在 上使用混沌工程來提高彈性並改善客戶體驗 AWS

AWS 方案指引

AWS 方案指引: 在 上使用混沌工程來提高彈性並改善客戶體驗 AWS

Table of Contents

在 上使用混沌工程來提高彈性並改善客戶體驗 AWS

Laurent Domb，Amazon Web Services 聯邦財務首席技術專家

2025 年 4 月 (文件歷史記錄)

混沌工程是對應用程式進行實驗的紀律，以建立對組織和應用程式在生產環境中承受亂流條件的能力的信心。這是一種主動的恢復能力方法，目標是透過跨人員、程序和技術引入受控制的故障，來驗證您的應用程式和組織是否能夠吸收、適應和最終從服務損害中復原。其目的是在可能導致生產中斷或其他中斷之前，識別並消除弱點。

在 Amazon，我們了解分散式系統中的故障是不可避免的，此時即使發生故障仍是正常的操作模式。由於服務之間的互動一定會失敗，因此您需要了解您的服務在各種故障模式期間如何反應，並建置可應對關鍵漏洞的服務，例如相依性故障、重試風暴、可用區域受損和主機資源耗盡。

讓我們以重試風暴為例。用戶端中的當地語系化故障可能會對多個 服務產生重大影響。這通常稱為平滑效果。重試風暴是滑蟲效果的資訊清單，其中失敗的相依性會觸發用戶端和這些用戶端的用戶端重試失敗的操作，導致流量呈指數增長。服務變得過載，因為除了重試流量之外，還必須回應一般流量，同時處理效能降低。

混沌工程已成為對分散式系統日益複雜度的回應。這是一種多學科方法，結合了混沌理論、系統思維和工程的原則，以設計和管理可應對意外事件和行為的複雜系統。混沌工程的核心在於了解和管理複雜系統在不確定性和不可預測性條件下的行為。它認識到，依賴預測和控制結果的傳統工程方法，通常不足以處理分散式系統的複雜和動態本質。隨著這些系統的成長，它們通常超過任何單一個人的理解範圍。

混沌工程提供概念、技術和工具，刻意將故障注入系統，以便在生產中出現弱點之前發現。這種主動方法可讓組織建立信心，讓他們的系統在壓力條件下執行。雖然混沌工程仍是一項不斷發展的實務，但它代表了設計、管理和操作現代運算系統的基本轉變，以便在面對日益增加的複雜性和互連性時保持彈性。

本指南的下列各節討論混沌工程的優點、說明如何進行混沌工程實驗，以及說明在組織中大規模實作混沌工程的方法。還包括範例實驗規劃和實驗結果文件，您可以用作混沌工程實驗的範本。

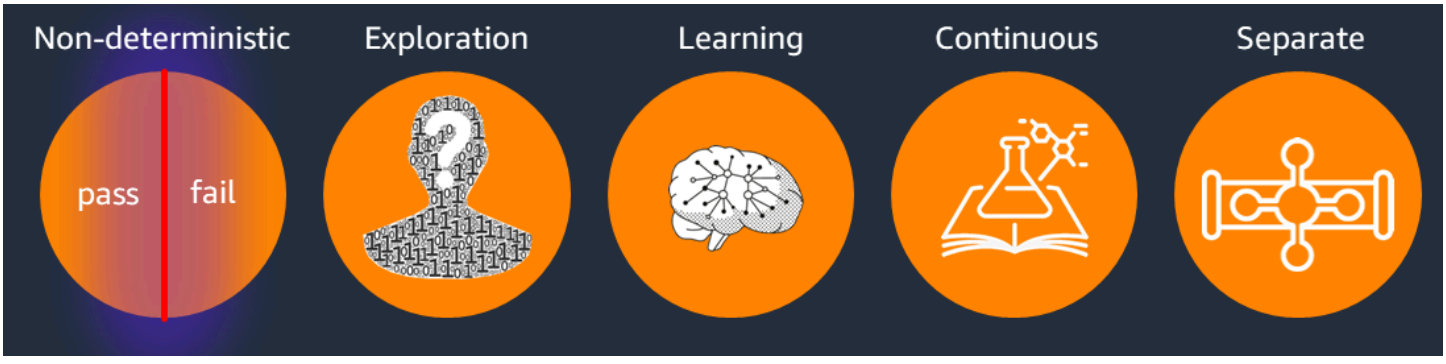
下一節探討混沌工程的特性與單元、煙霧或整合測試等傳統彈性測試有何不同。

概觀

比較彈性測試與混沌工程

彈性測試是確定性的。也就是說，它會驗證已在應用程式中實作的復原機制的已知特性，例如斷路器、重試、容錯移轉或備用。它確認了這些應用程式元件如何吸收受控中斷，並且對使用者的影響最小或無影響。因此，彈性測試著重於驗證注入應用程式元件的已知失敗模式，以產生通過/失敗結果。您應該在管道中持續執行彈性測試，以確保不會將迴歸引入彈性狀態。在彈性測試中，您通常不會對實際元件執行測試，但模擬特定元件APIs。這種方法允許在受控環境中對故障案例進行一致、可重現的測試，使其適用於自動化管道整合和迴歸測試。

相反地，混沌工程是非確定性的。也就是說，它以假設為基礎，並驗證您的心理模型，了解應用程式及其相依性（人員、程序和技術）如何吸收、適應意外故障模式，最終從中復原。因此，混沌工程著重於未知失敗模式的end-to-end驗證，目標是及早發現瑕疵，並在這些瑕疵變成大規模事件之前進行修復。混沌工程促進持續學習，應透過單獨的混沌管道或臨機操作實驗進行練習，讓您能夠隨時執行多個實驗，而不會阻礙開發人員在部署程式碼時的生產力。



混沌工程程序通常從混沌的遊戲日開始，這是一個專用事件，讓團隊刻意將受控的故障或故障注入其應用程式。遊戲日是漸進式的：它從較低層級的環境（例如開發或測試）開始，隨著可信度建置逐漸進展到更高層級的環境（例如預備和生產前）。透過系統地在這些環境中移動，團隊可以在到達生產環境之前，驗證其系統是否能夠正確容忍注入的故障。這種有條不紊的進展可確保在生產環境中執行混沌

實驗時，團隊對其系統的恢復能力建立了很大的信心。遊戲日程序是一種主動方法，用於識別應用程式架構和操作實務中的弱點和漏洞，同時消除意外生產中斷期間的學習壓力。

混沌工程的值

複雜系統在現今的世界中是無處不在的。從金融市場到醫療保健，它們在我們生活的許多層面都扮演重要角色。我們預期這些系統永遠可以運作。不過，複雜系統通常容易受到可能造成重大後果的意外事件和行為所影響。組織需要規劃中斷，而不是想知道是否會發生。他們可以將案例測試套用到其關鍵或任務關鍵業務服務。這是混沌工程發揮作用的地方。

Chaos 工程提供一種方法來管理複雜的系統，以協助降低風險並改善彈性。準備混沌實驗的過程需要團隊制定有關其系統行為的假設，這可以加深他們對系統建置方式及其操作方式的了解。此準備通常會顯示可能未發現的心理差距、架構洞見和操作知識。透過進一步了解複雜系統如何對故障做出反應，混沌工程可提升系統設計和管理的透明度和責任。組織練習混沌工程的頻率越高，操作的準備就越好。混沌工程可協助您建立最佳實務，以設計可承受元件故障的彈性應用程式，且不會影響使用者。這可確保關鍵應用程式在預期的服務水準和影響容錯範圍內運作，同時持續增強團隊對其系統的知識。

為不利條件做好準備

當您建置時 AWS，會使用不同類型的服務，包括區域服務，例如 Amazon Elastic Compute Cloud (Amazon EC2)、區域服務，例如 Amazon Simple Storage Service (Amazon S3)、全域服務，例如 AWS Identity and Access Management (IAM)、第三方軟體即服務 (SaaS) 服務，以及內部部署服務。每種類型的服務都會公開您需要考慮的不同故障網域。如何準備自我引發的事件，或組織無法控制的第三方所造成的事件？

為了協助了解應用程式可能如何回應不良情況，您可以使用 AWS Fault Injection Service (AWS FIS). AWS FIS 是一種全受管服務，以受控方式執行故障注入實驗。您可以使用此服務注入 AWS提供的案例，例如可用區域電源中斷和跨區域連線問題，或透過將服務提供的各種故障動作鏈結在一起來建置您自己的實驗。 AWS FIS 可讓您的團隊持續練習並了解其應用程式如何在偵測到常見故障時做出反應並修復缺陷。

練習控制混沌工程

受控混沌實驗的主要原則如下：

- 從類似生產環境的環境開始。
- 為您的實驗建立假設和停止條件。

- 從小開始。
- 對混沌實驗進行控制。
- 設定影響範圍。
- 了解您的服務基準。
- 排程實驗。
- 先修復，然後實驗。
- 密切監控實驗。
- 從結果中學習。
- 排定問題清單的優先順序、修復和驗證。
- 將學習內容傳播到整個組織。

若要成功擴展混沌工程，您必須以控制的方式實作混沌實驗。使用 時 AWS FIS，您可以使用 Amazon CloudWatch 警示建立停止條件。您可以將這些條件合併到實驗範本中，以確保實驗在超出範圍時停止，並復原回其最後已知狀態。 AWS FIS 也提供安全控制桿。當您使用這些控制桿時， 會 AWS FIS 停止並復原 帳戶中所有執行中的實驗 AWS 區域，包括多帳戶實驗，並防止新的實驗啟動。這可防止在特定期間發生故障注入，例如交易時間、銷售事件或產品啟動，或回應應用程式運作狀態警示。安全控制桿會保持接合狀態，直到手動分離為止。

當您進行混沌實驗時，您應該定義保護措施，以防止環境中的不良副作用，特別是如果實驗可能會影響生產中的應用程式時。當您規劃實驗時，請預測它可能對環境中其他應用程式造成的任何負面影響。例如，其他應用程式可能會從屬於實驗一部分的應用程式接收錯誤的訊息、遇到高請求量，或者如果他們共用基礎設施，可能會遇到資源爭用。在執行實驗之前，請記錄這些風險並解決任何已知或不可接受的問題。

混沌工程入門

在進行實驗之前，我們建議您備妥一些基本知識，以充分利用您的混沌工程實務。這些基本概念包括：

- 可觀測性（指標、記錄、請求追蹤）
- 您想要探索的實際事件或故障清單
- 透過領導層接受的組織彈性贊助
- 相較於執行混沌實驗時發現的新功能，根據潛在業務影響來排定重要調查結果的優先順序

混沌實驗的可觀測性

可觀測性包含指標、記錄和請求追蹤，在混沌工程中扮演重要角色。當您執行實驗時，您會想要了解業務指標、伺服器端指標、用戶端體驗指標和操作指標。如果沒有可觀測性，您將無法定義穩定狀態行為或建立有意義的實驗，以驗證您對應用程式的假設是否成立。

指標

下圖顯示您可以針對不同類型的應用程式追蹤混沌實驗的指標類型。

- 業務指標 – 穩定狀態表示系統的正常操作，並由您的業務指標定義。它可以由每秒交易 (TPS)、每秒點擊串流、每秒訂單或類似的測量來表示。您的應用程式在如預期般運作時會顯示穩定狀態。因此，在執行實驗之前，請確認您的應用程式運作狀態良好。穩定狀態不一定表示發生故障時不會影響應用

程式，因為一定百分比的故障可能落在可接受的限制內。穩定狀態是您的基準。例如，付款系統的穩定狀態可能定義為處理 300 TPS，成功率為 99%，往返時間為 500 毫秒。從視覺上來說，請將穩定狀態視為心電圖 (EKG)。如果系統的穩定狀態突然波動，您知道您的服務有問題。

- 伺服器端指標 – 若要了解資源在實驗期間的效能，您需要在實驗之前、期間和之後深入了解其效能。若要測量資源對 的影響 AWS，您可以使用 Amazon CloudWatch。CloudWatch 是一項服務，可監控應用程式、回應效能變更、最佳化資源使用，以及提供營運運作狀態的洞見。在實驗期間，您會想要擷取伺服器端指標，例如飽和度、請求磁碟區、錯誤率和延遲。
- 客戶體驗指標 – 在 上 AWS，您可以使用 CloudWatch RUM 透過 Locust、Grafana k6、Senlenium 或 Puppeteer 等工具來模擬使用者請求，以擷取真正的使用者指標。實際使用者指標對於執行混沌工程實驗的組織至關重要。透過監控實際使用者在實驗期間受到的影響，團隊可以準確了解故障和中斷如何影響生產中的客戶。主要用戶端體驗指標為首次位元組時間 (TTFB)、最大有內容的顏料 (LCP)、下一個顏料的互動 (INP) 和總封鎖時間 (TBT)。
- 操作指標 – 介入會測量您以自動化方式成功緩解錯誤的程度，例如，在重新啟動 Pod、任務或 EC2 執行個體時，透過複寫控制器或自動擴展等機制，成功降低用戶端請求延遲。能夠在故障期間自動介入，與良好的使用者體驗直接相關。了解隨著時間的推移，這些緩解機制是否有任何偏離至關重要。透過定義成功和失敗自動化緩解措施的指標，您可以建立指南，以協助識別整個系統的潛在迴歸。

記錄

在混沌實驗之前、期間和之後，集中式記錄是了解應用程式元件狀態的關鍵。我們建議您從所有應用程式元件收集日誌，以建立每個元件在注入實驗時正在執行的操作的合併檢視。這可提供end-to-end實驗流程的清晰影像。

請求追蹤

請求追蹤可讓您觀察應用程式中元件之間任何單一請求的流程，以全面了解注入失敗對系統及其相依性的影響。透過追蹤請求，您可以查看故障如何傳播到不同的服務和元件，以便更好地評估中斷的範圍。若要追蹤 上的請求 AWS，您可以使用 AWS X-Ray。

注入混沌實驗的失敗案例

將常見故障注入您的應用程式的目的是了解應用程式對這些意外事件的反應，以便您可以建立緩解機制並讓系統對此類故障具有彈性。此外，您應該使用混沌工程來重播歷史故障案例，以確認您的緩解機制仍然如預期運作，並且不會隨著時間而偏離。

當您規劃混沌工程實驗時，請考慮下列事件。

失敗模式	說明
伺服器受損	重新啟動 EC2 執行個體、刪除 Kubernetes Pod 或 Amazon Elastic Container Service (Amazon ECS) 任務，以了解應用程式對此類當機的反應。
API 錯誤	將錯誤注入 AWS 和您自己的服務 APIs，以了解應用程式行為。
網路問題	引入延遲或擁塞，或封鎖連線以模擬真實世界的網路問題。
AWS 可用區域受損	重播整個可用區域的損害，以驗證跨區域的復原。
AWS 區域 連線受損	重播網路中斷 AWS 區域 ，以驗證次要區域中的資源如何對此類事件做出反應。
資料庫失敗	容錯移轉資料庫複本或損毀的資料，或使資料庫執行個體無法連線，以了解對應用程式和復原策略的影響。
在資料庫和 Amazon S3 複寫中暫停	暫停資料庫或跨可用區域的 Amazon Simple Storage Service (Amazon S3) 複寫 AWS 區域 ，或了解下游應用程式的影響。
儲存體降級	暫停 I/O、移除 Amazon Elastic Block Store (Amazon EBS) 磁碟區或刪除檔案，以驗證資料耐久性和復原。
相依性受損	降低或降低您依賴的下游或上游服務的效能，包括第三方服務，以了解end-to-end流程和對客戶的影響。
流量突增	在使用者流量中產生峰值以測試自動擴展功能，並查看冷開機時間如何影響您的整體應用程式狀態。

失敗模式	說明
資源耗盡	將 CPU、記憶體和磁碟空間最大化，以驗證應用程式的正常降級。
串聯失敗	啟動層疊到下游應用程式和元件的主要故障。
部署錯誤	推出有問題的變更或組態，以驗證轉返機制。

組織彈性贊助

混沌工程會在整個組織套用時提供最大的價值。我們建議您與執行發起人合作，他們可協助設定整個組織的彈性目標、消除對網域的恐懼、不確定性和懷疑，並開始轉型程序，讓彈性成為每個人的責任。

為了支援建立混沌工程實務的業務案例，請將混沌工程工作連接至您的關鍵業務專案。讓彈性成為加速的資產和驅動因素，將有助於您追蹤一段時間的成功。從每月或每季的關鍵事件計數、平均復原時間，以及這些事件對客戶和組織造成的影響開始計算。與您的團隊設定目標，以減少 6 到 12 個月期間的事件數量，因為應用程式堆疊會進行改善，以回應混沌工程實驗。

測量事件是否具有高度重複性。例如，假設過期的 TLS 憑證會導致停機時間，因為用戶端無法建立信任的連線。如果由於多個 TLS 憑證過期而在一年內發生多個事件，您可以執行 TLS 憑證過期的實驗，並確認您的團隊收到提醒或能夠自動緩解此類問題。這將有助於確保您對此類故障具有彈性。

若要追蹤一段時間內的混沌工程進度，請擷取下列指標，以協助強調應用程式生命週期中的混沌工程值：

- 降低事件率 – 追蹤一段時間內的生產事件數量，並將此數量與採用混沌工程相互關聯。預期嚴重事件的速率將會下降。
- 改善平均解決時間 (MTTR) – 計算解決事件和追蹤此資料所需的平均時間，以查看隨著時間的推移，是否因混沌工程而改善。
- 提高應用程式可用性 – 使用服務層級指標，在應用程式彈性透過混沌實驗增加時顯示可用性改善。
- 更快的上市時間 – Chaos 工程可以提供信心，更快地推出創新產品，因為您知道應用程式具有彈性。追蹤產品發行速度的增加。
- 降低營運成本 – 量化警示噪音、營運負載和手動管理應用程式等指標是否隨著實施混亂實務而減少。
- 提高信心 – 調查開發人員、站點可靠性工程師 (SREs) 和其他技術人員，以測量混沌工程是否提高他們對應用程式彈性的信心。感知很重要。

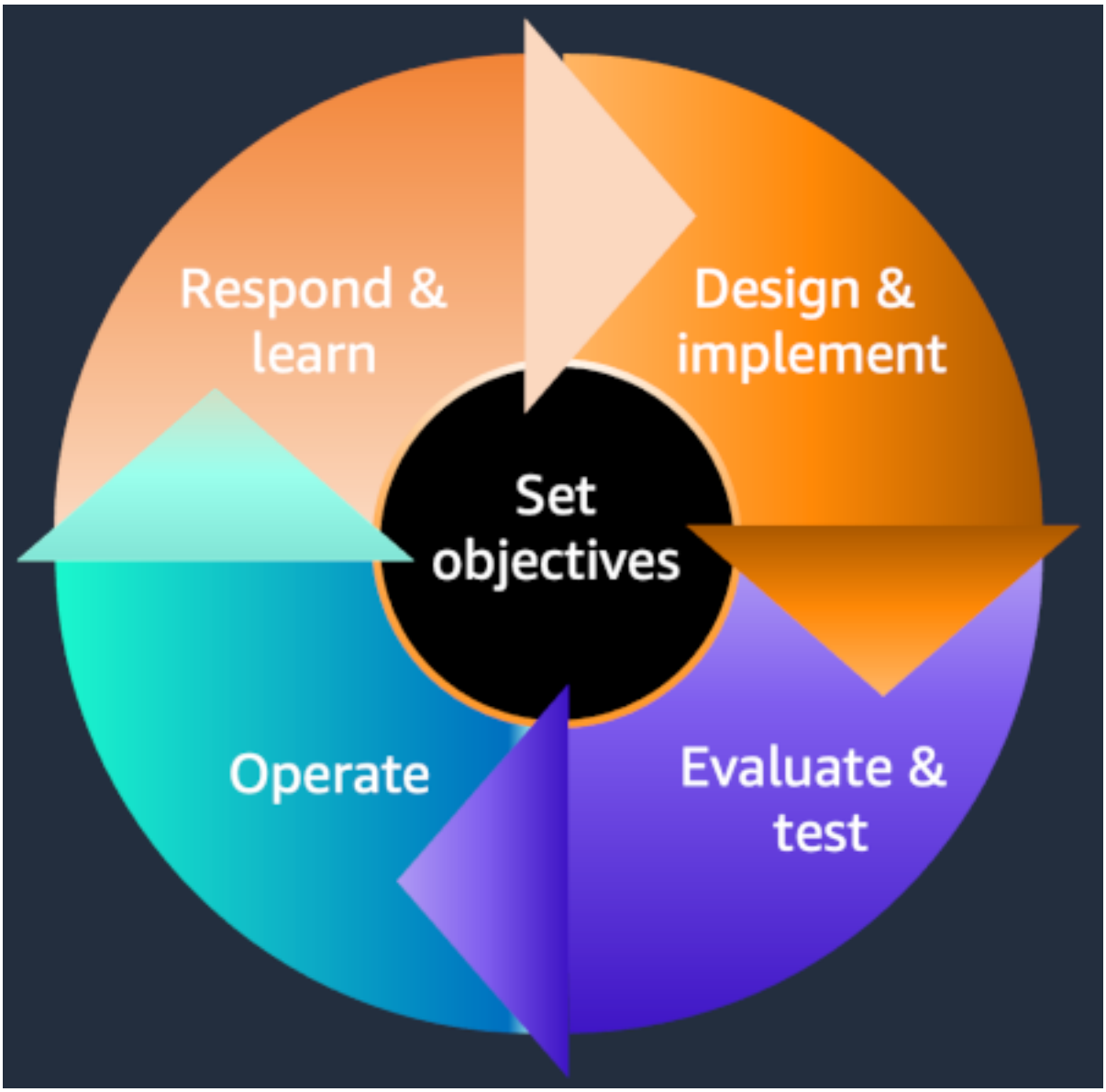
- 改善客戶體驗 – 將混沌工程與客戶正面成果連線，例如減少服務中斷、轉返和中斷。

排定修復的優先順序

當您執行混沌實驗時，您可能會找出應用程式未如預期執行的待改進領域。這類項目的修復會在您的待處理項目中運作，必須與其他工作一起排定優先順序，例如功能開發。我們建議您為這些增強功能騰出時間，以避免未來失敗。考慮根據這些學習和修復任務可能造成的影響程度，排定這些學習和修復任務的優先順序。直接影響應用程式彈性或安全性的問題清單應優先於新功能，以避免客戶受到影響。如果團隊在特徵開發上難以排定補救措施的優先順序，請考慮聯絡您的執行發起人，以確保根據業務風險承受能力設定優先順序。

在 上實作混沌工程 AWS

混沌工程是AWS 彈性生命週期評估和測試階段的一部分，如下圖所示。分散式應用程式不會與其他應用程式或用戶端分開運作，因此我們建議您檢閱整個彈性生命週期。隨著網路演進、上游和下游應用程式經歷轉移，以及用戶端用量隨時間變化，分散式應用程式的變更是固定的。



若要了解這些應用程式變更如何影響其彈性，請將混沌工程作為day-to-day操作的一部分。您可以採用不同的方式實作混沌實驗：

- 臨機操作 – 您可以將混沌實驗作為一次性實驗來執行，以解決特定問題或疑問。

- 混沌遊戲日 – 這些是結構化和週期性事件，旨在驗證應用程式的可靠性和彈性。混沌遊戲日的目的是識別人員、程序和技術的潛在彈性問題或缺陷，並練習識別、緩解和回應事件的程序和程序。
- 混沌管道 – 持續整合和持續交付 (CI/CD) 是關於建置新功能，並將其安全地部署到整個環境。若要實作混沌工程實驗，請建立與您的 CI/CD 管道分開的混沌管道。為了了解原因，假設您想要將單一混沌工程實驗新增至 CI/CD 管道，為下游元件注入增加的封包遺失。該實驗執行 3 次，每次需要 5 分鐘才能完成。每次執行時，封包遺失從 10% 增加到 20% 到 30%，實驗整體需要 15 分鐘才能完成。如果您有 100 個平行部署，則需要等待 1500 分鐘才能完成單一實驗。如果您有 10 個實驗要執行，對開發人員的影響將無法承受。大規模的混沌工程需要自己的管道，可讓您平行於軟體開發生命週期 (SDLC) 程序執行實驗。
- Canary 部署 – Canary 為混沌實驗提供測試環境。透過將一小部分的流量導向 Canary 服務，或使用流量鏡射或重播等方法，您可以驗證新的基礎設施或程式碼變更，而不會對穩定的生產系統造成任何影響。您可以針對 Canary 執行實驗並視需要注入錯誤，因為您可以限制對最終使用者的影響範圍。
- 排程實驗 – 您可以排程實驗來驗證應用程式的可預測復原機制。使用排程實驗重播常見事件，擷取您的系統如何從事件中復原，例如終止自動擴展群組後方的 EC2 執行個體、移除 Kubernetes Pod，或刪除 Amazon ECS 任務。

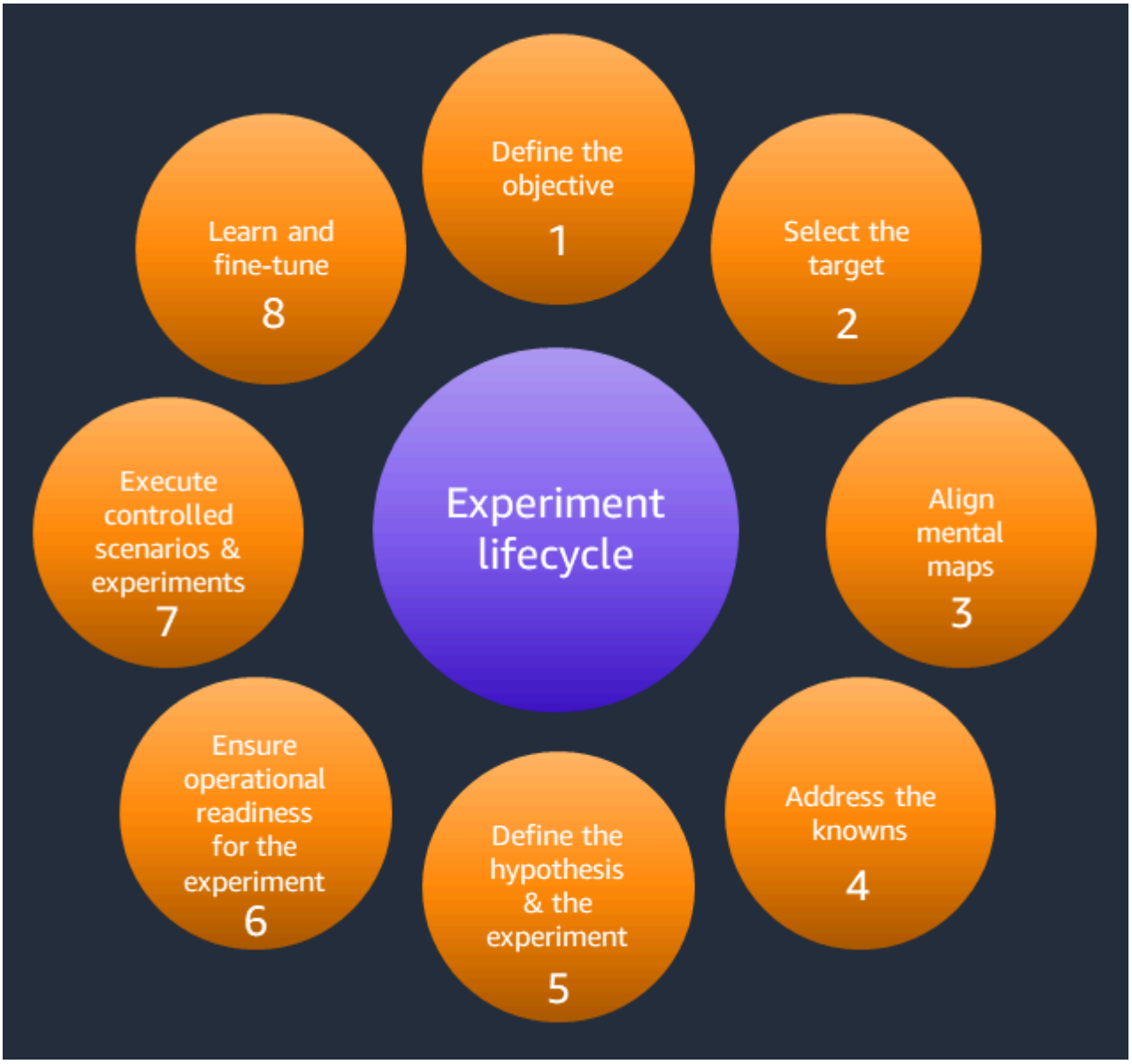
持續混沌工程實驗生命週期

如上一節所述，您可以採用不同的方式實作混沌工程實驗。在所有情況下，建置有意義的混沌實驗的關鍵在於了解應用程式、歷史事件和實作的修補，以及清楚了解要調查的領域，例如彈性或安全性。您對應用程式的了解可協助您制定應用程式潛在弱點的假設，並了解它在注入錯誤時如何偵測、修復和復原。

混沌實驗生命週期包含下列步驟：

1. 定義實驗的目標。
2. 選取目標應用程式。
3. 對齊心智圖。
4. 解決應用程式的已知問題。
5. 定義假設和實驗。
6. 確保實驗的操作準備度。
7. 執行受控案例和實驗。
8. 從 學習並微調實驗。

這些步驟會在圖表中說明，並在下列各節中討論。



定義目標並設定期望

在每次實驗之前，請確保您的目標和期望是具體的、可衡量的、可實現的、相關的和有時間限制的。明確定義下列項目：

- 識別系統和服務的可能故障或弱點，以了解它們如何影響使用者。這包括識別可能的故障模式，例如伺服器當機、網路故障或軟體錯誤，以及評估其對系統整體效能和可靠性的潛在影響。
- 透過定義關鍵風險指標 (KRIs) 來量化故障對系統和服務的影響。這包括當延遲、輸送量和錯誤率等指標偏離其穩定狀態時，測量失敗的影響。透過了解此類偏差的影響，您可以根據業務風險來排定緩解失敗的優先順序。

- 制定和驗證緩解或防止失敗的策略。這包括識別潛在的解決方案，例如備援、錯誤修正或備用策略，以及在受控環境中測試其有效性。透過驗證這些策略，您可以確保有效防止或緩解故障，並可以放心地在生產系統中部署它們。
- 改善事件回應和災難復原程序。透過在受控環境中重播失敗，您可以測試事件回應程序、識別潛在的瓶頸或差距，以及精簡災難復原程序。這有助於確保您準備好在發生意外故障時快速有效地做出回應。

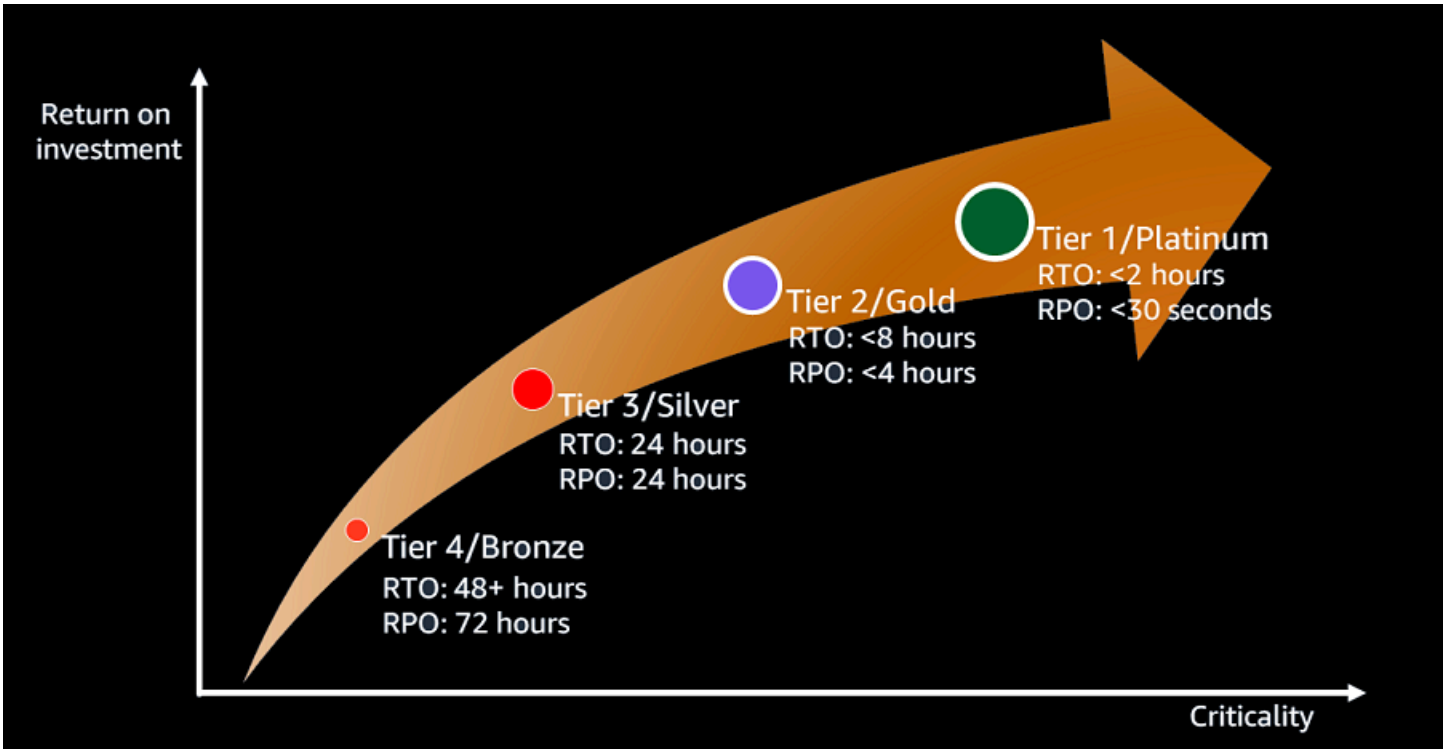
選取目標應用程式

混沌工程是一項強大的技術，但需要經過深思熟慮的優先順序才能最大化價值。在決定將混沌工程工作集中於何處時，請先考慮您業務的關鍵服務。要求您的團隊逐一完成軟體開發生命週期階段，並開始先在測試環境中注入故障。業務關鍵型應用程式直接與營收、客戶體驗和核心操作相關。這些服務的混沌實驗可以發現可能會對組織造成嚴重影響的漏洞，如果未解決這些漏洞，也可能影響整個市場。例如，專注於面向客戶的服務，例如交易系統或訂單系統。排定這些中央服務的優先順序可為每個時間投資提供最大的保護。

在關鍵服務之後，請查看資料庫、訊息佇列、網路和共用服務 APIs 等基本元件。這些可能會用作整個組織的共用元件或服務，因此它們的故障會導致廣泛的問題。確認基礎設施服務的彈性可確信它們不會打亂其上方的相依應用程式。例如，關閉 Kafka 叢集的混沌工程實驗顯示許多下游應用程式的容錯能力。雖然系統基礎設施不是直接面對客戶，但它是主要的混沌工程目標。

請不要忘記映射人員、程序、設施資訊和第三方相依性的心理差距，因為如果不符合組織的影響承受能力目標，可能會導致重大中斷。如需測量混沌工程投資報酬率的詳細資訊，請參閱將策略文件中混沌工程投資的 ROI 量化為策略必要性。

下圖顯示在不同服務層上執行混沌實驗的投資報酬率。



對齊心智圖（應用程式探索）

當您執行臨機操作實驗或遊戲日時，您將透過保留專注於映射應用程式詳細資訊的白板工作階段，開始應用程式探索程序。（如果您在混沌管道中執行實驗，您已透過定義目標應用程式來對齊該心智圖。）了解心理差距的好方法是讓最資淺的團隊成員先繪製應用程式的圖表，然後要求更資深的工作人員逐步新增至圖表。這將發現跨體驗層級理解的任何差距。

圖表應描述應用程式的直接上游和下游相依性，以及任何重要的第三方整合。請確定透過應用程式請求的預期流程相符。規劃關鍵工作流程和使用者旅程，以釐清客戶如何使用應用程式。考慮使用序列圖來擷取此資訊。

在此協作工作階段之後，團隊應該擁有應用程式的共同心理模型、其關鍵相依性及其監控功能，並了解做出明智決策以繼續進行或取消潛在混沌實驗的風險。

解決應用程式的已知問題

混沌工程實驗旨在主動找出應用程式中的瑕疵。透過插入延遲增加、伺服器重新啟動或可用區域電源受損等故障，您可以驗證應用程式容忍逼真的中斷的能力。不過，此程序會假設目標應用程式中的基本穩定性和運作狀態。在已經有問題的應用程式上執行混沌實驗，可能會掩護更深入的問題。

在進行混沌工程之前，團隊應該解決應用程式中任何已知的瑕疵、錯誤和效能問題。

定義假設和實驗

過去導致應用程式或組織內其他應用程式中斷的事件，可作為混沌實驗想法的絕佳來源。例如，先前的中斷是由組態錯誤或缺少彈性模式所觸發？ 透過混沌實驗檢閱事件歷史記錄並重播這些真實世界失敗的根本原因，是在未來針對類似問題開發彈性的有效方法。

另一個寶貴的實驗概念來源可以直接來自最熟悉 應用程式的工程師、架構師和操作員。允許團隊成員提交他們認為可能會嚴重中斷應用程式的假設失敗案例，讓您能夠根據內部知識收集想法。然後，應用程式團隊可以評估哪些提議的案例可能具有最大的潛在影響，或公開最大的未知風險。針對此類高風險、較不了解的案例進行混沌實驗，可以產生重要的學習並防止未來發生問題。

第三個想法來源是執行彈性建模，以預測可能導致已識別業務損失的條件。有些彈性建模練習具有以元件為基礎的方法來建置彈性模型，而有些則具有以系統為基礎的方法。以元件為基礎的方法詢問問題：「當元件 x 處於極端負載或失敗時會發生什麼情況？」 然後，開發彈性模型的團隊推測此類案例對更廣泛的應用程式的影響，並識別目前實施的監控和預防性控制，以偵測和緩解案例的影響。或者，系統型方法遵循由上而下的程序來強調應用程式的不良狀態，例如「Web Storefront 顯示過時的庫存水準」，並邀請應用程式團隊預測哪些條件或條件會導致應用程式以這種方式運作。

確保實驗的操作準備度

您需要可量化的指標來衡量不利條件對應用程式及其行為的影響，如先前有關可觀測性一節中所述。能夠測量應用程式的行為可讓您判斷不利條件是否會影響應用程式，以及影響程度。

了解應用程式是否有影響的最佳方法是測量其穩定狀態。穩定狀態會測量正常操作的外觀，並通常符合特定應用程式的業務和客戶體驗指標。在進行下一個步驟之前，請確定您已備妥可觀測性來了解影響，並準備好回復機制，以防實驗未如預期進行。

執行受控實驗和案例

在 AWS，我們不建議在生產環境中的應用程式上執行初始混沌實驗。混沌實驗的目的是了解應用程式在壓力條件下如何運作的新知識。應用程式的行為在實驗期間可能無法預測，因此第一次在生產環境中執行實驗可能會對客戶造成影響。因此，您應該一律在影響真實世界使用者的可能性最低的較低層級環境中執行初始混沌實驗，然後在驗證並確信您的應用程式可以吸收、調整並從注入的動作中復原之後，反覆瀏覽您的環境。

使用擷取金鑰詳細資訊的文件徹底規劃每個實驗，類似於附錄中提供的實驗規劃文件。要包含的一些關鍵欄位是穩定狀態定義、假設和失敗注入方法。混沌實驗的規劃、執行和分析可以在單一成品中涵蓋。

在您完成實驗的書面計畫之後，請準備任何必要的程式碼，以注入文件中概述的計劃中斷。

若要在實驗期間擷取潛在影響，請確保有可觀測性機制。如果您還沒有自動擷取實驗結果的方法，例如 AWS FIS 實驗報告、識別將在執行期間做筆記的團隊成員、擷取儀表板的螢幕擷取畫面，以及帶領團隊完成實驗。

學習和微調

在每次實驗之後，以團隊身分一起檢閱和思考混沌實驗。有意識地努力維持無責的思維。您的目標是要有一個開放、具建設性的對話，完全專注於獲得最大的學習，而不是指派責任。

首先，檢閱實驗的穩定狀態定義和假設。應用程式的行為是否如預期？ 是否有任何使假設失效的意外？ 討論應用程式在實驗期間如何反應的觀察，包括好和壞。收集的資料 – 指標、日誌、螢幕擷取畫面等 – 應該說明發生的情況。

以好奇而非判斷的方式進行此資料檢閱，並根據經驗找出可改善應用程式設計、文件、監控或其他功能的領域。這些動作項目會擷取為後續專案，讓應用程式更具彈性。

透過這種無責方法，您可以坦率地討論發生了什麼問題以及如何修正它。假設參與的每個人有正面意圖，並信任他們正在努力實現良好的成果。您的共同目標是透過持續學習和調整來實現組織成長和進展。以具建設性、無責的方式執行的混沌實驗審查，可為您的團隊提供安全空間，以取得寶貴的洞見，讓您的應用程式和組織長期更可靠且更具彈性。重點在於學習，而不是人員。若要將學習內容分散到您的團隊，請將實驗結果報告發佈到中央位置，並公告調查結果，以便其他人可以從中學習。

擴展整個組織的混沌工程

隨著您的組織採用混沌工程，標準化和實作將會帶來挑戰。在成熟的早期階段，不同的團隊可能會使用前面幾節所述的混沌工程程序的不同工具和變化。同時，有些團隊可能不會優先考慮或採用混沌工程，即使其可能帶來好處。下列各節提供如何克服這些挑戰的指引。

整體而言，您的混沌工程方法應設計為在集中式領導和分散參與之間取得平衡。此平衡有助於確保混沌工程整合到開發程序中，並在整個組織中共用學習。

建立混沌工程實務

將混沌工程實務標準化可以加速採用。跨團隊分享來自實驗的學習，可以擴大混沌工程投資的回報。

在混沌工程實務中建立集中式卓越中心，或組合一組主題專家。作為小型的集中式函數，此團隊可以跨軟體開發、基礎設施、安全和業務團隊運作，並維護這些團隊使用的標準。為了簡化， 卓越中心稱為集中式實務團隊，而套用混沌工程的群組在本指南的其餘部分稱為實務團隊。

集中式實務團隊的角色

集中式實務團隊負責在整個組織中開發和實作混沌工程實務。他們與實務團隊緊密合作，引導他們設計和執行實驗，並確保實驗對業務很有價值。集中式實務團隊也為開發、基礎設施和安全團隊提供指導和支援，以協助他們將混沌工程整合到他們的開發程序中。

集中式混沌工程實務團隊的主要責任包括下列項目：

- 啟用 – 集中式混沌工程函式做為主持人，透過遊戲日和研討會介紹混沌工程實務。他們在混沌工程過程中引導團隊，包括選擇失敗案例、定義假設，以及產生要與更廣泛的組織共用的報告。集中式實務團隊應擁有訓練資料，並努力提升實務團隊使用混沌工程的技能。
- 諮詢 – 集中式實務團隊也可以擔任諮詢角色，以監督實務團隊執行的實驗。他們的經驗和知識可以確保實驗為企業提供價值，並以安全的方式執行。同樣地，團隊可以監督實驗的執行和總結，以引導剛接觸混沌工程的人員。
- 行銷和價值追蹤 – 傳達混沌工程的商業價值是此類計劃成功的關鍵。參與混沌工程實驗的每個團隊都應從整個企業的實驗收集資料，並展示組織對混沌工程投資的價值。這包括量化和慶祝每個實驗期間避免的事件數量、實驗失敗時可能發生的停機時間，以及生產中發生故障案例時對業務的整體影響。透過從跨團隊收集和集中此類資料，並將資料提供給整個組織，集中式實務團隊可以追蹤和影響在整個組織中採用混沌工程所產生的價值。

- 標準 – 集中式實務團隊應擁有和維護執行混沌實驗的程序、規劃和報告實驗的範本，以及用於執行實驗的工具。

  中央團隊應擁有和管理實驗規劃範本、實驗報告範本、程序文件和啟用資料。最佳實務文件和啟用材料為團隊提供指引，以練習他們可以用來限制實驗影響的護欄、何時在生產中進行實驗，以及如何隨著時間發展他們使用混沌工程。如需範本和輸出的範例，請參閱附錄。

  集中式實務團隊也應該擁有執行實驗的程序，包括通訊和呈報，以及在實驗之前或期間與組織中其他團隊通訊的時間和方式。程序也應概述何時需要護欄。

  集中式實務團隊也應該選取並擁有執行混沌實驗的核心工具（例如，工具，例如 AWS FIS)。選擇和實作補充工具，例如負載產生工具，應該保留給練習團隊來決定。練習團隊應該能夠調整整體程序和工具，以符合其需求。

練習團隊的角色

集中式團隊負責推動整體混沌工程策略，而實務團隊則參與程序並擁有實驗的開發和執行。這有助於確保實驗與每個特定產品或服務相關，並且學習是可行的，並且可以套用以提高產品的可靠性和彈性。集中式實務團隊充當組織混沌工程標準和程序的指導者和擁有者。不過，為了防止集中式團隊成為瓶頸，個別練習團隊將需要從中央實務中學習，為自己執行混沌實驗。

建立實務社群

除了建立集中式團隊之外，我們建議您建立由對混沌工程有興趣的從業人員組成的非正式社群。此社群提供跨實務團隊和更廣泛的組織共用知識、最佳實務和體驗的平台。

實務社群可由集中式混沌工程實務團隊運作，但組織內的任何人都可以成為社群的成員。集中式團隊可以利用實務社群廣播更新和來源學習，以及從使用集中式團隊所管理標準和程序的實務團隊收集意見回饋。社群將充當回饋迴圈，通知集中式團隊有關實務團隊中混沌工程實務的有效性。然後，集中式實務團隊可以調整其文件和支援成品，以最佳支援產品團隊。

將混沌工程納入您的營運恢復能力

混沌實驗是您的企業為了防止生產中發生的事件所做的投資。需要確定企業可以實現此投資最大收益的位置。組織可以與集中式混沌工程實務團隊合作，更新其標準，並判斷哪些產品夠重要，需要混沌實驗。

系統開發程序

混沌工程和混沌實驗應作為應用程式生命週期的一部分重複執行。與團隊定期執行災難復原測試的方式類似，他們應該在一年中持續且定期地執行混沌實驗和遊戲日。此方法可改善組織預測、觀察和回應事件的方式。

結論

過去十年來，混沌工程的紀律已有很大的進展。它已在各種產業中採用，並協助組織建立彈性服務並提高客戶滿意度。（如需組織如何實作這些實務的範例，請參閱混沌工程案例。）混沌工程讓組織能夠透過在其應用程式堆疊的所有層級注入控制錯誤，包括雲端供應商服務，來降低其關鍵任務應用程式的風險。能夠以受控制的方式影響整個應用程式堆疊，可實現持續的彈性、改善卓越營運、可觀測性和復原導向的架構，而無須承受生產中斷的壓力。此方法也有助於提升整個組織的彈性測試實務。若要開始接受混沌工程，請執行混沌遊戲日或研討會，展示混沌工程可為您的組織提供的價值。

資源

AWS FIS 失敗模型實作：

- 使用 AWS Fault Injection Service 來示範多區域和多可用區域應用程式彈性 (AWS 部落格文章）
- AWS Fault Injection Simulator 支援 Amazon EKS Pod 上的混沌工程實驗 (AWS 部落格文章）
- 宣布 AWS Amazon ECS 工作負載的 Fault Injection Simulator 新功能 (AWS 部落格文章）
- 使用 Amazon EBS 磁碟區指標和 AWS Fault Injection Simulator 改善應用程式彈性 (AWS 部落格文章）
- 在多帳戶 AWS 環境中利用 Fault Injection Simulator AWS 的混沌工程 (AWS 部落格文章）
- 使用 Fault Injection Simulator 的 Amazon RDS 上的混沌實驗 AWS(AWS 部落格文章）
- 使用混沌工程建置彈性無伺服器應用程式 (AWS 部落格文章）
- 使用 FIS 中斷 Spot 執行個體 (AWS workshop)
- 在交付管道中自動化混沌工程 (AWS 社群文章）

其他資源：

- 投資混沌工程作為策略必要性
- 混沌工程案例
- Amazon CloudWatch 文件
- Amazon CloudWatch RUM 文件
- AWS X-Ray 文件
- AWS FIS 文件

附錄：範例文件

本節提供的範例文件使用虛構的寵物採用網站 (PetSite) 做為混沌實驗的目標應用程式。

- 實驗規劃文件
- 實驗結果文件

實驗規劃文件

穩定狀態

程序名稱	寵物採用網站
實體架構	（連結至架構圖）。
邏輯架構	（邏輯圖的連結）。
定義穩定狀態	平均頁面載入時間，測量方式為使用最大有內容顏料 (LCP)，對於寵物採用網站為 2.5 秒或更短，99 個百分位數延遲 (P99) 為 4.0 秒或更短，基準為 5000 個並行使用者。
穩定狀態指標	跨使用者基礎擷取的 LCP 指標，以及黃金指標（延遲、輸送量、錯誤率、飽和度）。
穩定狀態可觀測性	LCP 將由使用者的瀏覽器擷取、傳送至 Amazon CloudWatch，並使用 CloudWatch RUM 進行檢查。在 60 秒期間內，將會彙總該期間內所有請求的平均和 P99 LCP 時間。最上層黃金指標是使用 CloudWatch 擷取。
達到穩定狀態的程序	Grafana K6 將用於建立負載，模擬大約 5K 個並行使用者的正常生產流量層級。

可觀測性要求

團隊應該能夠檢視下列項目：

- 穩定狀態：將觀察哪些內容來驗證應用程式是否處於正常狀態？
- 失敗條件：失敗條件將如何顯示在儀表板中？ 例如：
  - 應觸發的警示
  - 應該產生的日誌
- 失敗影響：應觀察哪些項目來檢視預期會受到影響的元件 （影響範圍）？
- 復原：如何檢視和測量復原以擷取 MTTR？
- 偵錯：針對實驗失敗的詳細資訊進行故障診斷。

下表提供可觀測性需求圖表的建議和範例。您應該根據您的特定實驗，定義應該觀察的內容。

需要觀察的項目	連結至可觀測性工具	正在觀察到的內容
輸入來源	Grafana K6 儀表板	• 執行容器計數 • 每秒請求數
整體應用程式運作狀態	• 寵物採用 CloudWatch 儀表板 • 寵物採用使用者體驗儀表板 (RUM)	• Amazon EKS 運作狀態良好的節點計數 • Amazon EKS 節點 CPU 使用率
工作流程運作狀態	寵物採用 CloudWatch 儀表板	LCP 時間，黃金指標
追蹤	寵物採用 X-Ray 儀表板	• 請求延遲 • 請求計數 • 失敗計數

需要觀察的項目	連結至可觀測性工具	正在觀察到的內容
日誌	寵物採用 CloudWatch Logs	Pod 遇到的任何錯誤都會發佈至 CloudWatch Logs。

實驗定義

實驗名稱	Amazon EKS PetSite Pod CPU 壓力
實驗原始程式碼	（實驗來源儲存庫的連結。）
實驗描述	此實驗探索 PetSite 應用程式 Pod 的 CPU 用量增加將如何影響整體客戶體驗。 透過將 CPU 壓力注入每個執行中的 PetSite Pod，我們將能夠了解是否對客戶造成影響，以及影響的程度。
實驗需求或參數	應用程式負載：生產平均值 Pod 標籤選擇器： `app=petsite`
實驗持續時間	10 分鐘
Environment	Alpha 測試環境
實驗目標資源	PetSite 應用程式 Pod
透過負載產生工具引入的實驗基準	• 54% 的請求具有 <2.5 秒的 LCP。 • 46% 的請求具有 <4 秒的 LCP。 • 未觀察到錯誤。
退避條件	無

假設

如果	影響	復原
如果 PetSite 應用程式裝置在正常生產層級流量下 10 分鐘遇到或導致 CPU 使用率超過 60%，穩定狀態會發生什麼情況？	對於 P99 為 4.0 秒或更短的 P50 使用者，LCP 時間將維持在 2.5 秒以下。 P99 消費者應該能夠載入 PetSite 登陸頁面。	偵測： • 在 CloudWatch 中設定的警示會偵測到 CPU 壓力。 • LCP 指標也會針對使用者體驗的降級產生警示。 自我修復： • 微型服務架構的分散式性質表示許多 Pod 執行個體正在多個可用區域上執行。 • EKS 叢集控制平面會將流量移離受影響的 Pod，並在工作者節點上啟動新的 Pod。 復原： 當 CPU 使用率恢復正常時，LCP 應會自動復原。

實驗程序

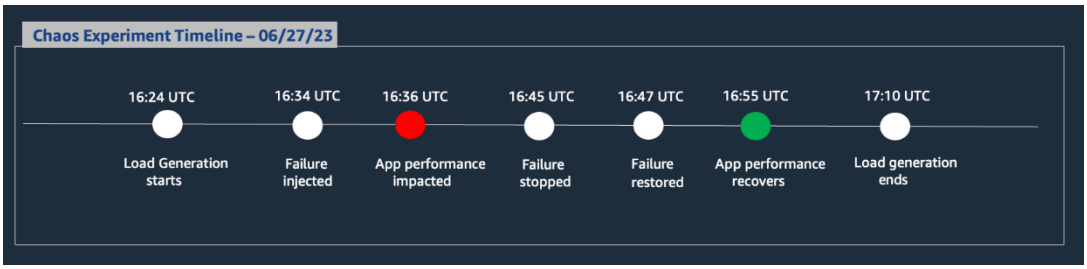
為您的特定實驗量身打造下列step-by-step程序：

1. 驗證所有 Amazon CloudWatch、CloudWatch RUM 和 AWS X-Ray 儀表板的存取權和功能。
2. 驗證應用程式環境的運作狀態：
   a. 使用 CloudWatch 儀表板確認 EKS 叢集運作狀態良好。
   b. 請造訪範例 URL 的測試寵物採用網站應用程式部署。

3. 啟動載入以達到穩定狀態：
   a. 確認負載產生器正在執行，每秒傳送 5000 個請求。
   b. 等待 5 分鐘，讓應用程式達到穩定狀態。
   c. 使用 CloudWatch RUM 儀表板確認應用程式的穩定狀態。
4. 啟動錯誤（實驗）：
   a. 開啟 AWS FIS 主控台。
   b. 執行 pet-adoption-pod-stress 實驗。
   c. 確認實驗正在 主控台中執行。
5. 觀察故障對應用程式的影響：
   a. 從 CloudWatch RUM 和 CloudWatch 儀表板擷取螢幕擷取畫面，並記下任何異常的資料點。
   b. 在實驗完成後 AWS FIS，擷取額外的螢幕擷取畫面，以記錄應用程式是否在沒有壓力的情況下返回穩定狀態，並記下資料點中的任何異常。
   c. 如果穩定狀態未繼續，請採取步驟來復原應用程式，並記錄採取的步驟。
6. 驗證環境已恢復正常：
   - 檢閱所有業務、使用者體驗、應用程式和基礎設施指標，以確認系統已返回已知狀態。如果有幫助，請擷取儀表板螢幕擷取畫面。

實驗時間軸

請確定您擷取end-to-end實驗的時間軸，從負載產生開始、注入錯誤、觀察影響和復原應用程式，以及在停止負載產生時結束。這會在下列範例中說明。



實驗結果

實驗執行 ID	實驗結果
PET-ADOPT-EXP-23	（實驗結果的連結。）

已識別的瑕疵

- Kubernetes 叢集未偵測到 PetSite Pod 的 CPU 受損，因此未排程其他部署。
- 此實驗不會增加 4XX 或 5XX 錯誤率。
- 我們需要調整 Pod 的運作狀態檢查，以便在有資源限制時考慮對 LCP 的影響。

實驗結果文件

組態

記錄實驗的特定組態。例如：

- 負載產生設定為模擬 5K 使用者每秒發出總共 85 個請求。

先決條件

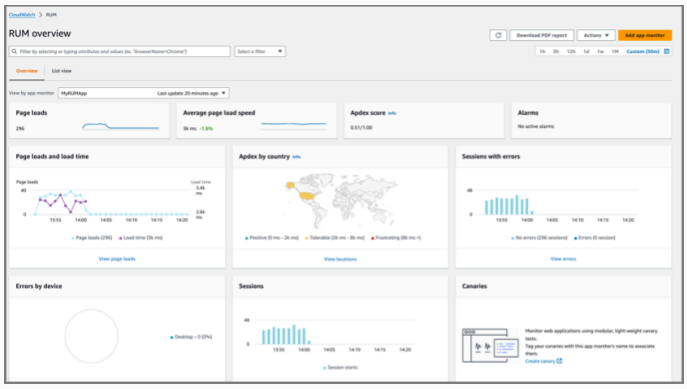
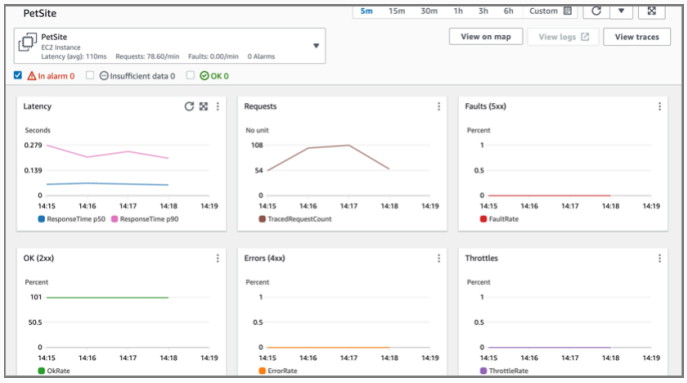
- 驗證寵物採用網站是否在 Alpha 測試環境中執行。
- 驗證實驗範本已設定為將 CPU 壓力套用至在 EKS 叢集中執行的 PetSite 應用程式 Pod。 應用程式 Pod 由 Kubernetes 標籤 識別`app=petsite`。
- 負載已確認正在執行，每秒產生 85 個請求。

穩定狀態

記錄為了達到穩定狀態所採取的步驟，以及驗證的方式。例如：

針對寵物採用站點的測試部署，會產生 85 個 RPS 的負載來模擬穩定狀態。已檢閱 CloudWatch RUM 和 CloudWatch 儀表板，以確認在執行實驗之前，所有業務和應用程式指標都在正常範圍內。

可觀測性資料：

預期	觀察到的
• P99 請求的 LCP 少於 4 秒。 • 回應延遲小於 500 毫秒。 • 沒有 4XX 或 5XX 錯誤。	 

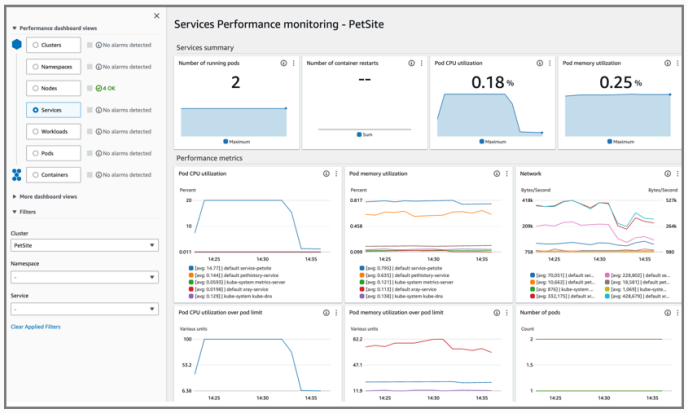
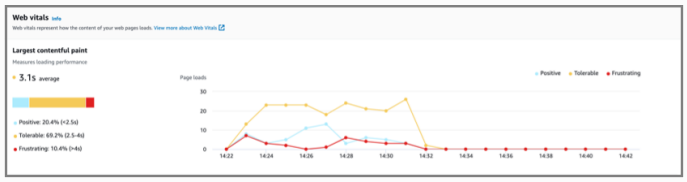
故障注入

AWS FIS 使用實驗範本（提供連結）來注入錯誤。實驗設定為執行 10 分鐘，如果工作者節點遇到 CPU 壓力超過 60%，則會設定回復。

故障觀察

已檢閱 CloudWatch RUM 和 CloudWatch 儀表板，以追蹤應用程式的穩定狀態（使用 LCP 指標定義）。 螢幕擷取畫面擷取於下表。

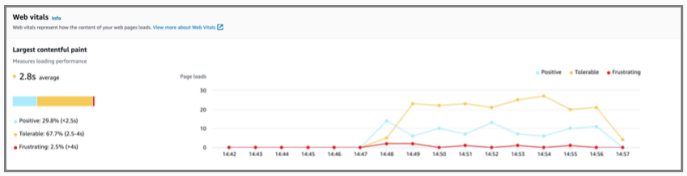
可觀測性資料：

預期	觀察到的
• P99 的 LCP 應保持在 4 秒以下。 • 回應時間應保持在 500 毫秒以下。 • 不應遇到 4XX 或 5XX 錯誤。	 

復原

移除壓力後 (AWS FIS 實驗已完成並從 Pod 移除 CPU 壓力) ，應用程式應恢復正常穩定狀態。 不需要手動介入。

可觀測性資料：

預期	觀察 （螢幕擷取畫面）
LCP P99 應低於 4 秒，平均值低於 2.5 秒。	

文件歷史紀錄

下表描述了本指南的重大變更。如果您想收到有關未來更新的通知，可以訂閱 RSS 摘要。

變更	描述	日期
初次出版	—	2025 年 4 月 4 日

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。