序

自從 Dave Farley 與 Jez Humble 寫了 Continuous Delivery 以來，持續交付社群就有一個廣泛的共識：工具並不重要。坊間有大量優秀的程式語言，也有很多組建、測試與部署程式碼的好工具，但社群的共識是：你使用什麼工具都不重要，只要避免糟糕透頂的就好了。

今年，Nicole Forsgren、Jez Humble 與 Gene Kim 博士更是鞏固了這個共識，他們的著作 Accelerate 歸納了多年來關於持續交付與 IT 性能的研究，其中一項結論是，團隊選擇工具的能力對持續交付有強烈、具統計意義的影響。因此，新的共識是：無論你使用哪些工具，只要你可以自行選擇它們，避開糟糕透頂的那些就可以了。

以我為例。我第一次在團隊中進行持續交付專案是在 2007 年的 Elsevier。我們用 TDD 與 CI 等 XP 做法，使用 Java 6、Spring 2 與 Tomcat 6 建構一個期刊網站，當時使用的管道（pipeline）是 Ant 與 Cruise Control，那個基礎程式（codebase）一直處於可釋出狀態，我們會每週將它部署到生產環境一次。

我第一次與整個公司一起做持續交付是在 2008 年的 LMAX。我們採取 XP 做法與領域驅動設計（Domain Driven Design），用 Java 6、Spring 3 與 Resin 3 來建構最先進的金融交易程式。當時使用的管道是 Ant 與 Cruise Control，以及許多自訂的儀表板，那個基礎程式一直處於可釋出狀態，我們每兩週將它部署到生產環境一次。

相信你可以看到它們的相似之處。這兩個例子都有一群聰明的人一起工作，XP 方法與完善的設計原則是不可或缺的。我們根據眼前的工作精心選擇適合的工具，而且在 LMAX 例子中，當時的開發主管寫了第一本與持續交付有關的書籍，那本書對我們的工作有很大的幫助。印象中，他的名字是 Dafydd 還是 Dev。

碁峰 www.gotop.com.tw

說了這麼多，我的重點是你可以使用 Java 或 PHP 或 .NET 成功地實作持續交付，你可以使用 Solaris Zones 或 Docker，也可以使用 AWS、Azure，或你的平台主管認為比 Azure 更便宜的內部資料中心。你只要針對眼前的問題選擇自己的工具就可以了，而且不必使用 MKS 來做版本控制、用 QTP 來做測試，或任何商用釋出管理工具，它們都很可怕。

既然對持續交付而言，工具沒那麼重要，你只要自行選擇不糟糕的選項就可以了，我為什麼要寫這篇前言呢？

如果你仔細探究，應該可以在上述內容中找到答案。工具不像持續交付的原理與做法那麼重要，但它們仍然非常重要。程式語言可以幫助人們快速地建立新功能與修復缺陷，進而減少產品開發的延遲成本。程式語言也可以實現可測試、可釋出的 app 架構，這是持續交付的關鍵推動因素。建構、測試與部署工具可以推動人們朝著正確的方向前進，執行 TDD 與 Trunk Based Development 等做法。

最近，當我整理小時候的臥房，看到大學所讀過 Ivor Horton 著作的 Understanding Java 2 時，我想起這種細微的差異。我在 1999 年與 Java 結緣，經過漫長的歲月，我們都忘了 20 周年紀念日快到了。在我看來，它是一種很棒的程式語言，多年來，Java、JUnit、Gradle、Spring 與許多其他工具已經幫我製作了許多歷經考驗、可釋出的 app，並鼓勵人們採取持續交付。

隨著雲端、容器化與無伺服器的發展，我們正朝著 Skynet 穩步前進，我們都需要經驗豐富的實踐者領導我們使用最新的工具以及實作持續交付。在本書，Daniel 與 Abraham 解釋如何使用 Java 與 Spring Boot、Kubernetes 以及 AWS EKS 等熱門的工具，頻繁地推出切合市場需求的現代 web app。使用 Java 和本書提到的任何其他工具的 IT 從業者都可以從 Daniel 與 Abraham 的教導中學到如何為 app 實作持續交付工具鏈。

— *Steve Smith*
Continuous Delivery Consulting 的持續交付顧問

碁峰 www.gotop.com.tw

所有工程團隊都應該將持續交付視為核心技術。很多人問我們，jClarity 與 adoptopenjdk.net 的 OpenJDK/Java 新 build farm 成功的關鍵因素是什麼？答案是，我們可以充滿自信地每天部署，而且只需要少數幾位工程人員。我們從 Dave Farley 與 Jez Humble 在 2010 年出版開創性的 *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*（Addison-Wesley Signature）以來，就開始使用持續交付方法了，但世上還沒有詳盡的指南教導 1000 萬位 Java 開發者做這件事。現在，這一本書問世了！

Daniel 與 Abraham 都是現實世界的實踐者，他們的書裡面有身為 Java 開發者的你在了解這個主題時需要知道的每一件事情，本書深入解釋“為什麼”你要採取持續交付方法、如何架構你的 app 來實現這種方法、如何結合組建、測試與部署管道，以及部署至雲端與容器環境的錯綜複雜之處。

“雲端原生”技術對 Java 的影響不容小覷，現代的 app 必須設法與更多外部元件連結（包括 JVM 與其他），以及使用全然不同的做法來處理傳統上由本地作業系統提供的資源（例如 I/O）。甚至連 app 的生命週期，以及它們所在的機器也發生變化，不可變的基礎設施與無伺服器等做法讓 Java 開發者必須改變思維，以充分利用這些交付 app 的新方法的新功能。

在這個全新的領域中，持續部署之類的技術、支援它的工具與架構思維，以及以雲端為中心的本地開發環境都至關重要。到目前為止，外界還沒有任何著作專門帶領 Java 開發者進入完全採用持續交付的旅程，以及持續交付提供的好處，這是第一本書。

— *Martijn Verburg,*
jClarity 與 LJC Leader CEO

— *Ben Evans,*
作者與諮詢 CTO

碁峰 www.gotop.com.tw

前言

我們為什麼要寫這本書？

我們都是 Java 開發者，在我們選擇的專業領域中，已經有夠長的年資見證並參與了幾項轉變。從我們寫出第一行 Java 程式以來，這種語言已經發展一段很長的時間了：Java 1.4 提供了 non-blocking I/O、Java 8 提供串流與 lambdas、Java 9 提供模組，最後，Java 10 提供區域變數型態推斷。部署平台也有突飛猛進的發展，雲端與容器的出現帶來許多機會與挑戰。不過有一件事沒有改變，那就是我們必須傳遞價值給最終用戶與顧客，我們必須盡量運用技術、工具與方法來提升交付軟體的效率（與樂趣），或許更重要的是，我們必須與團隊合作，並領導他們共同承擔這項責任。

隨著軟體開發、架構、部署平台的“最佳做法”越來越多，開發者通常都認同一件事：持續整合與持續交付可為軟體交付生命週期帶來巨大的價值。隨著顧客對交付的速度與穩定性的需求越來越高，你需要一個可以提供快速回饋並且實現品保及部署程序自動化的框架。但是，現代軟體開發者面臨的挑戰是多方面的，試著引進持續交付之類的方法（涉及軟體設計與交付的所有層面）代表你必須掌握新的技術，其中有些通常在開發者的舒適圈之外。

隨著職涯的發展，我們經常發現自己開始處理之前是由別人或別的團隊負責的工作，經過辛苦的過程，我們知道開發者在取得持續交付的好處之前，必須具備三項關鍵技能：

架構設計

正確地實作鬆耦合與高內聚的基礎，可以對持續測試與獨立部署軟體系統組件的能力產生顯著的影響。

碁峰 www.gotop.com.tw

自動品保

各公司對速度的需求越來越高，相關的架構風格也隨之演變（例如自成一體的系統、微服務、功能即服務等等），這代表你現在經常要測試分散的、複雜的自我調整系統，這些系統根本無法用傳統的手動程序重複且有效地驗證。

部署 app

雲端與容器技術已經為 Java app 的部署選項帶來革命性的變化，我們要學習新技能來利用它們，並建立自動化且安全的部署及釋出流程。

本書歸納我們的學習經驗，指導你如何掌握這些新技能。

你為什麼要讀這本書？

如果你是 Java 開發者，而且想要進一步了解持續交付，或正在努力接受這種軟體交付方式，這本書是為你而寫的。我們不但提供與持續交付有關的各種方法與工具的“how”與“what”，也會解釋“why”。我們相信解釋“why”很重要，因為當你了解動機之後，你就可以調整不太適合你的做法，了解做法背後的理由也可以幫助你建立紮實的基礎，並且協助你分享與教導這些概念給別人。正如同日本詩人松尾芭蕉所言“不要試圖追隨智者的腳步，而是要尋求他們尋求的東西。”

我們也希望透過這本書號召你這位 Java 開發者採取行動，離開舒適圈，學習更多關於架構、自動化與運維（operation）的知識。在目前的軟體開發職業領域中，我們認為單純編寫 Java 的職缺已經越來越少了，許多新職缺都要求具備持續交付、平台與運維工具知識。藉著投資自己，增加自己的軟體開發知識與技術，你不但可以獲得更多機會，也可以成為更好的程式員。

我們在寫這本書時，除了 Java 開發者之外，對這本書的對象類型沒有什麼概念，但是下面這些目標對象可能會引起你的共鳴：

傳統的企業 Java 開發者

你可能已經編寫 Java EE 或 Spring app 很多年了，但現在發現組織內的新 app 都圍繞著微服務風格的架構來設計，而且系統管理或運維團隊正在嘗試雲端、Docker 與 Kubernetes。你也想要進一步了解這些改變與 Java app 的建構有什麼關係，並且想要探索如何用自動化來減少測試與部署時遭遇的痛苦。

碁峰 www.gotop.com.tw

想要採取 *DevOps* 的 *Java* 開發者

你通常已經開發 Java app 好幾年了，一直都在閱讀關於雲端、DevOps 與 Site Reliability Engineering（SRE）的部落格文章、書籍與會議演說。你可能很羨慕 Netflix、Google 或 Spotify 等組織的開發方法，但也理解，他們的所做所為不一定都與你和你的團隊有關。但是，你渴望學到更多東西，並了解如何採取其中的一些概念，來逐漸轉換成 DevOps 工作風格。

剛畢業的大學或專科學生

你剛開始你的第一項專業軟體開發工作，雖然你在正規教育的歲月中學到許多具體的編程技術，但你發現自己不知道如何將所有的做法與工具結合起來，高效地交付軟體。你想要進一步了解整個軟體交付程序、填補知識的空缺，並將所有技術結合起來，以便進入下一個職業生涯階段。

不適合本書的對象

本書的重點是實現 Java app 持續交付的完整做法，因此，它不包括與架構、測試、雲端技術有關的所有內容。你當然會學到這些主題的要點，但是本書許多章節的主題都可以用一本書來說明，我們沒有時間或篇幅做到這一點。很多人都寫了專門探討特定主題的書籍了，我們會試著參考並推薦他們的心血。

本書編排慣例

本書使用下列的編排規則：

斜體字（*Italic*）

代表新的術語、URL、電子郵件地址、檔案名稱及副檔名。中文以楷體表示。

定寬字（`Constant width`）

代表程式，也在文章中代表程式元素，例如變數或函式名稱、資料庫、資料類型、環境變數、陳述式與關鍵字。

定寬粗體字（**`Constant width bold`**）

代表應由用戶逐字輸入的指令或其他文字。

碁峰 www.gotop.com.tw

第一章

持續交付：Why 與 What

本章介紹持續交付的核心概念，並說明它為開發者、QA、運維、業務團隊帶來的利益。在改變工作方式之前，你必須先問問“為什麼要這樣做？”在這裡，你將了解如何藉著實作快速回饋來減少背景切換（context switching）、為何將軟體交付給顧客時，使用自動化、可重複與可靠的釋出程序可以減少大量的壓力與挑戰，以及如何定義“完成”來實現快速驗證，並方便進行任何審核。最後，你將會看到典型的 Java 持續交付組建管道長怎樣，以及管道的各個階段的基礎概念。

背景

基本上，**持續交付**（*Continuous delivery*（*CD*））是一套有紀律的做法，可讓軟體交付團隊在短週期內生產有價值且穩健的軟體，採取這種做法時，團隊可謹慎地加入少量功能，讓軟體可隨時被可靠地釋出。從商業與技術的角度來看，這種做法可將快速回饋與學習的機會最大化。在 2010 年，Jez Humble 與 Dave Farley 出版了開創性的 *Continuous Delivery*（*https://continuousdelivery.com/*）（Addison-Wesley），書中有他們在世界各地部署軟體交付專案的經驗，這本著作仍然是 CD 的首要參考文獻，裡面有許多寶貴的技術、方法，以及從技術與組織的角度提供的建議。

在過去 20 年來，軟體開發與交付領域已經發生很大的變化了，商業需求與展望也有巨大的變化，把重點放在創新、速度與上市時間上。架構師與開發者也做出了相應的反應，為了支援這些需求而設計了許多新結構。有許多新的部署架構與平台被建立出來，並且與 DevOps、Release Engineering 與 Site Reliability Engineering（SRE）等新技術一起發展。此外還有一系列建立持續交付組建管道的最佳做法，它們都會隨著這些改變而共同演變。持續交付的核心概念是：軟體的任何更改都要經過組建、整合、測試與驗證，再決定它是否可以部署到生產環境。

碁峰 www.gotop.com.tw

在本書中，你的重點是為現代的 Java app 建立高效的組建管道（build pipeline），無論你要建立一個單體的、微服務的，或是“無伺服器”風格的功能即服務（Faas）app。

啟發開發者的能力：Why

當你開發軟體並採取任何一項重要的工作方式之前，都要問一個重要的問題：“為什麼？”身為一位 Java 開發者，為什麼你要把寶貴的時間花在持續交付與建立組建管道上呢？

快速的回饋可降低背景切換

當你處理複雜的系統時，回饋非常重要，幾乎所有軟體 app 都是複雜的自我調整系統，當今部署在 web 上，以組件為基礎的軟體系統更是如此，它們基本上都是分散式系統。回顧過去 20 年來的 IT 文獻，我們可以發現軟體開發問題通常只會在大型的失敗（而且成本高昂）發生時被發現。持續、快速與高品質的回饋可讓我們在早期偵測與修正錯誤，在問題還很小、代價還很低、還比較容易修復的時候偵測與修復它們。

快速的回饋可提供商業競爭優勢

Nicole Forsgren、Jez Humble 與 Gene Kim 在他們的著作 *Accelerate*（*https://itrevolution.com/book/accelerate/*）（IT Revolution Press）之中指出，各個產業的組織都已經開始從交付時間長的大型專案變成以小團隊來進行短週期的工作，並且評估用戶的回饋來建立取悅顧客及快速提供價值的產品。除了用數據來處理技術問題之外，你也可以和商務團隊緊密合作，找出可在 app 中實作的關鍵性能指標與數據，讓整個組織得到更多回饋。

從開發者的觀點來看，快速回饋有一項明顯的優點在於，它可以減少背景切換的成本，以及看到一段含有 bug 的程式碼時，努力回憶當初為何這樣寫的代價。不言可喻，修正五分鐘之前寫過的程式非常容易，但修正五個月之前寫過的就困難很多了。

碁峰 www.gotop.com.tw

自動化、可重複與可靠的釋出

組建管道必須快速提供回饋給開發團隊，讓他們可以在日常的工作週期中使用它們，而且管道的運維必須是高度可重複且可靠的。因此，大部分的團體都已經採取自動化了，他們的目標是實現 100% 的自動化，或務實地盡量接近它。以下是需要自動化的項目：

- 靜態分析軟體編譯與程式碼品質
- 功能測試，包括單元、元件、整合與端對端
- 所有環境提供的功能，包括 log（紀錄）、監視與警報勾點（hook）的整合
- 將軟體工件部署到所有環境，包括生產環境
- 遷移資料存放區
- 系統測試，包括非功能需求，例如容錯、性能與安全防護
- 追蹤與審核變動的歷史紀錄

“左移”思維與測試

你將會經常聽到人們談論“左移（shifting left）”程序，例如在討論安全驗證，或持續交付的驗收測試等做法時。*左移*的核心概念是“將後期的工作往前移”以做出高品質的作品，或降低“修正只會在交付程序中發現的問題”的成本。在本書提到的範例中，這代表在開始設計新功能時諮詢 InfoSec 團隊或建立一些威脅模型，或鼓勵開發者實作自動化的驗收測試，並且將它當成管道測試組的一部分來運行。持續交付是左移的催化劑，原因不但是管道可將所有的驗證階段視覺化，也提供了一個實作自動化驗證的框架。

將釋出程序自動化之後（並且讓它是可重複且可靠的），身為開發者或操作者的你，就可以相信自己能在不造成破壞或回歸的情況下，持續釋出新功能了。被迫使用不可靠且不穩定的部署程序會嚴重打擊士氣，讓團隊害怕部署程式碼，進而促使他們把大量的功能塞到一個“大霹靂”版本，最終導致更有問題的版本。我們必須打破這種負面的回饋迴圈，以小批量（理想情況下，使用單件流程（single-piece flow））來持續交付功能。

碁峰 www.gotop.com.tw

定義“完成”

快速取得回饋與將釋出程序自動化，對開發者本身有很大的幫助。但是建立組建管道的另一項好處是你可以自己定義“完成”。當一個軟體元件成功地通過組建管道時，應該就代表它可以進入生產環境、提供你原先規劃的價值，以及用可接受的運維參數（包括可用性、安全性與成本）來運行了。從歷史上看，團隊間很難就“完成”的定義取得共識，“完成”的定義很有可能成為組織內的開發與商務團隊之間的摩擦點。

確定你的目標、領域模型與用戶故事對映

雖然持續交付等做法可以協助為你和組織定義“完成”，但你也要確定需要建構什麼東西才能提供價值給顧客：

- Eric Reis 寫的 *Lean Startup*（*http://theleanstartup.com/book*）（Currency）借鑒設計思維、精益生產與敏捷方法，介紹如何藉著構思與測試商業概念與假設來持續創新。
- Jez Humble 等人合著的 *Lean Enterprise*（O’Reilly）根據 Eric Reis 和許多其他人提出的理論，展示如何用精益與敏捷原則和模式來協助組織快速且大規模地發展。
- Jeff Patton 與 Peter Economy 合著的 *User Story Mapping*（O’Reilly）可協助你釐清用戶在系統內的旅程，幫助你決定應提供的最低功能。
- Eric Evan 在 *Domain-Driven Design*（Addison-Wesley Professional）中提出的背景對映（*context mapping*，*http://dddcommunity.org/book/evans_2003/*）可用來了解你正在處理的（靜態）領域，並且幫你為“該領域與基礎程式的其他領域之間的互動”建立模型。
- Alberto Brandolini 著的 *Event Storming*（*http://eventstorming.com/*）介紹如何查明流經系統的商業事件，來以動態的方式建立領域模型。

如果你想要更了解商業價值的概念，我們推薦 Mark Schwartz 著作的 *The Art of Business Value*（*https://itrevolution.com/book/the-art-of-businessvalue/*）（IT Revolution Press）。

碁峰 www.gotop.com.tw

稍後的章節將會介紹，你可以將許多功能性與非功能（跨功能）屬性的斷言（assertion）編入現代的 Java 組建管道，包括容錯、確認已知的安全漏洞是否存在，以及基本的性能 / 負載特性（可以幫助你估算成本）。

探索典型的組建管道：What

了解持續交付管道各個核心階段的“What”或目的非常重要，因為一件事情的目標與原則通常比實作細節重要（例如，你究竟要使用 Jenkins、CircleCI、JUnit 還是 TestNG）。

組建管道的核心階段

圖 1-1 是典型的 Java app 持續交付組建管道。CD 程序的第一個步驟是持續整合（CI）。在這個步驟中，開發者將桌機內，已寫好的程式碼持續提交（整合）到一個共用的版本控制存放區，它會被自動組建與包裝成工件（artifact）。接著，CI 產生的工件會進入一系列的自動驗收與系統品質屬性驗證階段，完成後，再在越來越像生產環境的環境中進行手動用戶驗收測試與晉升。

組建管道的主要目的，是證明你對程式碼或組態做的任何修改所產生的結果都是可放入生產環境的。你提出的修改可能會在管道的任何一個階段失敗，這項修改可能會被拒絕，不會被標記成可以部署至生產環境。每一個通過驗證步驟的工件都可以部署至生產環境。我們在這個管道中收集技術與商業遙測數據，用來建立正面的回饋迴圈。

碁峰 www.gotop.com.tw

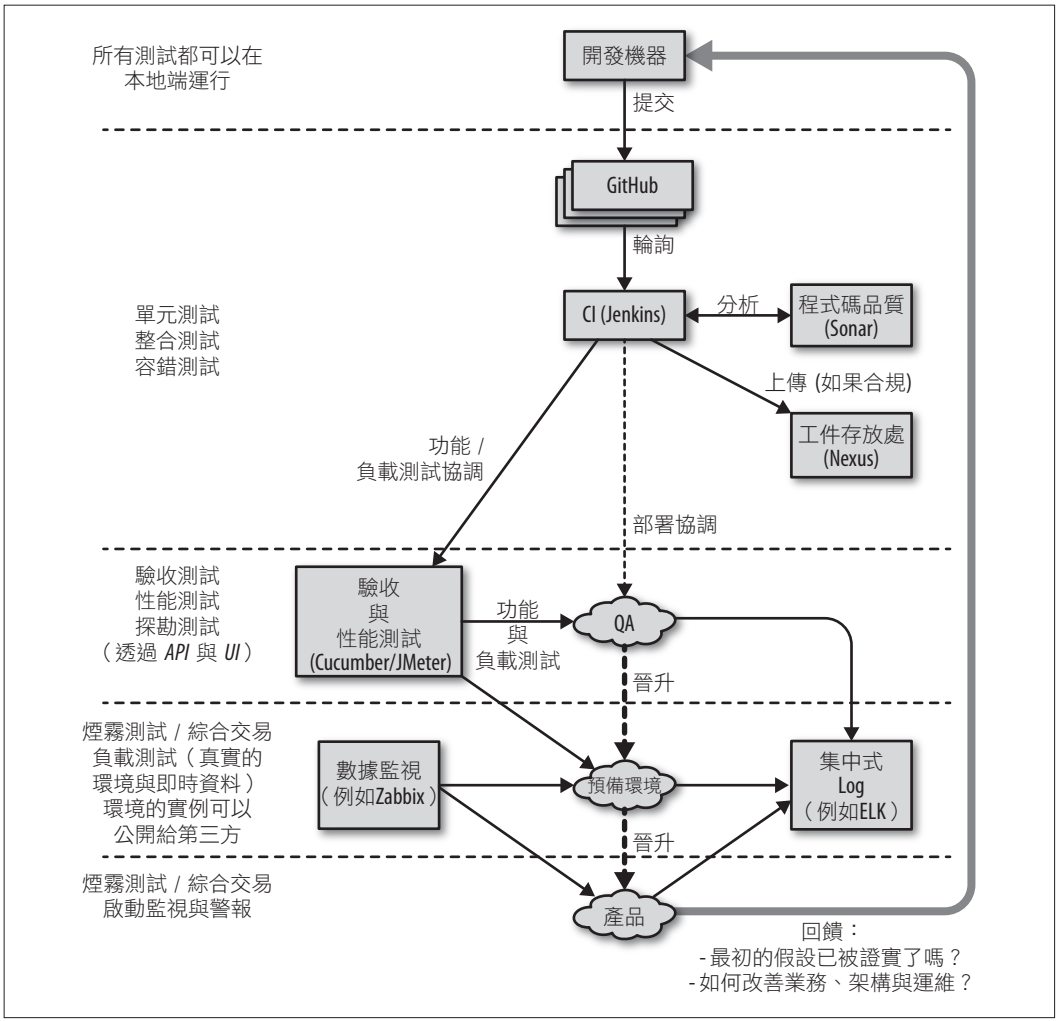

圖 1-1　典型的 Java 持續交付（CD）組建管道

接下來，我們要更深入地探討管道的各個階段的目的。

碁峰 www.gotop.com.tw

本地開發

開發者或工程師先對他們的本地版程式碼進行修改。他們可能使用行為驅動開發（BDD）、測試驅動開發（TDD）或其他極限編程（XP）做法（例如搭檔編程）來開發新功能。這個階段的核心目標就是讓本地開發環境與生產環境越像越好，例如，在一個被安裝在本地的虛擬或容器式環境中執行某些測試。

這個階段的另一個目標對較大型的 app 而言較具挑戰性：讓開發者不需要安裝所有的系統元件，或是按照順序執行它們，就可以在本地高效地進行開發工作。此時採取“鬆耦合與高內聚”之類的設計原則，對合約驗證、替身與服務虛擬化等支援做法進行測試有很大的幫助。

提交

在本地端工作的開發者，通常會將他們修改的程式碼與組態的變動提交（commit）至遠端託管的分散式版本控制系統（DVCS），例如 Git 或 Mercurial。這個程序可能需要與其他分支或主幹的變動合併，有時需要與負責基礎程式其他區域的開發者進行討論與合作，依團隊或組織已經完成的工作流程而定。

持續整合

這個階段對軟體 app 程式碼或組態變動進行持續整合（CI），使用版本控制系統（VCS）主幹的程式碼建構一個工件、單獨測試它，並且進行某種程式碼品質分析，可能使用 PMD、FindBugs 或 SonarQube 之類的工具。成功執行 CI 後，你會得到一個新的工件，它會被存放在一個集中的存放區，例如 Sonatype Nexus 或 JFrog Artifactory。

驗收測試

將成功經歷最初的單元與元件測試，以及程式碼品質評估的程式碼移往管道的右邊，在一個更大型的整合背景之下測試。此時會用少數的自動化端對端測試來驗證核心快樂路徑，或是對提供商業價值至關重要的用戶旅程。例如，如果你建立的是電子商務 app，嚴格的用戶旅程極有可能包括搜尋產品、瀏覽產品、將產品加入購物車，以及結帳和付款。

碁峰 www.gotop.com.tw

這也是驗證系統品質屬性（也稱為**非功能需求**）的階段。在這個階段執行的驗證包括可靠性與性能驗證，例如負載與浸泡（soak）測試；可擴展性，例如容量與自動擴展測試；以及安全性，包括掃描你寫好的程式碼、所使用的依賴關係，以及驗證及掃瞄相關的基礎結構元件。

驗收測試

在這個階段，測試員或實際的用戶會開始進行探勘測試。這個手動的測試會把重點放在人類認知的價值上，而不是只讓測試員照著大型的測試腳本操作。（電腦非常適合重複驗證腳本列舉的功能，那些動作應該要自動化才對。）

預備環境

被提出的修改通過驗收測試與其他基本的品保（QA）測試之後，你就可以將工件部署到預備（staging）環境了。這個環境通常很接近生產環境，事實上，有些組織也會在複製的生產環境，或真正的生產環境本身內進行測試。在這裡執行的自動化或探勘測試都要使用具代表性的資料，以及實際的資料量，工件與第三方或外部系統的整合也要盡量符合真實狀況，例如，使用沙箱或虛擬服務來模仿相關服務的特徵。

生產

經過充分驗證的程式碼終於從管道產出，被標記成“可部署至生產環境”了。有些組織會自動部署已經成功經歷組建管道，並且通過所有品質檢查的 app（這稱為持續部署），但你不一定要採取這種做法。

觀察與維護

當程式碼被部署到生產環境之後，別忘了觀察機制（監視、log 與警報），它不但可以幫商業與技術假設建立正回饋迴圈，也可以協助處理在生產環境出現的問題。

容器技術的影響

現在已經有越來越多軟體交付團隊將 Java app 包裝在 Docker 等容器技術裡面了，這種做法會改變本地開發、工件包裝與測試等工作的進行方式。圖 1-2 列出主要的四項改變：

1. 現在的本地開發，通常必須具備提供容器化環境的能力
2. 現在包裝部署工件時，會把重點放在容器映像的建立上面

碁峰 www.gotop.com.tw

3. 現在的初始化測試機制必須與容器執行環境互動，並且管理這種環境

4. 現在的開發環境通常要使用另一層的抽象，來做容器的動態協調[1]與調度

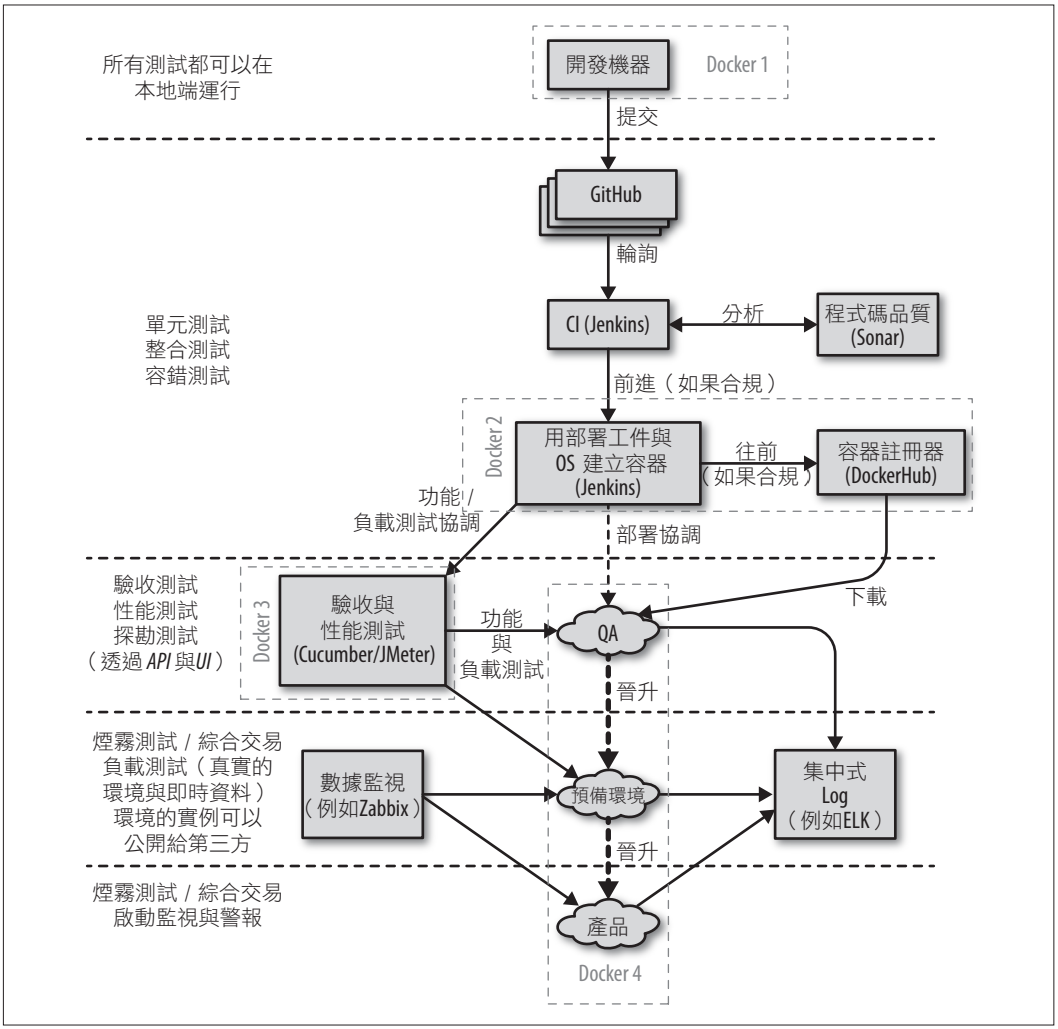

圖 1-2　使用容器技術的 Java 持續交付管道

1　orchestration，原意是管弦樂編曲，電腦科學領域用它來代表**自動**設置、協調與管理複雜的電腦系統和軟體。本書譯為“協調”。

碁峰 www.gotop.com.tw

當代架構變化

許多團隊也會用微服務或 FaaS 架構風格來建立 app，這些做法可能需要建立多個組建管道，讓每一個服務或功能使用一個。在使用這種類型的架構時，你通常要用一系列額外的整合測試與合約測試來確保你對一項服務的變動不會影響其他服務。圖 1-3 展示容器技術對組建管道步驟造成的影響，以及多服務整合帶來的挑戰（以大暗色箭頭表示）。

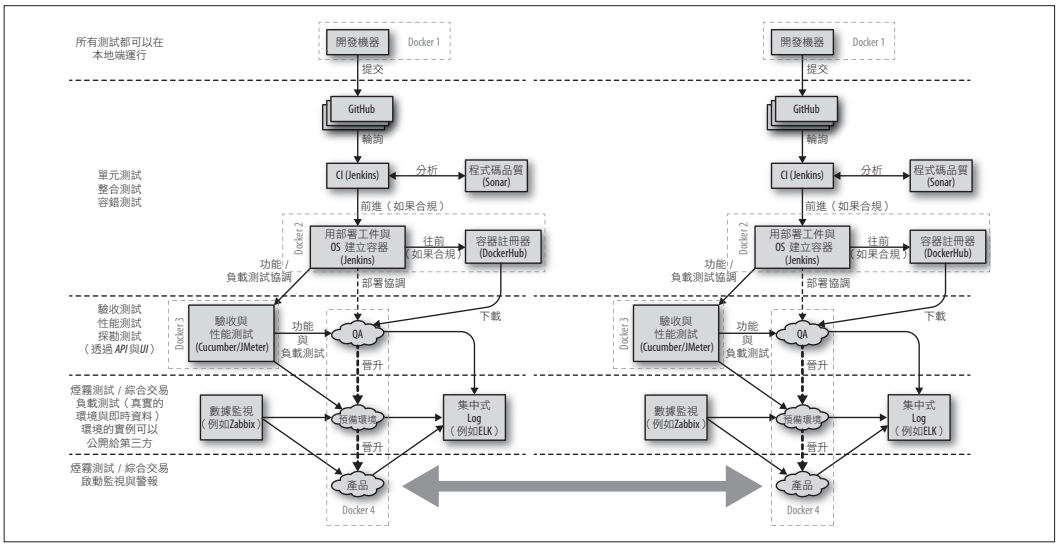

圖 1-3　容器技術與微服務結構對典型的 CD 組建管道造成的影響

本書將討論如何建立這幾種管道的各個階段，並分享我們的建議與經驗。

小結

這個介紹性的章節告訴你持續交付的核心基本概念，並探討相關的原則與做法：

- **持續交付**（*Continuous delivery*，CD）基本上是一組有紀律的做法，可讓軟體交付團隊在短週期內生產有價值且穩健的軟體。
- 對開發者而言，CD 可帶來快速的回饋（減少背景切換），自動化、可重複與可靠地釋出軟體，以及重新定義“完成”的意思。
- CD 組建管道包含本地開發、提交、組建、程式碼品質分析、包裝、QA 與驗收、非功能（系統品質屬性）測試、部署與觀察。

碁峰 www.gotop.com.tw

接下來要介紹過去 20 年來軟體交付的演變，我們把重點放在 Java app 的開發是怎麼變化的，以及持續交付如何緩解一些新的挑戰與風險。我們也會說明在 IT 領域中，持續變動與演化的需求、結構與基礎設施的最佳做法、不斷變化的角色如何改變現代軟體開發者需要的技能。

碁峰 www.gotop.com.tw