



## 前言

在國外，可觀測性 (Observability) 這個詞已經成為熱門話題引起熱烈討論，在 CNCF (Cloud Native Computing Foundation) landscape 中有一個區域專門介紹可觀測性 (Observability) 相關工具與技術。在國外研討會中，發現越來越多人探討如何透過可觀測性來實現 DevOps 更高效率。

近幾年，台灣也開始看到越來越多相關文章與研討會分享，分享有關可觀測性的觀念與實踐經驗。在吸收這些知識與寶貴知識後，對這主題有更深入的了解與認識。因此希望自己可以藉由 ITHome 鐵人賽的機會，整理並分享自己關於可觀測性的理解與小小心得。希望藉由透過這次不要臉分享，可以讓對這主題也有興趣的朋友們能夠更快了解可觀測性的重要觀念，一起入坑。

- 作者 : Marcus Tung
- 部落格 : [m@rcus 的學習筆記](#)
- 粉絲團 : [m@rcus 的學習筆記](#)

# 目錄

- 1. 起心動念
- 2. 關於主題
- 3. 淺談 Observability (可觀測性)
  - 3.1. 想要解決問題
- 4. 淺談 Devops
  - 4.1. 想要解決什麼問題
- 5. Devops 與 Observability 關聯
- 6. 小結
- 7. 參考連結
- 8. 業務連續性
- 9. 業務連續計畫 Business Continuity Plan
  - 9.1. BCP 核心概念
  - 9.2. BCP 計畫類型
  - 9.3. 計畫執行重要元素
- 10. 小結
- 11. 參考連結
- 12. 服務級別協議
  - 12.1. 可用性
  - 12.2. SLA 與 SLO
- 13. 如何計算 SLA
- 14. 小結
- 15. 參考連結
- 16. 災難復原 Disaster Recovery
  - 16.1. RTO : Recovery Time Objectives
  - 16.2. RPO : Recovery Point Objectives
- 17. SLA 與 RTO , RPO 關係
- 18. 小結
- 19. 參考連結
- 20. 災難復原計畫 Disaster Recovery Plan
- 21. 現況盤點
  - 21.1. Network & Application Server
  - 21.2. Database
- 22. 小結
- 23. 參考連結
- 24. 可用性與可靠性
- 25. 共同特性
  - 25.1. 兗餘 Redundancy
- 26. 小結
- 27. 參考連結
- 28. 什麼是監控
  - 28.1. 監控類型
  - 28.2. 監控指標(Framework)

- 28.2.1. USE
- 28.2.2. RED
- 28.2.3. Golden
- 29. 小結
- 30. 參考連結
- 31. SLA、DR、Availability 和 Reliability：系統穩定性的關鍵因素
- 32. 可觀測性是甚麼？
- 33. 為什麼可觀測性這麼重要：解決了哪些問題？
- 34. 小結
- 35. 參考連結
- 36. 遺留監控的挑戰
- 37. 可觀察性與監控有何不同
- 38. 小結
- 39. 參考連結
- 40. 演進史
  - 40.1. 1960：可觀測性和控制理論
  - 40.2. 2013：Twitter 可觀測團隊描述其使命
  - 40.3. 2016：Twitter 可觀察性的（四個）支柱
  - 40.4. 2017：可觀察性的三大支柱
  - 40.5. 2018：可觀測年
  - 40.6. 2019：激烈辯論隨之而來
  - 40.7. 2020：可觀察性（重新）定義
- 41. 小結
- 42. 參考連結
- 43. Observability Signals
  - 43.1. Metrics：指標
  - 43.2. Structured Log：結構化日誌
  - 43.3. Distributed Tracing：分散式追蹤
- 44. 三者關聯性
- 45. 超越「三大支柱」：Continuous Profiling
- 46. 小結
- 47. 參考連結
- 48. 可觀測性管道
  - 48.1. 獨立組件
  - 48.2. 統一採集 Unified Collection
  - 48.3. 統一儲存 Unified Storage
- 49. 核心階段：Collect 採集
  - 49.1. Agent
  - 49.2. Collectors
  - 49.3. Pipelines
- 50. 小結
- 51. 參考連結
- 52. 看見全貌
- 53. 草稿的進化
- 54. 如何選擇可觀測性工具
  - 54.1. SaaS 平台

- 54.2. [自行架設](#)
  - 54.2.1. [CNCF Landscape 是什麼？可以吃嗎](#)
- 55. [可觀測性專區](#)
- 56. [小結](#)
- 57. [參考連結](#)
- 58. [為什麼需要分散式追蹤](#)
- 59. [什麼是 OpenTelemetry](#)
- 60. [使命、願景和工程價值](#)
  - 60.1. [使命\(Mission\)：無所不在的高品質、可攜式遙測](#)
  - 60.2. [願景：有效的可觀測性世界](#)
  - 60.3. [工程價值：相容性、穩定性、彈性和效能](#)
- 61. [分散式技術追蹤的演進](#)
- 62. [核心元件](#)
- 63. [小結](#)
- 64. [參考連結](#)
- 65. [核心元件](#)
  - 65.1. [OpenTelemetry Specification：規格](#)
  - 65.2. [OpenTelemetry SDKs：程式語言的 API 實作](#)
- 66. [小結](#)
- 67. [參考連結](#)
- 68. [核心元件 \(2/2\)](#)
  - 68.1. [OpenTelemetry Protocol \(OTLP\)：傳輸可觀測性數據協定](#)
  - 68.2. [OpenTelemetry Collector：接收、處理和匯出遙測資料的方式](#)
    - 68.2.1. [組成](#)
    - 68.2.2. [不負責後端及呈現結果](#)
    - 68.2.3. [彈性與生態系](#)
  - 68.3. [OpenTelemetry Semantic Conventions：通用數據定義方法](#)
- 69. [參考連結](#)
- 70. [不明覺厲的 OpenTelemetry Demo 專案](#)
  - 70.1. [快速開始](#)
  - 70.2. [工具](#)
  - 70.1. [應用程式架構](#)
  - 70.2. [分散式重要觀念](#)
  - 70.3. [模擬異常：讓他爆](#)
  - 70.4. [Trace：使用 Jaeger 定位問題](#)
  - 70.5. [Metrics：使用 Grafana 查看 OTel Collector 狀況](#)
- 71. [參考連結](#)
- 72. [Telemetry Data Flow：遙測數據蒐集流程](#)
  - 72.1. [OpenTelemetry Demo](#)
    - 72.1.1. [OTel Collector](#)
    - 72.1.2. [Prometheus](#)
    - 72.1.3. [Jaeger](#)
    - 72.1.4. [Grafana](#)
- 73. [Manual Instrumentation](#)
- 74. [參考連結](#)
- 75. [可觀測性開源工具的挑戰](#)

- 76. [解決方案 : Observability Platform ?](#)
- 77. [參考連結](#)
- 78. [Grafana 起源](#)
- 79. [Grafana Cloud 是什麼](#)
  - 79.1. [Core Stack : LGTM](#)
  - 79.2. [Pricing 費用](#)
- 80. [起手式 : 註冊 Grafana Cloud 帳號](#)
- 81. [小結](#)
- 82. [參考連結](#)
- 83. [建立 .NET 範例專案](#)
- 84. [範例程式加上 OpenTelemetry SDKs](#)
- 85. [透過 OpenTelemetry 蔊集 Metrics 數據](#)
- 86. [將數據傳到 Grafana Cloud](#)
  - 86.1. [安裝 Grafana Agent : 執行檔](#)
- 87. [參考連結](#)
- 88. [什麼是 Grafana Pyroscope](#)
- 89. [如何進行分析](#)
- 90. [過去在 Windows 環境是怎麼做](#)
  - 90.1. [蒐集資訊方式](#)
- 91. [參考連結](#)
- 92. [問題排除流程](#)
- 93. [不只是工具](#)
- 94. [參考連結](#)
- 95. [過去怎麼做](#)
- 96. [Grafana Incident Response & Management \(IRM\)](#)
- 97. [Observability Maturity Model : 可觀測性成熟度](#)
- 98. [Observability Driven Development : 可觀測性驅動開發](#)
  - 98.1. [什麼是可觀測性驅動開發 ODD ?](#)
  - 98.2. [ODD 加入 SDLC 階段可能的挑戰 ?](#)
- 99. [小結](#)
- 100. [參考連結](#)
- 101. [學習資源](#)
- 102. [完賽心得](#)

## 前言

---

在國外，可觀測性 (Observability) 這個詞已經成為熱門話題引起熱烈討論，在 CNCF (Cloud Native Computing Foundation) landscape 中有一個區域專門介紹可觀測性 (Observability) 相關工具與技術。在國外研討會中，發現越來越多人探討如何透過可觀測性來實現 DevOps 更高效率。

近幾年，台灣也開始看到越來越多相關文章與研討會分享，分享有關可觀測性的觀念與實踐經驗。在吸收這些知識與寶貴知識後，對這主題有更深入的了解與認識。因此希望自己可以藉由 ITHome 鐵人賽的機會，整理並分享自己關於可觀測性的理解與小小心得。希望藉由透過這次不要臉分享，可以讓對這主題也有興趣的朋友們能夠更快了解可觀測性的重要觀念，一起入坑。

- 作者 : Marcus Tung

- 部落格：[m@rcus 的學習筆記](#)
- 粉絲團：[m@rcus 的學習筆記](#)

# Day01 前言 - 30 天心得分享

---

## 1. 起心動念

大家好，我是伐伐伐木工

首先，我想與大家分享自己為什麼選擇 **Observability 101** 這個主題，以作為參加鐵人賽的起點。

在國外，可觀測性 (Observability) 這個詞已經成為熱門話題引起熱烈討論，在 [CNCF \(Cloud Native Computing Foundation\) landscape](#) 中有一個區域專門介紹可觀測性 (Observability) 相關工具與技術。在國外研討會中，發現越來越多人探討如何透過可觀測性來實現 DevOps 更高效率。

近幾年，台灣也開始看到越來越多相關文章與研討會分享，分享有關可觀測性的觀念與實踐經驗。在吸收這些知識與寶貴知識後，對這主題有更深入的了解與認識。因此希望自己可以藉由 ITHome 鐵人賽的機會，整理並分享自己關於可觀測性的理解與小小心得。希望藉由透過這次不要臉分享，可以讓對這主題也有興趣的朋友們能夠更快了解可觀測性的重要觀念，一起入坑。

## 2. 關於主題

身為一個不專業的工程師，加上崇尚 DDD (Deadline Driven Development) 開發方式，因此開賽第一天還沒想清楚關於這主題每天的題目(被打)，以下是預計分享的主題方向

- DevOps x Observability  
可觀測性與 Devops 是相關聯的，這章節會介紹自己這幾年在工作上覺得 Devops 一些重要觀念。接著介紹可觀測性的基本知識與觀念，什麼是可觀測性(*What*)，為什麼這幾年大家開始重視(*Why*)？
- Observability 的實踐  
介紹可觀測性的工具以及如何實踐(*How*)，包括結構化日誌(Logging)、指標(Metrics)、分佈式追蹤(Tracing) 以及開發者們如何透過可觀測性開源(open source)工具與應用程式進行整合。
- 延伸議題 x 反思  
這主題將探討可觀測性延伸的議題。除了透過工具實踐之外，是否還有其他實踐可觀測性的方式與有趣議題討論，以及團隊在導入時可能會遭遇到的挑戰與問題，評估是否真的適合現在的團隊與解決遭遇到的問題。

以上是關於這次預計分享的內容，從什麼是可觀測性 (*What*) 到重要性 (*Why*)，接著再到如何透過工具實踐 (*How*)，再針對一些有趣的議題 (*When*) 進行討論，希望透過這樣的方式讓受眾全面的了解 Observability 的世界。

今年不知哪裡想不開報名參賽分享，過去都是以吃瓜角色在旁觀看比賽文章，看到很多厲害的大大跟前輩們分享技術受益良多，如果自己分享內容上或對可觀測性有任何感興趣的主題與想法，也歡迎留言一起討論與探討！

# Day02 - Observability in DevOps

---

大家好，我是伐伐伐木工

今天要與大家分享 DevOps & Observability，本篇內容的重點如下

- 淺談 Observability 可觀測性
  - 探討 Devops、Observability 與 Business Continuity 的關聯性
- 

### 3. 淺談 Observability (可觀測性)

先來談談什麼是 Observability 可觀測性，根據 CNCF (Cloud Native Computing Foundation) 的定義

Observability is a system property that defines the degree to which the system can generate actionable insights. It allows users to understand a system's state from these external outputs and take (corrective) action.

Observable systems yield meaningful, actionable data to their operators, allowing them to achieve favorable outcomes (faster incident response, increased developer productivity) and less toil and downtime.

Consequently, how observable a system is will significantly impact its operating and development costs.

#### 重點摘要

- 系統應具備 可被觀測 (Observable) 的能力
- 透過可觀測性的工具可以了解 CPU、Memory、硬碟、API 回應時間、錯誤等資訊。
- 更快的回應速度、增加開發者的生產力、減少停機時間
- 系統的可觀測性程度將影響運營與開發成本

備註：這裡先可觀測性做簡單介紹，後面會再針對可觀測性做詳細說明

#### 3.1. 想要解決問題

- 保持系統穩定運作，減少故障的影響及停機的時間
- 開發與運維同仁可以更高效的運營、提高開發效率

### 4. 淺談 Devops

我們來談談 DevOps，根據維基百科 Wiki 的定義

DevOps (Development和Operations) 是一種重視「軟體開發人員 (Dev)」和「IT運維技術人員 (Ops)」之間溝通合作的文化、運動或慣例。通過自動化「軟體交付」和「架構變更」的流程，來使得構建、測試、發布軟體能夠更加地快捷、頻繁和可靠

#### 4.1. 想要解決什麼問題

- 實現更快速的軟體交付，更高品質的軟體
- 團隊間更好的協作和溝通

### 5. Devops 與 Observability 關聯

DevOps 與 Observability 兩者的關聯是什麼呢？

Observability 是 Devops 其中的一部分。Observability 透過遙測數據收集，讓系統潛在問題有機會被更快速的發現並解決。DevOps 實現更快速的軟體交付，讓團隊間更高效。透過 Observability 與 DevOps 的協作，提高軟體開發與運營的效率，確保系統可靠性。

背後目的是實踐業務連續性(Business Continuity)，業務連續性目標是無論發生什麼故障或是事件，系統都能夠持續運行提供7x24的服務。系統為了達到業務連續性，需要具備可靠性(Reliability)、高可用(High Availability)及穩定性(Stability)。

以上是今天的分享。下一篇要來探討業務連續性(Business Continuity)，如果有任何疑問或想法，歡迎留言提出討論！

## 6. 小結

- Observability ( 可觀測性 ) 透過遙測數據的蒐集，幫助我們迅速識別和解決問題，提高系統的穩定性。
- DevOps 和 Observability 共同有助於實現業務連續性(Business Continuity)，確保系統可靠運行，並提高開發和運營的效率。

---

## 7. 參考連結

[Observability](#)

[DevOps](#)

[What is DevOps Observability \(Importance & Best Practices\)](#)

[What Is Business Continuity ?](#)

# Day03 - Business Continuity

---

大家好，我是伐伐伐木工

如果你是主管，公司突然遭遇天災、技術故障或緊急情況，系統是否能夠持續運作？如果你是開發者，你是否知道如何確保你的應用程序和數據在意外情況下是安全的？如果你是等著下班的人，享受生活的同時，如何確保公司不會出現災難性中斷？

今天要與大家分享的主題是業務連續性(Business Continuity)，本篇內容的重點如下

- 業務連續性的目的
- 什麼是業務連續計畫(Business Continuity Plan)

## 8. 業務連續性

業務連續性(Business Continuity)是指組織不間斷運營的能力，為了確保公司業務核心的服務可以持續運營，不會因為意外或是天災、數據洩漏造成服務中斷，公司管理層需要建立機制與流程，在最短的時間內恢復到意外發生前的狀態。

聽起來可能很遙遠對小小工程師的我們有些遙不可及，這裡舉幾個與開發者比較相關的情境

- 數據備份與還原：確保開發代碼與數據庫的資料的備份，以防止 server 故障時 source code 丟失，當系統異常時也可以透過備份的資料進行系統快速還原的動作。

- **遠端工作**: 忽然遭遇天災或疫情例如 covid 19，無法實體在辦公室工作開發，開發者要意外發生時可以遠端上班並可以正常使用需要的資源，以維持開發工作的連續性。
- **意外狀況**: 當系統發生緊急意外狀況無法正常運行，像是服務中斷、資安漏洞或 Server 故障，如何快速恢復不影響到用戶，以維持系統的可靠性。

業務連續性對於開發者意謂確保開發流程不會因為例外狀況而中斷，需要制定可行的計畫或行動方案來實現。

## 9. 業務連續計畫 Business Continuity Plan

為了達到業務連續性，其制定的計畫稱為 **業務連續計畫** (Business Continuity Plan)，根據維基百科 Wiki 的定義

營運持續計畫（英語：Business continuity planning，簡稱BCP）或稱業務連續性計劃，是指組織為因應突發災難事件而預先規劃的應變與復原作業流程，以確保組織在可接受的最低營運水準下可持續提供關鍵服務項目予重要客戶。計畫內容包含營運衝擊分析、最低資源需求、測試演練等

### 9.1. BCP 核心概念

在 BCP 中核心內容包括

- **營運衝擊分析**: 評估各種潛在意外與災難事件的影響，包含天災、技術故障與人為事故。透過分析與深入了解，可以知道公司面對這些風險與其可能的後果。
- **最低資源需求**: 確認緊急狀況下所需要的最低資源，包含人員、設備、技術與 Device。確保在極端情況下組織也能維持基本運作。
- **測試演練**: BCP 不僅是冰冷的文件，為了確保有效性須透過定期的測試跟演練。包括模擬雲端異常時，確保員工知道如何應對與快速處理，系統才能恢復運行狀態符合 SLA。

### 9.2. BCP 計畫類型

業務連續計畫分為不同類型，以因應不同面向的需求，包括(但不限

- 業務連續性計劃 (BCP): 核心業務運營的能力，確保業務不中斷。
- 災難恢復計劃 (DRP) : IT基礎建設與數據，確保系統與數據完整性
- 危機管理計劃 (CMP): 管理與應對各種緊急危機，包括公共關係與品牌管理
- 緊急響應計劃 (ERP): 專注於緊急情況的快速回應，以最小化損失

### 9.3. 計畫執行重要元素

另外，計畫要可以正確並有效的執行需要依賴下列三個元素

- **人 (People)**: 有經驗的團隊成員需要清楚理解自己在BCP的內容扮演的角色，以及不同階段該負責什麼樣的任務，才有機會在緊急狀況下有效的執行計畫。
- **流程 (Proces)**: 明確的流程和程序，包括警報通知、決策過程、數據備份和恢復流程等，以確保計畫的順利執行。
- **技術 (Technology)**: 支持BCP的技術工具，包括數據備份系統、通信工具和遠程工作設施，以確保業務能夠持續運作。

以上針對業務連續計畫 BCP 的內容說明，下一篇要來探討另外一個重要的觀念 SLA(Service Level Agreement)，如果有任何疑問或想法，歡迎留言提出討論！

OS : 越寫覺得洞越來越深有種寫不完的感覺 XD

## 10. 小結

- 業務連續性計劃目的是確定並減少危險可能帶來的損失，確保企業能夠持續經營
- 

## 11. 參考連結

[What Is Business Continuity ?](#)

[Business Continuity](#)

[Business Continuity Management Awareness Training](#)

# Day04 - Service Level Agreement

---

大家好，我是伐伐伐木工

如何判斷一個系統或服務的服務質量有多好？

今天要與大家分享 SLA，本篇內容的重點如下

- 什麼是 SLA (Service Level Agreement)
- SLA, SLO, SLI 三者的關係

## 12. 服務級別協議

服務級別協議 SLA (Service Level Agreement) 是服務提供者對於其服務水準的承諾。明確定義了衡量服務的標準跟方法，讓客戶或是使用者可以清楚知道當未達到服務水平時的處罰與補救措施。

常見的 SLA 會包含下列項目

- 服務內容
- 可用性
- 賠償條款
- 服務標準

聽起來有點抽象，為了讓大家可以更清楚理解，我們來看看 AWS EC2 SLA 作為範例，[連結](#)

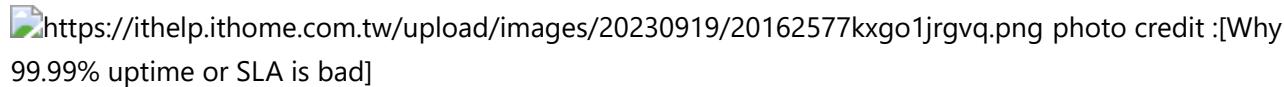
- 服務內容  <https://ithelp.ithome.com.tw/upload/images/20230919/201625775Jxc02Aw1p.png>
- 可用性：
  - **99.99%**
- 賠償條款  <https://ithelp.ithome.com.tw/upload/images/20230919/20162577JsnWEfHv4o.png>
- 排除事項  <https://ithelp.ithome.com.tw/upload/images/20230919/20162577CgCfDadJ9J.png>
- 提供技術支援服務內容

如果有興趣也可以參考其他雲端廠商 SLA 協議，[參考連結](#)

- [Azure SLA](#)
- [GCP SLA](#)

### 12.1. 可用性

SLA 是可用性中重要的衡量指標，服務提供商與使用者會透過此指標來客觀的評估服務的水準，服務正常運行為 UpTime，反之，當服務發生不可用或是中斷時間稱之 Downtime。其指標越高代表服務越可靠，停機時間越短。



接著我們來看可用性時間，下表為換算 SLA 中幾個 9 與 UpTime 及 Downtime 時間關係

SLA	Downtime	UpTime
99.999%	26 seconds	729 hours, 59 minutes, 34 second
99.99%	4 minutes, 22 seconds	729 hours, 59 minutes, 34 seconds
99.95%	21 minutes, 54 seconds	729 hours, 38 minutes, 6 seconds
99.90%	43 minutes, 49 seconds	729 hours, 16 minutes, 12 seconds
99.5%	3 hours, 39 minutes, 8 seconds	726 hours, 21 minutes
99.0%	7 hours, 18 minutes, 17 seconds	722 hours, 42 minutes

如果您所使用的系統服務提供的 SLA 為 99.99%，聽起來穩定，但對於服務提供商來說就是很大的挑戰，從上表可以得知，99.99% 意味著當系統發生異常時 5 分鐘之內必須修復好恢復使用，這其中還包含發現問題到修復完畢，然後系統重新上線完成的時間(壓力好大先睡一覺在說?)。

## 12.2. SLA 與 SLO

SLA 是對客戶的協議(Agreements)，是系統服務對外面使用者的承諾，對內而言 RD 所開發的服務需要設定目標來滿足 SLA，SLA 則和 SLO 與 SLI 有密切關係

SLI (Service Level Indicator)：衡量服務性能可靠性的指標

常見的 SLI 指標如下

- Availability、Latency
- Error Rate
- Throughput、MTTR(平均修復時間)

SLO (Service Level Objective)：為 SLI 設定可靠性的目標

在內部設定 SLOs 指標時會相較 SLA 更為嚴格。舉例來說 SLA 定義回應時間是 300ms，則在內部 SLO 目標則不會超過 200ms，才更有機會達標。



photo credit : [The Art of SLOs](#)

## 13. 如何計算 SLA

前面介紹了 SLA 基本概念，還有 SLA、SLO、SLI 三者的關係，接著探討 SLA 是如何計算出來的。假設服務架構如下，服務中有應用程式(Web)與中間應用程式(Application)和 SQL 數據庫(SQL)三個組件所組成。

 <https://ithelp.ithome.com.tw/upload/images/20230919/201625779Vjxu9ToUT.png> 單獨來看，各自的可用性分別是

- Web : 99.95%
- Application : 99.9%
- SQL : 99.95%

整體服務的可用性是多少呢？

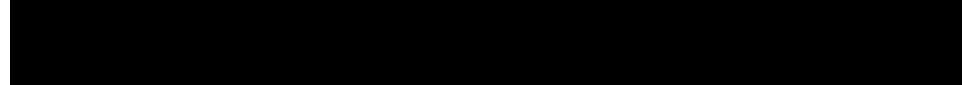
整體 SLA = Web SLA \* Application SLA \* SQL SLA  $99.95\% * 99.9\% * 99.99\% = 99.84\%$

最終答案是 **99.8%**，根據 [uptime.is](https://uptime.is) 的計算，允許 downtime 的時間為

 <https://ithelp.ithome.com.tw/upload/images/20230919/201625774cDQrBnb8x.png> photo credit : <https://uptime.is/>

假設對外服務的承諾 SLA 是 99.9%，那麼就要思考架構上要如何優化，才有辦法從 99.8% 變為承諾的 99.9%。要注意的是，SLA 目標設定越高所需要花費及投入的成本更高，如果身為管理者，希望減少停機的時間，勢必投入額外的基礎建設資源或是故障移轉機制，就需要額外思考其額外多出的成本是否值得投入。

如果想了解更多SLA、SLO、SLI 三者關係，個人推薦觀看下面影片，可以有更清楚的理解



以上是今天關於 SLA 的內容分

享，下一篇要來探討 DR(Disaster Recovery)，如果有任何疑問或想法，歡迎留言提出討論！

## 14. 小結

- SLA 是服務提供商與客戶之間定義的正式承諾
- SLO 是為了實現 SLA 的目標所設定的，SLI 用於衡量服務的性能指標

## 15. 參考連結

### 服務級別協定

[軟體技術架構如何正確與商業需求快速對齊：談 MAU 換算至 RPS，SLA 回推至 SLI](#)

Cloud SLAs punish, not compensate

Why 99.99% uptime or SLA is bad

How do you calculate the compound Service Level Agreement (SLA) for cloud services?

The Art of SLOs

SLA Vs SLO: Tutorial & Examples

## Day05 - Disaster Recovery

---

大家好，我是伐伐伐木工

今天要與大家分享的主題是災難復原(DR)，本篇內容的重點如下

- 什麼是災難復原 (Disaster Recovery)

### 16. 災難復原 Disaster Recovery

 <https://ithelp.ithome.com.tw/upload/images/20230920/20162577Q1SoiFJj3R.png> photo credit

<https://www.techtarget.com/searchdisasterrecovery/definition/Business-Continuity-and-Disaster-Recovery-BCDR>

災難復原是業務連續(BC)的一部分。業務連續性是指企業有應對風險、自動調整和快速反應的能力，以保證企業業務的連續運轉。為企業重要應用和流程提供業務連續性應該包括以下三個方面。

1. 高可用性 ( High availability)：指提供在本地故障情況下，能繼續訪問應用的能力。無論這個故障是業務流程、物理設施，還是IT軟硬體故障。
2. 連續操作 ( Continuous operations)：指當所有設備無故障時保持業務連續運行的能力。用戶不需要僅僅因為正常的備份或維護而需要停止應用的能力。
3. 災難復原 ( Disaster Recovery)：指當災難破壞生產中心時，在不同的地點恢復數據的能力。

簡單來說，BCP的目標只有一個，那就是確定並減少危險可能帶來的損失，有效地保障業務的連續性。今天所要介紹的是第三部分災難復原 DR 為出發，規劃當災難發生時所要進行的動作。在災難復原中 **RTO** 與 **RPO** 是很常聽到的概念，兩者分別是

#### 16.1. RTO : Recovery Time Objectives

- 當災難發生時恢復應用程式與數據所需要花費的時間
- 設定這指標目的是減少停機的時間。
- 手動恢復流程與自動恢復流程不同，會影響恢復時間長短。

#### 16.2. RPO : Recovery Point Objectives

- 當災難發生時允許遺(丟)失多少資料，白話就是接受噴掉多久時間的資料量
- 設定這指標目的是最小化數據丟失。
- 備份或是數據保護量頻率越高，丟失的數據量就會越少。

可能聽起來有些抽象沒那麼好理解，透過以下示意圖可以更清楚瞭解 RTO 與 RPO 兩者的差異



讓我們舉個實例看看其應用

前情提要

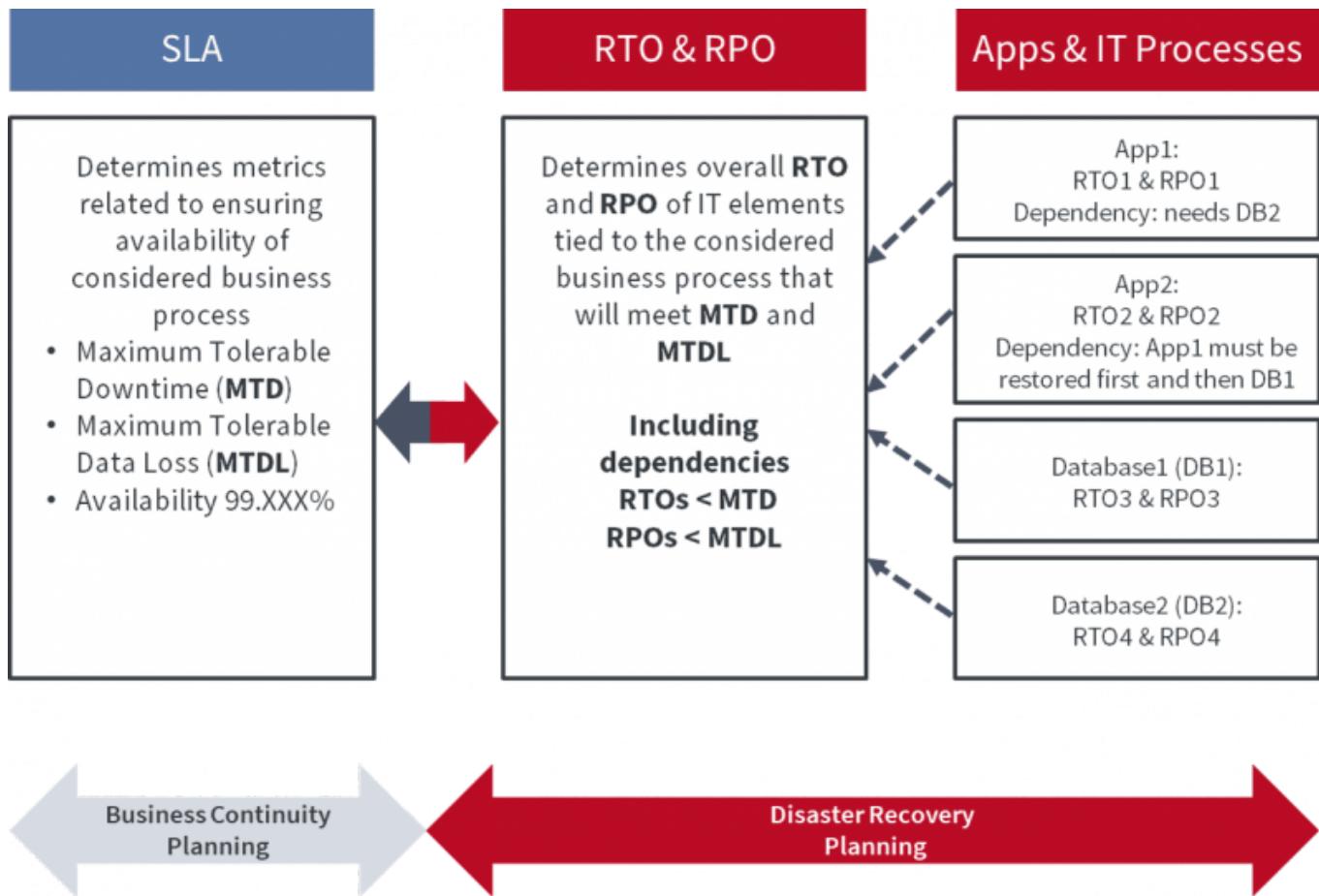
- 指標 : RTO 與 RPO 分別是 4 小時
- 備份時間 : 系統固定 4 小時進行備份，分別是 6:00、10:00、14:00 與 18:00

問題：如果 13:05 發生系統中斷時，RPO 會是多少？

當下午 13:05 發生意外中斷，最後數據備份時間是 10:00，因此 RPO 為 3小時5分鐘，資料庫恢復備份與重新佈署應用程式，4小時候系統可以正常恢復運作。

## 17. SLA 與 RTO , RPO 關係

前一篇文章中，我們提到透過 SLA 可以清楚的知道服務的可用性與服務內容與補償方案，幫助企業評估是否可以承受其風險，在災難復原的 SLA，主要的指標是 **RTO** 與 **RPO**，兩者是災難復原 DR 中重要的衡量指標，以時間單位為衡量值越低越理想。



在制定 RTO 與 RPO 之前需要考慮到 MTD(最大可容忍時間) 與 MTDL(最大可容忍數據丟失) 兩個業務連續性(BC)指標，並在設定 RTO 與 RPO 指標時，需小於 SLA 所設定的 MTD 與 MTDL，才有機會達到目標，其關係可以參考上圖。

以上是今天的分享。下一篇要來探討災難復原計畫 (Disaster Recovery Plan)，如果有任何疑問或想法，歡迎留言提出討論！

## 18. 小結

- 災難復原 (DR) 目的是確保組織在意外發生時能夠以最快速的方式來恢復，以確保業務連續性(BC)
- RTO 最大可允許中斷時間，RPO 數據損失可允許的最遠回溯時點

## 19. 參考連結

[Disaster Recovery \(DR\)](#)

[Risk Assessment, BIA, SLAs, RTOs, and RPOs: What's the Link? MTD and MTDL](#)

[Business continuity management](#)

## Day06 - 災難復原計畫 Disaster Recovery Plan

---

大家好，我是伐伐伐木工

今天要與大家分享災難復原計畫 DRP，本篇內容的重點如下

- 淺談難復原計畫 DRP

## 20. 災難復原計畫 Disaster Recovery Plan

災難復原計畫 DRP (Disaster Recovery Plan)是確保業務連續性的關鍵，這計畫將有助於當災難發生時，核心業務可以迅速且可靠的恢復正常運作。有效的 DRP 可能會包含下列要素

- **業務影響分析 (BIA)**：檢視當災難發生時哪些核心業務是至關重要的，並依據分析與識別後的結果設定優先權。
- **RTO 與 RPO**：透過 BIA 分析後可以了解核心業務功能的排序跟影響，制定各核心業務的 RPO 與 RTO 時間點，內部或是非關鍵功能可能 RTO/RPO 時間會較長。
- **恢復策略 (Recovery Strategies)**：制定可靠的恢復策略，需要考慮的點有網路、基礎建設、資料庫、應用程式及測試內容。以確保需要在特定時間點恢復時需要依賴那些重要元素或工具。
- **測試與演練 (Testing and Training)**：透過分析與計畫定期的練習，模擬當意外發生時DRP 計畫啟動到測試完成系統恢復正常



迷之聲：我是誰、我在哪、我要去哪裡

## 21. 現況盤點

以下是可能會遇到的問題

Q1 : 萬事起頭難，如何開始？

A : 先了解你的業務與IT資源使用，包含業務運作、IT 基礎建設、應用程式、雲端服務商或是數據庫等等，列出您目前擁有的所有資源跟系統。

範例

- 線上電子購物平台
- 使用 AWS 雲端服務，使用虛擬伺服器、數據庫、存儲和負載平衡器
- 應用程式包括網站前端、購物車、付款處理和庫存管理
- 數據庫存放在 IDC 機房，包含產品目錄、客戶訂單和用戶資訊

盤點後可能如下，將服務拆成不同類別，並依據既有服務可以拆分為

Service	Name	Description
Network	網路設定	AWS Elastic Ips、S3、Subnet、NAT Gateway、Route 53、Load balance
Application Server	網站應用程式	ECS、EC2、Service、Launch Template、Auto Scaling Group
Database	數據資料	產品目錄、客戶訂單和用戶資訊

就資料 (Data) 與非資料 (Network & Application) 層面拆開來看分析

## 21.1. Network & Application Server

- 盤點在 Network 與 Application 使用服務資源，支援自動化腳本程度、支援跨區服務
- 各服務恢復所需要的時間
- IP 費用，開啟白名單時需要同時建立 DR Site IP

## 21.2. Database

災難復原是恢復數據的能力，同步與備援機制如下

- 即時同步：透過 MS SQL Always on 機制，將數據即時同步到其他資料庫中。
- 備份：每天備份資料壓縮做備援 Backup，運維同仁會將備援檔案備份存放另一個 IDC

資料庫數據和日誌的容量整理如下，包括完整備份、數據傳輸時間估計、還原預估等

另外還需考慮

- 資料庫每天成長大小，ex: +1G/Day
- IDC 到雲端服務頻寬大小
- IDC upload Database full backup 時間
- 建置雲端機器設定與同步地端DB時間

透過現有狀況盤點後，可以得到系統恢復的真實時間，例如，上述盤點完光資料庫部分還原需要 6 小時，而產品服務的期待時間是 2 小時，因此我們要思考如何數據步驟是否有其他更快的方式進行還原，ex：數據庫進行拆分。

### Q2：如何進行 BIA (業務影響分析)

A：進行業務影響分析(Business Impact Analysis)，確定哪些業務功能對您的業務至關重要，以及它們的優先順序。這有助於確定災難發生時應首先恢復的部分。

順序	業務目標	功能	服務
1	登入	功能A、功能B	主站、登入 API
2	目錄	功能C、功能D	目錄 API
3	訂單	功能E、功能F	API
4	其他		

除了找到商業上重要的功能外，還要了解對應的服務資訊，以計算每個服務和數據恢復的關鍵時間。

### Q3 : 如何計算 Application 恢復時間

A : 盤點每個功能，在按下部署按鈕後到完成所需的步驟和時間。應用程式上線的總時間可以表示為：

AP上線總時間 = Script + 機器請求 + 移交機器 + Provisioned + Health Check

- Script：是執行Terraform 所需要的預估時間(無法控制的時間)
- 機器請求：從Auto Scaling Group向AWS請求EC2 instance所需要花費的時間(無法控制的時間)
- 移交機器：從AWS建立EC2 instance到執行Provisioned所需要的時間(無法控制的時間)
- Provisioned：執行Provisioned Script的所需時間
- Health Check：從Provisioned執行後到服務正式上線的心跳檢查

計算整個應用程式恢復所需的時間，包括腳本執行、機器請求、機器建立、Provisioned 腳本執行和健康檢查等步驟。這有助於確定可以控制的和無法控制的恢復時間因素。

### Q4 : RTO 與 RPO 時間如何定義？

A : 可以參考前一篇文章對於其定義與介紹 [Day5 DR](#)

## 22. 小結

- 災難復原計畫 (DRP) 是一項策略，在確保在災難性事件發生時，業務可以快速、可靠地恢復正常運作。
- DRP 包括業務影響分析BIA、RTO與RPO目標設定、恢復策略制定、測試與演練等，以確保業務在災難時能夠順利復原。

## 23. 參考連結

[Disaster Recovery Plan \(DRP\)](#)

[Business Continuity Plan \(BCP\) vs. Disaster Recovery Plan \(DRP\): What Are the Key Differences?](#)

# Day07 - 高可用性與可靠性 High Availability & Reliability

大家好，我是伐伐伐伐木工

今天要跟大家分享在討論系統中常會聽到的概念可用性(High Available)與可靠性(Reliability)，本篇內容的重點如下

- 可用性與可靠性

## 24. 可用性與可靠性

當我們在討論系統時，使用者通常只會遇到兩種情況，它「可以使用」或「無法使用」。「可以使用」是使用者順利完成他們的目標；不可用則是無法使用其功能，其背後可能是使用者自己的錯誤、系統 Bug 或是環境問題，例如網路瞬斷、設備異常或遭受攻擊。

接著回到主題，根據維基百科關於兩者的定義

可用性：系統在給定時間運行的概率，即設備實際運行的時間佔其應運行的總時間的百分比。可靠性：  
系統在某個給定時間t內產生正確輸出的概率

試著透過翻譯蒟蒻翻譯如下

可用性：正常運行時間的百分比(成功) 可靠性：特定的時間內，系統正常執行成功的機率(失敗)

兩者都是屬於抽象的概念，為了更好理解可用性與可靠性，可以使用兩個指標來衡量

Concept	Metric	Example
Availability	百分比	99.90%
Reliability	平均無故障時間 (MTBF)	20天10小時12分鐘

備註：

- 平均故障間隔時間(MTBF)：總正常運行時間/故障數量。
- 平均修復時間(MTTR)：總停機時間/故障數量

可靠性衡量系統正確運行的能力，包括避免數據損壞，而可用性衡量系統可用的頻率，即使它可能無法正常運行。

## 25. 共同特性

可用性與可靠性都是服務重要的關鍵特性，兩者具有一些共用的特性，這些特性有助於確保系統的穩定性，以下是列出個人覺得兩者共同具備的重要特性

- Monitoring and Alerting
- Automation
- Redundancy

監控機制跟告警機制(Monitoring and Alerting)，目的是當系統或服務有不穩定時可以在第一時間知道並進行處理，提高系統的穩定性與停機時間。自動化(Automation)可以加速故障恢復的時間，並減少在緊急問題人為錯誤的可能性。Redundancy 是什麼呢？以下就針對 Redundancy 進行更多的說明。

### 25.1. 夾餘 Redundancy

冗餘是刻意設置重複的零件或是功能，作為安全的緩衝。以確保如果一個服務組件發生故障，其他服務組件可以併行工作不影響其服務內容。

為了要提高系統可用性，可以先透過盤點方式了解系統架構中關鍵的組件，每個組件都會有其可用性(故障率)，如果單一組件可用性高(故障率低)，可以透過簡單的冗餘策略來提高可用性，例如在雲端環境中透過自動擴展(Scale)機制，確保單點發生故障時也能維持一定數量的 Instance。

例如，在雲端服務平台中設定規則，應用程式 Instance 的數量必須要同時維持 4 台。因此當雲端平台監控到某台機器已經關閉或是health check 沒回應時，會立即根據其配置建立副本應用程式，確保在最短停機時間並減少恢復可用性的手動干擾。

以上是關於高可用與可靠性的特點與解釋，如果有任何疑問或想法，歡迎留言提出討論！

## 26. 小結

- 可用性是正常運行時間的百分比，使用 SLA 來衡量
- 可靠性是特定的時間內，系統正常執行成功的機率。使用 MTBF 來衡量

## 27. 參考連結

[High availability](#)

[Available . . . or not? That is the question—CRE life lessons](#)

[Reliability, availability and serviceability](#)

[Impact of Redundancy on Availability](#)

[可靠性工程 \(Reliability Engineering\)](#)

# Day08 - 監控與指標分析 Monitor

---

大家好，我是伐伐伐伐木工

今天要與大家分享監控 Monitor，本篇內容的重點如下

- 監控的基本概念
- 主要的監控指標
- 監控框架：USE、RED、Golden Signals

## 28. 什麼是監控

監控可以幫助團隊觀察系統的效能並偵測已知的故障，有效的監控包含三個步驟

- 預先定義 "指標"
- 部署程式來收集指標
- 在儀表板中顯示這些指標

然而，監測也有其局限性。為了進行監控，您必須知道要追蹤哪些指標和日誌。如果您的團隊沒有預測到問題，則可能會錯過關鍵的生產故障和其他問題。

### 28.1. 監控類型

在現代分散式系統中，常見監控的元件有基礎建設、應用程式、資料庫、網路、資料流等，而我們想要的指標會因為監控類型而不同，例如

- 基礎設施：正常運作時、CPU 使用率、記憶體使用率
- 應用程式效能監控：吞吐量、錯誤率、延遲
- 資料庫監控：連線數、查詢效能
- 網路監控：往返時間、TCP、連線遺失

### 28.2. 監控指標(Framework)

#### 28.2.1. USE

由 Brendan Gregg 提出[文章](#)， USE 方法是基於三種度量類型和處理複雜系統的策略，其縮寫代表意義如下

- 使用率 (Utilization) : 資源繁忙的時間百分比，例如「一個磁碟以 90% 的利用率運作」
- 飽和度 (Saturation) : 與工作資源量相關，例如「CPU 的平均運行佇列長度為 4」
- 錯誤 (Errors): 錯誤事件計數，例如「此網路介面發生了 50 次遲到衝突」

核心概念

對於每個資源，檢查使用率、飽和度和錯誤。

### 28.2.2. RED

2015年，Grafana 的 Tom Wilkie 談到了監控微服務的RED方法。Tom 建議不要監視每個資源的使用率、飽和度和錯誤，而是對於每個資源，監控

- 速率 (Rate) : 您的服務每秒處理的請求數
- 錯誤 (Errors): 每秒失敗的請求數
- 持續時間 (Duration): 每個請求所花費的時間量的分佈

核心概念

透過使用 RED 方法，公司將更了解客戶的滿意度，並將幫助您建立有意義的警報並衡量 SLA

### 28.2.3. Golden

來自 Google SRE Book : [The Four Golden Signals](#)

監控的四個黃金訊號是延遲、流量、錯誤和飽和度。如果您只能衡量使用者導向的系統的四個指標，請專注於這四個指標。

- 延遲 (Latency): 處理請求所需的時間
- 流量 (Traffic): 對您的系統有多少需求
- 錯誤 (Errors): 失敗請求的比率
- 飽和度 (Saturation): 您的服務有多“完整”

以上是今天的分享，如果有任何疑問或想法，歡迎留言提出討論！

## 29. 小結

- 監控是一個策略和工具的組合，用於實時追蹤和評估資訊系統的性能和健康狀態
- 監控監控框架：USE、RED、Golden Signals

## 30. 參考連結

[4 SRE Golden Signals \(What they are and why they matter\)](#)

[USE vs RED vs The Four Golden Signals](#)

# Day09 - DevOps x Observability 小結

大家好，我是伐伐伐伐木工

前面介紹了 Business Continuity、SLA、DR、Availability & Reliability 等概念，那它們之間有什麼關聯呢？

## 31. SLA、DR、Availability 和 Reliability：系統穩定性的關鍵因素

在自己過去的經驗中，SLA（Service Level Agreement）、DR（Disaster Recovery）、Availability（可用性）、Reliability（可靠性）和BCP（Business Continuity Plan），在確保系統的穩定性有密切關係。

### • SLA (Service Level Agreement)：

- SLA 是服務提供商與客戶之間定義的正式承諾
- SLO 是為了實現 SLA 的目標所設定的，SLI 用於衡量服務的性能指標
- 可用性的指標，確保服務或是產品可以在特定時間提供所需要的功能

### • DR (Disaster Recovery)

- 目的是確保組織在意外發生時能夠以最快速的方式來恢復，以確保業務連續性(BC)
- RTO 最大可允許中斷時間，RPO 數據損失可允許的最遠回溯時點

### • 可用性 ( Availability )

- 正常運行時間的百分比，使用 SLA 來衡量
- 高可用(High Availability)意味系統能持續提供服務，減少停機時間

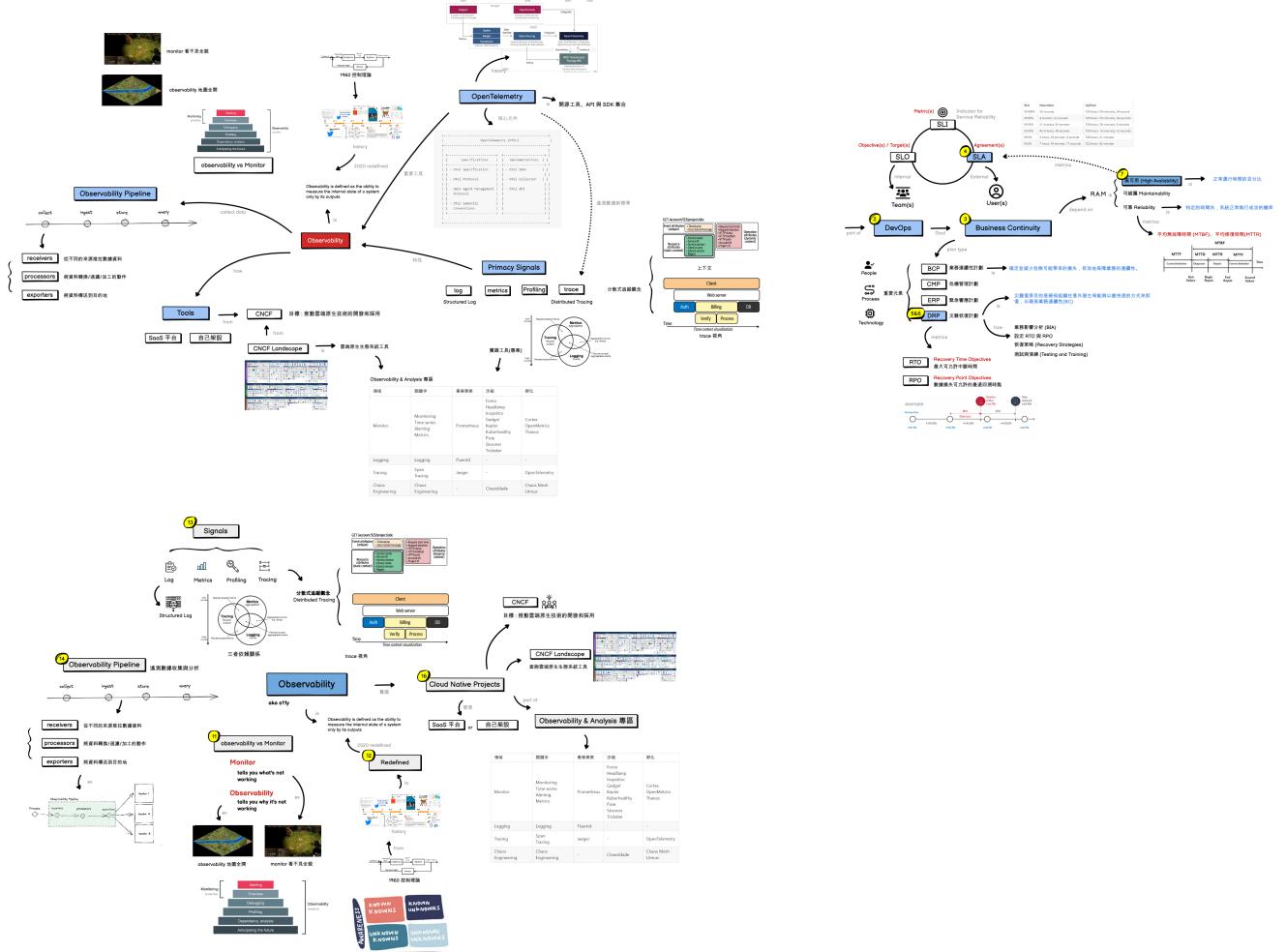
### • 可靠性(Reliability)

- 特定的時間內，系統正常執行成功的機率。使用 MTBF 來衡量
- 透過監控與告警機制，提醒團隊系統異常並緊急處理，減少 downtime 時間

### • BCP (Business Continuity Plan)

- 減少意外與危險可能帶來的損失，確保企業能夠持續經營
- 包含災難復原、備份策略、緊急回應計畫。

根據以上各點以及過去幾天分享文章的內容，我整理了以下重點資訊(偏維運)，方便大家更好理解



備註：

- 藍色框+黃色數字為比賽的文章標題與第幾天
- 眼尖的朋友可以發現缺少監控，會再補齊 XD

下一篇要來開始進入重點可觀測性 Observability 的世界，如果有任何疑問或想法，歡迎留言提出討論！

## Day10 - 可觀測性 Observability

大家好，我是伐伐伐伐木工

今天要與大家分享 Observability，本篇內容的重點如下

- 淺談 Observability 可觀測性
- 討論可觀測性想解決什麼問題

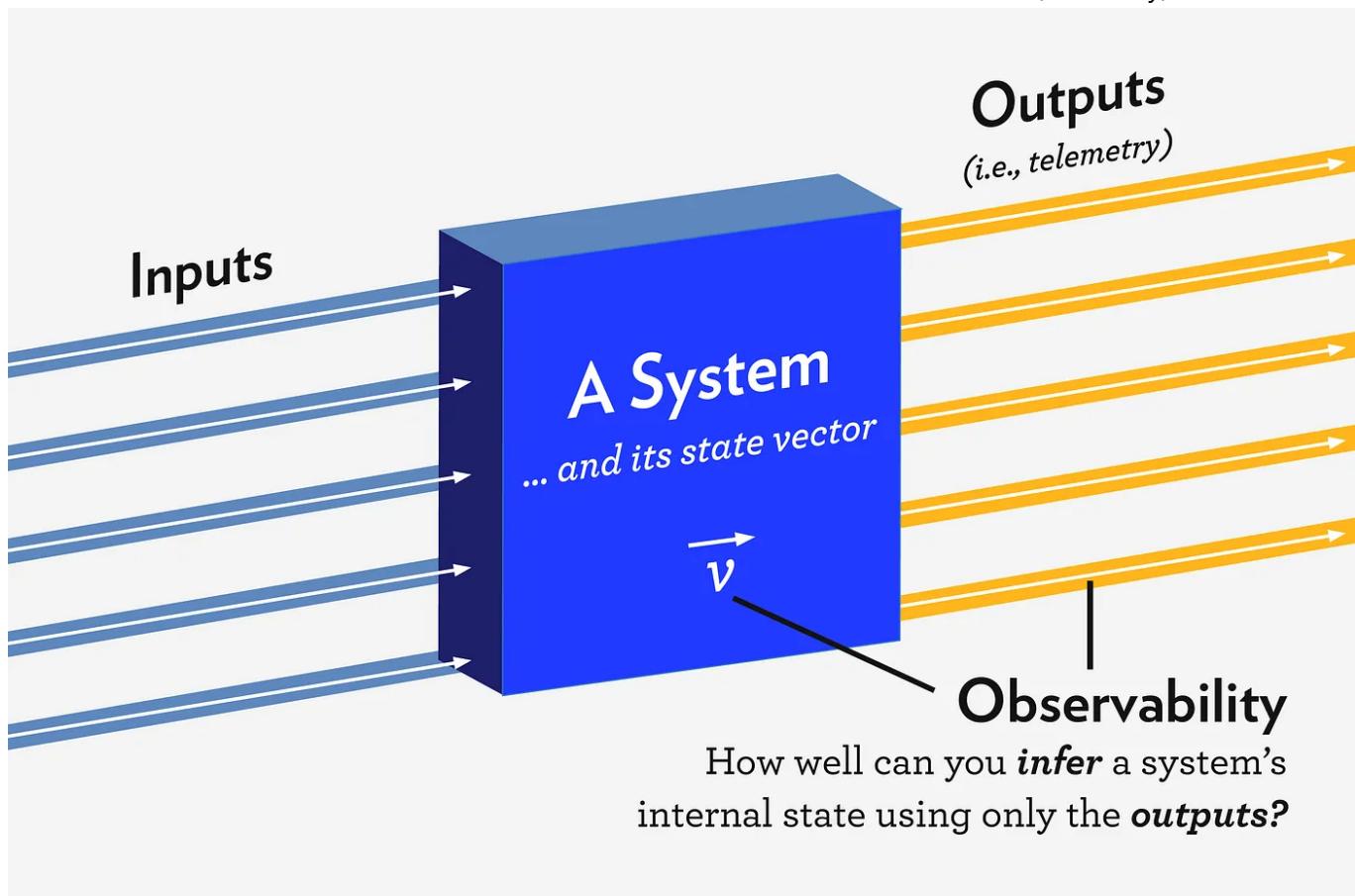
### 32. 可觀測性是甚麼？

終於要進入主題可觀測性，可觀測性 Observability 簡稱 **o11y**

可觀測性一詞是 1960 年由工程師 Rudolf E. Kálmán 創造，發展至今在不同的領域意味不同的描述。他在所發表的論文中介紹一個他稱為可觀測性的特徵用於描述數學控制系統，在控制理論中，可觀測性被定義如下

Observability is defined as a measure of how well internal states of a system can be inferred from knowledge of its external outputs.

從對系統外部輸出的資訊推斷系統內部狀態的能力，這裡的資訊指的是收集到的遙測(Telemetry)資料。



看過很多關於可觀測性的定義，自己最喜歡是 Lightstep 提供的定義

Observeability is defined as the ability to measure the internal state of a system only by its outputs

自己對於其理解是 measure 蔑集 log、metrics、trace 等遙測數據。internal state 是系統發生什麼問題，為甚麼發生及如何修復。outputs 是哪裡有問題、甚麼是慢的、需要做甚麼來提高性能。

要使系統具備可被觀測性，必須滿足以下

- 了解應用程式的內部運作原理
- 使用外部工具觀察來了解內部運作原理和系統狀態
- 了解內部運作狀態時，無須交付任何自訂程式來處理

### 33. 為什麼可觀測性這麼重要：解決了哪些問題？

隨著科技技術的不斷創新，軟體架構和基礎設施的演進速度也在快速變化。以下（但不僅限於此）是企業過去在開發系統或應用程式時可能經歷的幾個階段：

- 第一階段：單一應用程式架構

在這個階段，應用程式的架構設計相對簡單，商業需求還不是很複雜，因此架構通常不會對模組進行太多的切割。應用程式運行在單一執行緒（Process）中處理單一請求。測試和問題排除相對簡單。部署通常在虛擬機器（VM）或機房（IDC）中提供給使用者使用。

- 第二階段：雲端技術的崛起

隨著雲端技術的出現，企業開始嘗試將應用程式部署到雲端基礎設施即服務（IaaS）上，或結合公有雲和私有雲的優勢。它們可能在私有雲中處理機密資料，並在公有雲服務上處理其他資訊。開發人員在應用程式開發和整合方面有更多的機會與雲端服務整合。這一階段也涉及選擇在雲端上使用適合的服務。

- 第三階段：雲端技術的成熟

隨著雲端技術的逐漸成熟和穩定，企業可以利用越來越多的雲端基礎設施服務和新服務。架構設計變得更加複雜，分散式系統架構成為許多討論的焦點。資料庫從機房遷移到雲端上，並使用像RDS、DynamoDB、Redis等雲端資料庫服務，使雲端開發和管理更加便捷。架構也變得更加靈活，以滿足企業需求。同時，容器化技術（如Docker）的成熟也開始引入新的可能性。

- 第四階段：微服務和容器化的興起

軟體架構從單體式(Monolithic)演變為微服務(Microservices)、無伺服器 (serverless) 和服務網格 (service meshes) 等各種可能的組合。主要目標是減少服務之間的相依性，提高擴展能力與故障時的隔離與可用性。

帶來了好處也衍伸了一些有趣的議題，服務越切越細、分散式系統其也帶來複雜性，監控與測試變得更為複雜，如何在問題發生的第一時間定位異常的服務也變得更重要，在 Google 文章中 [DevOps measurement: Monitoring and observability](#) 提到要在監控和可觀測性方面做得出色，您的團隊應該具備以下特點：

- 報告系統整體健康狀況（我的系統是否運作正常？我的系統是否有足夠的可用資源？）。
- 報告顧客體驗到的系統狀態（我的客戶是否知道如果我的系統故障並且遭遇不佳體驗？）。
- 監控關鍵的業務和系統指標。
- 提供工具，幫助您在生產環境中了解和調試您的系統。
- 提供工具，用於查找您先前不知道的信息（即，您可以識別未知的未知）。
- 存取有助於追蹤、了解和診斷生產環境中基礎設施問題的工具和數據，包括服務之間的互動。

上述內容聽起來很抽象，白話點背後是希望可以回答以下問題

- 請求經過哪些服務？系統中哪個部分 loading 最大
- 當服務發生緩慢時，哪裡慢(Bottlenecks)以及可能原因是什麼
- 當服務無法使用時，錯誤及可能異常的原因是什麼
- 當使用者反映操作timeout，但在 Dashboard 上顯示平均請求都很快，要如何找到其可能緩慢原因

目的是希望工程團隊可以理解並解釋系統的現況(增加系統透明度，而不是靠通靈)，當系統出現問題時，可以第一時間了解爆炸範圍，進行緊急問題的處理已加速恢復的時間。且無須發布新的程式碼，提高系統的穩定性和可用性，也是現代化應用程式非常重要的特性之一。

有想了解更多關於可觀測性的內容可以參考 CNCF 的白皮書，Git 上面的是完整版，Google 文件的是眾多大神在討論時的版本，可以更了解細節，對其有興趣的朋友可以參考看看。

可觀測性白皮書：

- [Github 版本](#)
- [草稿版本](#)

以上是今天的分享。下一篇要來探討可觀測性與監控的差異，如果有任何疑問或想法，歡迎留言提出討論！

## 34. 小結

- 可觀測性是一種能力，允許工程團隊深入了解系統運作，並快速識別問題，不需事先定義具體的屬性或模式。
- 可觀測性成為確保系統可用性和穩定性的關鍵，透過監控、報告、和工具，團隊能夠有效解決問題，提高可用性與可靠性。

## 35. 參考連結

[DevOps measurement: Monitoring and observability](#)

[What is observability and why is it important?](#)

[What is DevOps Observability \(Importance & Best Practices\)](#)

# Day11 - 解析監控和可觀測性：從哨塔到全景地圖

---

大家好，我是伐伐伐木工

監控與可觀測性定義，兩者有什麼不同呢？

今天要與大家分享 DevOps & Observability，本篇內容的重點如下

- 探討監控與可觀測性的不同，以及兩者都對 DevOps 的重要性

## 36. 遺留監控的挑戰

過去幾年商業上的變化與數位轉型速度不斷加速，為了有效了解與管理系統的狀態，公司必須有正確的工具和方法，監控作法已在過去十幾年普遍使用，在今天仍然重要在可用性上佔有一席之地，但對於分散式系統或是團隊來說，監控仍有明顯的限制。

在雲端(Cloud)平台、容器技術化的普及與架構拆分更細的前提下，很多企業使用雲端原生(Cloud Native)技術來時間靈活性與敏捷性，來反映及面對市場上快速的變化。這對於傳統的監控方式太複雜了，造成可能會有下列問題

- 數據差距：傳統監控工具可能只能對數據進行取樣，這限制了用戶和任何在該數據上運行的分析算法的全面可視性。結果是對影響客戶的問題的可見度降低，這意味著解決問題所需的時間更長。
- 動作緩慢：內建於雲原生技術中的無伺服器 (Serverless) 功能在幾秒鐘內或更短時間內就會被調用。傳統的監控工具無法這麼快地捕捉動作，導致更多的數據丟失。
- 智能缺失：大多數監控工具並未建立來處理我們今天習慣的數據速率。即使它們收到了一些數據，這些工具也沒有內置的智能，導致過多的警報和不足夠的可操作洞見。
- 工具過多：如上所述，監控幾乎可以應用於任何數位領域，這意味著有太多的工具需要重疊和整合。這也是另一個錯過的機會。

## 37. 可觀察性與監控有何不同



監控是對系統的持續觀察，以檢測異常行為並發出警報。它涉及確保系統正常運作並在必要時採取糾正措施。

可觀察性是透過查看系統的輸出(指標、日誌和追蹤)來了解系統的內部狀態。它涉及了解系統內部正在發生的事情並預測它未來的行為。

可觀測性與監控的目標不同，**可觀測性並不能取代監控**，也不能消除監控的需要兩者是互補的。當使用了某個工具，就具備了某種形式的可觀測性。兩者都是為了協助系統在 debugged 更快找到可能的問題，是根據蒐集到的證據來解決其問題而不是靠直覺或是猜測。

監控可以告訴您何時出現問題，要解決已知的未知問題。而可觀察性可以告訴您發生了什麼、為什麼會發生以及如何修復，解決未知的未知問題。

可觀察性實踐者負責人Greg Leffler所說：

“可觀察性是一種心態，使您能夠回答有關整個業務的任何問題。”

當我們對於系統觀察到的越多，就可以理解其複雜的方式，不再需要假設各種系統服務是我們看不到的「黑盒子」。

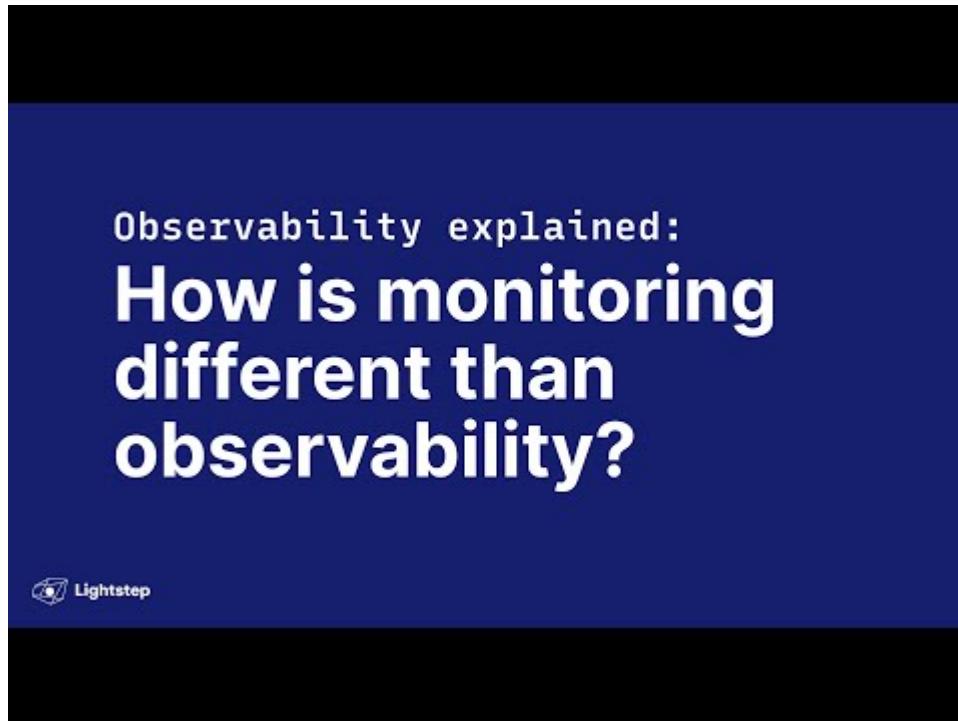
舉個玩世紀帝國的例子，監控就像是在已經打開的地圖前提下，在覺得危險的位子設立哨塔及哨兵，當有人來攻擊時就會有叮叮叮聲響來提醒玩家派兵防守



可觀測性就是一打開地圖就已經全開，當別的玩家派兵攻打自己的範圍就可以看到派人的兵種是甚麼及人數，可以做更有效的化解對方玩家的攻擊



推薦 Observability 與 Monitor 不同的影片



以上是今天的分享。下一篇要來探討可觀測性信號(Observability Signals)，如果有任何疑問或想法，歡迎留言提出討論！

## 38. 小結

- 透過監控可以了解系統健康狀況，但面對現代分散式系統有其局限。
- 監控專注於解決已知問題，而可觀測性則提供對未知問題的深入洞見。

## 39. 參考連結

[Observability vs. monitoring: What's the difference?](#)

[Monitoring and Observability](#)

# Day12 - 可觀察性的演進史：從控制理論到重新定義

---

大家好，我是伐伐伐伐木工

今天要與大家分享關於可觀測性過去幾年是如何不斷演進與重新定義的歷程

## 40. 演進史



### 40.1. 1960：可觀測性和控制理論

- 在技術領域，可觀測性一詞 起源於控制理論，這是動態工程和機械系統的數學領域
- 在系統中，可觀測性是衡量系統內部狀態可以從外部輸出的知識推斷出來的程度的指標。
- [魯道夫·E·卡爾曼 \(Rudolf E. Kálmán\)](#) 引入了該術語來描述系統可以透過其產出來衡量的程度。

### 40.2. 2013：Twitter 可觀測團隊描述其使命

- 2013.9 : Twitter 的工程師撰寫了一篇名為Twitter 的可觀察性的部落格文章，「可觀察性」一詞首次在 IT 系統中登場，內文如下

Twitter 的工程師需要確定其服務的性能特徵、對上游和下游服務的影響，並在服務未按預期運行時收到通知。可觀察性團隊的使命是利用我們用於收集、儲存和呈現指標的統一平台來分析此類問題。解釋了如何「捕獲、儲存、查詢、視覺化和自動化整個過程」

#### 40.3. 2016：Twitter 可觀察性的（四個）支柱

- Twitter 可觀測性工程團隊的 [Anthony Asta](#) 創建了一篇名為Twitter 可觀測性：技術概述，第一部分的部落格文章。

Twitter 的可觀測性工程團隊為我們的內部工程團隊提供全端程式庫和多種服務，以監控服務運作狀況、發出問題警報、透過提供分散式系統呼叫追蹤來支援根本原因調查，並通過創建聚合應用程序/系統日誌的可搜尋索引。

- 其中概述了他們團隊章程的四大支柱
  - 監控
  - 警報/可視化
  - 分散式系統追蹤基礎設施
  - 日誌聚合/分析

#### 40.4. 2017：可觀察性的三大支柱

- 2017.2 : [Peter Bourgon](#) 出席2017分散式追蹤高峰會。他參與了關於追蹤如何幫助提供可觀察性的定義和範圍的討論
- 在一篇名為「[指標、追蹤和日誌記錄](#)」的部落格文章中，他描述了他認為他們如何可能將儀器或可觀察性領域繪製成一種維恩圖



#### 40.5. 2018：可觀測年

- 可觀察性以及日誌記錄、指標和追蹤三大支柱成為主流對話的一部分。
- 2018.6, Humio 執行長 Geeta Schmidt 在一篇名為 [數據驅動的可觀察性和日誌](#)》的Medium 文章中加入了她的想法。

僅僅擁用於日誌管理、指標和追蹤的工具並不足以從中獲得價值。她堅持認為需要進行文化轉變，重視事實和回饋，在調試過程中以數據為驅動。並利用這種思維方式來迭代、改進和解決問題。

- 2018.7, [Cindy Sridharan](#)為 O'Reilly 出版了一本權威書籍《[分散式系統可觀察性](#)》。本書概述了可觀察性的三大支柱，並詳細介紹了使用哪些工具以及何時使用。
- 2018 年底，可觀測三大支柱模式開始出現裂痕。
- 2018.9 Honeycomb 技術長 [Charity Majors](#) 警告稱

可觀察性描述為三大支柱限制了討論。大膽地感嘆“[可觀測性不存在三大支柱](#)”，並補充道，“事實上，每個人都在盲目地重複這個口頭禪（以及貨物崇拜這些原語），這可能就是為什麼我們的可觀測性工具落後了10 年。”我們軟體工具鏈的其餘部分。”她進一步指出：「事件是程式碼通過系統的執行路徑。這是從內到外了解您的系統的正確視角。」

許多業內人士注意到了這一點，並開始考慮除了三大支柱之外的可觀察性。

## 40.6. 2019：激烈辯論隨之而來



隨著越來越多人重視與實際導入，可觀測性議題開始慢慢浮現

當公司購買並部署工具來從三個支柱中收集數據時，他們發現他們可以存取更多的系統數據，但他們仍然沒有實現基礎設施的 100% 可觀察性。事實證明，承諾可觀察性的工具無法處理來自 TB 級非結構化資料的資料量。工程師必須限制他們保留的數據，以保持在昂貴的許可證所限制的範圍內。他們發現他們使用的工具太慢，因為索引延遲和其他問題導致即時觀察變得不可能。而且新系統過於複雜，部署困難、維護不切實際且成本高、介面不一致、難學、難用。

整個產業的技術領導者加入了重新定義可觀察性的對話。

2019.2, LightStep 執行長兼聯合創始人 Ben Sigelman 發表了一篇名為《零答案的三大支柱：可觀察性的新記分卡》的部落格文章。

他描述了高基數指標如何壓倒系統。他指出，如果您想要攝取、保留和儲存來自分散式系統的大量數據，日誌就會變得太昂貴。如果您無法確定正確的樣本，那麼痕跡就會變得不切實際。他指出，即使所有這些都可以克服，但這些都不能直接解決特定的痛點、用例或業務需求。

2019.8, Glitch 的站點可靠性工程師 Mads Hartmann 在幾個月內深入研究了可觀察性，並在《可觀察性之旅：閱讀材料》中發表了他的想法。他的結論是 您無法僅透過系統產生的遙測（記錄和傳輸儀器讀數的過程）來實現可觀察性。

他提出了可觀察性的另一種觀點：

.....可觀察性就是能夠向系統提出問題並根據其產生的現有遙測數據獲得答案；如果您必須重新配置或修改服務才能獲得問題的答案，那麼您還沒有實現可觀察性。

Mads Hartmann 同意麥茲的前進方向。2019 年 8 月，她在 [The New Stack: Observability — A 3-Year Retrospective](#) 上發表文章。在其中，她領導了一項挑戰，為可觀察性提出了新的定義：

如果我們不使用「可觀察性」來表示已知與未知的未知之間、被動監控與探索性調試之間的差異，則不清楚我們還可以使用哪些其他術語（也不清楚同樣的命運不會降臨到它們身上）

她接著建議，實現可觀察性的承諾來自於能夠檢查系統的內部狀態並能夠提出任何問題來理解它。

透過使用事件並傳遞完整的上下文，相反，我可以詢問我的系統的任何問題並檢查其內部狀態，因此我可以理解我的系統已經進入的任何狀態- 即使我以前從未見過它，從未想到過它之前！我可以理解系統內發生的任何事情、系統可能處於的任何狀態，而無需發布新程式碼來處理狀態。這是關鍵。**這就是可觀察性。**

## 40.7. 2020：可觀察性（重新）定義

2020.11 CNCF TAG 小組開始撰寫 [可觀測性白皮書](#)，至今內容仍在撰寫中

可觀測性不僅僅是三種類型資料的部署和收集。可觀察性是你要麼擁有，要麼沒有的東西。只有當您擁有所有數據來回答任何問題（無論是否可預測）時，它才能實現。我們同意那些這樣想的人：

可觀察性是指您能夠從系統提供的資料中了解系統的內部狀態，並且您可以探索該資料來回答有關發生了什麼以及為什麼發生的任何問題。

就像是世紀帝國中，主城鎮在演進的過程中隨著不同時代(封建時期、城堡時期、帝王時代)而在外觀上會有不一樣的變化，其核心價值與精神是保持不變的，要真正了解所面臨到的問題跟挑戰，我們更需要的是深入探討這些演變的過程與其目的。



接下來會怎麼發展，近幾年自己觀察到有些議題越來越火

- OpenTelemetry 是分散式追蹤的新的規範，開發者可以深入了解應用程式的運行狀態，透過工具的配合可以更有效的掌握系統的全貌。
- 數據可觀察性 ( Data Observability ) 是希望透過收集數據，能夠更直觀的理解與解讀數據。
- 可觀測性驅動開發 ( O.D.D ) 是提供新的思考方式 (左移)，期許開發者可以從可觀測性的角度出發，確保系統在任何狀態下都可以提供清晰、有意義的呈現與回饋。
- eBPF 是一項革命性的技術，讓開發者可以在不改變核心代碼的情況下，對 Linux 系統進行深度觀察與修改。其特性也將成為下一代系統診斷與優化的關鍵工具。

OpenTelemetry 與可觀測性驅動開發之後(應該)會有章節來介紹，未來會怎麼發展讓我們繼續看下去 XDDD

以上是今天的分享。下一篇要來探討 Observability Signals，如果有任何疑問或想法，歡迎留言提出討論！

## 41. 小結

- 可觀察性是指您能夠從系統提供的資料中了解系統的內部狀態，並且您可以探索該資料來回答有關發生了什麼以及為什麼發生的任何問題。

## 42. 參考連結

[Observability is Also Programmed](#)

[Observability \(Re\)defined](#)

# Day13 - 可觀測性信號(Signals)的進化之旅

## 43. Observability Signals

可觀測性目的是希望能更了解你的系統，為了更了解系統運行中的狀態跟行為，會從不同的角度、不同的時間和管道蒐集不同的遙測(Telemetry)數據資料，透過這些資料類型我們可以推斷出系統的情況，將這些稱之為信號，在可觀測性中常見的三個信號包括 Metrics、Logging 與 Tracing。

在各大研討會與文章都可以看到上述為可觀測性三支柱 (尤其是工具商 XD)，但如果要定義得更明確，實際執行上更為清楚個人應該是 **Metrics、Structured Log 與 Distributed Tracing**，以下針對三者做簡單說明

### 43.1. Metrics : 指標

- 用來衡量和監控系統性能的關鍵數據指標，通常是數字化的方式呈現(可聚合數字)。
- 這些數據通常用於即時監控系統，以確保它們在正常運行範圍內。
- 例如：CPU使用率、記憶體使用率、請求速率、錯誤率等都可以通過指標來衡量

### 43.2. Structured Log : 結構化日誌

跟 Logging 有什麼分別？

日誌通常會包含有用的資訊，當程式出現異常時方便開發人員進行盤查查找問題，可能包含事件的描述，發生的時間，嚴重類型與其他像是用戶ID、IP等各種訊息。

傳統的日誌設計上是非結構化的，可能會是以行為單位為了方便人類閱讀，例如

```
2023-03-16T12:02:00 - info: Request 1234 started from user 5678. GET /my/endpoint
2023-03-16T12:02:00 - info: User 5678 authenticated - name foo
2023-03-16T12:02:01 - info: Processing request 1234
2023-03-16T12:02:02 - info: Request 1234 finished with status code 200. It took 2
seconds
```

結構化日誌是在原有的資訊中使用 key/value 加在其資訊中，方便機器進行解析的動作，將上面日誌換成結構化格式

```
timestamp=2023-03-16T12:02:00 level="info" message="Request started"
requestId="1234" userId="5678" path="/my/endpoint" httpMethod="GET"
timestamp=2023-03-16T12:02:00 level="info" message="User authenticated"
requestId="1234" userId="5678" userName="foo" userPlan="professional"
timestamp=2023-03-16T12:02:01 level="info" message="Processing request"
requestId="1234" rateLimited="false"
timestamp=2023-03-16T12:02:02 level="info" message="Request finished"
requestId="1234" durationMs="2000" statusCode="200"
```

結構化日誌是可觀測性除錯的基礎。這些日誌更容易被日誌系統取得，團隊也可以透過日誌系統任意的標準搜尋相關資訊。

- 日誌包含有關離散事件的結構化或可讀的詳細資訊。用於提供請求的細節與上下文。
- 日誌通常被用來瞭解系統中特定事件的發生情況，以及在事件發生時提供上下文。
- 例如，錯誤日誌、訪問日誌、應用程式日誌等都是常見的日誌類型。

### 43.3. Distributed Tracing : 分散式追蹤

- 用於追蹤複雜分佈式系統中請求的過程，以便了解請求從一個元件到另一個元件的傳遞情況。
- 它有助於識別性能問題和瓶頸，並提供有關請求流程的詳細信息。
- 通常，追蹤由一個唯一的標識符（例如 traceID）來關聯相關事件。



分散式追蹤的偵測有兩個主要目的：上下文傳播(context propagation)和跨度映射(span mapping)。上下文傳播是透過使用可與 HTTP 用戶端和伺服器整合的程式庫完成。在這一部分中，可以使用 OpenTelemetry API/SDK、OpenTracing 和 OpenCensus 等專案、工具和技術。

## 44. 三者關聯性



如果你是對可觀測性略有研究的朋友，相信一定看過上面這張圖，Peter Bourgon 在 2017 年參加完分散式追蹤研討會後所繪製的圖，說明 Metrics, tracing 及 logging 訊號彼此的關聯性與重要性。並在可觀測性中發揮不

同且互補的作用。

這句話聽起來好像有道理但又似乎有些模糊，這裡來舉個範例讓大家更容易理解，如上圖所示。

- 收到伺服器錯誤過多的警報提示，超過了我們所設定的 SLO 標準。
- Metrics：從錯誤計數器來看，發現是由於請求量變高導致伺服器回傳 501 比例變高。
- Logs：導航到 log 日誌，看來錯誤來自許多後面的內部微服務。
- Trace：透過 traceID 相同的 requestID 可以導航到 Trace，明確地找到多個服務中哪個緩慢的原因。

透過三個信號的結合，團隊才能清楚的確切地知道哪個服務或流程導致了問題，並進行了更多挖掘可能的問題動作。

## 45. 超越「三大支柱」：Continuous Profiling

These three pillars continue to be critically important. But it's important not to be confined by the "three pillars" paradigm and to choose the right telemetry data for your needs. from logz.io

OS：只有賽亞人才能超越超級賽亞人 XDDD

關於可觀察性的討論通常會提到「可觀察性三大支柱」，這兩年開始越來越多人開始討論 **連續性分析** (**Continuous Profiling**) 作為一種新的可觀測性信號

- 它是實時監控應用程式性能和效率的方法，與靜態性能分析不同，它是在應用程式運行時持續進行的。
- OpenTelemetry 已開始整合連續性分析，提供全面性能的可觀測性
- 透過連續性分析幫助優化性能、排查問題和提高系統穩定性

關於更多的 Continuous Profiling 資訊，可以參考 OpenObservability Talks 的介紹



以上是今天的分享，如果有任何疑問或想法，歡迎留言提出討論！OS：越來越長篇字數已爆，不知道有沒有休刊 OR 暫停連載的選項 XDDD

## 46. 小結

- 可觀測性三個重要的 Signals，分別是 Metrics（指標）、Structured Log（結構化日誌）和 Distributed Tracing（分散式追蹤）。
- 連續性分析（Continuous Profiling）是可觀測性新的趨勢，提供運行時間控應用性能的能力。

## 47. 參考連結

[Correlating Signals Efficiently in Modern Observability](#)

[Observability Signals](#)

[OpenTelemetry Roadmap and Latest Updates](#)

[5 Key Observability Trends for 2022](#)

[Continuous Profiling: A New Observability Signal](#)

# Day14 - 可觀測性管道：解析現代數據收集與分析

---

今天要與大家探討「可觀測性管道 Observability Pipeline」主題

## 48. 可觀測性管道

在上一篇提到可觀測性信號(Observability Signals)，為了處理這龐大的遙測數據資訊與分析，我們需要有效的方式來整合這些遙測數據，可觀測性管道的概念也因應而生，資料從它的來源地匯出到使用者需要查詢的地方，這包含以下四個不同的階段



- **收集(collection)**：接收可觀測資料，通常在主機上運行的代理程式進行。
- **攝取(ingestion)**：遙測資料在目的地處理，可能會經過批次、壓縮和其他轉換，以使資料處於最佳儲存形式。
- **儲存(storage)**：保存可觀測資料，可能會涉及索引以加快查詢速度。
- **查詢(query)**：尋找可觀測資料，可能會涉及將查詢條件轉換為對底層儲存系統的 Get/List 請求。

### 48.1. 獨立組件

當有多種可觀測性信號時，意味需要不同的收集、儲存跟查詢層獨立的系統處理方式。例如在 Open Source 的世界中，大家所重視的三個可觀測性信 Metrics、Traces、Logs 可以分別採用 Prometheus、ElasticSearch 和 Jaeger 作為解決方案。這裡列出每項服務的管道



Prometheus for Metrics

collect (prometheus scraper) -> ingest(prometheus) -> store (prometheus) -> query (prometheus)

Elasticsearch for logs

collect (logstash) -> ingest (elasticsearch) -> store (elasticsearch) -> query (elasticsearch)

Jaeger for traces

collect (jaeger collector) -> ingest (jaeger) -> store (cassandra) -> query (jaeger)

## 48.2. 統一採集 Unified Collection



- 在過去統一採集的標準有 OpenTracing 與 OpenCensus，各自有不同擁護者與供應商支持其格式。
- 2019 年 OpenTelemetry (aka OTel) 發布，為收集遙測數據的統一標準，並合併 OpenTracing 與 OpenCensus 重要專案。
- 2023 年 OpenTelemetry 已逐漸成熟，提供供應商中立的規範與實踐，可以從應用程式(支援的程式語言)來源收集指標、日誌和追蹤並將其送到目的地。

## 48.3. 統一儲存 Unified Storage

當可觀測性資料越來越多，隨之而來的挑戰是儲存資料空間與查詢效能的議題浮上檯面。在 Open Source 中可以透過以下解決方案

- Metrics : Prometheus with Cortex/Mimir
- Logs : Loki
- Trace : Tempo

## 49. 核心階段：Collect 採集

前面提到的四個階段中，資料來源地收集是相對重要的。它決定了我們可以收集多少資料，以及我們可以如何使用這些資料，這裡針對 Collect 階段做進一步的探討。在應用程式蒐集可觀測性資料時可能有下面流程



- receivers: 從不同的來源推拉數據資料
- processors: 將資料轉換/過濾/加工的動作
- exporters: 將資料傳送到目的地

以上組合共同創建出收集的可觀測性管道。透過簡單範例讓大家更清楚



- 收集 Log 資料將傳送到 AWS S3 的管道，並將錯誤的 Log 設定傳送到 datadog。
- 管道中將主機與容器資訊做為 Log 的自定義屬性添加到每筆資料中。
- 從可觀測性資料中刪除未使用的屬性。

接下來，我們來探討可觀測性管道的三個重要組件

### 49.1. Agent



- 代理主要運作於他們收集資料的或是應用程式上。
- 主要職責：拉取數據、進行輕量級的資料聚合，並將數據發送到預定的目的地。
- 每個可觀察性供應商都有其客製化的代理。

### 49.2. Collectors



- 收集器可以部署於它監控的相同系統上或作為獨立的部署。
- 主要職責：整合來自不同來源的可觀測性數據、應用基本過濾器，並將數據路由到一個或多個目的地。
- 常見的收集器：FluentD 和 Fluent Bit，兩者在社群中擁有多種整合解決方案。

### 49.3. Pipelines



- 可觀測性管道是收集器的進階版。
- 可以在同一系統、作為獨立部署或雲端上運作。
- 主要職責：能夠從任何來源接收數據，對蒐集到的數據進行多種轉換，並將其匯出到任何指定的目的地。

以上是今天的分享。下一篇休刊，如果有任何疑問或想法，歡迎留言提出討論！

## 50. 小結

- 可觀測性管道四個階段：收集(collection)、攝取(ingestion)、儲存(storage)和查詢(query)
- 三個重要組件：Agent、Collectors、Pipelines

## 51. 參考連結

[The Architecture of Modern Observability Platforms](#)

[First Mile Observability and the Rise of Observability Pipelines](#)

## Day15 - 開賽至今的回顧與反思



今天不聊可觀測性，來談對自己這半個月來的回顧與反思

## 52. 看見全貌

專案開始之初，首重[看見全貌](#) by Ruddy 老師

自從想不開報名鐵人賽後，一直不斷的思考主題拆程30天變成系列文後，反覆思考問自己下面的問題

從開賽到現在快進行一半，是不是有甚麼可以更好的地方？排版與標題需不需要再精確點？文章的內容跟用詞是不是夠淺顯易懂？

希望閱讀者可以對其主題可以更容易理解之外，也希望自己透過這過程加強自己表達能力，主要目的如下



透過將資訊系統化的整理，才能吸收它並轉化為自己的知識。透過不斷練習與改善，我們將它才會變成自己的技能。

自己在比賽初期自己有設定每天發文的 Template，其目的是讓大家更好的理解每天的主題、內容重點與小結，與其他文章的關聯性，也將草稿紀錄在 hackmd 上方便在不同地方撰寫，如下所示



## 53. 草稿的進化

撰寫 Day12 Observability 重新定義時忽然提醒了自己，平時團隊在開發時會定期兩周進行一次回顧，在自己的回顧(retrospective)與反思後Template 改版如下：(~~OS : format 很像軟體改版時發的 Release Note~~)

- **圖片**：選擇合適的圖片來匹配文章的主題，希望可以讓閱讀的人對於其內容有不同的了解，maybe 透過 OpenAI 產生圖片。
- **主題**：過去直球切入直接簡短兩三句，結果瀏覽點擊率慘不忍睹，會在用點心思想在主題名字上(看完一輪參賽者的主題文字，發現自己的好像不太用心 XDDD)ex：可觀測性 Observability > 現在化監控可觀測性 Observability
- **內容**：多用條列式呈現方便閱讀
- **小結**：每篇文章加上小結，透過精簡文字說明文章的核心內容
- **參考連結**：文章內容都是參考網路上文章，自己整理後呈現出來的，附上參考的文章內容與出處讓有興趣的人可以一起深入探討。

目的是希望大家在閱讀時可以更輕鬆，更可以理解每個章節想要表達的含意，比賽進行了一半也希望自己可以堅持下去。Happy reading！

---

PS：看到文章被瀏覽有 87 次(87分不能再高了)，無聊記錄一下 XDDD



## Day16 - 可觀測性與它的工具夥伴們



在雲端原生的時代，如何選擇最適合的可觀測性工具？

## 54. 如何選擇可觀測性工具

可觀測性旨在幫助開發人員、IT 團隊管理複雜的系統、應用程式和基礎設施。前面幾個章節了解完可觀測性想達成的目的與發展史之後，下一個大家所關心的就是在實務工作上要如何落地，在實踐上開發人員可能會有兩大方向，**SaaS 平台或是自己架設(Self-Host)**

### 54.1. SaaS 平台

使用可觀測性平台廠商 SaaS 服務(滿滿的大平台)，例如

- Splunk
- Honeycomb
- Lightstep
- New Relic
- Datadog
- 其他...

優點：快速上手、維護成本低 缺點：會有延伸費用

## 54.2. 自行架設

透過 CNCF 基金會底下的 Open source 專案，開發者可以透過 CNCF Landscape 工具搜尋到所需要類型的專案，應用在自己公司的開發專案上。

優點: 高度客製化、掌握度高 缺點: 維護成本、學習曲線

如果你是雲端開發者，相信對 CNCF 及上述提到的名詞已經而熟到不要不要的，但如果是雲端開發新人可能會有些陌生，以下就針對 CNCF Landscape 做簡短的說明

### 54.2.1. CNCF Landscape 是什麼？可以吃嗎

CNCF 是雲端原生運算基金會 (Cloud Native Computing Foundation) 是成立於 2015 年的非營利組織，旨在推動雲端原生技術的開發和採用。



CNCF Landscape 是一個基於 Web 的動態互動式工具，可提供雲端原生生態系統的概述。它的目標是將所有雲端原生(Cloud Native)開源專案和專有產品進行分類整理和組織，其專案內容不斷發展並定期更新，以反映雲原生空間的最新發展。提供雲端原生生態系統的全面概述。

目前在 CNCF 一共有 1,235 個專案，像是著名的 K8S、Grafana 和 Prometheus 都是在 CNCF 的專案中，專案分為四個等級

- **沙箱(Sandbox)**：尚未廣泛測試的實驗項目
- **孵化(Incubating)**：少數用戶在正式中成功使用的項目
- **已畢業(Graduated)**：穩定且被廣泛採用的項目
- **已存檔(Archived)**：不活躍、即將結束的項目。

決定使用 SaaS 平台還是自行架設可觀測性資料基礎架構取決於您團隊的可用資源、成本考量與對其熟悉程度等。SaaS 通常適合希望快速上手且不希望投入大量維護資源的團隊。而自行架設則適合希望有更大客製化空間，並願意投入資源維護的團隊。

## 55. 可觀測性專區

在 landscape 有分不同的領域專區，例如 App Definition and Development、Platform、CI/CD 等各向類型的專案。其中與 Observability 相關的有**可觀測性與分析(Observability and Analysis)** 區，在這領域又依據使用需求分為 **Monitor、Logging、Tracing、Chaos Engineering** 與 **Continous Optimization**，上述在 CNCF 都有推薦的工具，整理相關資訊如下

領域	關鍵字	畢業專案	沙箱	孵化
Monitor	Monitoring	Prometheus	Fonio	
	Time series		Headlamp	
	Alerting		Inspektor Gadget	Cortex
	Metrics		Kepler	OpenMetrics
			Kuberhealthy	Thanos
			Pixie	
			Skooner	
			Trickster	

領域	關鍵字	畢業專案	沙箱	孵化
Logging	Logging	Fluentd	-	-
Tracing	Span Tracing	Jaeger	-	OpenTelemetry
Chaos Engineering	Chaos Engineering	-	Chaosblade	Chaos Mesh Litmus

選擇合適的可觀測性工具是重要的一步。上述提到了各領域推薦的工具接下來將會選擇挑選重要的來介紹，在下個章節，我們將深入探討這幾年火紅的 OpenTelemetry！

## 56. 小結

- SaaS 通常適合希望快速上手且不希望投入大量維護資源的團隊。而自行架設則適合希望有更大客製化空間，並願意投入資源維護的團隊。

## 57. 參考連結

[DevOps Institute](#)

[What is the CNCF?](#)

[A Guide to Enterprise Observability Strategy](#)

[Avoiding the Roadblocks: How to Choose the Right Tools for Your Observability Stack](#)

# Day17 - OpenTelemetry : 在收集遙測數據中加上一點新標準



為什麼我們需要分散式追蹤？日誌 (Log) 跟指標 (Metrics) 不夠用嗎

## 58. 為什麼需要分散式追蹤

隨著科技的進步，測量系統指標和發送警報也變得更加複雜；許多組織每天處理數百甚至數千萬個請求，以 Uber 為例內部系統有 [4,000 個微服務](#) 彼此關聯。

如果你的架構圖像是下面圖示，一個請求可能會經過數十或是數百個網路節點。架構切分這麼細的情況下，這使得了解請求所經過的路徑變得相當困難，如果你只有日誌跟指標，要進行故障排除也是相當複雜的。在問題發生時，對開發或是運維團隊來說，需要了解的是怎麼找到根本問題(Root Cause)及經過哪些服務。



分散式追蹤可協助您查看整個請求 (Request) 期間服務之間的交互，並讓您深入了解系統中請求的整個生命週期。它幫助我們發現應用程式中的錯誤、瓶頸和效能問題。

意味著當使用者透過瀏覽器發送請求到伺服器應用程式的那一刻起，我們可以可看到整個請求的過程。

(追蹤)資料(以跨度的形式)產生遙測資料，可以幫助了解延遲或錯誤是什麼，以及為什麼會發生還有對整個請求的影響。



## 59. 什麼是 OpenTelemetry

近期很多在看很多 Open Source 專案都用相似方式來在介紹，為了跟上這股潮流我們也不能輸，以下參考官網說明

### OpenTelemetry 是什麼

OpenTelemetry 是一個雲端原生運算基金會(CNCF)專案。OpenTelemetry 是一個可觀察性框架和工具包，旨在創建和管理遙測數據，例如追蹤、指標和日誌。OpenTelemetry 與供應商和工具無關，這意味著它可以與各種可觀察性後端一起使用，包括 Jaeger 和 Prometheus 等開源工具以及商業產品。

### OpenTelemetry 不是什麼

OpenTelemetry 不是像 Jaeger、Prometheus 或商業供應商那樣的可觀察性後端。OpenTelemetry 專注於遙測資料的產生、收集、管理和匯出。該資料的儲存和視覺化有意留給其他工具。

## 60. 使命、願景和工程價值

了解其設計初衷跟方向，才會知道為什麼這樣設計與其它想要試圖解決的問題

### 60.1. 使命(Mission)：無所不在的高品質、可攜式遙測

### 60.2. 願景：有效的可觀測性世界

OpenTelemetry 的願景是實現一個有效的可觀測性世界。這個願景的核心思想是，有效的可觀測性需要高品質的遙測技術，以及使其成為可能的高性能、一致的儀器。在這樣的世界中，遙測應該是 **容易的、通用的、與供應商無關的、鬆散耦合的、內建的**。

### 60.3. 工程價值：相容性、穩定性、彈性和效能

OpenTelemetry 重視以下四個關鍵價值：

- **相容性** OpenTelemetry 致力於相容性，這意味著遵循規範並實現互通性非常重要。我們希望 OpenTelemetry 能夠跨語言和組件一致，並保持供應商中立。
- **穩定性** API 穩穩定性和向後相容性對於 OpenTelemetry 的程式庫來說至關重要。我們不會引入新概念，除非確信廣泛子集需要它們。
- **彈性** OpenTelemetry 強調技術彈性，即使在資源稀缺或其他環境挑戰下，也能適應並繼續運作。我們設計 OpenTelemetry 能夠優雅地應對應用程式行為異常，並持續收集遙測訊號。
- **效能** 高效能是 OpenTelemetry 的要求，我們不希望用戶在高品質遙測和高效能應用程式之間做出選擇。OpenTelemetry 確保高效能，並且不接受主機應用程式中的意外干擾。

## 61. 分散式技術追蹤的演進

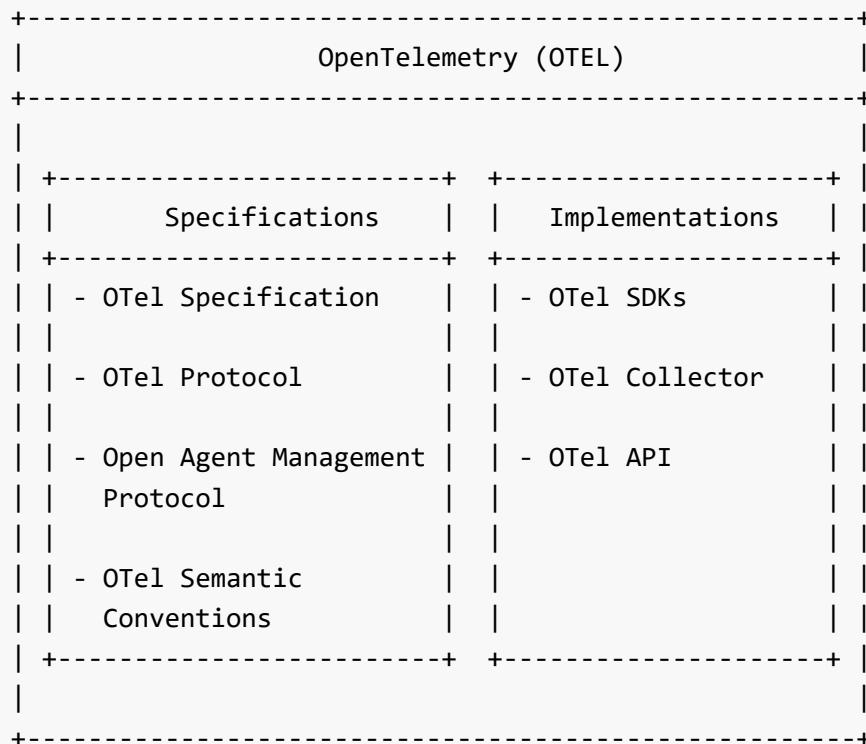
在過去為了嘗試解決分散式追蹤 (Distributed Tracing) 問題，有多種不同的 Open Source 專案，以下是簡單的整理 ([圖片來源](#))

- 2010 年 : Google 發表一篇名為 Dapper 的論文，描述分散式追蹤系統的運作。
- 2016 年 : CNCF 發表 OpenTracing 專案目的是為了分散式追蹤提供標準化。
- 2017 年 : Google 再次創新，推出 OpenCensus 作為分散式追蹤和監控的 library
- 2019 年：
  - OpenCensus 及 OpenTracing 進行整合，OpenTelemetry 成為新一代的遙測數據蒐集標準。
  - OpenTelemetry 實作 W3C Distributed Tracing 所定義的規範與標準

## 62. 核心元件

OpenTelemetry (aka OTel) 在可觀測性的領域中扮演越來越重要的角色，近幾年已成為 CNCF 中第二個活躍的 Open source 專案，受歡迎程度僅次於 Kubernetes，它的貢獻者遍布所有主要的可觀測性供應商，其協議在可觀測性供應商中幾乎被普遍採用。有興趣可以參考 [CNCF Dev Status](#)。

OpenTelemetry 核心分為兩個部分，分別是規範與實作



由於資料內容過多體力不支，下一章節再繼續介紹核心的元件 XDDD

## 63. 小結

小結呼應主題搭配梗圖  <https://ithelp.ithome.com.tw/upload/images/20231002/20162577f9xqUEfRhW.png>

## 64. 參考連結

[What is OpenTelemetry?](#)

[Observability Whitepaper](#)

[OpenTelemetry mission, vision, and values](#)

[OpenTelemetry Up and Running](#)

[OpenTelemetry in 2023](#)

# Day18 - OpenTelemetry : 核心元件大解密 (1/2)

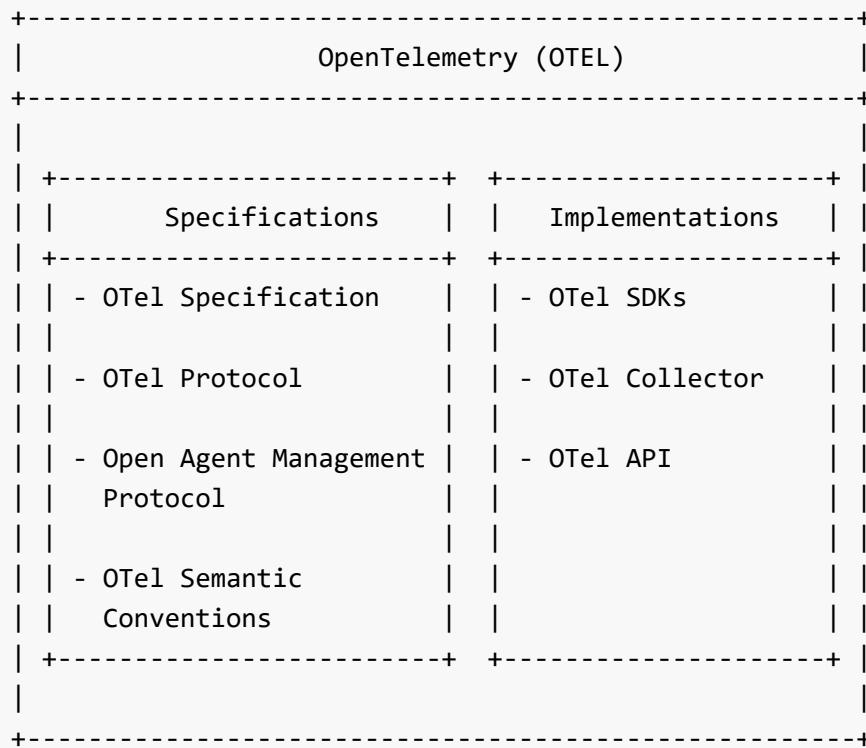


OpenTelemetry 看起來很厲害，背後是如何做到的呢？

OpenTelemetry 是一個開源工具、API 與 SDK 集合。用於雲原生 (Cloud Native) 應用程式、為服務和分散式系統，透過它可以協助收集與匯出遙測資料(Trace、Metrics和Logs)。使用這些數據以及對分散式系統的可觀測性，開發人員可以排除故障、進行 Debug、測試和監控應用程式。提高應用程式的可靠性和可擴展性。

今天我們將介紹 OpenTelemetry 的各核心元件。簡稱 OTel，在下面內容中使用 OTel 作為替代 (自以為專業)。

## 65. 核心元件



### 65.1. OpenTelemetry Specification : 規格

OpenTelemetry Specification 是 OTel 的基礎，提供所有 OTel 標準所衍生的 API、SDK 和資料模型。

從 2019 開始至今，發展時間表

- 2019.09 Tracing stable
- 2021.11 Metrics stable
- 2023.04 Log stable

對於可觀測性重要的信號(Signals)是穩定支援的，想了解信號可以參考 [Day17 : 可觀測性信號的進化之旅](#)

## 65.2. OpenTelemetry SDKs : 程式語言的 API 實作

提供基於 OTel 規範的客戶端工具。對於每種程式語言提供其 SDK，幫助開發人員更快速的收集、處理和導出遙測(Telemetry)資料，在不同信號中都有其成熟度等級。



追蹤數據可以使用兩種方式產生，分別是 **自動(automatic)** 或 **手動(manual)**

**自動**：Auto Instrumentation SDK 會自動將信號或屬性欄位注入到應用程式中，例如上下文傳播和語義約定，不需要太多額外的程式碼來收集，透過這方式開發人員可以簡單、輕鬆地對應用程式進行分散式追蹤。

**手動**：Manual Instrumentation 指為應用程式加上特定的程式碼，透過 OTel API 向應用程式添加可觀測性代碼的過程，可以更有效的滿足客製欄位上的需求，例如：新增自定義的屬性和事件內容。

程式語言支援細節可參考官方網站的 [Status and Releases](#)

應用程式收集遙測數據資料後，下一步是將其數據傳送到某個地方

以上是 OTel 相關核心的元件關於 spec 與 SDKs 的介紹，如果有任何疑問或想法，歡迎留言提出討論！

## 66. 小結

- OpenTelemetry (OTel) 是一個開源工具、API 與 SDK 集合，用於雲原生應用程式和分散式系統，幫助收集和匯出遙測資料。
- 核心元件包括 Specification ( 規格 ) 、SDKs ( 程式語言的 API 實作 ) 、Protocol ( 傳輸協定 ) 、Collector ( 資料處理方式 ) 、和 Semantic Conventions ( 通用數據定義方法 ) 。

## 67. 參考連結

[OpenTelemetry: A full guide](#)

[awesome-opentelemetry](#)

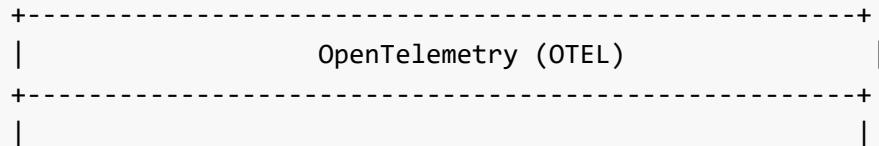
[What Are Semantic Conventions in OTEL?](#)

[A beginner's guide to OpenTelemetry](#)

## Day19 - OpenTelemetry : 核心元件大解密 (2/2)

 <https://ithelp.ithome.com.tw/upload/images/20231004/20162577lg2o0FkDPZ.jpg> 上篇介紹負責蒐集應用程式遙測數據的元件，這篇將介紹後續處理的原件 OTel Collector

## 68. 核心元件 (2/2)



Specifications	Implementations
- OTel Specification	- OTel SDKs
- OTel Protocol	- OTel Collector
- Open Agent Management Protocol	- OTel API
- OTel Semantic Conventions	

## 68.1. OpenTelemetry Protocol (OTLP): 傳輸可觀測性數據協定

開放遙測協定 (OTLP) 規格描述了遙測源、採集器等中間節點和遙測後端之間遙測資料的編碼、傳輸和交付機制 [name=open-telemetry/opentelemetry-proto]

OTLP 協定描述如何編碼和傳輸遙測數據，白話意思是可以在接收、處理或導出 OTEL 資料的任何服務上實現。每種語言 SDK 都提供一個 OTLP 匯出器，您可以將其配置為透過 OTLP 匯出資料。然後 OpenTelemetry SDK 將事件轉換為 OTLP 資料。

## 68.2. OpenTelemetry Collector: 接收、處理和匯出遙測資料的方式



OTel Collector 是 OTel 重要核心元件之一，用於收集、轉換處理並發送遙測資數據，與供應商無關(vendor-agnostic)。也可依據團隊收集數據資料的需求進行採樣(sampling)設定。

### 68.2.1. 組成

OTel Collector 由以下組件組成

- Receivers : 接收器接受指定格式的數據，將其轉換為內部格式，並將其傳遞給適用管道中定義的處理器和匯出器
- Processors : 轉換/過濾/加工/發送數據
- Exporters : 將資料傳送到下游(一個或多個)目的地
- Connectors : 依據其需求，可以將多個管道彙總一起
- Extensions : 提供處理遙測數據之外的附加功能，例如基本驗證、執行狀況檢查等

各組件搭配一起成為可觀測性管道(Observability Pipeline)，可以讓使用者從不同來源 (例如OpenTelemetry SDK、代理或導出器) 收集遙測數據資料，並在收集資料過程中進行轉化加工處理，再將處理後的資料傳送到任何目的地 (destinations)。

OpenTelemetry 不負責儲存後端或呈現結果。如果需要儲存或可視化，您可以參考 Awesome OpenTelemetry 中的存儲解決方案。

## 68.2.2. 不負責後端及呈現結果

另外提醒，OpenTelemetry 不提供儲存後端及呈現結果，如果需要儲存或可視化，您可以參考 [Awesome OpenTelemetry 中的存儲解決方案](#)。

## 68.2.3. 彈性與生態系

OpenTelemetry 提供了一種跨各種程式語言、平台和雲端供應商檢測、收集和匯出遙測資料的標準方法。這種標準化可確保在不同環境和系統中以一致的方式收集和報告遙測數據，減少整合上的成本並使其更易於使用，使用上也具備彈性可因不同需求來搭配不同的Receivers、Processors 及 Exporters。

- 支援的接收器列表
- 支援的處理器列表
- 支持的出口商名單



另外可以參考 [OpenTelemetry 生態系統](#)，了解更多支援 OTEL 的工具、追蹤器實作、實用程式和其他有用的項目。

## 68.3. OpenTelemetry Semantic Conventions : 通用數據定義方法

提供了一種定義如何在系統中的不同元件之間建構和交換資料的方法。透過使用語意約定，各種工具和服務可以一致地解釋和處理資料。白話就是定義了可觀測性資料的一組通用屬性。它們涵蓋了廣泛的領域，包括雲端資源、資料庫、異常和系統。

以上是 OTEL 相關核心的元件介紹與分享，如果有任何疑問或想法，歡迎留言提出討論！

## 69. 參考連結

[OpenTelemetry: A full guide](#)

[awesome-opentelemetry](#)

[What Are Semantic Conventions in OTEL?](#)

[A beginner's guide to OpenTelemetry](#)

[OpenTelemetry in 2023](#)

[OpenTelemetry in 2023](#)

## Day20 - OpenTelemetry : Demo 專案快速入門 (1/2)

---



前面兩天介紹關於 OpenTelemetry 重要的功能，身為一位專業的村民還是需要看到程式碼會比較有安全感，今天我們就透過 Demo 專案來體驗 OpenTelemetry 帶來的幫助

## 70. 不明覺厲的 OpenTelemetry Demo 專案

在 OpenTelemetry 官網有提供 **OpenTelemetry Astronomy Shop** 範例專案，它是一個基於微服務的分散式系統，模擬了線上商店的操作，以下是關於這專案的重點介紹

- 提供用於 Demo OpenTelemetry 儀器和可觀測性的分散式系統的實際案例與架構介紹。
- 包含 15 種不同的服務，使用 10 多種不同的程式語言，以模擬網路商店操作行為。
- 為供應商、工具作者和開發者建立一個基礎，以擴展他們的 OpenTelemetry 整合。
- 它使用負載產生器（Locust）不斷向前端發送模仿真實使用者購物流程的請求
- 該專案使用 Jaeger 與 Grafana 等工具來診斷異常問題

## 70.1. 快速開始

在開始之前的前置作業，需要先在電腦上安裝 Docker。

Clone Demo repo `git clone https://github.com/open-telemetry/opentelemetry-demo.git`

移駕到該 repo 目錄 `cd opentelemetry-demo/`

執行 Docker Compose 指令 `docker compose up --no-build`

開啟瀏覽器輸入網址 `http://localhost:8080/`，看到下列圖片就代表網站啟動成功



## 70.2. 工具

- 網上商店：`http://localhost:8080/`
- Grafana：`http://localhost:8080/grafana/`
- 功能標誌 UI：`http://localhost:8080/feature/`
- 負載產生器 UI：`http://localhost:8080/loadgen/`
- Jaeger 使用者介面：`http://localhost:8080/jaeger/ui/`

### 70.1. 應用程式架構



#### 重點摘要

- 15 個以上服務
- 10 種以上程式語言開發
- http + grpc
- 使用負載產生器（Locust）模仿真實使用者購物流程的請求

### 70.2. 分散式重要觀念

OpenTelemetry 中基本的對像是事件（event）。事件只是一個時間戳和一組屬性。大多數屬性對單個事件來說並不獨特。相反，它們是一組事件所共有的。例如，`http.target` 屬性與作為 HTTP 請求的每個事件相關。如果在每個事件上反覆記錄這些屬性，效率會很低。相反，我們把這些屬性拉出到圍繞事件的封裝中，在那裡它們可以被寫入一次。我們把這些封裝稱為上下文（context）。

有兩種類型的上下文靜態和動態，如下圖所示



- 靜態上下文**：定義了事件發生的物理位置。在 OpenTelemetry 中，這些靜態屬性被稱為資源。一旦程序啟動，這些資源屬性的值通常不會改變。
- 動態上下文**：定義了事件參與的活動操作。這個操作層面的上下文被稱為跨度（span）。每次操作執行時，這些屬性的值都會改變。



**span** 是我們描述因果關係的方式。TraceID、SpanID 和 ParentID，這三個屬性是 OpenTelemetry 的基礎。

屬性	類型	描述	範例
traceid	16字節數組	識別整個交易	bf92f3577b34da6a3ce929d0e0e4736
spanid	8字節數組	識別當前操作	00f067aa0ba902b7
parentid	8字節數組	識別父操作	3ce929d0e0e4736

通過添加這些屬性，我們所有的事件現在可以被組織成一個圖。這種類型的圖被稱為追蹤（trace）。透過 TraceID 索引，可以找到該請求中所有的 Log 資訊。

### 70.3. 模擬異常：讓他爆



在這範例專案中，有提供模擬故障的功能可以到 [功能標誌設定頁](#) 啟用需要失敗的功能設定，將所有的設定改為 **true**（讓他爆）

範例專案有負載產生器（Locust）不斷發送模仿真實使用者購物流程的請求，加上 1/10 的錯誤發生機率，我們等待 5~10 分鐘再來觀看可能有哪些錯誤。

### 70.4. Trace：使用 Jaeger 定位問題

**Jaeger** 是 CNCF 畢業的專案，用於分散式系統監控和故障排除的開源工具，透過視覺化的工具追蹤性能、定位故障點、優化性能，提升系統穩定性。

開啟 [Jaeger UI](#) 畫面，在 Tag 輸入 error = true，按下 Find trace 可以看到部分請求有 error 的狀況發生



點擊軌跡，您可以查看該請求經過的服務列表、執行順序以及每個不同服務所花費的時間。這可以幫助您定位問題。

以下是正常的請求，當按下購物車後的請求，可以看到該發送請求(Request) 後與 9 個服務 (span) 關聯與執行順序，其每個 span 的 Service Name 與時間長短。



錯誤請求時，點擊進去則可以看到哪一段發生異常，這範例是我們先使用設定頁讓 adservice 功能一定機率異常，遇到時發生的錯誤，在點擊異常紅色的 span 則可以看到像是 error message 等 debug 需要的資訊



### 70.5. Metrics：使用 Grafana 查看 OTel Collector 狀況

範例專案可以開啟 Grafana 查看內建的 Dashboard 儀表板資訊，了解遙測數據的收集狀況，目前有四個部分

- Process Metrics
- Traces Pipeline
- Metrics Pipeline
- Prometheus Scraping

### OpenTelemetry Collector Dashboard



透過此 Dashboard 可以了解可觀測性管道 Receivers、Processors、Exporters 收集/發送數據的狀況 (資料流) 以及每個管道的導出比例。

### Opentelemetry Collector Data Flow 觀看可觀測性信號遙測數據的資訊，看到來源有 http 與 grpc, Trace



### Prometheus Scrape

反思：沒有這些工具，要如何快速定位問題呢

以上是針對 OpenTelemetry 專案簡單的介紹，對開發人員帶來的幫助是使用 Jaeger 和 Grafana 進行故障排除和監控，有效的縮短異常事件處理的時間。下一篇我們再繼續介紹是如何做到的。

## 71. 參考連結

[OpenTelemetry in Action](#)

[Development](#)

[opentelemetry-demo](#)

## Day21 - OpenTelemetry : Demo 專案快速入門 (2/2)



OpenTememtry Demo 專案是如何做到的呢？

## 72. Telemetry Data Flow : 遙測數據蒐集流程



上圖是 OTel 範例專案的數據流程圖，從圖中可以得知重要元件有 OTel Demo Application、OTel Collector、Prometheus、Jaeger 及 Grafana。以下就針對這些元件做簡單說明

### 72.1. OpenTelemetry Demo

OTel Demo是由 10 多種不同的程式語言開發的微服務架構應用程式，在 Day18 的介紹中有提到蒐集遙測數據方式有自動跟手動兩種，以 .NET 開發的 Cart Service 為例，其是使用自動(automatic)採集，再將蒐集好的資料使用 HTTP、grpc Protocol 傳遞到 OTel collector。

在 .NET 專案 Program.cs 可以在程式碼中可以看到使用 `AddOpenTelemetry()` 方法後加上 `AddAspNetCoreInstrumentation` 與 `AddRedisInstrumentation` 自動蒐集 dotnet 應用程式與 Redis 的遙測資料

```
    Action<ResourceBuilder> appResourceBuilder =
        resource => resource
            .AddDetector(new ContainerResourceDetector());

    builder.Services.AddOpenTelemetry()
        .ConfigureResource(appResourceBuilder)
        .WithTracing(tracerBuilder => tracerBuilder
            .AddRedisInstrumentation(
                cartStore.GetConnection(),
                options => options.SetVerboseDatabaseStatements = true)
            .AddAspNetCoreInstrumentation()
            .AddGrpcClientInstrumentation()
            .AddHttpClientInstrumentation()
            .AddOtlpExporter());
```

### 72.1.1. OTel Collector

- Receivers :
  - 透過兩個不同的端點接收數據進行監聽
  - HTTP : `http://localhost:4318`
  - gRPC : `grpc://localhost:4317`
- Processors
- Exporters
  - OTel 使用兩種方式導出數據
  - OTLP HTTP Exporter 將數據發送到 `http://localhost:9090/api/v1/otlp`
  - OTLP Exporter grpc 數據發送到 Jaeger

設定檔 : otelcol-config.yml、[otelcol-observability.yml](#)

```
# Copyright The OpenTelemetry Authors
# SPDX-License-Identifier: Apache-2.0

exporters:
  otlp:
    endpoint: "jaeger:4317"
    tls:
      insecure: true
  otlp/logs:
    endpoint: "dataprepper:21892"
    tls:
      insecure: true
  otlphttp/prometheus:
    endpoint: "http://prometheus:9090/api/v1/otlp"
```

```

tls:
  insecure: true

service:
  pipelines:
    traces:
      exporters: [otlp, logging, spanmetrics]
      metrics:
        exporters: [otlphttp/prometheus, logging]
      logs:
        exporters: [otlp/logs, logging]

```

OTel Collector 詳細配置設定介紹請參考：[官網](#)

### 72.1.2. Prometheus

- 收集器使用OTLP協議將數據發送到 Prometheus。
- 存儲在其時間序列數據庫 (TSDB) 中。
- 可以通過瀏覽器訪問 <http://localhost:9090> 開啟 Prometheus UI 進行操作。

### 72.1.3. Jaeger

- Jaeger 收集器在 <grpc://jaeger:4317> 上監聽遙測數據。
- 數據存在 Jaeger DB 中。
- 可以通過瀏覽器訪問 <http://localhost:16686> 開啟 Jaeger UI 進行操作。

### 72.1.4. Grafana

- 資料來源
  - Prometheus : <http://localhost:9090/api>
  - Jaeger : <http://localhost:16686/api>
- 可以通過瀏覽器訪問 <http://localhost:3000/dashboard> 開啟 Grafana UI 進行操作。

## 73. Manual Instrumentation

當需要自定義 Span 屬性時，該如何設定呢？

在 [Cart Service](#) 中有幾個自定義屬性

Name	Type	Description
app.cart.items.count	number	Number of unique items in cart
app.product.id	string	Product ID for cart item
app.product.quantity	string	Quantity for cart item

在 .NET 中我們可以使用 [Activity](#) 類別，設定其 tag 進行自定義屬性設定，將重要資訊透過設定在既有的請求上，將屬性一直往後傳遞下去

例如 : `cart.items.count`

```
var activity = Activity.Current;
activity?.SetTag("app.user.id", request.UserId);
activity?.AddEvent(new("Fetch cart"));

var cart = await _cartStore.GetCartAsync(request.UserId);
var totalCart = 0;
foreach (var item in cart.Items)
{
    totalCart += item.Quantity;
}
activity?.SetTag("app.cart.items.count", totalCart);
```

例如 : 清空購物車時 , 紀錄 UserId 。當遇到異常的狀況時亦可將錯誤資訊記錄下來 , 如下所示

```
public override async Task<Empty> EmptyCart(EmptyCartRequest request,
ServerCallContext context)
{
    var activity = Activity.Current;
    activity?.SetTag("app.user.id", request.UserId);
    activity?.AddEvent(new("Empty cart"));

    try
    {
        if (await _featureFlagHelper.GenerateCartError())
        {
            await _badCartStore.EmptyCartAsync(request.UserId);
        }
        else
        {
            await _cartStore.EmptyCartAsync(request.UserId);
        }
    }
    catch (RpcException ex)
    {
        Activity.Current?.RecordException(ex);
        Activity.Current?.SetStatus(ActivityStatusCode.Error, ex.Message);
        throw;
    }

    return Empty;
}
```

關於資料蒐集的重要觀念之前已介紹過 , 可以參考下列章節回顧

- Day14 - 可觀測性管道 : 解析現代數據收集與分析
- Day18 - OpenTelemetry : 核心元件大解密 (1/2)
- Day19 - OpenTelemetry : 核心元件大解密 (2/2)

最後關於 Demo 範例專案內容可以參考官網 [OpenTelemetry Document 文件](#)，內有更豐富的資源提供參考，詳細說明就整理到這邊不在追述。

## 74. 參考連結

## OpenTelemetry in Action

## OpenTelemetry Demo Documentation

## Day22 - 可觀測性摘要 cheatsheet 及小結

## 以下列出這幾天可觀測性文章相關重點文字整理

- **可觀測性 Observability**
    - 可觀察性是指您能夠從系統提供的資料中了解系統的內部狀態，並且您可以探索該資料來回答有關發生了什麼以及為什麼發生的任何問題。
  - **監控和可觀測性差異**
    - 監控是告訴你什麼沒有工作。
    - 可觀察性是告訴你為什麼它不工作。
  - **可觀察性的演進史**
    - 從 1980 年代的控制理論演變到現今的高度結構化和詳細的可觀察性過程
  - **可觀測性信號 Observability Signals**
    - 可觀測性三個重要的 Signals，分別是 Metrics ( 指標 ) 、 Structured Log ( 結構化日誌 ) 和 Distributed Tracing ( 分散式追蹤 ) 。
    - 連續性分析 ( Continuous Profiling ) 是可觀測性新的趨勢，提供運行時間控應用性能的能力。
  - **可觀測性管道 Observability Pipeline**
    - 可觀測性管道四個階段：收集(collection)、攝取(ingestion)、儲存(storage)和查詢(query)
    - 三個重要組件：Agent、Collectors、Pipelines
  - **可觀測性工具**
    - Cloud Native Projects ( 原生雲專案 )
    - Observability & Analysis 專區清單整理

文字理解幫助還是有限，因此自己整理圖形化的 cheatsheet 關係圖，方便大家更好的理解彼此依賴關係



邁入鐵人賽第 22 天也開始慢慢接近尾聲，如第一天提到 Observability in Devops 與可觀測性 (Observability) 內容，也簡單介紹可觀測性中扮演重要元素的 OpenTelemetry 遙測數據收集新的標準，接著就是要進入工具實踐介紹的章節，也希望自己可已順利完賽！

Day23 - 可觀測性工具的整合與挑戰



## 75. 可觀測性開源工具的挑戰

在 [Day16 - 可觀測性與它的工具夥伴們](#) 中，我們介紹了可觀測性信號 (Signals) 的概念，以及 CNCF 可觀測性推薦的 open source 工具清單，進而從不同類型的應用程式架構 (單體式或是分散式系統)、不同來源的遙測數據

信號 (metrics、log、trace) 匯集到特定的資料湖 (data lake) 後，方便開發人員可以更了解系統的運作與回答相關問題。

### 在實際導入可觀性工具會遇到那些挑戰呢？

根據 2020 年 9 月 CNCF 可觀測性報告中調查所呈現的技術雷達結果



- 最常用的可觀測性工具是開源的
  - 如 Prometheus、Grafana、Elastic、Jaeger 和 OpenTelemetry。
  - 可觀測性工具的多樣性，很難一個工具滿足所有的維度
- 工具之間沒有整合
  - 一半的公司使用5種以上，三分之一的公司使用10種以上。
  - 每個工具有不同優勢，用戶傾向於使用多種工具。
  - 轉換或整合工具的成本高昂
- Prometheus 和 Grafana 同時使用
  - 三分之二的受訪者同時使用 Prometheus 和 Grafana。
  - 這兩個工具之間的高度相關性是由於它們背後的強大推動力以及大量的教程和安裝指南。

在可觀測性中 Open source 工具扮演著重要的地位，使用開源的可觀測性工具帶來的好處時，其實也應該關注開源背後所帶來的問題與挑戰

- 工具整合的挑戰
  - 不同工具帶來整合問題，如 CNCF 報告顯示，半數使用5種以上，三分之一使用10種以上工具
  - 工具的蔓延不只帶來操作上的困難，還會造成資料孤島，進一步阻礙資料分析。
- 開源軟體重新授權
  - 近年來，許多領先的開源專案進行了重新授權，可能導致使用者不得不重新評估其選擇。
  - 這些變化包括更嚴格的許可證、Copyleft 許可證（例如 GNU AGPL）或非開源許可證。
  - 有些公司甚至禁止使用某些許可證，如 AGPL，因為它們認為風險超過了收益。
- 學習與維護成本
  - 使用多種工具可能導致更高的培訓、維護和更新成本。
  - 頻繁切換工具可能中斷工作流程並增加錯誤的風險。

工具的多樣性與整合是可觀察性領域的挑戰之一。同時，此外，開源軟體的重新授權趨勢也影響了開發者的選擇和公司的策略。

## 76. 解決方案：Observability Platform？

國際研究機構 Gartner 定期整理在可觀測性與 APM (Application Performance Monitoring and Observability) 領域中解決方案的分析，每年都會整理該領域中評比後的領導者，以下是 Gartner 2023 報告列出的一些解決方案（包含但不限於）https://ithelp.ithome.com.tw/upload/images/20231008/20162577Zx9KCY8IHw.png

上述解決方案可以在 [這裡](#) 查看更多介紹與細節，這裡就不在多加說明。

在選擇可觀測性的解決方案上，可以參考 CNCF Observability 專區推薦的工具以及上面 Gartner 中整理的解決方案，在近幾年可觀測性報告中都有提到**工具的整合性**是首要問題（也可能是產品銷售策略放大問題 XD）

以下是實際情況遇到的問題

- 當系統異常時，開發人員為了查找 Metrics、Log 與 Trace 資訊，需要在不同工具中進行切換動作找 root cause
- 當系統維運上，開發人員要評估工具出新版本，包含安全性與升級計畫，還有平常不同工具的維護成本
- 當人員招募時，職缺內容上需要要求熟悉滿滿的 open source 工具，這也會增加招募的困難 ~~員工心累但員工不敢說~~

這些問題可能是是在實際會遭遇的問題，也是不能忽略的隱形成本。如果有一個可觀測性平台整合可觀測性信號數據，可節省更多開發人員學習及維運的成本與時間，有機會解決 CNCF 報告中提到的工具之間沒有整合問題。

接下來幾天將簡單介紹 Grafana Cloud 的可觀測性平台，兩者除了火紅之外，相關的學習資源也是豐富的，無論是想要進入可觀測性領域或是改進監控方案上，這兩個都可以幫助團隊實現更好的可觀測性，如果有任何疑問或想法，歡迎留言提出討論！

## 77. 參考連結

[Extracting the Signal: Rethinking Network Observability](#)

[Open Source for Better Observability](#)

[Technology Radar, 2020.09](#)

## Day24 - Grafana Cloud：雲端監控的未來

---



## 78. Grafana 起源

在 GrafanaCON 2023 keynote 開幕創辦人 Torkel Ödegaard 回顧一切是如何開始的故事。

Grafana 起源來自兩個偉大的開源專案 Graphite 和 Kibana，Graphite 能查詢分析指標並繪製圖表，但圖表無法互動僅是 png 圖片，在編輯查詢與建置圖表上是複雜的。Kibana 是 open source 專案，提供可視化儀表板並儲存在 Elasticsearch 日誌中，2013 年誕生後改變集中式日誌與可視化儀表板的遊戲規則。



Torkel : 如果 Kibana 可以查詢 Graphite 並加上 Metrics 呢？

~~爭什麼爭，攜在一起做撒尿牛丸不就好了嗎？~~

但 Kibana 專注於 Elasticsearch 所提供的數據資料，並不支援其他數據來源，因此 Torkel 將 kibana 專案 fork 下來並完全專注於指標與時間序列，核心概念為 **Make querying easier**，加上自己想法調整完後初版在 2014.1 在 github 推出初版



當時已經有很多 Graphite Dashboard 解決方案，作者認為推出後並不會發生什麼，但 Grafana 乾淨好看的用戶介面，快速查詢與好用的編輯查詢介面支持，讓儀表板可以重複使用，造成在監控社群上廣大的好評。

作者也分享當時的 Logo 想法 (都有動物 XDDDD

 <https://ithelp.ithome.com.tw/upload/images/20231009/20162577lsZfsbQh2m.png>

更多關於 Grafana 的歷史與重大里程碑可以看影片



## 79. Grafana Cloud 是什麼

Grafana is an open source interactive data-visualization platform

Grafana 是一個開放源碼的監控視覺化工具，提供不同的時間序資料庫來源，用於監控和可視化指標數據。

Grafana Cloud 是 Grafana Labs 所提供的雲端可觀測性平台，讓開發人員可以收集、儲存、視覺化應用程式的 Log、Metrics、Trace，並發送告警警報。並與 100 多個外部數據來源集成，包括 AWS CloudWatch、Elasticsearch、Graphite、和 InfluxDB。它與 Grafana K6 整合進行效能測試、Grafana Incident 和 Grafana OnCall 以管理事件議題處理與回應流程。

### 79.1. Core Stack : LGTM

核心專案由 **LGTM**，分別是 Loki、Grafana、Tempo 與 Mimir，以下針對核心專案做簡短說明其用途

- **Logs :**
  - 由高效能日誌聚合和儲存系統 Grafana Loki 專案負責
  - 集中日誌且與 Prometheus 指標一致，且 Loki 使用 LogQL 查詢日誌
- **Grafana Dashboard :**
  - Grafana 是每個 Grafana Cloud stack 的中心
  - 提供查詢、視覺化、發出警報並了解您的指標，並顯示在儀表板上。
- **Grafana Tempo :**
  - 提供易於使用、高度可擴展且經濟高效的分散式追蹤系統
  - Tempo 依賴 Grafana 內的深度整合，能夠在指標、日誌和追蹤之間無縫切換
- **Metrics :**
  - Grafana Mimir 專案負責
  - 並具有高可用性、多租戶、持久儲存以及長時間內極快的查詢效能。

- **Grafana Agent :**

- 輕量級收集器，用於將遙測資料傳送到 Grafana Cloud

備註：以上皆為 Open Source 專案，如果要自架也可以。

## 79.2. Pricing 費用

分 FREE、Cloud Pro 及 Cloud Advanced 三種，其支援功能與限制如下



## 80. 起手式：註冊 Grafana Cloud 帳號

要開始使用 Grafana Cloud，需要先在官方網站註冊，點擊註冊頁面 [SIGN UP](#)，可選擇透過其他帳號登入，或是建立新的帳號名稱



選擇地區及輸入名稱



按下 Finish setup，進行相關資料建立動作



建立完成後，可以看到目前資源的資訊 (Free account 前 15 天免費，可以好好利用 XD)



按下右上方 Connect data，可以選擇將資料進行整合到 Grafana Cloud 在 Dashboard 上呈現



以上是今天的分享，明天來繼續介紹如何與應用程式整合將資料上傳到 Grafana Cloud！

## 81. 小結

- Grafana 是一個開放源碼的監控視覺化工具，提供不同的時間序資料庫來源，用於監控和可視化指標數據。
- Grafana Cloud 是雲端可觀測性平台，讓開發人員可以收集、儲存、視覺化應用程式的 Log、Metrics、Trace，並發送告警警報。

## 82. 參考連結

[Grafana Cloud](#)

[Grafana Cloud 介紹](#)

# Day25 - Grafana Cloud：蒐集應用程式遙測數據 (1/2)



接下來透過程式直接看如何將資料送到 Grafana Cloud，在開始之前來看遙測數據要如何傳送，根據官方的建議架構如下 

應用程式使用 OpenTelemetry SDKs 自動蒐集方式將 traces, metrics 和 logs 資訊透過 OTLP 協定送到 Grafana Agent, Agent 再將資料送到 Grafana Cloud。

Grafana Agent 是本地安裝的代理，具有遠端寫入的功能，可以將指標、日誌及追蹤數據發送到後端儲存，例如 Mimir、Loki 和 Tempo。以下用 .NET 作為範例。

## 83. 建立 .NET 範例專案

我們透過範例程式來整合 OpenTelemetry，在 .NET 中有內建 Web API 範例程式可以做為 Demo 程式不用自己寫，在開始我們透過下列指令建立 .NET 應用程式

```
dotnet new webapi
```

上述指令將在資料夾底下建立 .NET 範例應用程式，該範例程式會包含 Swagger UI 及 WeatherForecast WebAPI。

## 84. 範例程式加上 OpenTelemetry SDKs

需要透過設定 OpenTelemetry for .NET 的 SDKs 自動蒐集應用程式的遙測數據資料，在 .NET 是透過 nuget 套件管理中心進行下載動作，要使用 openTelemetry 需要安裝下列套件

```
<PackageReference Include="OpenTelemetry.Exporter.Console" Version="1.6.0" />
<PackageReference Include="OpenTelemetry.Exporter.OpenTelemetryProtocol"
Version="1.6.0" />
<PackageReference Include="OpenTelemetry.Instrumentation.AspNetCore"
Version="1.5.0-beta.1" />
<PackageReference Include="OpenTelemetry.Instrumentation.Runtime" Version="1.5.0"
/>
```

套件安裝完成後，需要在啟動程式 `program.cs` 加上下列程式碼設定 openTelemetry

```
using OpenTelemetry.Metrics;
using OpenTelemetry.Resources;
using OpenTelemetry.Trace;

builder.Services.AddOpenTelemetry()
    .ConfigureResource(res => res.AddService(
        serviceName: builder.Environment.ApplicationName,
        serviceVersion:
        Assembly.GetEntryAssembly()?.GetName().Version?.ToString(),
        serviceInstanceId: Environment.MachineName)
    .AddAttributes(new Dictionary<string, object>
    {
        { "deployment.environment", builder.Environment.EnvironmentName }
    }))
    .WithTracing(tracing => tracing.AddAspNetCoreInstrumentation()
    .AddConsoleExporter());
```

## 上述程式碼重點

- AddOpenTelemetry : 使用 openTelemetry sdks 並設定其 service Name 、環境與版本資訊
- AddAspNetCoreInstrumentation : 自動蒐集 .NET 應用程式 **Trace** 數據資訊，包含傳入的 http 請求
- ConsoleExporter : 將追蹤輸出到控制台上

執行專案 dotnet run 或是在 Visual Studio IDE 按下 F5，在瀏覽器輸入 <https://localhost:5000/weather>，即可看到畫面上出現發送請求的相關遙測數據

```

Activity.TraceId:      5d8094a157d928778b5fbe5dc85126eb
Activity.SpanId:       732fb5979e3905b3
Activity.TraceFlags:    Recorded
Activity.ActivitySourceName: OpenTelemetry.Instrumentation.AspNetCore
Activity.DisplayName:   /weather
Activity.Kind:         Server
Activity.StartTime:    2023-10-10T15:25:04.1451306Z
Activity.Duration:     00:00:00.0871028
Activity.Tags:
  net.host.name: localhost
  net.host.port: 5000
  http.method: GET
  http.scheme: http
  http.target: /weather
  http.url: http://localhost:5000/weather
  http.flavor: 1.1
  http.user_agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/117.0.0.0 Safari/537.36
  http.status_code: 200
Resource associated with Activity:
  service.name: dotnet6-example
  service.instance.id: 2363b8ac-1e0f-44aa-9531-bf6a402b0d34

info: Microsoft.AspNetCore.Hosting.Diagnostics[1]
  Request starting HTTP/1.1 GET http://localhost:5000/favicon.ico --
Activity.TraceId:      26b7f657aaaffd67b0ea144c1d516a7b
Activity.SpanId:       c164340727e4b0e7
Activity.TraceFlags:    Recorded
Activity.ActivitySourceName: OpenTelemetry.Instrumentation.AspNetCore
info: Microsoft.AspNetCore.Hosting.Diagnostics[2]
  Request finished HTTP/1.1 GET http://localhost:5000/favicon.ico -- - 404 0
- 0.5362ms

```

## 85. 透過 OpenTelemetry 蒐集 Metrics 數據

在啟動程式 `program.cs` 加上下列程式碼設定 `WithMetrics`

```
.WithMetrics(metrics => metrics.AddAspNetCoreInstrumentation()
            .AddRuntimeInstrumentation()
```

```
.AddConsoleExporter());
```

看起來內容與 trace 很像，主要功能如下

- 從 dotnet core 應用程式收集 metrics 資訊
- 應用程式運行時蒐集 runtime 資訊
- 蒉集後透過 console 顯示

透過 IDE 或是 Command 執行 `dotnet run`，即可看到 trace 與 metrics 數據。

## Day26 - Grafana Cloud：蒐集應用程式遙測數據 (2/2)



### 86. 將數據傳到 Grafana Cloud

前面有提到資料會透過 Grafana Agent 送到 Grafana Cloud 上面，因此我們需要先安裝 Grafana Agent。Grafana Agent 可以在 Docker、Kubernetes、Linux、macOS 或 Windows 上以靜態模式 (static mode) 安裝，可以選擇在專案使用 Docker 運行或是安裝在 windows 服務上 [連結點擊](#)，我手邊是 window 電腦，因此接下來步驟以 window 為主，其他方式像是 docker 可以參考官方文件說明

#### 86.1. 安裝 Grafana Agent：執行檔

- 到 [Github Grafana 頁面](#)
- 移動到 Assets 區塊
- 下載 `grafana-agent-installer.exe.zip` 檔案，並解壓縮
- 點擊安裝執行檔 `grafana-agent-installer.exe` 安裝 Grafana Agent
- 開啟 Windows 服務管理員 (`services.msc`)，確認 Grafana Agent 服務是否為 Running 狀態
- 在執行時可以透過事件檢視器 (`eventvwr`) 來觀察其 log 輸出

或者新增 OTLP Connections



1. 選擇 OS platform，以我手邊環境為例是 windows
2. 輸入 API Token Name 產生 Grafana Agent 組態設定檔，這裡產生會一併產生 username 與 password 資訊
3. 下載 windows 安裝程式，如果安裝過這裡可以跳過
4. Agent 會安裝成功後目錄會在 `C:\Program Files\Grafana Agent Flow\config.river`，到此目錄下開啟 config 檔案將步驟 2 資訊貼上
5. 重啟 Grafana Agent Flow 服務

補充 [Grafana Agent 組態設定檔](#) 重點如下

- `otelcol.receiver.otlp`：
  - 配置 Grafana Agent 的 OTLP 接收器，接受應用程式的 trace、log 和 metrics 數據資料

- 預設監聽 gRPC 0.0.0.0:4317 和 HTTP/Protobuf 0.0.0.0:4318 endpoint
- `otelcol.processor.batch`
  - 指定資料的輸出目標 (批次)
- `otelcol.exporter.loki`
  - 設定 Loki exporter
  - Log 數據輸出到 Grafana Cloud Loki。透過 `forward_to` 指定 Loki 的接收器
- `otelcol.exporter.prometheus`
  - 設定 Prometheus exporter
  - Metrics 資料輸出到 Grafana Cloud Prometheus。透過 `forward_to` 指定 Prometheus 的接收器
- `prometheus.remote_write`
  - 設定 Prometheus 資料寫入 Grafana Cloud Prometheus 的細節
  - 包含目標的 URL 與基本身分驗證資訊
- `loki.write`
  - 設定將 log 資料寫入 Grafana Cloud Loki 的細節
  - 包括 Loki 的URL和基本身份驗證資訊
- `otelcol.exporter.otlp`
  - 設定 OTLP exporter
  - 將 trace 數據輸出到 Grafana Cloud Tempo。
  - 指定 Tempo endpoint 和基本身分驗證資訊
- `otelcol.auth.basic`
  - 設定基本身分驗證詳細資訊
  - 用於與 Tempo 進行身分驗證

到程式 `program.cs` 加上下列程式碼多設定輸出 exporter 為 otlp

```
.WithTracing(builder => builder
    .AddAspNetCoreInstrumentation()
    .AddConsoleExporter()
    .AddOtlpExporter()
)
```

登入 Grafana Cloud 並到 explore 頁籤，按下 run query 即可在 tempo 看到結果。例如在此範例中的 tempo 查詢結果  <https://ithelp.ithome.com.tw/upload/images/20231011/20162577y4aawYsvRY.png>

Log 與 Metrics 可以參考此範例，寫法差不多這裡就不在多說明 傳送門：[Observability with Grafana Cloud and OpenTelemetry in .net microservices](#)

## 87. 參考連結

[Application Observability recommended architecture](#)

# Day27 - Grafana Cloud : 使用 Grafana Pyroscope 優化應用程式性能



## 88. 什麼是 Grafana Pyroscope

在 Day13 - 可觀測性信號 Observability Signals 中提到常見的三個信號分別是 Metrics、Log 與 Trace，應用程式的性能分析 Profiling 也是在追查問題的 root cause 重點項目之一，Grafana Cloud 提供解決方案名為 Grafana Pyrscope。是由 Plare 與 Pyrscope 兩個開源專案合併而成，目的是為了提供開源社群大規模的分析線上應用程式資訊，以了解程式碼在上線之後資源的使用狀況，幫助開發人員在問題定位或是效能優化。

## 89. 如何進行分析

 <https://ithelp.ithome.com.tw/upload/images/20231012/201625778LWWiAIDMB.png> 在官網有提供分析步驟，說明如下

- 應用程式：負責收集數據動作，在 Grafana 中可以透過 Grafana Agent 或是應用程式加上 language 的 sdks 蒐集 profiling 資訊
- Pyroscope：負責資料儲存與查詢
- Grafana：負責資料的可視化與查詢，可透過 Dashboard 火焰圖、直方圖提供不同維度的視覺分析數據。
- 優化：透過上述資訊，可以識別應用程式中更慢或是消耗記憶體的部分，以提供更快的應用程式或找到 OOM 問題。

## 90. 過去在 Windows 環境是怎麼做

 如何在正式環境找到應用程式的性能瓶頸？

這一直是個複雜的任務，需要了解不同的應用程式組件像是前端、後端、資料庫或外取各自的瓶頸，才有機會制定其優化策略達到提高性能目的，在 .NET 應用程式優化部分以自己維運經驗可以透過 [Debug Diagnostic](#) 或是 [WinDbg](#) 來追查應用程式性能問題

 <https://ithelp.ithome.com.tw/upload/images/20231012/20162577ljqj6WYixn.png> Debug Diagnostic

 <https://ithelp.ithome.com.tw/upload/images/20231012/20162577BQ5YW0MJmN.png> WinDbg

以 Debug Diagnostic 為例，所需要步驟大概如下

- 收集應用程式資訊：將執行中的應用程式 Memory Dump 下來，可以選擇在執行當下或是特定規則，例如 CPU 超過 60% 時進行 Dump 動作，Dump 資訊會是一個 dump file
- 工具分析：開啟工具選擇下載的 dump file，按下分析按鈕，工具會針對 Dump 檔案內資訊進行分析
- 產生報表：分析結果透過報表方式呈現，例如網站 hand 住 CPU 100%、OOM (out of memory)、Memory leak，方便開發者可以了解網站當下使用 Memory/Thread/Callstack 等線上程式執行狀況資訊。以及彙整這些資訊後分析可能的問題與建議

 <https://ithelp.ithome.com.tw/upload/images/20231012/20162577LP4CIBptdh.png>

但這些在問題處理上可能都是屬於落後指標，意味著在處理上可能會是問題發生後才有機會發現及後續處理，舉例來說發現 CPU 使用量飆高，團隊來建立規則達到特定 % 後，下一次發生時觸發其規則時自動 Dump 檔案，團隊取得檔案進行後續分析當下程式執行狀況。

透過即時與不斷的蒐集線上應用程式資訊，可以更有效的定位可能發生的問題並針對問題進行調整，也可縮短 MTTR(Mean time to recovery) 平均修復的時間。

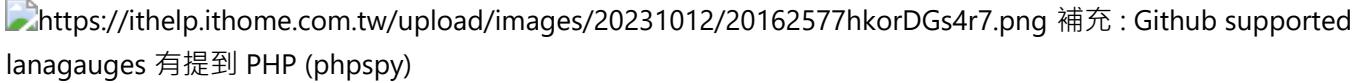
### 90.1. 蒐集資訊方式

 <https://ithelp.ithome.com.tw/upload/images/20231012/20162577ZHbf9bhKg3.png>

在蒐集遙測數據方面，有兩種方式

- Pyroscope SDKs : 在應用程式中使用各自程式語言的 SDKs，收集即時的 Profiling data 到 Grafana Cloud
- Grafana Agent : 架設 Grafana Agent 定期從中收集分析數據

在 Grafana Pyroscope 中有提供不同程式語言的範例，目前有以下幾種

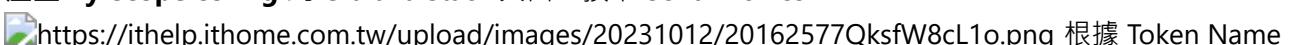
補充：Github supported lanagages 有提到 PHP (phpspy)

自己對於 .NET 比較熟悉，接著使用該程式語言進行範例說明，如果是要將既有的 .NET 專案加入 Profiling 可以參考官方說明：[連結](#)

以下是直接使用範例程式進行步驟說明

- **Clone 範例專案**

`https://github.com/grafana/pyroscope.git`

- 產生 **Pyrscope config** 到 Grafana Stack 頁面，按下 Send Profiles
- 修改 **API Token** 設定 將剛剛產好的 User Name 與 API Token 複製到應用程式設定中的 PYROSCOPE\_BASIC\_AUTH\_USER 與 PYROSCOPE\_BASIC\_AUTH\_PASSWORD
- 執行應用程式 到 sample 專案底下 Fast-slow 目錄，執行下列語法

`docker-compose up`



透過以上方式即可在 Grafana Cloud 看到應用程式 profiling 資訊。

另外在官網也有提供 [Pyroscope Demo](#)，有興趣可以看到更多資訊感受其強大威力。



## 91. 參考連結

[documentation](#)

[Pyroscope Language SDKs](#)

[pyroscope](#)

# Day28 - Grafana 事件管理：從告警到修復的問題排除流程 (1/2)



前面文章討論很多關於可觀測性的觀念與工具，最重要的是在實務面上我們如何將理論實際應用，來解決真實世界上開發團隊所遇到的問題

透過可觀測性，我們如何具體幫助線上問題釐清呢？

## 92. 問題排除流程



如果你是開發或是運維團隊人員，當線上系統或負責應用程式遇到問題時，會做哪些事情來協助定位問題，上圖是當問題發生到被修復可能的流程

- 警報 (Alert)**: 當系統監控偵測到異常或是特定指標 (Metrics) 超過預定的閥值或水位時，系統會產生告警警報，對於 RD 及團隊來說，這是首先被告知問題的方式。
- 儀表板 (Dashboard)**: 收到警報後，RD 可能會先查看 Dashboard 了解異常背後的數據。Dashboard 提供即時的數據視覺化，幫助 RD 快速定位問題範圍和影響程度。
- 查詢 (Adhoc Query)**: 若 Dashboard 所提供的資訊不足，或是為了要更精準確定問題原因，RD 會使用 adhoc Query 進一步的來查找特定時間的數據資料來確認問題。
- 日誌 (Log Aggregation)**: 確認問題範圍後，RD 會查找相關的系統或是應用程式 Log，Log 中可能包含更明確的應用程式執行資訊，或是問題的詳細描述與錯誤訊息，幫助 RD 更有機會找到問題的具體原因。
- 分散式追蹤 (Distributed Tracing)**: 如果應用程式架構是分散式系統，RD 會使用分散式工具來找特定請求如何在多個服務間流動過程，有助於找到性能瓶頸或可能失敗的原因。
- 修復 (Fix)**: 一旦問題被定位，就有機會定位問題並進行緊急修復的動作。可能解決方式式調整配置設定檔、修正錯誤的程式或是重新啟動服務(重開治百病)

透過上述的流程與步驟之後了解到每個步驟的細節，接著整理各個步驟其主要目的如下

步驟	核心目的	目的
警報 (Alert)	When	立即通知團隊，讓他們迅速回應問題
儀表板 (Dashboard)	What	即時視覺化系統狀態，助於定位問題
查詢 (Adhoc Query)	How	深入探索數據，查詢特定的數據點或時間範圍。
日誌 (Log Aggregation)	Where	提供應用或系統的詳細日誌
分散式追蹤 (Distributed Tracing)	Why	在分散式系統架構中，請求如何在多個服務之間流動，找到可能的瓶頸
修復 (Fix!)	Action	一旦問題被確定，立即採取措施來解決它，如代碼修復、配置調整或服務重啟。

以上流程可以看到可觀測性中強調的信號 metrics、logging、trace 在問題定位上扮演重要的角色，可以幫助問題發生當下縮短系統異常的恢復時間 (MTTR)，也可以看出前面文章提到的監控與可觀測性是相符相乘的，無法取代彼此。

Grafana 生態系提供了一系列強大的工具，可以輔助開發團隊預防、定位及後續的分析跟優化，以下是針對各階段的工具說明

- 事前預防**
  - 預警系統**：使用 Grafana Alter，基於當前指標的警報外，也可以設定基於數據趨勢的警報，例如，當系統的記憶體使用率持續上升時，可以提前通知，即使當前使用率還在正常範圍內。

- 動態儀錶板 : Grafana Dashboard , 建立動態儀錶板，展示系統的各種指標，從而實時了解系統的健康狀態
  - 持續性能測試: 使用 k6 定期執行性能測試，確保不會隨時掛掉
  - 系統健康檢查: 定期的系統健康檢查，識別任何潛在的問題或不足之處。
- 事中處理
    - 使用可觀測性相關工具像是 Grafana、Loki、Tempo 工具整合，快速定位問題與盤查
  - 事後分析
    - 數據視覺化: 利用工具進行數據分析，找出可能的隱患或問題根源。
    - 使用 Grafana Pyroscope 進行線上的應用程式性能分析。

## 93. 不只是工具



透過事前預防、事中處理與事後分析，加上結合 Grafana 可觀測性的工具，可以更有效的處理管理與加速問題處理速度。但實務面上除了工具的使用之外，更重要的是 **團隊成員的合作** 與 **清晰的 SOP 機制** 也不能忽視。

不同人代表有各種不同的工作方式與思維。舉例來說在寫程式時 Junior 與資深的開發人員對於程式寫法不同，為了確保程式的可讀性，就會需要定義制定統一規範並在 Code Review 落實，才有機會大家在看 Code 跟閱讀時更加輕鬆。

同理，在緊急問題處理時需要定義清楚異常事件的 SOP，以便問題發生時知道該採取哪一些步驟，包含該使用那些工具分析與定位、定時回報的機制事後事故報告(RCA)內容寫法，減少混亂並確保團隊成員都可以依照相同方式進行操作，在新成員加入時也不會因為認知落差而影響到處理時效性，提高其處理問題的效率。

補充：在處理線上問題時，核心目的是**先止血**，不是急於尋找根本原因。這也是為什麼我們需要清晰的流程和工具來協助我們在問題發生時快速應對，而不是盲目地尋找問題的原因(思路：止血、暫時解、根本解)。

下一篇將介紹 **Grafana 提供 Grafana Incident Response & Management (IRM)** 解決方案，如果有任何疑問或想法，歡迎留言提出討論！

## 94. 參考連結

[Beyond the 3 Pillars of Observability](#)

[面向故障处理的可观测性体系建设](#)

## Day29 - Grafana 事件管理：從告警到修復的問題排除流程 (2/2)

## 95. 過去怎麼做

1. 口耳相傳，緣分到了就會修好
2. 各自通靈
3. (文件)定義流程 + 系統

## 96. Grafana Incident Response & Management (IRM)

 <https://ithelp.ithome.com.tw/upload/images/20231016/201625779oZRa27rax.png> Grafana 解決方案：  
Grafana IRM

- Grafana IRM = Incident Response & Management
- 回應 (Response) :
  - Grafana Alter : 這部分涉及設定警報和通知系統，以便在系統或應用程式出現問題時能夠及時通知相關團隊成員。這有助於快速回應問題並採取必要的措施。
  - Grafana Oncall : Grafana Oncall是一個關鍵元件，它確定了誰在特定的時間段內負責解決問題。這確保了團隊中的成員在需要待命時都自己知道處理問題。
  - Grafana Incident : Grafana Incident模組用於記錄和追蹤發生的問題和事件。它可幫助團隊建立問題的歷史記錄，以便記錄更好地了解問題的模式和趨勢。
- 管理 (Management) : using Grafana Incident
  - 問題協作：管理團隊成員之間的協作，確保每個人都知道問題的狀態和處理進度。
  - 問題解決：提供工具和資源，幫助團隊成員更輕鬆解決問題，包括文件、故障排除指南和知識庫(KM)。
  - 問題分析：幫助團隊分析問題的根本原因 Root cause。

由什麼所組成  <https://ithelp.ithome.com.tw/upload/images/20231016/20162577ZTXOlvxLfp.png>

## Day30 - 從傳統開發到可觀測性驅動開發 (ODD)

 <https://ithelp.ithome.com.tw/upload/images/20231016/20162577oEILFd6gbt.jpg>

傳統的軟體開發方法如何面對現代化系統複雜的挑戰？或者，有沒有一種方法能夠建構更可靠、高性能的應用程式？

## 97. Observability Maturity Model : 可觀測性成熟度

監控 (Monitor) 做為團隊了解系統可用性與效能這方式已存在數十年，傳統的監控工具有助於捕捉一組特定的指標，解決已知的未知問題，可觀測性是監控的再進化，透過蒐集可觀測性重要的指標像是 metrics、trace、log 及 profiling 等信號資訊，希望解決未知的未知問題。前面在可觀測性工具的整合與挑戰文章也提到了工具在整合時所遭遇的問題，超過一半以上公司使用 5 種以上工具，數據上的整合或是維運人員在不同工具的 context switch 都會是個潛在的問題。

解決這些問題，就是實踐可觀測性了嗎？

 <https://ithelp.ithome.com.tw/upload/images/20231016/20162577lhvEv6D0MP.png>

在 [Observability Maturity Model](#) 文章中定義了可觀測性演化的四個不同的層級，每個層級各有不同的缺點、挑戰與說明事項，還有如何影響系統的可靠性 (Reliability)。這四個層級分別是 **Monitoring**、**Observability**、**Causal Observability** 與 **Proactive Observability with AIOps**。透過表格整理四個層級各自重點與目的如下

等級	主要目的	摘要
Monitoring	確保各個組件如預期運作	<ul style="list-style-type: none"> <li>- 追蹤 IT 系統中各個元件的基本運作狀況</li> <li>- 查看事件；觸發警報和通知</li> <li>- 告訴您出了問題</li> </ul>

等級	主要目的	摘要
Observability	確定系統不工作的原因	<ul style="list-style-type: none"> <li>- 透過觀察系統的輸出來深入了解系統行為</li> <li>- 重點關注從指標、日誌和跟蹤中推斷出的結果，並結合現有的監控數據</li> <li>- 提供基線數據以幫助調查出了什麼問題以及原因</li> </ul>
Causal Observability	尋找事件的原因並確定其對整個系統的影響	<ul style="list-style-type: none"> <li>- 提供更全面的見解，幫助確定導致問題的原因</li> <li>- 基於 1 級和 2 級基礎構建，並增加了跟蹤 IT 堆疊中拓撲隨時間變化的能力</li> <li>- 生成廣泛的相關信息，有助於減少識別問題所所需的時間，問題為何發生、何時開始以及哪些其他領域受到影響</li> </ul>
Proactive Observability with AIOps	分析大量數據，自動回應事件，並防止異常成為問題	<ul style="list-style-type: none"> <li>- 使用 AI 和 ML 尋找大量資料中的模式</li> <li>- 將 AI/ML 與 1-3 級資料結合，提供整個堆疊中最全面的分析</li> <li>- 及早檢測異常並發出足夠的警告以防止故障</li> </ul>

隨著成熟度等級越高，團隊在處理問題上就可以更快地找到故障異常的服務，更快速的找到問題根本原因，提供客戶更好的體驗與提高系統可靠性 (Reliability)，縮短 MTTR 的時間。

## 98. Observability Driven Development : 可觀測性驅動開發

### 98.1. 什麼是可觀測性驅動開發 ODD ?



權威調查機構 Gartner 每年會定期公布技術趨勢，在 2022 年 新興技術成熟度公布有三項與可觀測性相關的項目分別是

- OpenTelemetry :
  - 是由一系列規範、工具、API 和 SDK 組成的可觀察能力框架，進而支援開源儀器及實現軟體的可觀察性。
  - 到達成熟需要的時間 (預估) : 2 ~ 5 年
- 數據可觀測性 (Data observability) :
  - 是指透過持續監測、追蹤、警報、分析和故障排除的記錄，來了解組織的資料圖景、資料工作流程和資料基礎設施等健康狀況的能力。
  - 到達成熟需要的時間 (預估) : 5 ~ 10 年
- 可觀測性驅動開發 (Observability-driven development, ODD) :
  - 一種軟體工程實踐，透過設計可觀察的系統，為系統的狀態和行為提供細粒度的可見性與背景。
  - 到達成熟需要的時間 (預估) : 5 ~ 10 年

在 2023 年也把可觀測性技術列為戰略技術策略趨勢之一。

過去曾經聽過測試驅動開發 TDD (Test-Driven design)、領域驅動開發 DDD (Domain-driven design)、死線驅動開發 DDD (Deadline-driven design)，那麼驅動開發這個詞究竟是怎麼來的呢？「驅動開發」詞源自於 1989 年 Rebeckah Wirf-Brock 的責任驅動設計 (Responsibility-driven design)。

可觀測性的目的是讓團隊可以更了解系統運行的狀況，那麼可觀測性驅動開發是甚麼呢？

可觀察性驅動開發 (ODD) 是一種將可觀察性需求整合到軟體開發生命週期(SDLC)早期階段的方法。

ODD 可觀測性驅動開發是一個新的術語，強調在整個軟體開發週期中加入可觀測性的需求，像是可觀測性重要信號日誌、指標和追蹤。這種收集跟分析是在應用程式的開發過程中完成的，而不是在其中一個階段像是上線後或是當問題發生後才加入。

 <https://ithelp.ithome.com.tw/upload/images/20231016/20162577b9cCdxOrn.png>

就像 TDD 測試驅動開發強調在寫程式碼前要先寫測試案例，以提高程式碼的保護與品質。ODD 在建立可觀測性系統方面也是如此，可觀察性驅動開發意味著開發人員在編寫程式碼之前考慮可觀察性信號或是監控方法，適用於元件級別或是整個系統。利用工具和實踐開發人員來觀察系統的狀況和行為，讓團隊可以更了解系統，包含可能的弱點或是潛在問題。

TDD 在測試(Test)和設計(Design)之間創建了反饋(feedback)循環，而 ODD 擴展了反饋循環，確保功能按預期運行，改進部署流程(Deploy)並向規劃(Plan)提供反饋。

當功能或是程式上線之後，就可以透過遙測數據分析程式上線後，程式有按照預期執行嗎？還有什麼看起來很奇怪的嗎？適時地放入麵包屑，以便未來當問題發生時可以有效的追溯到問題的源頭。

## 98.2. ODD 加入 SDLC 階段可能的挑戰？

前面提到可觀測性驅動開發左移觀念，以下列出在各 SDLC 階段可能會遇到的問題

 <https://ithelp.ithome.com.tw/upload/images/20231016/20162577pz4cBNBffB.png>

- 設計 (Design)
  - 在設計階段確定系統的運作方式和功能 (outcome)，可能是系統或是業務 KPI 相關問題。
  - 確定要持續監控和追蹤的關鍵數據點或指標，例如 MTTR、平均回應時間等。
  - 透過公用程式碼或 AOP 方式，方便統一從容器、服務等收集遙測數據。
- 開發 (Development)
  - 為儀器數據提供上下文，使原始數據有意義。
  - 使用統一的上下文標準規範，降低蒐集與開發人員手動處理
  - 使用 OpenTelemetry 作為遙測數據收集的標準 (10篇文章會有 12 篇推薦你使用 XDDD)
- 建置與部署 (Build & Deployment)
  - 遵循「[可觀察性即程式碼](#)」的實踐。
  - 通過部署管道 pipeline 實現可觀察性的自動化。
- 維運 (Operate)
  - 實踐可觀察性使您能夠進行準確的 debugging 和主動檢測，提高生產系統的效率。
  - DevOps 團隊應專注於主動監控標準最佳實務，將營運期間的觀察結果回饋給開發團隊。

當開發人員 Day1 在開發程式時就建立營運思維，在系統營運上就有機會更快的解決跟發現可能的問題，讓使用者更滿意。

## 99. 小結

- 觀察性驅動開發 ( ODD ) 強調在整個軟體開發生命週期中加入可觀察性方面的需要。鼓勵從早期階段就將可觀察性所需的活動左移。

## 100. 參考連結

[Observability-Driven Development: From Development to DevOps](#)

[Resilient software delivery through observability-driven development](#)

[Observability-driven development](#)

[Observability Maturity Model](#)

[Moving to Observability Driven Development](#)

Gartner 公布 2022 新興技術成熟度曲線，這些技術趨勢最值得关注

[What Observability-Driven Development Is Not](#)

## Day31 - 30 天之後？

---

 標題是參考最喜歡的漫畫灌籃高手 10 天後

## 101. 學習資源

在這三十天中整理一些基礎的知識讓大家入門，但可觀測性議題不僅僅於這些議題，學習有興趣的事物也不會因為鐵人賽結束而停止，因此整理自己過去覺得有幫助的學習資源，方便有興趣的夥伴一起研究下去

- 白皮書
  - 草稿版
  - github
- 書籍
  - [Observability Engineering](#)
  - [The Future of Observability with OpenTelemetry](#)
  - [Cloud Observability in Action](#)
- 電子報
  - [O11y news](#)
  - [Observability news](#)
  - [The new stack](#)
- 研討會
  - [ArchAsuhmmit](#) : Arch Summit 是 InfoQ 定期舉辦的全球架構師峰會，有專門可觀測性議題議程軌，也可下載分享議程的 [投影片檔案](#)
  - [Grafana con](#) : 這應該不用介紹，大家對 Grafana 都比我熟 XDDD

- Awesome 系列
  - [awesome-observability](#)
  - [awesome-opentelemetry](#)
- Other
  - [o11y](#)
  - [CNCF](#)
  - [CNCF Landscape](#)

## 102. 完賽心得

忽然有天看到鐵人賽沒想太多就報名了，報名題目沒太多時間考慮就直接寫可觀測性 101，後來有時間思考分享大綱後比較偏向 Observability in Devops，想要分享的是 Observability 最初其實是從 DevOps 的 ops 為出發，系統上線後才是挑戰的開始，因此在前10天想要 ops 相關聯的基礎知識，後面才帶到 observability 主軸，工具介紹在時間有限的情況下以應用程式整合為主，工具介紹比我詳細的大大跟文章們太多了，這系列文章想要表達的核心觀念為

- Observability 與 Devops 是密不可分的
- Observability 只是個新名詞，目的是希望透過可觀測性可以更了解系統的實際運行狀況
- OpenTelemetry 很重要請務必關注
- Observability 還在持續演進(連載)中
- 了解其核心價值是必要的(why)，工具只是解決的手段之一(how)

一直以來都有想有結構性的整理 Observability 相關議題，經過鐵人賽這30天的奮鬥終於有小小的成果。想要分享的很多但時間有限，體驗比較深的是從第七天後存稿已耗盡，在工作滿檔/團隊雜事不斷/家庭大小事都得顧的情況下，每天的分享文都有種在趕專案 deadline 的感覺，連趕 23 天後終於完賽(躺平)，回想這過程有很多東西可以更好的，反思如果有下次再參加的話會怎麼做呢？..... 沒有然後了 XDDDDDD

最後謝謝家人的支持，跟有閱讀這系列文的讀者們 希望這些文章閱讀後有讓你們對於可觀測性有進一步的認識若有任何問題歡迎提出來討論，Happy Coding !