前言

當我們在 2017 年開始撰寫 *Continuous Delivery in Java* 這本書時，Abraham Marín-Pérez 和我都知道 DevOps 將成為我們書裡的重要部分。從那時起，Java 開發人員認識和理解操作概念的重要性只增不減。隨著雲端和容器等技術、以及可觀察性（observability）和網站可靠性工程（site reliability engineering, SRE）等支援概念的興起，我們其中絕大多數的人不再「只是」開發人員；我們現在經常要負責應用程式的設計、交付和執行。因此，開發人員擁抱 ops 會是有意義的事，反之亦然。

DevOps 並不是新的詞。它已經被使用了 15 年左右。該概念最初基於**敏捷基礎架構**（*agile infrastructure*），該基礎架構源自於 2008 年多倫多的 Agile Conference，在此會議中 Patrick Debois、Andrew Clay Shafer 和其他許多人討論了傳統系統管理方法所面臨的一些挑戰。能夠對基礎架構進行「程式設計」的願望意味著軟體工程在這個領域上一直存在著影響力。在 2009 年 O'Reilly Velocity 會議上，John Allspaw 和 Paul Hammond 發表了他們的著名演講，題目為「每天 10 次部署：Flickr 的開發和營運合作（10 Deploys a Day: Dev and Ops Cooperation at Flickr）」，這鞏固了開發人員與營運人員協作的重要性。

我在 Ambassador Labs 的日常工作中看到越來越多的組織開始建構平台，以使開發人員能夠將他們的想法和程式碼快速地投入生產並呈現在客戶面前。兩個最重要的目標是快速且安全地獲得回饋，並且沒有發生當機或安全性事件。在我看來，這些平台是開發人員和營運人員有效協作的產物。

我們從 Nicole Forsgren、Jez Humble 和 Gene Kim 博士收錄在必讀書籍 *Accelerate*（IT Revolution Press，2018 年）中的研究得知，高效能組織具有較高的部署頻率、較低的改革啟動時間、更小的改革失敗率、更短的回復生產問題的平均時間。所有這些因素都受到您的平台、流程和人員的影響。開發人員必須像營運人員一樣思考，營運人員也必須接受開發原則。這一切都與共享所有權有關，而這將從共享的理解開始。這本書將幫助您發展這種共享的理解。

作為一名開發人員，在過去的十年中，您無疑一直在使用版本控制來儲存和管理您的應用程式碼。像 GitOps 這樣的現代 DevOps 實務往前更進一步，將應用程式和基礎架構程式碼以及配置都儲存在版本控制中。因此，學習本書前幾章中所介紹的版本控制系統技能非常重要。

微服務（microservice）的興起給開發人員帶來了很多機會，同時也帶來了很多挑戰。其中一些挑戰可以透過使用適當的技術來緩解。容器（container）和相關的排程（scheduling）框架，像是 Kubernetes，有助於以標準化和自動化的方式來封裝（packaging）和執行微服務。

當然，您還需要瞭解這些「原生雲端」技術對持續整合和持續交付實務的影響。從套件管理到保護工件及部屬，本書的作者在引導您完成所有相關步驟這方面做得非常出色。

我很幸運能直接從所有作者那裡學習。我已經數不清有多少次我參加 DevOps 會議時和人群四處遊蕩著，只為了看到 Baruch 獨特的大禮帽在人群中緩慢地移動著，或者撞見一個穿著皮夾克的摩托車手，才意識到那是 Stephen。從那時起，我就知道很快我就會全神貫注在有關持續交付和 Java 生態系統的有趣對話中。

作為 Melissa 在 JFrog 工作的一部分，我也密切關注她關於建構和保護容器的教導。在 Jfokus 2021 現場（而且出乎意料地！）看到她獲得 Java Champion 認可是一個美好的時刻。Ixchel 長期以來一直在 Java 和 JVM 領域具有指導性的影響力。自從我在 2013 年第一次參加 JavaOne 以來——在那裡我看到她與 Andres Almiray 共同討論多語言 JVM 的未來——我從她那裡學到了很多東西，這讓我成為了一個更好的開發人員。

多年來，我還有幸與 Ana-Maria Mihalceanu（她編寫了第 8 章）一起參與幾個小組討論，最近討論的主題包括容器、Kubernetes、和 Quarkus。我總是喜歡聽她談論新技術對 Java 平台的營運影響，而且在我離開討論後通常都會有一堆新工具要來進行研究。

無論您是對 DevOps 領域陌生的 Java 程式設計老手，還是熱衷於想要善用您所擁有的容器和雲端知識的程式設計新手，我知道您將會從本書中學到很多東西。那您還在等什麼呢？坐下來，拿起你最喜歡的飲料（也許是一杯 Java 咖啡？），並準備在和 Java 開發人員的 DevOps 工具相關的所有層面上升級吧。

— Daniel Bryant
Ambassador Labs DevRel 負責人
Java Champion
2022 年 3 月

第一章

利於（或可能不利於）開發人員的 DevOps

Baruch Sadogursky

當你酣然熟睡的時候，
眼睛睜得大大的「陰謀」，正在施展著毒手。
假如你重視你的生命，
不要再睡了，你得留神；
快快醒醒吧，醒醒！

—威廉·莎士比亞，暴風雨

有些人可能會問，DevOps 運動是否只是一場由營運人員引發來針對開發人員的陰謀。大多數（如果不是全部的話）會這樣問的人不會期望得到認真的回應，尤其是因為他們只打算把這個問題當作是半開玩笑的話。這也是因為——無論您的出身是在等式的開發那方還是營運那方——當有人開始談論 DevOps 時，大約需要 60 秒才會有人問到「DevOps 究竟是什麼？」

您應該會認為，在這個詞出現 11 年後（在這十年內，業界的專業人士已經對它進行談論、辯論和大聲呼喊），我們應該已經得出一個標準的、嚴肅的、普遍理解的定義。但事實並非如此。事實上，儘管企業對 DevOps 人員的需求呈現指數級成長，但是任何五名隨機選出來的 DevOps 員工是否能夠準確地告訴您 DevOps 是什麼，其實是非常令人懷疑的。

所以，如果您在這個話題出現時仍然摸不著頭腦，請不要感到尷尬。概念上，DevOps 可能不容易理解，但也不是完全不可能。

但不管我們是怎麼來討論這個術語或者我們同意了它的什麼定義，有一件最重要的事情要牢記：DevOps 完全是一個被發明出來的概念，發明者來自等式的營運方。

DevOps 是 Ops 方發明的概念

我關於 DevOps 的假設可能會引起爭議，但它是可以證明的。讓我們從事實開始吧。

物證 1：鳳凰專案

Gene Kim 等人（IT Revolution）的鳳凰專案（*The Phoenix Project*）自近十年前出版以來已成為經典著作。它不是一本操作手冊（無論如何，至少不是傳統意義上的）。這部小說講述了一家問題重重的公司及其 IT 經理的故事，他突然被指派來實施一項已經超出預算且落後時程數個月的攸關企業成敗的專案。

如果您生活在軟體領域的話，那麼本書其餘的中心人物對您來說會很熟悉。不過，現在讓我們先來看看他們的職稱：

- IT 服務支援總監
- 分散式技術總監
- 零售銷售經理
- 首席系統管理員
- 資訊安全長
- 財務長
- 執行長

請注意到它們之間的結締組織了嗎？他們是有史以來關於 DevOps 的最重要書籍之一的主角，而且其中沒有一個是開發人員。即使開發人員確實有出現在故事情節中，嗯……這麼說吧，他們並沒有被特別熱情地提到。

當勝利到來時，故事中的英雄（與支持他的董事會成員一起）發明了 DevOps，把專案從水深火熱中拯救出來、扭轉了公司的命運、並榮升為企業的資訊長（chief information officer, CIO）。從此每個人都過著幸福的生活——如果不是永遠如此的話，那麼至少在兩三年內，這樣的成功故事往往會讓您安居於這個產業，之後就要重新證明您的價值。

物證 2：DevOps 手冊

最好在 Gene Kim 等人（IT Revolution）的 *DevOps 手冊*（*The DevOps Handbook*）之前閱讀**鳳凰專案**，因為後者會讓您處於非常可信的人性化場景中。要沉浸在人物的性格類型、職業困境、人際關係中並不難。使用 DevOps 的方法和原因是對一些狀況的不可避免且理性的回應，而這些狀況很容易就會導致企業崩潰。利害關係、角色和他們做出的選擇似乎都很合理。不難找出和您自己的經歷相似之處。

DevOps 手冊允許您更深入地探索 DevOps 原則和實務的概念部分。正如其副標題所示，這本書在解釋**如何在技術組織中創造世界級的敏捷性、可靠性和安全性**方面大有幫助。但這難道不應該和開發有關嗎？它是否應該如此是可以辯論的。但無可爭辯的是，這本書的作者是聰明又才華橫溢的專業人士，他們可以說是 DevOps 之父。但是，在這裡呈堂的物證 2 並不是為了讚美他們，而是為了仔細研究他們的背景。

我們從 Gene Kim 開始。他創立了軟體安全和資料整合公司 Tripwire，並擔任其技術長（chief technology officer, CTO）十多年。作為一名研究人員，他專注於研究和理解大型、複雜的企業和機構中已經和正在發生的技術變革。除了合著**鳳凰專案**外，2019 年他還合著了**獨角獸專案**（*The Unicorn Project*）（稍後我會詳細介紹）。他職業生涯的一切都沉浸在營運中。即使**獨角獸**說它是「關於開發人員」，但它仍然只是從營運人員的眼中所看到的開發人員！

至於**手冊**的其他三位作者：

- Jez Humble 擔任過的職位包括網站可靠性工程師（site reliability engineer, SRE）、技術長、交付架構和基礎架構服務副總監以及開發人員關係部副總監。他就是一個營運人員！儘管他的最後一個頭銜提到了開發，但這份工作並非和它**有關**。它是和開發人員**的關係**有關。它是關於縮小開發人員和營運人員之間的鴻溝，他對此主題進行了廣泛的寫作、教授和演講。
- Patrick Debois 曾擔任 CTO、市場策略總監和 Dev♥Ops 關係總監（這個愛心是他加的）。他將自己描述為「透過在開發、專案管理和系統管理中使用敏捷技術來彌合專案和營運之間的差距」的專業人士。這聽起來確實像個營運人員。
- 在撰寫本文時，John Willis 擔任 DevOps 和數位實務副總裁。在此之前，他曾擔任生態系統開發總監、解決方案副總裁，尤其是 Opscode（現稱為 Progress Chef）的訓練和服務副總裁。儘管 John 的職業生涯更深入地參與了開發，但他的大部分工作都與營運有關，尤其是當他將注意力集中在拆除曾經將開發人員和營運人員分開成不同陣營的高牆這件事。

如您所見，所有作者都有營運背景。這是巧合嗎？我不認為。

您還是不相信 DevOps 是由營運人員推動的？讓我們看看現在試圖向我們推銷 DevOps 的領導者吧。

用 Google 查一下

在撰寫本文時，如果您在 Google 搜尋時只是為了看看會出現什麼而輸入「什麼是 DevOps？」的話，您的第一頁結果可能包括以下內容：

- Agile Admin，一家系統管理公司
- Atlassian，其產品包括專案和問題追蹤、列表製作和團隊協作平台
- Amazon Web Services（AWS）、Microsoft Azure 和 Rackspace Technology，它們都銷售雲端營運基礎架構
- Logz.io，銷售日誌管理和分析服務
- New Relic，其專長是應用程式監控

所有這些都是非常聚焦於營運的。是的，第一頁包含一個比較偏向開發方面的公司，另一個和這個搜尋沒有直接關係。關鍵是，當您嘗試尋找 DevOps 時，您會發現大部分內容都傾向於營運。

它有什麼作用？

DevOps 是一件事物！它的需求量很大。因此，很多人會想知道，具體來說，DevOps 做了什麼、它實質上產生了什麼。與其直接回答這個問題，不如從結構上來看它，將其概念化為橫向的八字形無限符號。從這個角度來看，我們看到了一個流程循環，從編寫程式碼到建構（build）到測試到發布到部署到營運再到監控，然後再返回開始來規劃新功能，如圖 1-1 所示。

如果這對某些讀者來說看起來很熟悉，那應該是因為它在概念上與敏捷開發週期（Agile development cycle）相似（圖 1-2）。

這兩個永不結束的故事之間沒有任何深刻的區別，只是營運人員將自己嫁接到了敏捷圈的舊世界，基本上將其延伸為兩個圓圈，並將他們的擔憂和痛苦硬塞到曾經被認為只屬於開發人員的國度。

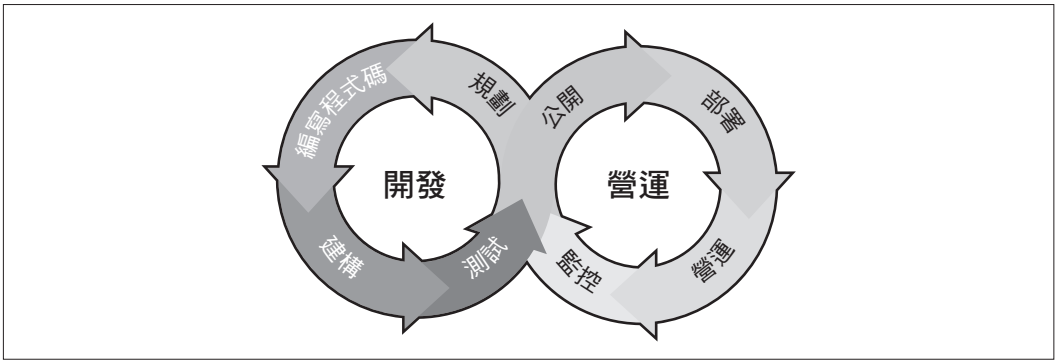

圖 1-1　DevOps 無限迴圈

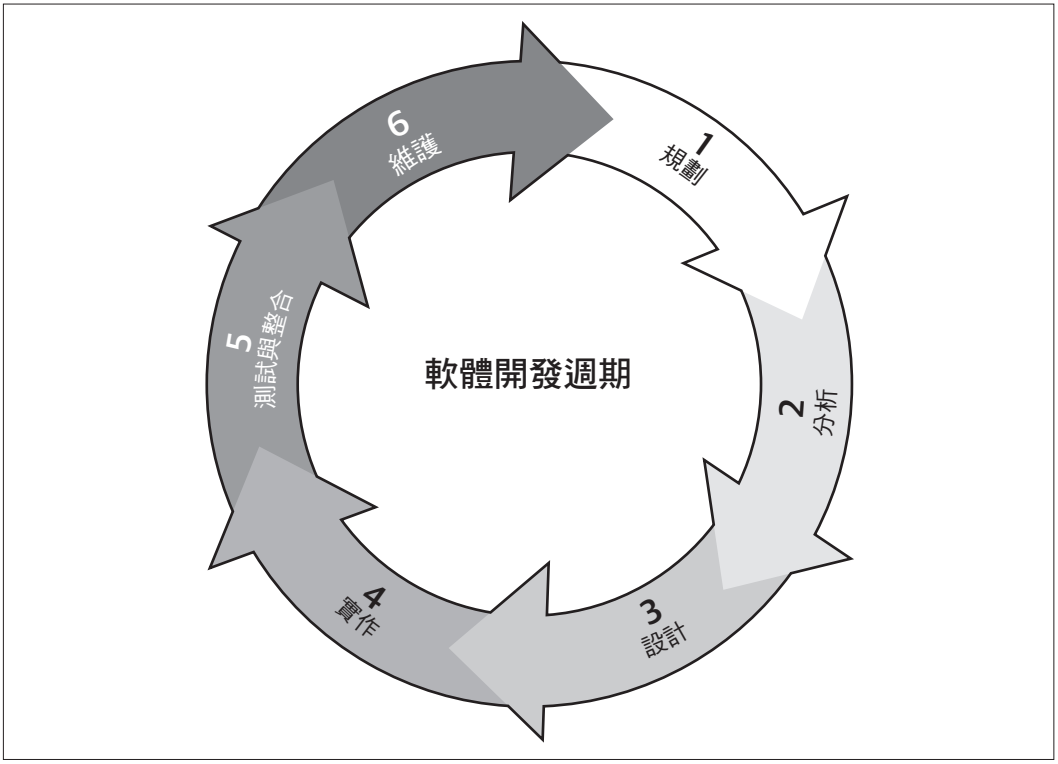

圖 1-2　敏捷開發週期

產業現狀

自 2014 年以來，DevOps 是一種被營運驅動之現象的這個事實的進一步證據，被以易於閱讀的年度資料摘要的形式出現，該摘要收集、分析和總結了來自於全球數以萬計的產業專業人士和組織的資料。「加速：DevOps 狀態（Accelerate: State of DevOps）」報告主要是 DevOps 研究和評估（DevOps Research and Assessment, DORA）的工作，是軟體行業衡量 DevOps 的位置和可能走向的那份最重要的文件。例如，在 2018 年版（*https://oreil.ly/jWjvX*）中，我們可以看到對以下問題的認真關注：

- 組織多久部署一次程式碼？
- 從程式碼提交後到在生產版本中成功執行通常需要多長時間？
- 當出現損壞或中斷時，恢復服務通常需要多長時間？
- 多少百分比的已部署變更會導致服務降級或需要修復？

請注意，所有這些都是非常以營運為中心的問題。

什麼構成工作？

現在讓我們來看看「加速：DevOps 狀態」報告和鳳凰專案是如何定義工作的。嗯，首先，有計劃的（*planned*）工作側重於業務專案和新功能，而它們涵蓋了營運和開發。內部專案（包括伺服器遷移、軟體更新和由來自已部署專案的回饋而驅動的變更）可以是廣泛的，並且可能會或可能不會比較重視在 DevOps 等式的某一方。

但是未計劃的（*unplanned*）活動，例如支援升級和緊急中斷呢？它們很偏向營運，編寫新功能的程式碼、錯誤修復和重構也是如此——這些都和怎麼透過讓開發人員參與 DevOps 的故事來簡化營運人員的生活相關。

如果我們不關心部署和營運，那麼我們的工作是什麼？

顯然的，DevOps 並不是開發人員曾經（或正在）要求的東西。這是一項讓其他人更加努力工作的營運發明。假設這個事實成立，讓我們思考一下，如果開發人員站起來說：「你的營運問題是你的，而不是我們的。」時會發生什麼呢。好吧。但在這種情況下，向叛逆的開發人員詢問他們對「完成」的定義是正確和恰當的。他們認為需要達到什麼標準才能說「我們把工作做得很好，現在我們的部分已經完成了」？

這不是一個輕率的問題，我們可以尋找一些資源來尋找答案。其中之一是軟體工藝宣言（Manifesto for Software Craftsmanship）（*https://oreil.ly/mTAUe*），雖然它不完美且經常受到批評，但它提出了應該能夠激勵開發人員的四個基本價值觀，一起來看看它們：

精心製作的軟體

是的，品質確實很重要。

穩定的附加價值

這點不會有異議。當然，我們希望提供人們需要、想要、或會想要的服務和功能。

專業人士社群

概括地說，誰會反對這件事呢？業內同行之間的熱情讓他們在專業上關係更緊密。

具有生產力的伙伴關係

協作（collaboration）當然是這個遊戲的名稱。開發人員並不反對品質保證（quality assurance, QA）、營運或產品本身。所以，在語境中，這只是對每個人都要友善（只要其他團隊不開始規定他們的工作應該是什麼）。

究竟是什麼構成了「完成」？

利用到目前為止我們已經建立的東西，我們可以有把握地說，我們需要產出簡單、可讀、可理解且易於部署的程式碼。我們必須確定非功能性要求（例如，效能、生產量、記憶體耗用量、安全性、隱私等）已經被滿足了，我們應該努力工作以避免產生任何技術包袱，如果幸運的話，可以在此過程中擺脫一些包袱。我們必須確保所有測試都已通過，我們有義務與 QA 團隊保持高成效的關係（當他們高興時，我們就高興）。

有了高品質的程式碼，加上積極的團隊領導和同儕審查之後，一切都應該會很好。透過定義了價值和附加價值標準的產品團隊，可以牢固地建立基準。透過他們的回饋，產品負責人可以幫忙確定這些基準是否已經達成（或未達成）以及達成了多少。對於優秀的軟體開發人員已經「完成」了他們需要做的事情這件事來說，這是一個很好的縮圖定義。它還展示了如果沒有營運人員的參與並與營運人員進行清晰溝通的話，就永遠無法充分衡量（甚至知道）什麼叫做「做得好」。

對抗？

所以是的，儘管可以證明 DevOps 確實不是開發人員所追求的東西，但同樣可以證明它的無限實踐對每個人都有好處。不過仍然有一些頑固分子；那些想像開發人員和 QA 測試人員之間具有競爭甚至對抗關係的人。開發人員在他們的創作上努力工作，然後覺得 QA 團隊幾乎就像駭客一樣，只為了證明一些事情而不斷地挖掘，直到發現問題為止。

這就是 DevOps 諮詢的用武之地。每個盡職盡責的開發人員都希望為自己的產品感到自豪。發現缺陷可能看起來像是批評，但實際上它只是來自另一個方向的認真工作而已。開發人員和 QA 人員之間良好、清晰、開放和持續的溝通有助於加強 DevOps 的好處，但它也清楚地表明每個人最終都在朝著同一個目標努力。當 QA 人員發現錯誤時，他們所做的只是要幫助他們的開發人員同事編寫出更好的程式碼、成為更好的開發人員。這個營運端的一些人和開發端的另一些人之間互動的例子展現了兩個世界之間的不同與區隔的有用的模糊性。他們的關係必然是共生的，並且再一次的會沿著無休止的連續活動進行，其中一方會通知另一方是為了所有人的共同利益。

比以往任何時候都多

對 DevOps 日益增長的需求既來自外部力量，也來自軟體公司本身。這是因為作為生活在 21 世紀世界的人們，我們的期望，我們所有的期望，持續地迅速變化著。我們越依賴不斷改進的軟體解決方案，我們會花在資訊和溝通的差距，以及開發人員和營運人員之間的延誤上所浪費的時間就越少。

以銀行業為例。十年前，大多數主要銀行都有相當適宜的網站。您可以登錄查看您的帳戶、您的報表和最近的交易。也許您甚至會開始透過銀行提供的電子服務來進行電子支付。雖然這些服務很好並且提供了一定程度的便利，但您可能仍然需要去（或者，至少感覺去會更舒服）當地的分行來處理您的銀行事務。

當時不存在的是如今天這樣完全數位化的體驗——包括行動應用程式、自動帳戶監控和警報、以及足夠的服務，使一般的帳戶持有者越來越普遍地在線上進行所有事情。您甚至可能不在乎您是否會再次看到實體分行的內部，甚至不知道那個分行在哪裡！更重要的是，銀行正在透過整合和關閉實體分支機構來應對這些迅速變化的銀行習慣，並為客戶提供將銀行業務轉移到線上領域的激勵措施。在 COVID-19 危機期間，這種情況進一步加快，當時分支機構的訪問僅限於預約服務、受限的入內訪問、以及更短的訪問時間。

因此，10 年前，如果您的銀行網站因為 12 小時的維護而停機，此時銀行正在部署一個更好、更安全的網站，您可能還可以從容應對。如果能產生更高品質的服務，那麼十幾個小時又算什麼？您不需要全年無休的線上銀行服務，此外，當地分行也可以隨時可以為您服務。但今天，情況並非如此。半天的停機時間是無法被接受的。本質上，您希望您的銀行**始終保持開放和可用**，這是因為您（以及全世界）對**品質**的定義已經改變，而這種變化比以往任何時候都更需要 DevOps。

容量和速度

推動 DevOps 增長的另一個壓力是儲存和處理的資料量。這是合乎邏輯的。如果我們的日常生活越來越依賴於軟體，那麼軟體所產生的資料量顯然會大幅增加。2020 年時，整個全球資料圈（datasphere）達到近 10 皆位元組（zettabyte）。而十年之前，只有 0.5 皆位元組。到 2025 年時，據合理估計（*https://oreil.ly/hvghC*）它將以指數方式膨脹到超過 50 皆位元組！

這不僅僅關乎 Google、Netflix、Facebook、Microsoft、Amazon、Twitter 等龐然大物，以及其他變得越來越大、越來越好的公司，因此需要處理大量的資料。這一預測證實，越來越多的公司將進入巨量資料（big data）世界。隨之而來的是對資料負載大幅增加的需求，以及從提供給定生產環境的精確副本的傳統模擬伺服器（staging-server）環境的離開。這種離開是基於這樣一個事實，也就是維持這種配對的方案在大小或速度方面上都不再可行。

過去那種所有東西都可以在投入生產之前進行測試的日子是快樂的，但這已經不可能了。那些軟體公司沒有 100% 信心的東西已經也將會越來越多地投入生產。這應該引起我們恐慌嗎？不用。快速發布和保持競爭力的必要性應該會激發創新和創造力，以最佳方式來執行受控的翻轉、測試程序和更多的生產內測試（in-production testing）——現在稱之為**漸進式交付**（*progressive delivery*）。這伴隨著功能旗標和可觀察性工具，例如分散式追蹤（distributed tracing）。

有些人將漸進式交付等同於爆炸裝置的爆炸半徑。這個想法是，當我們部署到生產環境中時，應該會發生爆炸。因此，要優化此類部署，我們所能期望的最好的就是盡量減少人員傷亡，盡可能減小爆炸半徑的大小。這一直是透過提高伺服器、服務和產品的品質來實現。如果我們同意品質是開發人員會關心的問題，並且它的成就是開發人員對「完成」這個詞的定義的一部分，那麼這意味著**在開發的完成時刻和接下來營運的生產時刻之間不會有停頓或脫節**。因為一旦發生這種情況，我們就會循環回到開發中，在其間修復錯誤、恢復中斷的服務等等。

完成了

也許越來越清楚的是，從營運環境中產生的期望和需求必然推動了我們進入 DevOps。因此，對開發人員的期望和要求的增加並不是來自營運人員對他們的開發人員同事的仇恨，也不是剝奪他們睡眠的陰謀的一部分。相反的，**所有**這一切，所有 DevOps 所代表的，是對我們不斷變化的世界以及它們對軟體行業全面施加的變化的現實政治（realpolitik）商業反應。

事實是，每個人都有新的責任，其中一些需要專業人員（當然是來自許多部門）**隨時**準備回應，因為我們的世界是一個不間斷的世界。這是另一種說法：我們對「完成」的舊定義已經過時了！

我們的新定義是**網站可靠性工程**（*site reliability engineering, SRE*）。這個 Google 創造的術語透過彌合開發人員與營運人員之間任何揮之不去的感知差距，而永遠的將兩者結合在一起。雖然 SRE 的重點領域可能由 DevOps 等式的一方或雙方的人員來負責，但如今，公司通常擁有專門的 SRE 團隊，專門負責檢查與效能、效率、緊急反應能力、監控、容量規劃等相關的問題。SRE 專業人員會像軟體工程師一樣思考來為系統管理問題設計出策略和解決方案，他們是讓自動化部署愈來愈能發揮作用的人。

當 SRE 員工感到高興時，這意味著建構（build）變得越來越可靠、可重複和快速，特別是因為場景是在無狀態環境中執行的可縮放（scalable）、向後和向前相容的程式碼之一，這些環境仰賴於爆炸性增加的伺服器並發出事件流以允許即時性觀察能力，並在出現問題時發出警報。當新建構出現時，它們需要被快速地啟動（我們完全能夠期望其中有些會同樣迅速陣亡）。服務需要盡快恢復完整功能。當功能無法發揮作用時，我們必須能夠立即透過 API 以程式設計方式來將其關閉。當新軟體發布並且使用者更新他們的客戶端、但隨後卻遇到錯誤時，我們必須具備執行快速無縫回轉（rollback）的能力。舊客戶端和舊伺服器需要能夠與新客戶端進行溝通。

雖然 SRE 正在評估和監控這些活動並制定戰略反應，但所有這些領域的工作完全是開發人員的工作。因此，當開發人員正在**做**（*doing*）的時候，SRE 就是今天對**完成**（*done*）的定義。

像蝴蝶一樣漂浮……

除了上面已經提到的所有考慮因素之外，現代的 DevOps（和相關的 SRE）時代還必須定義一個基本特徵：**精實**（*lean*）。這裡我們所談論的是省錢。「但是，」您可能會問，「程式碼和省錢有關係嗎？」

舉個像是雲端供應商向公司收取過多的離散服務費用的例子。其中一些成本直接受到那些企業雲端服務訂閱者所輸出的程式碼的影響。**因此，成本降低可以來自創新性開發工具的建立和使用，以及編寫和部署更好的程式碼。**

全球性、我們永不關閉、軟體驅動的社會不斷渴望著更新、更好的軟體功能和服務，這意味著 DevOps 不能只關注生產和部署。它還**必須**關注業務本身的底線。儘管這似乎是另一個負擔，但下次老闆說必須削減成本時請考慮一下。與其採取消極的、下意識的解決方案（例如裁員或減少工資和福利），不如透過積極的、增強業務概況的措施（例如改為無伺服器（serverless）環境和遷移到雲端）來達成所需的節省。然後，沒有人會被解僱，而且休息室裡的咖啡和甜甜圈仍然是免費的！

精實不僅可以節省資金，還可以讓公司有機會提高其市場影響力。當公司可以在不裁員的情況下達到效率時，他們可以保持最佳的團隊實力水平。當團隊繼續得到良好的補償和照顧時，他們將更有動力去產出他們最好的工作成果。當輸出取得成功時，這意味著客戶會開心。只要**客戶由於更快的部署而繼續快速獲得運行良好的新功能時**，那麼……他們會不斷回來尋求更多資訊並將資訊傳播給其他人。這是一個良性循環，意味著銀行裡會有錢。

完整性、身分驗證和可用性

與任何和所有 DevOps 活動攜手並肩作戰是永不消失的**安全性**（*security*）問題。當然，有些人會選擇聘請資訊安全長來解決這個問題。這很好，因為當出現問題時，總會有個人可受責備。不過更好的選擇可能是在 DevOps 框架**內**實際分析個人員工、工作團隊，和公司作為一個整體如何**思考**安全性以及如何加強安全性。

我們將在第 10 章中對此進行更多討論，但現在考慮一下：漏洞（breach）、臭蟲（bug）、結構化查詢語言（Structured Query Language, SQL）注入、緩衝區溢出（buffer overflow）等並不是新鮮事。不同的是它們出現的速度越來越快，數量越來越多，以及惡意個人和實體採取行動的聰明程度。這並不奇怪。隨著越來越多的程式碼被發布，隨之而來的問題也越來越多，每種類型都需要不同的解決方案。

隨著部署速度的加快，對風險和威脅做出更多反應變得越來越重要。2018 年發現的 Meltdown 和 Spectre 安全漏洞清楚地表明，某些威脅是無法預防的。我們正在競賽，**唯一要做的就是盡快部署修復程式**。

緊迫感

現在應該很清楚了，*DevOps 不是一個密謀，而是對進化壓力的一種反應*。它是達成以下目的的一種手段：

- 提供更好的品質
- 節省開支
- 更快地部署功能
- 加強安全性

誰是否喜歡、誰先提出這個想法、甚至它的初衷都無關緊要。重要的東西將在下一節中介紹。

軟體產業已全面擁抱 DevOps

現在，每家公司都是 DevOps 公司（*https://oreil.ly/tkSSZ*）。所以，上船吧……因為您別無選擇。

如前文所述，今天的 DevOps，也就是 DevOps 演變的結果，是一個無限循環。這並不意味著群組和部門不再存在。這並不意味著每個人都只對自己關心的領域負責，就像這個連續體中的其他人一樣。

這*確實*意味著每個人都應該一起工作。這確實意味著給定企業內的軟體專業人員必須去瞭解並合理考量所有其他同事正在做的工作。他們需要關心同行所面臨的問題、這些問題如何影響*他們*所做的工作、他們公司提供的產品和服務、以及這*整個*是如何影響他們公司的市場聲譽的。

這就是為什麼 *DevOps 工程師*（*DevOps engineer*）是一個沒有意義的術語的原因，因為它意味著存在著可以完全地和勝任地進行（或至少完全精通）在 DevOps 無限循環中發生的所有事情的人。這樣的人現在並不存在。也永遠不會存在。事實上，即使*嘗試*成為 DevOps 工程師也是一個錯誤，因為它與 DevOps 的本質完全背道而馳，DevOps 會消除隔離了程式碼開發人員與 QA 測試人員、 QA 測試人員與發布人員…等的孤島。

DevOps 是努力、興趣和回饋的結合，不斷努力建立、保護、部署和完善程式碼。DevOps 是關於*協作*（*collaboration*）的。由於協作是有機的、交流性的努力，嗯……就像協作工程並不是一回事一樣，DevOps 工程也不是（無論任何機構或大學做出了什麼承諾）。

讓它表現出來

瞭解 DevOps 是（和不是）什麼只是建立一個概念。問題是，如何在廣泛的軟體公司中合理有效地實施和維持它？最好的建議是？我現在開始說明。

首先，您可以擁有 DevOps 推動者、DevOps 傳播者、DevOps 顧問和教練（我知道 Scrum 是怎麼破壞了所有的這些詞彙，但沒有更好的了）。沒關係。但是 DevOps 不是一門工程學科。我們需要網站 / 服務可靠性工程師、生產工程師、基礎架構工程師、QA 工程師等等。但是一旦一家公司擁有了 DevOps 工程師之後，接下來幾乎肯定會擁有一個 DevOps 部門，而這將只是另一個穀倉（silo），可能只不過是一個被重新更名的現有部門，以讓公司看起來有搭上 DevOps 順風車。

DevOps 部門並不是進步的標誌。相反的，它只是回到未來。然後，接下來需要的是一種促進 Dev 和 DevOps 之間合作的方法，而這將需要創造另一個術語。*DevDevOps* 看起來如何？

其次，DevOps 是關於細微差別和小事的。就像文化（尤其是企業文化）一樣，它與態度和關係有關。您可能無法清楚地定義這些文化，但它們都是一樣的。DevOps 也與程式碼、工程實務或技術實力無關。我們沒有任何的工具可以購買，沒有分步（step-by-step）手冊，也沒有家庭版桌遊可以幫助您在組織中建立 DevOps。

這和公司內部鼓勵和培養的行為有關。其中大部分只是關於普通員工的待遇、公司的結構以及人們持有的頭銜。這是關於人們多久才有機會聚在一起（尤其是在非會議環境中），在其間他們坐下來吃飯、談論商店和非商店、講笑話等。文化是在這些空間中形成、發展、和改變的（而不是在資料中心內）。

最後，公司應該積極尋找和投資 T 型人（Ж 型更好，正如我的俄語讀者可能建議的那樣）。與 I 型人（在某一領域絕對是專家）或通才（瞭解很多，但不精通任何特定學科）相反，T 型人在至少一件事上擁有世界級的專業知識。這就是「T」字裡的長垂直線，牢牢紮根於他們的知識和經驗的深度。「T」字的上面則橫跨了其他領域所積累的能力、技術訣竅和智慧。

神人（total package）是展現出清晰而敏銳的環境適應、學習新技能、和迎接目前挑戰的傾向的人。事實上，這是對理想的 DevOps 員工近乎完美的一種定義。T 型人使企業能夠有效地處理優先事項的工作，而不僅僅是公司認為其內部能力可以承受的工作。T 型人可以看到大局並被它所吸引。這使他們成為了不起的合作者，從而導致了賦權團隊的建立。

我們都收到了訊息

好消息是，在 ops 發明 DevOps 十年後，他們完全明白這不僅與他們有關，而是與**每個人**都息息相關。我們可以親眼看到這種變化。例如，2019 年的「加速：DevOps 狀態」報告（*https://oreil.ly/vICAO*）讓更多的開發人員參與了這項研究，而不是 ops 或 SRE 人員！為了找到更深刻的證據來證明事情已經發生了變化，我們又回到了 Gene Kim 身上。同樣在 2019 年，這個透過**鳳凰專案**的幫助將等式的營運方新穎化的人發布了**獨角獸專案**（IT Revolution）。如果那本較早期的書對開發人員不屑一顧，那麼我們的英雄就是 Maxine，她是公司的**首席開發人員**（也是最終的救星）。

DevOps 始於 ops，這是毫無疑問的。但動機不是要征服開發人員，也不是在推崇營運專業人士的至高無上。它的過去和現在都是基於每個人都能看到其他人、欣賞他們對企業的價值和貢獻——不僅僅是出於尊重或禮貌，而是出於個人利益和企業的生存、競爭力和成長。

如果您害怕 DevOps 會讓你在大量的 ops 概念中不知所措，那麼實際上很可能正好相反。看看 Google（*https://sre.google*）（發明該學科的公司）對 SRE 的定義：

> 當您將營運視為軟體問題時，您就會得到 *SRE*。

那麼，營運人員現在想成為開發人員嗎？歡迎。軟體問題**始終屬於所有**軟體專業人員。我們從事解決問題的業務——這意味著每個人都有一點點 SRE、一點點開發人員、一點點營運……因為它們都是一樣的。它們都是相互交織的面向，使我們能夠為今天的軟體以及明天的個人和社會問題設計解決方案。