在部署期間確保轉返安全

Sandeep Pokkunuri

在 Amazon，我們建構解決方案的指導方針之一為避免單向門決策。也就是說，遠離不可逆轉或無法擴展的決策。我們在軟體部署的所有步驟中應用此方針，從設計產品、功能、API，乃至後端系統到部署。在本文中，我會說明我們如何在軟體部署時應用此方針。

部署會使軟體環境從某個狀態 (版本) 成為另一種。軟體可能在這兩種狀態下皆運作良好。但是，在向前轉換 (升級或前滾) 或向後轉換 (降級或轉返) 期間或之後，軟體可能無法妥善運作。軟體運作不良時，會造成服務中斷，也因此無法可靠提供給客戶。在本文中，我假設兩種軟體版本都如預期般運作。我的焦點會擺在如何確保部署期間前滾或回滾不會造成錯誤。

在釋出新軟體版本之前，我們會在 Beta 版或 Gamma 版測試環境中進行多種維度的測試，例如功能、並行、性能、擴展和下游失敗處理。此測試有助我們找出新版本中的問題並加以修正。但有時候，這樣的測試不足以確保成功部署。我們在實際執行環境中，可能會遭遇預期外的狀況或不太理想的軟體行為。

在 Amazon，我們希望避免陷入部署轉返可能導致客戶錯誤的處境。為了避免這樣的狀況，我們在每一次部署之前，會做好完備的轉返準備。轉返不會導致前一版功能發生錯誤或導致前一版功能中斷的軟體版本，稱為向後相容。我們的每一個修訂版，都會規劃並驗證軟體能夠向後相容。

在詳述 Amazon 的軟體更新取向之前，讓我們先討論獨立和分散式軟體部署的差異。

獨立與分散式軟體部署

對於在單一裝置上以單一程序執行的獨立軟體，部署是不可部分完成的。兩個軟體版本絕對不會同時執行。如果獨立軟體保持狀態，新版本就必須讀取 (也就是還原序列化) 舊版寫入 (即序列化) 的資料，反之亦然。滿足此條件可保持部署的前滾和回滾安全。

在分散式系統中，部署變得複雜許多。部署是透過滾動更新進行，因此可用性不受影響。新版會一針對部分主機試運行，因此其他主機可以繼續處理服務請求。一般而言，這些主機會透過遠端程序呼叫 (RPC) 或共用持續狀態 (例如：中繼資料或檢查點) 彼此通訊。該通訊或共用狀態可能帶來其他挑戰。寫入者和讀取者執行的可能是不同的軟體版本。因此，會以不同方式解讀資料。讀取者甚至可能無法一併讀取資料，因此造成中斷故障。

協定變更的問題

我們發現無法復原最常見的原因是協定變更。例如：試想一段程式碼變更，開始在將資料保留至磁碟時壓縮資料。新版寫入部分已壓縮資料後，轉返不是一種選擇。舊版不知道從磁碟讀取之後，必須解壓縮資料。如果資料儲存於 Blob 或文件存放區，其他伺服器就會讀取失敗，即使部署正在進行中。如果該資料在兩項程序或兩部伺服器之間傳送，接收者將會讀取失敗。

有的時候，協定變更可能相當細微。例如：試想有兩部伺服器透過連線非同步通訊。為了確保得知彼此存在，這兩部伺服器同意每五秒彼此傳送活動訊號。如果伺服器沒有在規定時間內看見活動訊號，就會假設另一部伺服器已當機並關閉連線。

現在，試想將活動訊號期增加為 10 秒的部署。此程式碼遞交看似微不足道，只是改變數字而已。但改變之後，前滾和回滾都不再安全。在部署期間，執行新版的伺服器每 10 秒傳送活動訊號。因此，執行舊版的伺服器會有超過五秒沒看見活動訊號，並終止與執行新版伺服器之連線。如為大型叢集，此情況可能發生於數個連線，因而導致可用性滑落。

像這種細微的改變，很難透過讀程式碼或設計文件進行分析。因此，我們會明確驗證每一次部署的前滾和回滾安全性。

兩階段部署技術

我們確保得以安全復原的方法之一，是使用通常稱為*兩階段部署*的技術。試想以下的 Amazon Simple Storage Service (Amazon S3) 資料管理服務假設情境。該服務是在跨多個可用區域的伺服器叢集上執行，以便擴展和提供可用性。

目前，該服務使用 XML 格式保留資料。如下圖所示，在版本 V1 中所有伺服器都會寫入和讀取 XML。基於商業理由，我們希望以 JSON 格式保留資料。如果在一次部署中進行變更，取用變更的伺服器將會以 JSON 寫入。但是，其他伺服器還不知道如何讀取 JSON。這樣的情況會造成錯誤。因此，我們將此變更拆成兩部分，執行兩階段部署。

如上圖所示，我們將第一階段稱為準備。在此階段中，我們會藉由部署版本 V2，讓所有伺服器做好讀取 JSON 的準備 (除了 XML 之外)，但它們會繼續寫入 XML。從營運觀點來看，此變更並未改變任何事情。所有伺服器仍然可以讀取 XML，所有資料也依然以 XML 寫入。如果我們決定復原此變更，伺服器會恢復成無法讀取 JSON 的狀態。這一點不成問題，因為還沒有任何資料是以 JSON 寫入。

如上圖所示，我們將第二階段稱為啟動。在此階段中，我們會藉由部署版本 V3 啟動伺服器使用 JSON 格式寫入。隨著每部伺服器取用此變更，它們會開始以 JSON 寫入。尚未取用此變更的伺服器，還是可以讀取 JSON，因為它們在第一階段已做好準備。如果我們決定復原此變更，暫時處於啟動階段的伺服

器寫入的所有資料，都會是 JSON 格式。尚未進入啟動階段的伺服器所寫入的資料，則是 XML 格式。這樣的情況不成問題，因為如 V2 所示，伺服器在轉返後還是可以讀取 XML 和 JSON 兩種格式。
雖然上圖顯示序列化格式從 XML 變更為 JSON，一般技術仍適用於之前在 *協定變更*一節中描述的所有情況。例如：想一想先前提及，伺服器之間的活動訊號期必須從五秒增加至 10 秒的情境。在準備階段中，我們可以讓所有伺服器將預期的活動訊號期放寬為 10 秒，雖然所有伺服器會繼續每五秒傳送活動訊號一次。在啟動階段，我們將頻率改變成每 10 秒一次。

兩階段部署預防措施

接下來，我要說明依循兩階段部署技術時的預防措施。雖然是以上節描述的範例情境說明，實際上這些預防措施適用於多數的兩階段部署。

若取用變更的主機達到下限且回報運作狀態良好，許多部署工具會讓使用者認為部署成功。例如：AWS CodeDeploy 有一個部署組態稱為 minimumHealthyHosts。

在此兩階段部署範例中，有一個重要的假設是在第一階段結束時，所有伺服器都會升級為能夠讀取 XML 和 JSON。如有一部或多部伺服器在第一階段中升級失敗，在第二階段期間及之後，就無法讀取資料。因此，我們會明確驗證所有伺服器均在準備階段中取用變更。

我以前負責 Amazon DynamoDB 業務時，我們決定針對橫跨多項微型服務的大量伺服器，改變伺服器之間的通訊協定。我負責協調所有微型服務之間的部署，讓所有伺服器都能先達成準備階段，然後再進入啟動階段。我採用的預防措施是在各階段結束時，明確驗證每部單一伺服器已部署成功。

雖然兩個階段個別來看都可安全復原，但我們無法同時復原兩者。在先前的範例中，啟動階段結束時，伺服器會以 JSON 寫入資料。準備和啟動變更之前使用中的軟體版本，並不知道如何讀取 JSON。因此，我們讓準備和啟動階段間隔相當的一段時間，以此作為預防措施。這段時間我們稱之為製作期，持續時間通常是數天。我們等待是為了確保不需要復原至前一版。

在啟動階段後，我們就無法安全移除軟體讀取 XML 的能力。移除並不安全的原因在於，在準備階段前寫入的所有資料都是 XML 格式。我們只能在確認已經以 JSON 重新寫入每個單一物件後，才能移除其 XML 讀取能力。此程序稱為回填。可能需要可在服務寫入和讀取資料的同時執行的額外工具。

序列化最佳實務

多數軟體涉及資料序列化—無論為了保存還是透過網路傳輸。隨著技術演進，序列化邏輯改變更是家常便飯。變更內容包括加入新欄位乃至完全變更格式。這幾年來，我們逐漸採取一些序列化最佳實務：

- 我們通常避免發展自訂序列化格式。

  自訂序列化的初始邏輯看似簡單，甚至可提供更佳效能。但是，後續的格式反覆就帶來一些挑戰。許多已妥善建置的框架，例如：JSON、協定緩衝區、Cap'n Proto 和 FlatBuffers 已能克服前述挑戰。若適當使用，這些框架可提供逸出、向後相容性和屬性積存追蹤 (即是否明確或暗示指派欄位預設值) 等安全功能。

- 透過各改變，我們明確指派不同版本給序列化程式。

  這些改變與原始碼或版本控制無關。我們也會將序列化程式版本與序列化資料一併儲存，或儲存於中繼資料中。較舊的序列化程式會在新軟體中繼續運作。我們發現針對寫入或讀取資料的版本發出數據能帶來實益。若發生錯誤，可提升操作人員的可見性並為其提供故障排除資訊。前述亦一律適用於 RPC 和 API 版本。

- 我們會避免對無法自行控制的資料結構進行序列化。

  例如，我們可以使用反射將 Java 的集合物件序列化。但是，我們試圖升級 JDK 時，該類別的基礎實作可能改變，導致還原序列化失敗。同樣的風險也適用於跨團隊共用程式庫的類別。
- 一般而言，我們設計的序列化程式允許不明屬性存在。

  若可行，我們的序列化程式會在保留不明屬性的同時回寫資料。利用此調整，即使執行新版軟體的伺服器在序列化時納入新資料屬性，執行舊版的伺服器在更新相同資料時，仍不會抹除屬性。因此，不需要進行兩階段部署。

與我們的許多最佳實務相同，我們在分享時會特別強調這一套指南不見得適用於所有應用程式和情境。

驗證變更可安全轉返

一般而言，我們會透過稱為升級-降級測試的方式，明確驗證軟體變更可安全前滾和回滾。針對此程序，我們會設置一個代表實際執行環境的測試環境。多年來，我們已找出設置測試環境時應避免的幾種型態。

我曾經碰過部署變更在實際執行環境中造成錯誤，但該變更在測試環境中順利通過所有測試的情況。有一次，測試環境中的每項服務只有一部伺服器。因此，所有部署是不可部分完成的，這一點排除了同時執行不同軟體版本的可能性。現在，即使測試環境的流量不及實際執行環境，但我們會使用來自不同可用區域的多部伺服器，如同實際執行環境一樣。我們喜歡 Amazon 的節儉樸實，但為了確保品質時不可如此。

還有一次，測試環境有多部伺服器。但是為了加快測試速度，一次對所有伺服器進行部署。此法同樣造成軟體新舊版本無法同時執行。所以未能偵測前滾的問題。現在我們在所有測試和實際執行環境中使用相同的部署組態。

對於涉及微型服務間協調的變更，無論測試或實際執行環境，我們都會在不同的微型服務間保持相同的部署順序。不過，前滾和回滾的順序可能不同。例如，我們通常會在序列化內容中依循特定順序。也就是說，前滾時讀取者先於寫入者，在回滾時，則是寫入者先於讀取者。在測試和實際執行環境中，共同依循適當的順序。

測試環境設置與實際執行環境類似時，我們會儘可能模擬實際執行流量。例如，我們會接連快速建立和讀取多項記錄 (或訊息)。所有 API 會連續實行。接著，我們在環境中完成三個階段，每個階段分別持續合理的時間，目的是抓出潛在錯誤。持續時間足以讓所有 API、後端工作流程和批次任務至少執行一次。首先，我們將變更部署至約一半的叢集，這是為了確認軟體版本共存。其次，完成部署。第三步，我們會展開轉返部署，並依循相同的步驟，直到所有伺服器執行舊版軟體。如果前述階段皆未發生錯誤或預期外的行為，我們就認定測試成功。

結論

確認復原部署不會對客戶造成任何服務中斷，是維持服務可靠性的關鍵。明確測試轉返安全，可減少仰賴人工分析的需要，人工分析就容易出錯。若發現復原變更不安全，我們通常會將一個變更拆成兩個變更，且都可安全前滾和回滾。