序獨立失敗。當一個進程確實失敗時，軟件應該使用「進程對」來重試工作，這意味著一個新的進程可以承擔一個失敗的工作。為了維護工作負載的可靠性和完整性，每個操作都應被視為一個ACID事務。

這可讓處理程序失敗，而不會中斷交易並復原所做的任何變更，而不會損毀工作負載的狀態。這可讓復原程序從已知良好狀態重試交易，並正常重新啟動。這就是軟件對海森錯誤的方式。

但是，您不應該旨在使軟件對 Bohrbug 進行容錯。在工作負載進入生產環境之前，必須先找到並移除這些瑕疵，因為沒有任何等級的備援能夠達到正確的結果（見吉姆·格雷，「為什麼計算機停止，可以做些什麼？「，一九八五年六月，串聯技術報告 85.7。）

增加的最後一種方法MTBF是減少故障造成的影響範圍。透過模組化使用故障隔離來建立容錯容器，是先前在容錯和容錯隔離中所述的主要方式。降低故障率可提高可用性。 AWS 使用諸如將服務劃分為控制面和數據平面，可用區域獨立性（AZI），區域隔離，基於細胞的架構和洗牌分片等技術來提供故障隔離。這些也是 AWS 客戶也可以使用的模式。

例如，讓我們回顧一個案例，即工作負載將客戶置於其基礎架構的不同容器中，該容器最多為總客戶總數的 5% 提供服務。其中一個故障容器遇到的事件會增加延遲超過 10% 的要求的用戶端逾時。在此活動期間，對於 95% 的客戶，該服務 100% 可用。對於其他 5%，該服務似乎是 90% 可用。這會導致 1 的可用性-(5% 或 F C U s o 米電子 s × 10% 或 F t 我的 R 電子問題 u s s) = 99.5% 而不是 10% 的請求失敗 100% 的客戶 (導致 90% 的可用性)。

規則十一

故障隔離可降低整體故障率，從而減少影響範圍並增加工作負載。MTBF

增加相依性 MTBF

增加 AWS 依賴關係的第一種方法MTBF是通過使用故障隔離。許多 AWS 服務在 AZ 提供一定程度的隔離，這表示一個 AZ 中的故障不會影響不同 AZ 中的服務。

在多個中使用冗餘EC2執行個體可AZs提高子系統可 AZI在單一區域內提供備用功能，可讓您提高AZI服務的可用性。

不過，並非所有 AWS 服務都在 AZ 層級運作。許多其他人提供區域隔離。在這種情況下，如果區域服務的可用性設計不支援工作負載所需的整體可用性，您可以考慮採用多區域方法。每個區域都提供服務的獨立實例化，相當於備用。

有各種服務可以幫助您更輕鬆地構建多區域服務。例如：

- Amazon Aurora 全球數據
- Amazon DynamoDB 全球表
- Amazon ElastiCache (RedisOSS) — 全球資料存放區
- AWS 全球加速器
- Amazon S3 跨區域複寫
- Amazon 路線 53 應用程序恢復控

本文件並未深入探討建置多區域工作負載的策略，但您應該權衡多區域架構的可用性優勢，以及符合您所需的可用性目標所需的額外成本、複雜性和作業實務。

下一個增加依賴性的方法MTBF是將工作負載設計為靜態穩定。例如，您有一個提供產品資訊的工作負載。當您的客戶提出產品要求時，您的服務會向外部中繼資料服務提出要求，以擷取產品詳細資訊。然後，您的工作負載會將所有這些資訊傳回給使用者。

不過，如果無法使用中繼資料服務，您的客戶提出的要求就會失敗。相反地，您可以在本機以非同步方式將中繼資料提取或推送至服務，以便用來回應要求。這樣可以消除關鍵路徑對中繼資料服務的同步呼叫。

此外，由於即使沒有中繼資料服務，您的服務仍然可以使用，因此您可以在可用性計算中將其作為相依性移除。這個範例取決於中繼資料不會頻繁變更，而且提供過時的中繼資料比要求失敗更好的假設。另

一個類似的例子是服務過時DNS，它允許數據保存在緩存中超過TTL到期，並在刷新的答案不容易獲得時用於響應。

增加依賴性的最後一種方法MTBF是減少失敗造成的影響範圍。如前所述，失敗不是二進制事件，有失敗程度。這是模塊化的效果; 失敗僅包含由該容器服務的請求或用戶。

這會導致事件發生的失敗次數減少，最終藉由限制影響範圍來提高整體工作負載的可用性。

減少常見的影響來源

在 1985, 吉姆·格雷發現, 在串聯計算機的研究, 失敗主要由兩件事驅動:軟件和操作. （見吉姆·格雷，「為什麼計算機停止，可以做些什麼？「，一九八五年六月，串聯技術報告 85.7。）即使在 36 年後，這仍然是真實的。儘管技術有進步，但這些問題並不是一個簡單的解決方案，並且主要的故障來源並沒有改變。在本節的開頭討論了解決軟件故障的問題，所以這裡的重點將是操作和減少故障的頻率。

穩定性與功能相比

如果我們回到本節中的軟件和硬件圖表中的故障率the section called “分散式系統可用”，我們可以注意到每個軟件版本中都添加了缺陷。這表示工作負載的任何變更都會增加失敗的風險。這些變化通常是像新功能一樣的東西，它提供了必要的結果。較高的可用性工作負載比新功能更有利於穩定性 因此，提高可用性的最簡單方法之一就是減少部署頻率或提供較少的功能。部署頻率較高的工作負載本質上會比沒有的工作負載具有較低的可用性。但是，無法新增功能的工作負載無法跟上客戶需求，而且隨著時間的推移可能會變得不那麼有用。

那麼，我們如何繼續安全地創新和發布功能？ 答案是標準化。什麼是正確的部署方式？ 您如何訂購部署？ 測試的標準是什麼？ 您在階段之間等待多長時間？ 您的單元測試涵蓋了足夠的軟件代碼嗎？ 這些是標準化將回答並防止由於不加載測試，跳過部署階段或部署太快到太多主機等問題引起的問題。

您實作標準化的方式是透過自動化。它減少了人為錯誤的機會，並讓計算機做他們擅長的事情，這是做同樣的事情，每次一遍又一遍地做同樣的事情。您將標準化和自動化結合在一起的方式是設定目標。目標像沒有手動更改，只能通過臨時授權系統訪問主機，為每個編寫負載測試API等。卓越運營是一種文化規範，可能需要重大的改變。根據目標建立和追蹤效能，有助於推動文化變革，從而對工作負載可用性產生廣泛影響。AWS Well-Architected 的卓越營運支柱提供了全面的最佳實踐，以實現卓越的營運。

操作員安全

引入失敗的操作事件的其他主要貢獻者是人。人類會犯錯誤。他們可能會使用錯誤的認證，輸入錯誤的命令，過早按 Enter 鍵，或錯過關鍵步驟。採取手動操作會持續導致錯誤，從而導致失敗。

導致操作員錯誤的主要原因之一是混淆，不直觀或不一致的用戶界面。Jim Gray 在 1985 年的研究中還指出：「要求操作員提供信息或要求他執行某些功能的接口必須簡單，一致且操作員容錯。」（見吉姆·格雷，「為什麼計算機停止，可以做些什麼？」，一九八五年六月，串聯技術報告 85.7。）這種見解今天仍然是真實的。在過去三十年中，整個行業中有許多例子，其中令人困惑或複雜的用戶界面，缺乏確認或說明，甚至只是不友好的人類語言導致操作員做錯事。

> 規則十二
>
> 讓運營商輕鬆做正確的事情。

防止過載

影響的最終常見因素是您的客戶，也就是工作負載的實際使用者。成功的工作負載往往會被大量使用，但有時使用量超過了工作負載的擴展能力。可能會發生許多事情，磁碟可能已滿、執行緒集區可能會耗盡、網路頻寬可能已飽和，或是可以達到資料庫連線限制。

沒有故障防護方法可以消除這些問題，但是透過「作業 Health 全狀況」指標主動監控容量和使用率，可在發生這些失敗時提供早期警告。諸如負載脫落、斷路器以及具有指數輪詢和抖動的重試等技術可以幫助減少影響並提高成功率，但這些情況仍然代表失敗。根據作業 Health 狀態指標自動調整可協助減少因超載而導致的故障頻率，但可能無法快速回應使用率變更。

如果您需要確保客戶持續可用的容量，則必須取捨可用性和成本。確保容量不足不會導致無法使用的一種方法是為每位客戶提供配額，並確保您的工作負載容量可擴展，以提供 100% 的配額。當客戶超出其配額時，他們會受到限制，這不是失敗，也不會計入可用性。您還需要密切追蹤客戶群並預測 future 的使用率，以保持佈建足夠的容量。這可確保您的工作負載不會因為客戶過度使用而導致故障情況。

- Amazon Builders' Library-使用負載脫落以避免過載
- Amazon Builders' Library — 多租戶系統的公平性

例如，讓我們檢查提供儲存服務的工作負載。工作負載中的每個伺服器每秒可支援 100 次下載，為客戶提供配額或每秒 200 次下載，並有 500 個客戶。為了能夠支援此數量的客戶，此服務需要提供每秒 100,000 次下載的容量，因此需要 1,000 部伺服器。如果有任何客戶超出其配額，則會受到限制，以確保每個其他客戶都有足夠的容量。這是一個簡單的例子，說明了避免過載而不拒絕工作單位的一種方法。

結論

我們在本文件中建立了 12 個高可用性規則。

- 規則 1 — 較少的故障頻率 (較長的 MTBF)、更短的故障偵測時間 (縮短 MTTD)，以及更短的修復時間 (縮短 MTTR) 是用來改善分散式系統可用性的三個因素。
- 規則 2 — 工作負載中的軟體可用性是工作負載整體可用性的重要因素，而且應與其他元件相同。
- 規則 3 — 減少相依性可能會對可用性產生積極影響。
- 規則 4 — 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。
- 規則 5 — 使用備用來增加工作負載中相依性的可用性。
- 規則 6 — 備用的成本效率有一個上限。利用所需的最少備用裝置來達到所需的可用性。
- 規則 7 — 請勿依賴資料平面中的控制平面，尤其是在復原期間。
- 規則 8 — 鬆散耦合依賴關係，以便儘管依賴性受損，您的工作負載仍可以正確運行，盡可能。
- 規則 9 — 觀察性和儀器儀表對於減少 MTTD 和 MTTR 至關重要。
- 規則 10 — 專注於緩解影響，而不是解決問題。以最快的路徑回到正常操作。
- 規則 11 — 故障隔離可減少影響範圍，並透過降低整體故障率來增加工作負載的 MTBF。
- 規則 12 — 讓操作員輕鬆做正確的事情。

透過減少 MTTD 和 MTTR，以及增加 MTBF 來改善工作負載可用性。總之，我們討論了以下方法，以提高涵蓋技術、人員和流程的可用性。

- MTTD
  - 透過主動監控您的客戶體驗指標來減少 MTTD。
  - 利用精細的運作狀態檢查，快速進行容錯移轉。
- MTTR
  - 監視影響範圍和作業健康狀態指標。
  - 按照 1 /重新啟動、2/ 重新開機、3 /重新映像/重新部署和 4 /取代以減少 MTTR。
  - 通過了解影響範圍繞失敗的路由。
  - 透過虛擬機器或實體主機，利用重新啟動時間較快的服務，例如容器和無伺服器功能。
  - 盡可能自動復原失敗的部署。
  - 建立診斷操作和重新啟動程序的手冊和操作工具。
- MTBF

- 通過嚴格的測試，在軟件發布到生產之前消除錯誤和缺陷。
- 實施混亂工程和故障注入。
- 利用相依性中的適當數量備用來容忍失敗。
- 透過故障容器將故障期間的影響範圍降至最低。
- 實作部署和變更的標準。
- 設計簡單、直觀、一致且記錄良好的操作員介面。
- 設定卓越營運目標。
- 當可用性是工作負載的關鍵維度時，有利於發行新功能的穩定性。
- 透過節流或負載脫落或兩者實作使用配額，以避免過載。

請記住，我們永遠不會完全成功地防止失敗。專注於具有最佳故障隔離的軟體設計，這些隔離會限制影響範圍和大小，最理想地將影響保持在「停機時間」閾值以下，並投資於非常快速、非常可靠的偵測和緩解措施。現代分散式系統仍然需要將故障視為不可避免的情況，並在各個層面進行設計，以實現高可用

附錄 1 — MTTD 和 MTTR 的關鍵指標

以下是儀器和可觀察性的標準化框架，可以幫助減少事件期間的 MTTD 和 MTTR。

客戶體驗指標。這些指標反映了服務具有回應能力，可用來滿足客戶要求。例如，控制平面延遲。這些指標會測量錯誤率、可用性、延遲、數量和節流率。

影響評估指標。這些指標可讓您深入瞭解事件期間的影響範圍。例如，受資料平面事件影響的客戶數量或百分比。測量受影響事物的數量或百分比。

操作健康指標。這些指標反映了服務具有回應能力，可用來滿足客戶要求，但著重於一般基礎架構子系統和資源。例如，EC2 叢集的 CPU 使用率百分比。這些指標應衡量使用率、容量、輸送量、錯誤率、可用性和延遲。

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

深入閱讀

如需其他資訊，請參閱：

- 架構良好的可靠性支柱
- 架構良好的卓越營運支柱
- 亞馬遜建設者庫 — 確保部署期間的回滾安全
- 亞馬遜建造者圖書館 — 超越五個 9s：我們最高可用數據平面的經驗教訓
- Amazon 建置者程式庫 — 自動執行安全、免提部署
- Amazon 建置者程式庫 — 大規模架構和操作彈性無伺服器系統
- 亞馬遜建造者圖書館 — 亞馬遜的高可用性部署方法
- 亞馬遜建造者圖書館-亞馬遜建立彈性服務的方法
- 亞馬遜建造者圖書館-亞馬遜成功失敗的方法
- AWS建築中心

文件歷史紀錄

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
初始出版	白皮書首次出版。	2021 年 11 月 12 日

> **Note**
>
> 若要訂閱 RSS 更新，您必須啟用所使用瀏覽器的 RSS 外掛程式。

注意

客戶有責任對本文件中的資訊進行自行獨立評估。本文件：(a) 僅供參考，(b) 代表目前的AWS產品供應項目和做法，如有變更，恕不另行通知，且 (c) 不會向其關聯公司、供應商或授權人建立任何承諾或保證。AWS 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。客戶的責任和責任由AWS協議控制，本文件不屬於與客戶之間AWS的任何協議的一部分，也不會修改。AWS

© 2021 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。