推薦序

在經歷多年數次跳槽之後，我終於在一家大型金融機構得到一份系統管理員的全職工作。負責管理伺服器和執行 Shell 腳本程式的人是在二樓，而負責開發應用程式的人員位在三樓。我從未想過為什麼我們之間會隔著一部電梯，也未曾質疑過為何多次的溝通是透過支援工單而非面對面交流。但，這就是職務架構。

在冬季的假日，我有幸成為這家公司的一員，猜猜看是誰沒有休假時間？我發現自己總是孤伶伶地坐在二樓處理支援工單。開發人員開啟了部署請求工單，而我要在每台伺服器上執行腳本程式來完成這些請求並將工單結案。終於，我對這個流程感到厭倦，於是寫了一個腳本程式，基本上能夠替我自動完成任務。在一組不同機器上執行腳本程式，現在不用 30 分鐘，一分鐘之內就能搞定，使得工單結案速度和開啟速度幾乎一樣快。有一天，一位來自三樓的人來訪，詢問處理工單為何這般迅速。我向他展示了如何透過撰寫腳本程式自動化工作流程，結果意外地獲得了一個更佳的職務。隨著工作不斷面對新的挑戰，因此我決定跳過中間環節，把自己的辦公桌搬到了三樓，使開發和維運之間的界線不再涇渭分明。

可惜好心未必有好報。管理部門告誡，我已經違背業界規則所規範的流程，並對團隊其他成員帶來錯誤的示範，因為工作內容根本沒有要求系統管理員學習程式設計、協助 Q/A 自動化測試，或者定義全新的工作方式。越是不拘一格、標新立異，就越是遭到那些墨守成規之人的反彈。

碁峰 www.gotop.com.tw

經過大約一年的時間成功完成專案，我逐漸開始理解自己那種非傳統工作風格的用處和重要性，同時也領悟到獨行俠的工作方式非長久之計，再棒的工具也無法取代一個優秀的團隊。這個時候協理（我曾經未經他的許可就擅作主張而且違背過他）剛開完一場會議回來，拉著我到一旁說：「我終於明白你在做什麼了，原來有一個詞可以形容它：DevOps（開發及維運）。」

過去十年來，DevOps 常被錯誤地用來描述現代系統管理；但實際上，為了在日新月異的環境下生存，DevOps 只是我們必須採取的最新做法之一。現代系統管理不僅僅局限於一種做法，也不能完全被一種工具或個人貢獻所決定。雖然對某些人而言，DevOps 成為專業道路發展上的一顆北極星；但太多人只走 DevOps 的路並迷失方向。這本書就像一份地圖，象徵現代系統管理的眾多起點和道路，由一位曾走過許多路的人所記錄。Jennifer 不僅提供指引方向，還提供了相關資訊，協助讀者瞭解這條路存在的原因，不僅讓你能夠追隨他人的腳步，還能幫助你開拓出屬於自己的道路。

— Kelsey Hightower

碁峰 www.gotop.com.tw

前言

當我開始做第一份系統管理員工作時，前輩告訴我需要研讀紅皮書，即 Evi Nemeth 等人所著的 *UNIX System Administration Handbook* 第二版（Addison-Wesley 出版），並出席 USENIX LISA 會議（這是首次專門針對大型網站和系統管理的會議，在當時指的是服務超過一百位使用者的規模）。那些前輩的說法沒有錯，從這兩次經歷中我學到了很多東西。研讀紅皮書讓我對硬體和 Unix 服務打下了扎實的基礎。由於紅皮書蘊涵了作者們的實務經驗和智慧結晶，它比任何現有的手冊都更有價值。在首次參加 USENIX LISA 會議時，我從 Evi Nemeth 的「系統管理的熱門主題」課程中瞭解到終身學習的重要性，並從 Mike Ciavarella 的課程中學會了系統管理文件的撰寫技巧。我在非正式聚會和資訊分享會上遇到許多其他志同道合的系統管理員，彼此在大廳走廊上交流。

除了具體的技能或技術之外，我還學到了以下幾件事：

- 系統管理工作通常涉及多個專業領域，需要不同類型團隊之間的合作。
- 冷知識有時會出奇不意地派上用場。
- 經驗對於學習和教學相當重要，這也是冷知識變得實用的原因。

儘管如此，我仍然感覺到系統管理的實務和理論之間彷彿存在一種隔閡和距離。從那時起，我發現永遠不會有*一本書*能夠指導你如何面對可能遭遇的一切狀況。當然，我們可以透過分享經驗來學習，但每個人都是在特殊環境下，為了維護特定系統而走出屬於自己的道路。

碁峰 www.gotop.com.tw

如今，系統管理員在建立、部署和執行擁有數千甚至數百萬使用者的系統時，必須學習和使用越來越多的技術以及第三方服務。

有鑒於此，我想在本書中分享一些自己的經歷，並著重在一套概論和實務精華，幫助讀者展開進行組裝、執行、擴充及最終移交系統之旅。

本書適合的對象

我特地撰寫此書，為所有系統管理員、IT 專業人士、支援工程師和其他維運工程師提供指南，一窺當代維運技術和實務的全貌。

本書也適合開發人員、測試人員以及任何希望提升維運技能的人閱讀。我明白有時團隊內的成員並非專司維運，然而這群人有義務清楚瞭解系統，才能提升自己的工作表現。

我試著將重點放在支援所有現代維運工作的準則與實務。同樣的，我也知道自身的經驗（主要是有關分散式系統的 Unix 風格管理）塑造了個人的觀點。本書所有內容與絕大多數系統管理員的工作相關。每個組織皆有不同的需求，這些需求將主導系統管理團隊的行動。舉例來說，假設你主要負責管理站台基礎設施（如 Wi-Fi 熱點、印表機和手機），那麼第三篇的內容也許與你無關。

本書不適合的對象

這本書並非工具、軟體應用程式或特定作業系統的參考指南，因為市面上有許多優質的參考資料可供深入研究這些特定主題。然而在適當的場合，我會推薦一些資料來幫忙提升讀者的技能水準。

假如你在尋找特定工具的使用手冊，渴望按部就班的學習系統管理指南，那麼本書並不適合你。在這方面，有很多關於作業系統和應用程式的書籍和資源可供參考，這裡有一些我推薦的書籍：

- 針對一般 Unix 管理方面的書籍，建議參考紅皮書 *UNIX and Linux System Administration Handbook* 第五版，由 Trent R. Hein 等人所著（Addison-Wesley 出版）。

碁峰 www.gotop.com.tw

- 針對一般系統和網路管理擁有數十年經驗的讀者，建議參考 Thomas A. Limoncelli 等人所著的兩本書籍：
  — *The Practice of System and Network Administration: V olume 1: DevOps and Other Best Practices for Enterprise IT* 第三版（Addison-Wesley 出版）
  — *The Practice of Cloud System Administration: DevOps and SRE Practices for Web Services* 第二版（Addison-Wesley 出版）
- 對於需要深入研究資料應用系統的讀者，建議參考 *Designing Data-Intensive Applications*，作者 Martin Kleppmann（O'Reilly 出版）。
- 如果你看重的是微服務管理，建議參考 *Building Microservices* 第二版，作者是 Sam Newman（O'Reilly 出版）。

本書範圍

身為系統管理員的我們，時間都集中在系統層面及整體合作上（對於我們負責的系統範圍而言）。沒有人能告訴你如何做好這一切，但我可以指導讀者善用某些方法和工具，協助掌握這門領域，令你更有自信並與其他同事建立良好關係。

若是只能告訴你一件事

系統，基本上就是凌亂的。假設在某個地方，有人已完美地掌握了管理系統的方式，他們的流程和工具能夠維持系統的完好如初。當然，一些有經驗的人會樂於分享建議，儘管這些建議有所幫助，但仍須牢記下列幾點：

- 他們的經驗可能不適用於你的環境或困難挑戰。
- 他們不懂所不知道的事情。他們或許不曉得影響他們成功結果的其他因素。
- 他們的最佳方案也許可行，因為他們的系統反映了其設計和執行方式。

你不再是一人孤軍奮戰。有時候，光靠直覺不見得行的通。科技發展日新月異，規則不斷改變，世上沒有真正的萬事通。無論你是涉獵廣泛、擁有通才的知識，或是擁有一門深入、專精的知識，但知識仍然有限。採取合作方式可以讓你從多個角度增加見識，有效地管理你的系統。與他人合作表示採取不同於平常的做法。合作需要傳達意圖，讓他人能夠更加理解你正在處理的問題，以及解決問題的重要性與過程。

碁峰 www.gotop.com.tw

若是只能告訴你另一件事

當系統出現錯誤且問題無可避免，不要默默地獨自承受壓力。既然錯誤已成既定事實，那就勇於尋求協助。一旦肩負著維護系統方面的重大責任，這種壓力就會對身心健康產生負面影響。有諸多方法可以讓你的系統運作漸入佳境，毋需為了追求完美無瑕的系統而犧牲健康。唯有善待自己，才能擁有悠長的職業生涯。

本書結構

本書使用了以下排版格式：

斜體（*Italic*）

　表示新名詞、網址、電子郵件地址、檔名和副檔名。

等寬字體（`Constant width`）

　用於程式碼顯示，以及段落內用於表示程式碼的相關敘述，例如變數和函式名稱、資料庫、資料類型、環境變數、陳述式以及關鍵字。

等寬粗體（`Constant width bold`）

　用於顯示應由使用者直接輸入的指令或其他文字。

此圖示代表提示或建議。

此圖示代表一般注意事項。

此圖示代表告誡或需要注意的事項。

碁峰 www.gotop.com.tw

致謝

撰寫一本書極為不易。在一場數百萬人喪生、全球系統被壓垮，造成疾病大流行期間下寫一本書，內心實在五味雜陳（特別是寫一本關於管理系統的書）。

由衷感謝眾多人的協助，令本書得以付梓。

非常感謝 Evi Nemeth（*https://oreil.ly/vmPXm*），以她自己的「系統管理」聖經和座談指導，在系統和網路管理領域建立了分享和終身學習的文化。

感謝那些審閱初稿並提供回饋意見的人們：Chris Devers、Yvonne Lam、Tabitha Sable、Brenna Flood、Amy Tobey、Tom Limoncelli、David Blank-Edelman、Bryan Smith、Luciano Siqueira、Steven Ragnarök、Æleen Frisch、Jess Males、Matt Beran 和 Donald Ellis。對於定稿內的任何錯誤，我都會承擔全部的責任。

感謝 Chris Devers，從早期的初稿章節開始，你就一直參與其中，提供你個人的想法、措辭和經驗觀點。

特別感謝 Tomomi Imura 為本書繪製的插圖，他的才華真是令人嘆為觀止。

感謝 O'Reilly 整個團隊，你們使得本書得以實現。特別感謝 Virginia Wilson，她是一位非常有耐心的開發編輯，在幫助我找到合適措辭一事上發揮關鍵的作用。有了她的支援，本書內容和我的寫作能力都有飛躍性的進步。

在本書撰寫期間，我對給予出來的關愛和包容感到無比感激。如果沒有 Brian 的支持，維持家庭的運轉和幫忙照顧 Frankie，並成為我的第一位讀者，這本書就不可能完成。謝謝你 Frankie，讓我始終保持樂觀和無限的想像力。Frankie、Brian 和 George，我實在太愛你們了。

非常感謝所有積極參與 USENIX LISA、SREcon、CoffeeOps 和 DevOps 社群的人，分享他們的故事並為產業技術的發展做出貢獻。在此，我要向你們所有人致上深深的敬意。

碁峰 www.gotop.com.tw

現代系統管理簡介

系統是由一組元件及它們之間的關係所組成，形成一個複雜的完整體系。你的基本目標是從系統引發的混亂當中，以永續方式有效管理你的系統。系統管理方法沒有標準答案，但在理解系統的過程中，你可以選擇不同的路徑來減少身體和心理上的負擔，並建立一個終身的事業，面對令人感興趣的挑戰。

本書編排之目的旨在提供讀者準備旅程所需的資源，採用現代系統管理的技術、工具和實踐方法。在這個簡介當中，將替各位提供一些更高層次的目標，幫助你制定自己的路徑，以可靠和永續的方式管理你的系統。

展開你的地圖之旅

在很多方面，系統管理員就像踏上荒野的徒步者。如圖 I-1 所示，我們希望在某個地方有一張地圖，確切告訴我們該做什麼及何時去做；只要遵循這張地圖，就能實現完美的系統維護。想像自己即將踏上明亮的道路，慶幸找到的地圖具備了明確的里程碑和目標。

然而現代系統管理更像是圖 I-2，你可以透過一些萬用工具來為旅程做好準備：基本和關鍵實務包括組裝、監控系統及增減系統的規模。你無法預測旅程上具體需要的工具有哪些，以及如何使用它們。一旦機會降臨，就能準備好做出這些決策並付諸實行，而且你不必獨自一人去做這些事情！

碁峰
www.gotop.com.tw

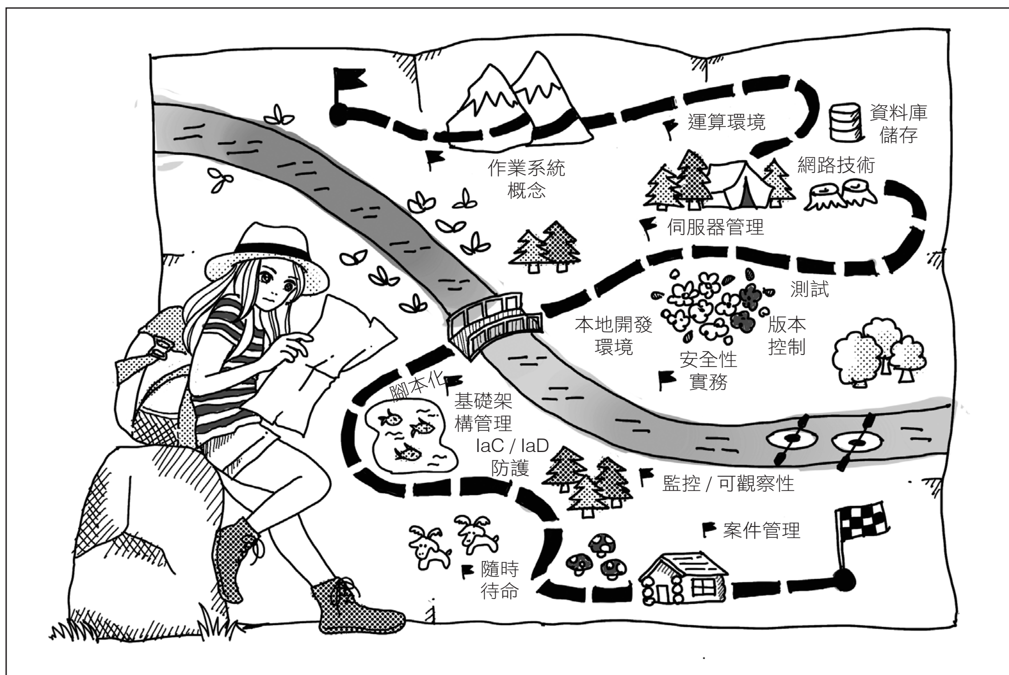

圖 I-1　這張圖片是大多數人心中想像的情景：一張清晰的地圖，具有明確的目標和獨自的旅程，幫助我們找到正確的資源並學到正確的觀念。很遺憾，這並不現實（圖像作者 Tomomi Imura）。

你必須根據每個組織和加入團隊的需求來調整實現高效系統管理的旅程。最終的結果，里程碑和目標會有所不同。

在徒步旅行時，你不會知道沿途道路的每一個轉向。即使走過相同的路徑，可能會遇到新的挑戰：道路被沖毀，或者你不想引起野生動物的注意。在系統管理中，你會遇到意料之外的問題（如同道路曲折和轉向），這些問題會影響你的努力結果。因此，你要從錯誤中吸取教訓，嘗試不同的路線，尋求幫助，並不斷努力，直到抵達目的地。

碁峰 www.gotop.com.tw

圖 I-2　沒有任何資源可以確切告訴我們該如何管理我們的系統。前方的道路不明，地形永遠不會和地圖相符；但憑藉著正確的工具和合作夥伴，我們可以充滿信心地前進，因為我們相信能夠應付未來的任何挑戰（圖像作者 Tomomi Imura）。

這本書提供讀者建立模式和行為的訣竅，把時間和精神集中在需要的地方，以便建構出優質、可靠和永續的系統。你所肩負的責任規模和範圍不盡相同，或許你需要負責一切，平衡整個組織和特定工程的主導權；也可能管理「IT 骨幹」以及公司的業務運作，甚至必須支援一個產品的特定基礎設施。

當問題浮現時，你要在不傷害自己身體和心理健康的情況下維護系統。當你達到目標時，你的工作並未完成。對於終身職業來說，隨著技術和實務的演變，你必須不斷調整以適應新的道路和地形。

碁峰 www.gotop.com.tw

接受心態的轉換

為思維的成長開始做好準備，相信自己可以隨著時間增長自己的能力和才華。你可以持續更新自己的技能和知識，並堅持面對失敗和挑戰。

在整本書裡，我分享了不同的模型，讓你能夠思考你所管理的系統。模型有助於理解和交流，有助於解釋概念和想法，並提供彼此共同的交談方式。沒有一種模型是完美的，也不可能做到。當你思考這些模型所代表的系統時，請記住梵谷所寫的話：「你的模型不是最終目標」[1]。當維護你的系統時，若模型未能替你提供一個良好的框架，那就要當心。

利用架構程式碼和五層互聯網這類的模型，可以處理、建立視覺化和解釋你的系統。從經驗中獲得啟發，制定新模型，促進系統管理的實踐和技術發展。

現代系統管理的關鍵事實是系統的規模和複雜度日益增加，因為「軟體在吞噬這個世界」。無論是採用新的實務還是技術，為了提高效率，必須認識變化，增進對實際工作的理解。

這份工作是什麼？

你的職責是建立、配置和維護可靠且永續的系統，而系統可以是特定的工具、應用程式或服務。儘管組織內的每個人皆應關心系統的運行、效能和安全性，但你的角度更應專注於這些數據的測量，同時考慮組織或團隊的預算約束，以及工具、應用程式或服務使用者的具體需求。

無論你是管理數百個還是數千個系統，只要在系統上具有升級的權限，你就是系統管理員。不幸的是，許多人從工作相關的任務或個人工作內容來定義系統管理。一般情況下，這是因為這個角色界定模糊，常常超出自己的責任範圍，承擔所有其他人不想做的工作。

1 梵谷在給弟弟的信中引述狄更斯說：「你的模型不是最終目標，而是賦予你思想和靈感的形式和力量的手段」（*https://oreil.ly/5nkDi*）。

碁峰 www.gotop.com.tw

許多人將系統管理員形容為「工友」的角色[2]，負責在系統出現問題時進行修繕的工作，尤其是在系統無法正常運行的時候。儘管公司裡的清潔工角色至關重要，然而將這兩個職位劃上等號，對兩者都不公平。

對系統管理員更貼切的比喻包括水電工或空調專家。人們往往把現代住宅和企業具備水、電和空調控制系統視為理所當然，但這些系統需要訓練有素的專家來建置、安裝、維護和修復，以確保它們的安全和運作正常。

角色的各種講法

在過去 10 年裡頭，我對「系統管理員（sysadmin）」的角色產生了不諧調的感覺。「系統管理員」究竟是什麼，這困惑我很久了。系統管理員是維運者嗎？系統管理員是擁有 root 權限的人嗎？隨著人們試圖拋下過去，頭銜和稱謂方面出現了爆炸性的成長。當有人對我說：「我不是系統管理員，我是基礎架構工程師」，我意識到不僅僅是我有這種感覺。

一些組織已經重新命名了他們的系統管理員職位，以便跟上業界變化的潮流。不要因為職稱而限制了你的機會。

形形色色的系統管理

負責管理系統的職稱包羅萬象，如系統管理員、網站可靠性工程師（SRE）[3]、DevOps 工程師、平台工程師及雲端工程師等。角色的各種稱謂表示需要稍微不同的技能。例如，「SRE」通常是指工程師同時是具備維運技能的軟體工程師。對於 DevOps 工程師的看法，人們通常假定工程師至少擅長一種現代程式語言，並擁有持續整合和部署方面的專業知識。更常見的情況是，這只是一種稱呼，不一定是統稱。有時候團隊會定義完全不同的角色，並根據組織的需求來要求特定的技能。為了避免和期望有落差，在評估某個角色是否適合自己時，應直接與團

2 請翻閱 Thomas Limoncelli 等人所寫的書《系統和網路管理實務》（Addison-Wesley 出版），在附錄 B（*https://oreil.ly/JYWCK*）中列出許多系統管理員的角色。

3 請從 Alice Goldfuss 的部落格文章〈如何成為 SRE〉（*https://oreil.ly/wALwU*）和 Molly Struve 的部落格文章〈成為網站可靠性工程師的意義〉（*https://oreil.ly/35Es6*）中，瞭解更多有關成為 SRE 的資訊。

碁峰 www.gotop.com.tw

隊進行確認。例如在不同組織內，SRE 縮寫可能表示網站、系統、服務可靠性或是彈性工程的意思。

作為一門工程專業來說，系統管理的一部分是藝術，另一部分是科學。它是一種對工作的處理方法（如設計、建置及監控系統），需要考量到安全性、人為因素、政府法規、實用性和成本的影響。有數百種不同的方式可以完成某件事。你的知識、技能和經驗將決定你的做法，同時利用你的分析能力來監控影響和成功率，決定何時花費（或節省）金錢或時間，並考慮支援系統所需的人力成本。

擁抱不斷演變的實務

隨著技術的發展，管理技術的實務也在不斷適應。要隨時準備好採用新技術，才能跟上不斷變化的平台，減少系統的影響和維護性。

當衡量系統的可靠性，且組織為了改善可靠性而進行轉型時，基本的系統管理和開發就會自動發生變化。如今更常見的做法是每個人參與提高產品的可靠性，而非一個團隊獨自承擔大部分支援工作以保持系統或服務的運行。SRE 團隊有權幫助減少系統的總工作量[4]。

擁抱合作

對於環境變化的速度、複雜性以及失敗固有的風險，建議採取以下行動：

- 聚集來自不同領域的專業人才（如開發、維運、防護和測試）。
- 以整合提案取代妥協，使最終解決方案能夠涵蓋多個角度。

建立信任感和心理安全感需要真正的努力，以鼓勵人們表達他們的意見和觀點。當團隊成員在彼此之間產生心理安全感時，他們會感到放心、勇於冒險、不怕失敗並且願意表現脆弱的一面。例如，團隊中的個人如果感受到高度的心理安全感，他們將會積極分享自己需要幫助的情況。這是因為建立了相互支援機制，有助於防止系統的失敗。

4 請從 Stephen Thorne 發表有關 SRE 原則的 Medium 文章中，來瞭解減少辛勞及其對團隊的影響（*https://oreil.ly/SpiwZ*）。

碁峯 www.gotop.com.tw

鼓勵提問的文化讓人們勇於提出深入的問題，幫助每個人達成共識（朝著同一個目標努力），並增加智慧與勇氣（專家也會犯錯）。一些問題包括：

- 為什麼？為什麼我們要這樣做？為什麼它運作方式是這樣的？
- 你能幫助我理解你的觀點嗎？
- 你有考慮過其他解決這個問題的方式嗎？

這是 Google 人力營運部門透過「re:Work」研究計畫，找出高效團隊的首要關鍵動力，供讀者瞭解更多關於心理安全感的資訊（*https://oreil.ly/uTpZU*）。

擁抱合作有助於和他人共事愉快。當你需要他們的時候，你的合作夥伴就會提供支援且樂於這麼做，因為你已經和他們建立了良好的人際關係，等待的就是這一天。

擁抱永續性

*永續性*是衡量一個系統的能力，使系統中的人們能夠成長茁壯，在工作的同時過上健康的生活。無論你的工作規模如何，有八種指標可以衡量工作的永續性：

效能

衡量系統在一段時間內執行實質工作的能力。系統效能的定義視建構的服務或產品而異。

可擴展性

衡量系統在新增和移除個別元件的適應能力。

有效性

衡量系統按預期運行的時間長短。

可靠性

衡量系統在一段時間內如何始終有效地執行其特定目的。

可維護性

衡量部署、更新和淘汰系統的輕鬆程度。

碁峯 www.gotop.com.tw

簡單性

衡量新進工程師理解系統的輕鬆程度。

易用性

衡量用戶對系統的滿意度。

可觀察性

當系統出錯時，衡量你對系統觀察的瞭解程度。不過，並非所有系統都需要高度的觀察性。

在接下來的章節中，我將分享不同的技術和實務，改善讀者對衡量標準所設定的目標，進而提高系統的永續性。

總結

你的旅程會根據系統和支援這些系統的人而有所不同。無人能提供完美定義的檢查清單來告訴你需要學習或執行的項目及執行時機。但你可以透過適當的工具箱（理解基本原則和關鍵實務及組建、監控和擴展系統）來替自己做好準備。

成為一名系統管理員的意義不斷在修正。運用新的技術和實務，對於思維的成長與維持終身職業所需的才能和技能將有所助益。

請求他人的協助並築起合作關係，透過建設心理安全感來有效地與團隊協力合作。運用模型來補足你的理解，並以此基礎來增進系統管理實務。

擁抱永續性可以使你成長茁壯，並擁有一個支撐你管理系統的完整職業生涯。

碁峰 www.gotop.com.tw

第一篇

關於系統推論

第一篇共分四個章節，主要介紹系統的基礎知識以及面對不同解決方案時如何做出抉擇。用「最佳」一詞來談論解決方案的意義並不大。相反地，讀者需要瞭解有哪些可用的方案，對什麼情況是「最適合」的，以及它在系統中的環境。這個環境涵蓋了一系列不斷變化而相互衝突的目標、人員和不同要素，彼此構成運轉的系統。「系統思維」鼓勵你思索系統的各項組成，及它們和目前問題之間的互動關聯，使你更透徹的理解系統的演變。

碁峰
www.gotop.com.tw

第一章

互連模式

打個比方，設想一下你正在與朋友一起製作蛋糕。你已按照食譜來混合所有的食材（油、麵粉、雞蛋和糖），且外表看起來相當不錯，可是當你嚐味道時，有些地方卻顯得不太對勁。要成為一位成功的烘培師，你必須掌握蛋糕所有的成分（麵粉與脂肪的比例等），以及它們如何影響成品的品質（例如味道與口感）。舉例來說，如圖 1-1 範例，我們的烘培師沒有意識到芝麻油對於以製作蛋糕來說，芝麻油並不是合適的油品。

換個角度來看，把*烘培師*換成*系統管理員*，將烘培師手中原料的黃金比例換成你系統內相互連接的電子元件（譬如，智慧型手機、嵌入式裝置、大型伺服器和儲存陣列）。要成為一位成功的系統管理者，你必須瞭解這些元件如何以協同模式進行連結，從而影響你的系統品質（例如可靠性、可擴充性與可維護性）。

本章將協助讀者對系統進行分析，並檢視其中的互連模式，以理解系統設計背後的思維。

碁峰 www.gotop.com.tw

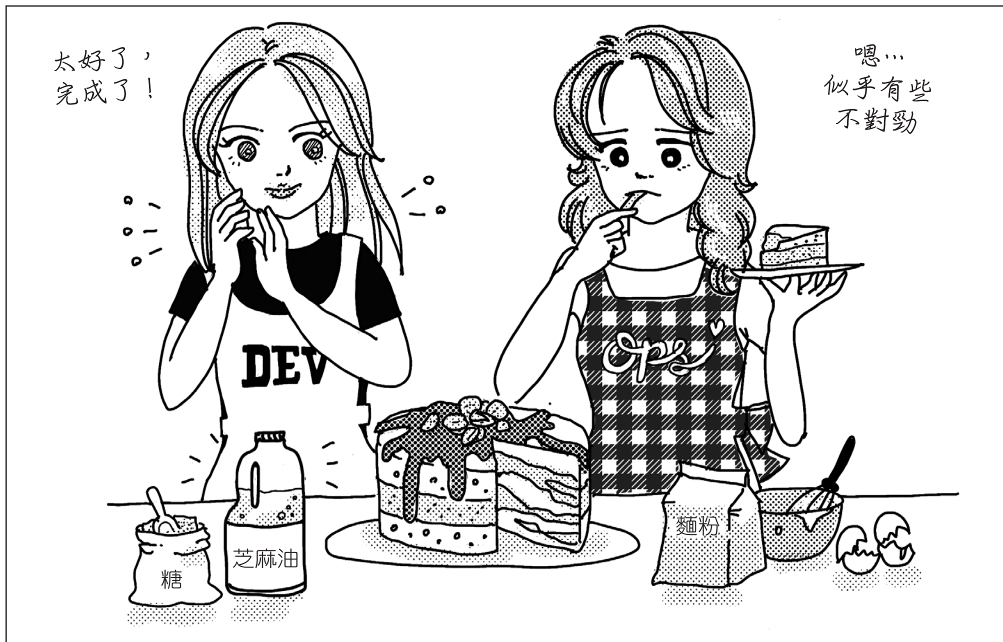

圖 1-1　系統建模的知識（圖像作者 Tomomi Imura）

如何建立連結

工程師們選擇架構模式，可處理代表工作負荷的非一次性解決方案（例如批次處理、網路伺服器和快取）。這些模式亦稱為模型，其中傳達了對系統設計的理念共識[1]。

從企業內部到雲端運算環境，可重複使用的解決方案正不斷發展茁壯，能夠支援切割開來的小型服務[2]。這些模式取決於系統元件及元件彼此間的連接來塑造系統的可靠性、可擴充性與可維護性。

1　見 Martin Fowler 在 martinfowler.com 網站所發表的「Software Architecture Guide（*www.martinfowler.com/architecture*）」最後一次修訂於 2019 年 8 月 1 日）

2　詳見 Sam Newman *Building Microservices* 第三章（*https://oreil.ly/SSx0B*）（O'Reilly），以瞭解更多關於拆解服務的詳細資訊。

碁峰 www.gotop.com.tw

我們將針對系統設計使用到的三種常見架構模式進行探討，讓讀者可以看到這些架構的使用如何影響（和限制）它們的變革與系統品質（可靠性、可擴充性與可維護性）。

分層式架構

最廣為人知的模式是通用分層式或分層架構模式。工程師們通常將這種模式應用於用戶端——伺服器應用程式，諸如網頁伺服器、電子郵件和其他商業應用。

工程師將元件組織到水平層之中，每一層扮演特定角色，將每一層的關注點與其他層分開。各層通常緊密耦合，具體取決於與相鄰層的請求和回應。因此，你可以在每一層當中更新和部署元件。

雙層系統由用戶端和伺服器組成。三層系統包括一個用戶端和另外兩層的伺服器；表現層、應用層和資料層經由抽象化而成為不同的元件。在多層次架構系統中，每一層可以拆分為獨立的邏輯層。根據系統的需求（例如彈性、安全性和可擴充性），可能超過三層以上。隨著每一層的加入，可擴充性和可靠性也會隨之增加，因為各層在關注點之間分隔開來，能夠單獨進行部署和更新。

微服務架構

微服務系統是一種分散式架構，並非分層式架構，而是由眾多小型獨立的業務程式單元所組成。微服務屬於小型的自主服務。由於每個服務都是獨立的，因此可以單獨開發和部署程式。此外，每個服務都可以針對其使用案例來採用最適合的語言或框架。

微服務提高了系統的可擴充性和可靠性，因為它們可以根據需要來獨立部署，並與系統中的故障點隔開。

將服務拆解為微服務會降低可維護性，因為這會增加系統管理員的認知負荷。若要瞭解系統，讀者需要理解每個獨立服務（即語言、框架、建置和部署管道以及任何相關環境）的所有詳細資訊。

碁峰 www.gotop.com.tw

事件驅動架構

事件驅動架構是一種分散式非同步模式，支援應用程式之間的鬆散耦合。不同的應用程式不知道彼此的詳細資訊。相反的，它們是透過發佈和處理事件進行間接交流。

事件定義為發生了某事，是可以被追蹤的事實[3]。系統產生事件。在事件驅動的系統中，事件生產者會建立事件，代理者引入事件，而事件消費者收取和處理事件。

事件驅動系統有兩種主要模式：訊息傳遞（或發佈 / 訂閱）和串流。

事件生產者或發佈者將事件發佈到事件訊息傳遞系統內的代理者。代理者將所有已發佈的事件傳送給事件消費者或訂閱者。訊息代理者從發佈者接收已發佈的事件，維護接收到的訊息順序，使其可供訂閱者使用，並在事件被處理後加以刪除。

在事件串流系統中，事件被發佈到分散式日誌（僅允許附加的永久性資料儲存）。因此，事件消費者可以處理他們想要的串流中的事件，並且可以回溯該事件。此外，分散式日誌會在事件被使用後保留事件，這意味著新訂閱者可以在訂閱之前訂閱已發生的事件。

由於元件的組成屬於鬆散耦合，因此系統的各部分不必顧慮其他元件的運作情況是否良好。鬆散耦合的元件可以獨立部署和更新，從而提高了整個系統的彈性。事件持續性允許系統回溯發生失敗時的事件。

表 1-1 歸納了讀者將在系統看到的三種常見架構模式比較：可靠性、可擴充性和可維護性。

表 1-1　架構的可靠性、可擴充性和可維護性比較

	分層式	微服務	事件驅動
可靠性	中（與系統緊密耦合）	高	高
可擴充性	中（限制於各層）	高	高
可維護性	高	低（降低簡潔性）	中（降低可測試性）

3 CloudEvents（*https://cloudevents.io*）是一項由社群推動的工作，旨在定義以一種標準方式描述事件資料的規範，以實現橫跨不同的服務和平台。

碁峰 www.gotop.com.tw

想當然爾，這些並不是讀者在系統設計範疇所看到的唯一模式。詳情請參考 Martin Fowler 的網站 the Software Architecture Guide（*https://oreil.ly/Sf5IC*）。

元件之間的交流方式

系統的元件並非自我隔離於外，每個元件都會與系統的其他元件進行通訊，這種通訊可能由架構模式來通知，譬如用於多層式架構的 REST（*https://oreil.ly/CmRCT*），還有用於事件驅動架構的 gRPC（*https://oreil.ly/MzO9n*）。

有若干不同的模型用於表示元件如何通訊，例如網際網路模型、五層式網際網路模型、TCP/IP 五層式參考模型，以及 TCP/IP 模型。雖然這些模型看起來非常相似，但它們有細微的差異，這些差異也許會告知人們如何思考它們為通訊而建立的應用程式與服務。

當個人或一組工程師發現需要改善的地方時，他們會撰寫建議需求（Request for Comment，RFC），並呈交給同儕進行審查。網際網路工程小組（IETF）（*https://oreil.ly/ydfJn*）身為一個開放性的國際標準組織，採用一些建議的 RFC 作為定義官方規範和協定的技術標準，致力於維護和改善網際網路的設計、易用性、可維護性和互通性。這些協定規範定義了設備之間如何相互通訊，同時大致遵守網際網路模型。隨著網際網路的發展和人們需求的變化，這些協定也不斷日新月異（有關這方面的範例，請參閱附錄 B）。

如表 1-2 所示，五層式網際網路網模型顯示了五個不連續的分層。每一層以自己特有的協定來透過介面與上下層做溝通。對系統進行分層能夠劃分每一層的責任，並允許建立（和修改）不同的系統部分，也允許各層之間的差異化。

表 1-2　五層式網際網路模型與協定範例

分層	協定範例
應用層	HTTP, DNS, BGP
傳輸層	TCP, UDP
網路層	IP, ICMP
資料鏈結層	ARP
實體層	銅纜、光纖

碁峰 www.gotop.com.tw

就像圖 1-1 的蛋糕一樣，沒有任何脆皮層能夠準確地告知你哪裡出現了問題。實作協定並不要求嚴格遵守規範，也不需要層層相疊。例如，邊界閘道協定（BGP）決定了傳輸資料的最迅速和最有效率的路由協定。由於該協定的實作，人們可以將 BGP 歸納為應用層或傳輸層協定。

網際網路模型的各層提供了一種框架環境，令讀者能將注意力集中於應用程式的要素：原始碼和其相依性，或位在較低的實體層，將複雜的通訊模型簡化為易於理解的單元。然而，也許偶爾會遭遇環境簡化的情形，卻仍無法協助你理解究竟發生何事的狀況。欲要提升理解能力，需瞭解各部分如何相互運作，以及它們如何影響系統的品質。

瞭解更多協定的相關資訊

如果你的職責需要管理網路或實作協定，請查閱 Andrew Tanenbaum *Computer Networks*（Pearson），Kevin R. Fall 和 W. Richard Stevens *TCP/IP Illustrated, Volume 1: The Protocols*，2nd edition（Addison-Wesley）和 RFC 系列網站（*https://oreil.ly/1DYQT*）。

接下來讓我們更詳細地瞭解應用層、傳輸層、網路層、資料鏈結層和實體層。

應用層

首先從網際網路模型的頂層開始探討應用層。應用層描述了應用程式經常直接互動的所有上層協定。此層的協定處理應用程式介面該如何與下層的傳輸層傳送和接收資料。

若要瞭解應用層的觀念，請將重心放在基於應用層協定所實作的函式庫或應用程式。譬如，當客戶使用坊間流行的瀏覽器存取你的網站時，將執行以下步驟：

1. 啟用瀏覽器呼叫函式庫，使用網域名稱系統（DNS）獲得網頁伺服器的 IP 位址。
2. 瀏覽器發起 HTTP 請求。

DNS 和 HTTP 協定是在網際網路模型的應用層之內運作。

碁峯 www.gotop.com.tw

傳輸層

Internet 模型的下一層（傳輸層）負責處理主機之間的網路流量。有兩種主要協定：傳輸控制通訊協定（TCP）和使用者資料封包通訊協定（UDP）。

從過往經驗來看，UDP 源自於基礎協定，如 PING/ICMP、DHCP、ARP 和 DNS，而 TCP 一直是備受矚目的協定（如 HTTP、SSH、FTP 和 SMB）之根本。然而協定不斷在改變，因為在特殊背景下，較 UDP 更加可靠的 TCP 會談品質，此時存在著效能瓶頸。

UDP 是一種無狀態協定，竭盡所能的嘗試傳輸訊息，但不會驗證網路遠端裝置是否接收到訊息；另一方面，TCP 是一種連接導向的協定，它使用三向交握與網路遠端裝置建立可靠的會談。

UDP 的基本特徵如下：

非連接式

UDP 不屬於會談導向的協定。網路遠端裝置無須事先建立會談即可進行封包交換。

損失性

不支援錯誤檢測或內容修正。應用程式必須自行提供容錯機制。

非阻塞性

TCP 容易受到「佇列前端擁塞（head of line blocking）」問題的影響，其中遺失封包或非持續性接收資料可能導致會談卡關，需要從發生錯誤的地方重新傳輸。使用 UDP 時，非持續性傳輸不會造成問題，應用程式可能會選擇性要求重新傳輸遺失的資料，而無須重新傳送已成功送達的封包。

相較之下，TCP 的基本特性如下：

確認回應

接收方會通知傳送方每個封包的接收狀況。接收方的確認回應（Acknowledgment）並不代表應用程式已接收或處理資料，僅表示它到達了預定目的地。

碁峰 www.gotop.com.tw

連接導向

傳送方在傳輸資料之前先建立會談。

可靠性

TCP 負責追蹤傳送和接收的資料。接收方的確認回應可能會遺失，因此接收方不會依照順序去檢查資料段；反倒是會傳送上次監測到的有序封包或重複增加的確認回應。這種可靠性會增加延遲時間。

流量控制

接收方通知傳送方能夠接收多少資料。

請注意，TCP 或 UDP 的設計並未涵蓋安全性。換句話說，這些協定的初始設計缺乏安全性，將造成應用程式和系統實作的複雜性增加，以及協定內容的額外更迭。

網路層

網路層位於網路模型的中間層，網路層在傳輸層和資料鏈結層之間進行傳輸，使得封包基於特有的分級式位址進行傳送，即 IP 位址。

網際網路協定（IP）掌管兩個系統之間的位址，以及資料分割的規則。它要處理網路介面的唯一識別碼，以使用 IP 位址傳送封包。當資料以較小的最大傳輸單元（MTU）傳送通過鏈路，IP 會根據需要拆解和重組封包。IPv4 是最為被廣泛部署的 IP 版本，其 32 位元定址空間是由四組八位數的二進制數字或四組十進位數字（又稱為四段式十進位位址表示法）的字串所組成。IPv6 標準相較之下則更具有優越性，諸如 128 位元的定址空間、更加複雜的路由能力，以及對群播位址的支援更全面。不過，IPv6 推廣的速度很慢，部分原因是 IPv4 和 IPv6 不具互通性，且相較於將所有設定移植到新標準而言，彌補 IPv4 的缺陷通常會更容易些。

底層二進位定義會影響到十進位表示法的範圍，即 IPv4 位址的範圍從 0 到 255。此外，RFC 還為私有網路（*https://oreil.ly/QkHVN*）制訂了保留範圍（私有網路無法透過公共網際網路進行路由）。

網路層的 IP 協定著重於提供遠端裝置唯一的位址，以便與之進行互動，但它不負責資料鏈結層的傳輸，也不處理會談管理（會談管理是在傳輸層進行）。

碁峰 www.gotop.com.tw

資料鏈結層

接下來，資料鏈結層利用實體層來傳送和接收封包。

位址解析協定（ARP）處理探索已知 IP 位址的硬體位址。硬體位址亦稱為媒體存取控制（MAC）位址。每個網路介面控制器（NIC）為它指派一個唯一的 MAC 位址。

業界希望 MAC 位址是全球唯一的硬體位址，因此網路管理設備及軟體為了設備身分驗證和安全性，而假定此位址為獨一無二。否則出現在同一網路上的重複 MAC 位址可能會導致問題。由於硬體製造的生產環節失誤（或軟體設計有意使 MAC 隨機化）[4]，確實有機會出現重複的 MAC 位址。

儘管如此，虛擬化系統也可能會發生這種情況，例如從參考映像檔複製的虛擬機器（VM）。倘若多台網路主機回報它們具有相同的 MAC 位址，則會發生網路機能障礙且增加回應的延遲。

人們可以透過軟體來掩飾網路上的 MAC 位址。這種方法稱為 *MAC* 欺騙（MAC spoofing）。一些攻擊者利用 MAC 欺騙展開第 2 層攻擊，試圖劫持兩個系統之間的通訊以入侵其中一個系統。

反向位址解析協定（RARP）會檢查 IP 位址與硬體位址的對應關係，有助於找出同一個 MAC 位址是否具有多個 IP 位址的回應。若是讀者認為網路存在兩台設備共享一個 IP 位址的問題，可能是因為某人設定了靜態 IP，而 DHCP 伺服器早已事先將相同的位址指派給另一台主機的關係。RARP 有助於查明始作俑者。

實體層

實體層將來自上一層的二進位資料流轉換為底層硬體傳輸的電子、光波或廣播訊號。每種類型的物理介質都擁有不同的最大長度和速度。

4 有關 MAC 隨機化問題的更多資訊，請參閱「MAC 位址隨機化：以犧牲安全性和便利性為代價的隱私」（*https://oreil.ly/31Hsf*）。

碁峰 www.gotop.com.tw

即使在使用雲端服務時，讀者不必管理機架上的實體伺服器，但仍然須要關心實體層。兩節點之間的實體路由延遲可能會突然增加。假如資料中心出現突發狀況，譬如雷擊或松鼠損毀電纜（*https://oreil.ly/miV4u*），資源就會被重新導向到遠方的備援服務。此外，資料中心內的網路設備可能需要重新啟動，或者纜線降速，或者網卡出現故障。因此透過這些實體元件傳送的請求可能會遇到更多的延遲。

總結

在讀者的環境內，你會遇到這些模式（分層、微服務和事件驅動）與協定。理解系統的架構可以明白元件如何相互關聯和通訊，而你的需求會影響系統的可靠性、維護性和可擴充性。

在下一章，我將分享如何思考這些模式與互連，以及它們如何左右你的機構對運算環境做出的選擇。

碁峰 www.gotop.com.tw

第二章

運算環境

讓我們從運算環境開始，一步步深入鑽研推理系統的基本原理。

本章將探討系統的基石：運算。**運算**是一個通用術語，用於描述與一套資源（係指處理能力、記憶體、儲存設備和網路）相關的執行個體。現代的運算不僅僅是系統的技術實現；它還涉及支援建立、設定和部署組織時所需的協同處理方法。在本章裡，我們將討論如何區分運算基礎架構的類型和環境，讓讀者能夠量身打造組織或團隊的需求和技術。

常見的工作負載

工作負載的特性泛指應用程式對系統所佔用資源的壓力值與類型。

讀者管理的系統會有許多應用程式或服務是在生產環境內進行安裝、維護和執行。你所掌控的每個應用程式或服務皆有一套最基本及建議的運算需求（CPU、記憶體和儲存設備），透過這些需求可將應用程式歸類成：CPU 密集型、記憶體密集型和儲存密集型。

我將分別在第 3、4 章分享更多有關儲存和網路的知識。

碁峰 www.gotop.com.tw

CPU 密集型的應用程式直接受益於高效能處理器。工作負載的例子包括：

- 批次處理
- 遊戲伺服器
- 高效運算（HPC）
- 多媒體轉碼
- 機器學習
- 科學建模
- 網頁伺服器

記憶體密集型的應用程式直接受益於容量更高的記憶體，且大部分的執行時間耗費在讀取和寫入資料。工作負載的例子包括：

- 快取
- 資料分析
- 資料庫

儲存密集型的應用程式直接受益於低延遲存取時間和隨機 I/O 運作。工作負載的例子包括：

- 資料倉儲
- 資料湖泊
- 資料庫
- 分散式檔案系統
- Hadoop
- Log 或資料處理應用程式

瞭解系統中包含哪些類型的工作負載，有助於評估建立系統所需的資源。就雲端架構來看，通常會根據系統的工作負載最佳化（CPU、記憶體和儲存最佳化）來做出選擇。

碁峰 www.gotop.com.tw

在雲端上建置系統時，毋須做出萬無一失的選擇。只要有合理的預算開支，儘管採用的方案並非完美，但亦相去無幾。請參閱第 11 章，瞭解如何利用 infracode 來建置基礎架構。

選擇運算工作負載的位置

讀者的運算環境可能位於採用自動化維運和控管的私人資料中心，有時稱為*本地部署*（on-prem）。或者，你可以利用服務供應商所提供的*雲端運算*服務。

本地部署

在本地運算環境中，可以透過租賃或購買硬體配備來代管所需的服務，並滿足組織的需求。

你可以針對每個應用程式的工作負載狀況來部署不同的資源。標準化的硬體可簡化部署和設定，但可能會使某些應用程式導致系統資源的不足，同時造成其他應用程式無法充分利用。針對特殊的應用程式需求部署專用的配備，能夠更佳合理化資源的消耗和效益，但會增添管理基礎架構的複雜性。

採用專屬的硬體配備時，可針對不同應用程式來部署不同規格的配備，具體取決於它們是 CPU、記憶體還是儲存密集型應用程式。擁有不同的硬體配備會增加伺服器部署和系統軟體設定的複雜性。

倘若你的組織專營資料中心管理，或者你有客製化的需求，而傳統雲端服務供應商無法提供這些服務時（譬如法規要求），維護私有的資料中心也許相當具有優勢。

碁峰 www.gotop.com.tw

權衡新的處事法則

管理資料中心（從採購設備到部署和管理硬體）是我職涯當中感到新鮮有趣的一件事。我負責管理價值數百萬美金的設備，以支援需要橫跨各種儲存和網路設備，進行硬體測試和評估的開發環境。自從有了這次經歷，我瞭解到自己擅長解決複雜的專案和廠商管理的問題，以及掌控交貨與裝機的時程表。但我真的不喜歡所有涉及文書處理的工作，也不喜歡鋪設電纜和追蹤系統元件這些枯燥乏味的工作。我很慶幸雲端運算的問世和唾手可得的運算資源。

在第 17 章，我將會談論到工作上的監控，從中找出你喜愛的工作。從事自己擅長卻不喜歡做的工作，真的很容易令人陷入迷惘。

部署標準化硬體時，可能會發現在同一系統上部署相互輔助的應用程式大有助益。舉個例子，在同一系統上代管網頁伺服器和資料庫伺服器（兩層式系統架構）可能運作良好，因為它能使服務之間的網路延遲降到最低。另一方面，同時執行這些服務有可能會導致 CPU、記憶體或儲存之間的競爭，使得縱向或橫向改變的決策變得更加困難。留意資源受限服務的熱點和閒置服務的冷點，並思考如何取得最佳平衡，提高系統的能力以滿足未來的需求。

雲端運算

服務供應商通常會使用不同的術語來介紹他們所提供的服務。某些情況會引起混淆，尤其是同一個專有名詞往往代表不同的涵義[1]。

請檢視圖 2-1。此術語表顯示了這些不同概念的重疊性。在堆疊的最上方，功能即服務（FaaS）代表使用者對執行的運算基礎架構擁有最小的控制權。在堆疊的最底部（使用 VM），代表在硬體方面擁有最大的控制權和靈活性，同時也得付出最多的維運成本。

1 以此概念為例，讀者可以參考 Julia Evans 所撰寫的部落格文章（*https://oreil.ly/bCBhT*），其中解釋了 Google Cloud 和 Amazon Web Services 所提供的身分識別與存取管理（IAM）工具之間的差異。

碁峯 www.gotop.com.tw

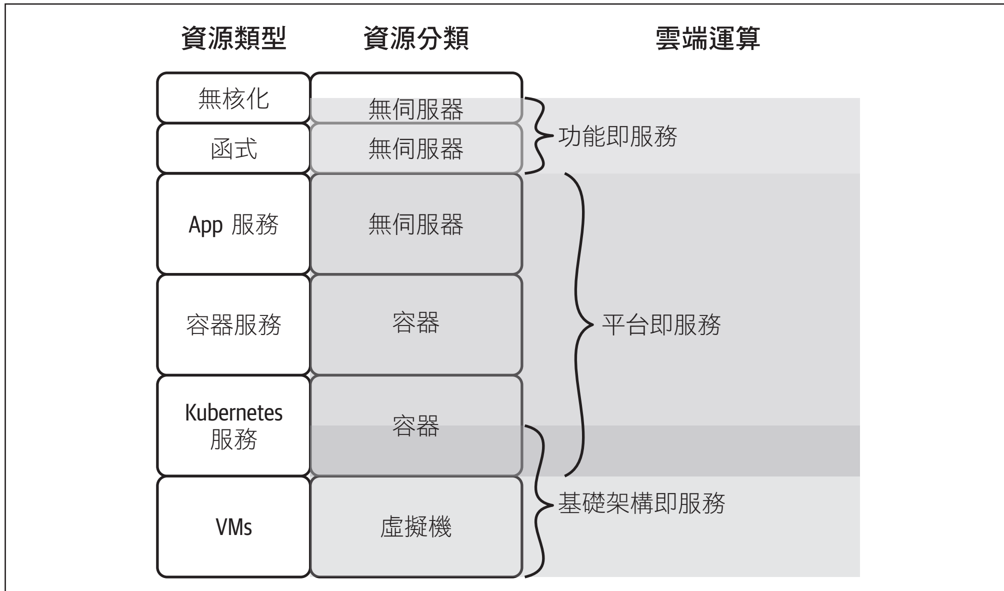

圖 2-1　雲端運算環境

運算選擇

讓我們看一下運算環境內不同類型的運算（無伺服器、容器和虛擬機），以便有一個標準的參考架構。

無伺服器化（Serverless）

無伺服器化架構涵蓋了無核化、函式和應用程式服務（視供應商而定，有時也包括容器）。

無核化（Unikernelss）

無核化屬於輕量級且不可改變的作業系統（OS）映像，被編譯為執行單獨的程序。基於其特殊性和佔用的空間大小，我們將它列入在內。

MirageOS 是其中一個用於建立無核化最早的程式庫作業系統。如想親身體驗學習更多有關無核化的知識，歡迎使用 MirageOS 的教學網站（*https://oreil.ly/fwZex*）。

碁峰 www.gotop.com.tw