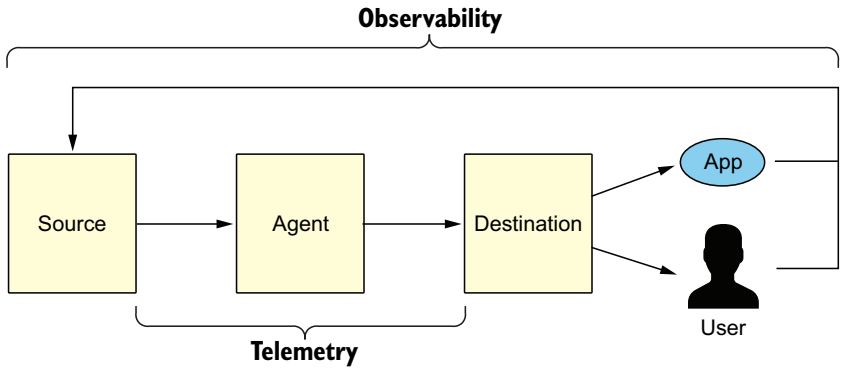
Cloud Observability IN ACTION

邁克爾·豪森布拉斯

MANNING



Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Cloud Observability in Action

邁克爾·豪森布拉斯

mannings
HELTER i SLAND

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

有關在線信息和訂購此和其他曼寧書籍的訂購，請訪問www.manning.com。出版商在訂購數量時為本書提供折扣。 有關更多信息，請聯繫

特別銷售部Manning Publicati
ons Co. 20 Baldwin Road Po B
ox 761 Shelter Island，NY 119
64電子郵件：orders@mannin
g.com

©2024 Manning Publications Co.保留所有權利。

未經發布者事先事先書面許可，任何形式或通過電子，機械，複印件或其他方式都可以復制，存儲在檢索系統中或以電子，機械，影印本或其他方式傳輸的任何部分。

製造商和賣家用來區分其產品的許多名稱被稱為商標。這些名稱出現在書中的地方，而Manning出版物則意識到商標聲明，這些名稱已以初始上限或所有帽子印刷。

♾ 認識到保留所撰寫的內容的重要性，曼寧的政策是將我們出版的書籍印刷在無酸紙上，而我們為此盡了最大的努力。 同樣，認識到我們有責任保存地球的資源，曼寧書籍印在紙上，至少在不使用元素氯的情況下進行了15%的回收和處理。

作者和出版商已竭盡全力確保本書中的信息在發稿時正確。作者和出版商不承擔，特此不承擔任何責任，無論是由於疏忽，事故或任何其他原因，還是因此處的任何信息而導致的錯誤或遺漏造成的任何損失，損害或中斷。

曼寧出版公司
鮑德溫路20號
郵政信箱761
該島的藝術，紐約11964。

開發編輯：伊恩·霍夫（Ian Hough）
技術編輯：Jamie Riedesel
評論編輯：Aleks DragoSavljevic ´
生產編輯：Deirdre Blanchfield-Hiam複製編輯：Ch
ristian Berk校對者：Katie Tennant技術校對者：Ernest
Gabriel Bossi Carranza Carranza發行者：Dennis Dalinnik
封面設計師：大麻Tudor

ISBN: 9781633439597
Printed in the United States of America

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

To my family: my wife Anneliese; our kids Iannis, Ranya, Saphira;
as well as Snoopy the dog and Charles the cat

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

brief contents

1 ■端到端可觀察性1 *2* ■信號類型14 *3* ■來源44 *4* ■代理和儀器66 *5* ■後端目的地95 *6* ■前端目的地120 *7* ■ cloud Operations 142 *8* ■ cound Operations 142 *8* ■分佈踪跡178 {v11 { *11* ■信號相關211

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

contents

preface *xi*
acknowledgments *xiii*
about this book *xv*
about the author *xix*
about the cover illustration *xx*

1 *End-to-end observability* *1*

1.1什麼是可觀察性？ 2 1.2可觀察性用例4 1.3角色和目標5 1.4示例Microservices App 6 1.5挑戰以及可觀察性如何幫助10

Return on investment *11* ▪ *Signal correlation* *12*
Portability *12*

2 *Signal types* *14*

2.1參考示例15 2.2評估儀器成本16 2.3日誌17

Instrumentation *19* ▪ *Telemetry* *20* ▪ *Costs and benefits* *24*
Observability with logs *26*

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

2.4指標26
Instrumentation 28 ▪ Telemetry 29 ▪ Costs and benefits 31 Observability with metrics 34

2.5痕跡34
Instrumentation 36 ▪ Telemetry 37 ▪ Costs and benefits 39 Observability with traces 41

2.6選擇信號42

3 *Sources 44*

3.1選擇資源45

3.2計算相關的來源46
Basics 46 ▪ Containers 49 ▪ Kubernetes 51 ▪ Serverless compute 55

3.3與存儲相關的資源58
Relational databases and NoSQL data stores 58 ▪ File systems and object stores 60

3.4與網絡相關的資源61
Network interfaces 61 ▪ Higher-level network sources 62

3.5您的代碼63
Instrumentation 64 ▪ Proxy sources 64

4 *Agents and instrumentation 66*

4.1日誌路由器67
Fluentd and Fluent Bit 68 ▪ Other log routers 73

4.2指標集合74
Prometheus 74 ▪ Other metrics agents 76

4.3 opentelemetry 77
Instrumentation 79 ▪ Collector 81

4.4其他代理90 4.5選擇代理91
Security for and of the agent 91 ▪ Agent performance and resource usage 92 ▪ Agent nonfunctional requirements 93

5 *Backend destinations 95*

5.1後端目的地術語96

5.2日誌的後端目的地97
Cloud providers 97 ▪ Open source log backends 99 Commercial offerings for log backends 100

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

5.3指標的後端目的地101
Cloud providers 103 ▪ Open source metrics backends 104
Commercial offerings for metrics backends 106

5.4痕跡106
Cloud providers 106 ▪ Open source traces backends 107
*Commercial offerings for trace backends 108*的後端目的地

5.5柱狀數據存儲108 5.6選擇後端目的地114
Costs 114 ▪ Open standards 116 ▪ Back pressure 116
Cardinality and queries 116

6 *Frontend destinations 120*

6.1前端121
Grafana 121 ▪ Kibana and OpenSearch Dashboards 128
Other open source frontends 130 ▪ Cloud providers and commercial frontends 130

6.2 All-Inson 131
CNCF Jaeger 131 ▪ CNCF Pixie 134 ▪ Zipkin 135
Apache SkyWalking 136 ▪ SigNoz 136 ▪ Uptrace 136
Commercial offerings 137

6.3選擇前端和全副範圍139

7 *Cloud operations 142*

7.1事件管理143
Health and performance monitoring 143 ▪ Handling the incident 145 ▪ Learning from the incident after the fact 146

7.2警報146
Prometheus alerting 146 ▪ Using Grafana for alerting 154
Cloud providers 154

7.3用法跟踪155
Users 155 ▪ Costs 156

8 *Distributed tracing 158*

8.1簡介和術語160
Motivational example 160 ▪ Terminology 163
Use cases 164

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

8.2使用Microservices App 164
*Example app overview 164 ▪ Implementing the example app 165 ▪ The "happy path" 167 ▪ Exploring a failure in the example app 168*中使用分佈式跟踪

8.3實際考慮169
Sampling 170 ▪ Observability tax 171 ▪ Traces vs. metrics vs. logs 172

9 *Developer observability 174*

9.1連續分析176
The humble beginnings 177 ▪ Common technologies 180 Open source CP tooling 184 ▪ Commercial continuous profiling offerings 186 ▪ Using continuous profiling to assess continuous profiling 187

9.2開發人員生產力190
Challenges 190 ▪ Tooling 191

9.3工具注意事項194
Symbolization 194 ▪ Storing profiles 194 ▪ Querying profiles 195 ▪ Correlation 195 ▪ Standards 195 Using tooling in production 195

10 *Service level objectives 197*

10.1 Slos 198 *Types of services 200 ▪ Service level indicator 200 Service level objective 201 ▪ Service level agreement 202* 10.2實現 Slos 202 *High-level example 202 ▪ Using Prometheus to implement SLOs 203 ▪ Commercial SLO offerings 208* 10.3考慮209

11 *Signal correlation 211*

11.1相關基礎212
Correlation with OpenTelemetry 213 ▪ Correlating traces 215 Correlating metrics 216 ▪ Correlating logs 216 ▪ Correlating profiles 216

11.2使用Prometheus，Jaeger和Grafana進行相關性217
Metrics–traces correlation example setup 217 ▪ Using metrics–traces correlation 219

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

11.3商業產品中的信號相關支持221 11.4注意事項222
Early days 222 ▪ *Signals* 223 ▪ *User experience* 224 11.5結論224

appendix A Kubernetes end-to-end example 226
index 237

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

preface

我們真正生活在激動人心的時代！從大約10年前開始使用Docker和Kubernetes開始的雲本地技術的興起，以及雲產品的可用性，使您能夠根據微服務體系結構運行大規模應用程序，從而改變了我們編寫和操作軟件的方式。

我很幸運地成為旅途中的一部分，從2015年的容器空間開始，然後在Kubernetes領域工作，直到2021年直到2021年。雲本地人對我來說很突出：考慮到容器和功能與服務的動力，如果您無法對系統上的洞察力，並且您無法對您的系統中的興趣進行綜合的問題，那麼您就可以願意接受AD COSS的問題，並且您會發現Ad cor的問題，並且您會遇到良好的問題。當我將AWS的團隊更改為專注於可觀察性時，剛剛形成了開放式遙測，並且空間正在迅速發展。現在，在出版時，可以公平地說，可觀察性已經成為主流。

我僅在事後才意識到的一件事是，除了雲本機計算基礎（CNCF）項目的生態系統的開源性質之外，讓我進入了觀察能力空間，這是一個事實，即觀察力本質上是數據工程的應用領域。這是關於基於管道的生成，收集，存儲和查詢數據。我為什麼要指出這一點？在進入容器的世界之前，我在數據工程上花費了十多年的時間，首先是應用研究，然後在一家初創企業中，在那裡我必須在“大數據”時期應用一些經驗教訓。

當有機會分享我過去20年中我學到的東西時，無論是在數據工程還是雲端空間，

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

關於可觀察性指南，我很明顯這是正確的時間和地點。基本想法是涵蓋整個可觀察性空間，從生成數據到如何收集和處理該空間，再到人類和軟件的消耗方式，所有這些目的都是了解可觀察性的基本原理和方法，使用開源軟件進行演示，以便任何人對這個話題感興趣的人都可以自己嘗試，而無需擔心成本。

我希望這本書是您在組織中介紹可觀察性的旅程的參考和指南。如果它可以幫助您創建解決方案，從而使您的團隊能夠從雲本地產品中受益，而無需盲目，則它將達到其目的。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

acknowledgments

寫書是長期的承諾，通常是一年或更長時間。雖然這不是我的第一本書，而且我能夠從過去的經驗中汲取教訓，但不用說，結果是我自己並沒有實現的，因為許多人幫助塑造和改進了這本書。

首先，我要感謝我的家人，他一直支持和激勵我！接下來，我要對曼寧的編輯伊恩·霍夫（Ian Hough）表示非常感謝您的所有指導（耐心）。 While I spent most of the time with Ian, there are sev- eral folks at Manning who helped make this book a reality, and I am grateful for every- thing you did: Malena Selic, Marina Matesic, Ivan Martinovic´, Rebecca Rinehart, Stjepan Jurekovic, Ana Romac, Susan Honeywell, Mike Stephens, and Marjan Bace.我還要感謝我的項目編輯Deirdre Blanchfield-Hiam； 我的副本克里斯蒂安·伯克（Christian Berk）；我的校對凱蒂·坦南特（Katie Tennant）；還有我的技術校對者Ernest Gabriel Bossi Carranza。

我出色的技術編輯傑米·里德賽爾（Jamie Riedesel）值得大喊大叫！傑米（Jamie）是Dropbox的員工工程師，擁有超過20年的經驗。她對這本書的影響和塑造很大，為如何解釋事物，反饋技術方面以及更加努力的動力提供了指導。謝謝。但是，我還要感謝許多在各個章節上提供反饋的人，分享了寶貴的見解：弗雷德里克·布蘭奇克（Frederic Branczyk），馬修斯·洛伊布（Matthias Loibl），基特·默克（Kit Merker）； 和曼寧評論家阿德里安·布魯加（Adrian Buturuga），亞歷山德羅·坎佩斯（Alessandro Campeis），巴文·塔克（Bhavin Thaker），鮑比·林（Bobby Lin），博科·德喬爾科維奇（Borko Djurkovic），克里斯·哈格斯特羅姆（Chris Haggstrom），克利福德·瑟伯（Clifford Thurber），多伊爾·特納（Doyle Turner），厄爾尼斯托·鮑斯迪（Ernesto Bossi），埃內斯托·波斯迪

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Warczarek, Jan Krueger, Jorge Ezequiel Bo, Juan Luis, Ken Finnigan, Kent Spill Er, Kosmas Chatzimichalis, Maciej Drozdzowski, Madhav Ayyagari, Machael Bright, Michale Di Pede, Miguel Montalvo, Onofrei George, Pablo Chacin, Rahul Modpur, Rui Liu, Sander Zegveld，Sanjeev Jaiswal，Satadru Roy，Sebastian Czech，Stefan Turalski，Stephen Muss，Vivek Dhami和Wesley Rolnick。

最後，感謝我在AWS的很棒的同事的支持和回饋，以及我參與其中的開源社區，尤其是在CNCF的背景下。這是一種榮譽和榮幸。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

about this book

可觀察性是根據觀察到的（雲本地）系統的信號連續生成和發現可行的見解的能力，目的是影響系統。 我們從投資回報的角度了解該主題：我們研究成本和收益，從源到遙測（包括代理）到信號目的地（後端）（包括時間序列數據存儲），例如Prometheus和Grafana等前端。

在整本書中，我都使用開源工具，包括但不限於Opentelemetry（收集器），Prometheus，Loki，Jaeger和Grafana來展示不同的概念，並使您無需任何費用，而無需任何費用。

Who should read this book

該書主要關注正在使用雲本地應用程序的開發人員，DevOps/網站可靠性工程師（SRE）。它是針對有興趣運行雲本地應用程序的任何人，無論是在Kubernetes中還是使用功能即服務產品，例如AWS Lambda。

另外，我相信，如果您是發行經理，IT架構師，安全和網絡工程師，技術領導者或云本地空間中的產品經理，則可以從這本書中受益。這本書可以與任何公共雲（我使用AWS用於多個演示，純粹是為了熟悉），以及與任何云的本地設置（例如，數據中心的Kubernetes）一起使用。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

How this book is organized

該書有11章和一個附錄，其中包含以下內容：

第1章為您提供了端到端的示例，並定義了終端，從來源到代理到目的地。它還在觀察性的背景下討論了用例，角色和挑戰。 第2章討論了不同的遙測信號類型（日誌，指標和痕跡），何時使用信號，如何收集信號以及相關的成本和收益。 第3章涵蓋了生成遙測的信號源。我們討論存在的來源類型，以及何時選擇哪個來源，如何從為任務選擇正確的源來獲得可行的見解，以及如何處理您擁有的儀器代碼（包括供應鏈方面）。 第4章討論了從日誌路由器到OpentEleme- try的不同遙測代理。您將學習如何選擇和使用代理，並強調開放式遙測為統一遙測管理帶來什麼。 第5章重點介紹了遙測信號的後端目的地，作為真理的來源。您將學習使用並選擇對日誌，指標和痕蹟的後端，並深入研究時間序列數據庫，例如Prometheus和麵向列的數據存儲，例如ClickHouse。 第6章討論可觀察性前端是您消耗遙測信號的地方。您將了解純粹的前端和多餘的前端，以及如何選擇它們。 第7章涵蓋了稱為*cloud operations*的雲本地解決方案的一個方面，包括如何檢測何時無法按照其應有的方式工作；對異常行為的反應； 並從以前的錯誤中學習。 您還將學習有關警報，使用和成本跟踪的信息。 第8章深入研究分佈式跟踪，以及如何幫助您進行維修和對微服務進行故障排除。 第9章深入研究開發人員的可觀察性，涵蓋了連續的分析和開發人員生產力工具。 第10章討論了服務級別的目標，向您展示瞭如何使用它們來解決服務消費者的滿意度。 第11章深入探討了信號相關性，解決了Single遙測信號類型的挑戰，通常無法回答您所有的觀察問題，以及您可以做些什麼來應對這一挑戰。 附錄可以使用開放式遙測，Prometheus，Jaeger和Grafana介紹一個完整的端到端示例。

第2至第6章提供了概念基礎，因此，如果您是可觀察到的空間的陌生，我建議您先完成這些空間。第7章

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

11專注於可觀察性，最佳實踐的某些操作或發展相關的方面，如果您願意的話，您可以將其刪除。

About the code

本書包含許多源代碼的示例，包括編號列表和與普通文本一致。在這兩種情況下，源代碼均以這樣的固定寬度字體格式化，以將其與普通文本分開。

在許多情況下，原始源代碼已經重新格式化； 我們增加了線路斷路並重新設計凹痕，以適應書中的可用頁面空間。在極少數情況下，即使這還不夠，列表包括線路連接標記（ ）。此外，源代碼中的評論通常已從文本中描述時從列表中刪除。代碼註釋伴隨著許多清單，突出了重要的概念。

您可以在https://livebook.manning.com/book/cloud-observability-in-action中獲取本書的LiveBook（在線）版本的可執行代碼段。本書中的示例中的complete代碼可從Manning網站https://www.manning.com/books/cloud-observability-in-in-action下載，並從https://github.com/github.com/mhausenblas/o11y-in-action.cloud.cloud/tree/tree/tree/main/main/main/code。

liveBook discussion forum

購買*Cloud Observability in Action*包括免費訪問Manning的在線閱讀平台LiveBook。使用LiveBook的獨家討論功能，您可以在全球或特定部分或段落上附加註釋。為自己做筆記，詢問和回答技術問題並從作者和其他用戶那裡獲得幫助是一種快照。 要訪問論壇，請訪問https://livebook.manning.com/book/ cloud-observability-in-action/討論。 您還可以在https://livebook.manning.com/discussion上了解有關曼寧論壇和行為規則的更多信息。

曼寧對我們的讀者的承諾是提供一個場所，在該場所，個人讀者之間以及讀者與作者之間的對話可以進行。這不是作者對任何特定數量參與的承諾，作者對論壇的貢獻仍然是自願的（和無薪）。我們很高興您嘗試問他一些具有挑戰性的問題，以免他的興趣流浪！只要本書印刷，就可以從發布者的網站訪問論壇和先前討論的檔案。

Online resources

如果您想深入研究某些主題，請查看以下在線資源：

本書的進一步閱讀部分（https://o11y-cloud.cloud/further-rading/），列出了文章，書籍和工具

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

“Return on Investment Driven Observability” (https://arxiv.org/abs/2303.13402), a sh ort article I published that discusses challenges that arise when rolling out observability i n organizations and how you can, grounded in return on invest- ment (ROI) analysis, add ress said challenges The OpenTelemetry blog (https://opentelemetry.io/blog/)

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

about the author

Michael Hausenblas在Amazon Web Services（AWS）開源觀測服務團隊中工作，在那裡他主持了OpentElemetry活動。他在數據工程和雲本地系統方面擁有20多年的經驗。在AWS之前，Michael在Red Hat上的Kubernetes，Mesophere（現為D2IQ）在Mesos和Kubernetes上工作，MAPR（現為HPE的一部分）是首席數據工程師，並在符號AI領域的應用研究中度過了十多年。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

about the cover illustration

*Cloud Observability in Action*封面上的圖是“ Cauchoise”或“ Caux中的女人”，這是從1797年出版的Jacques Grasset de Saint-Sauveur的收藏中獲取的。每個插圖都是精心繪製的，並用手彩色。

在那些日子裡，很容易確定人們的住所，以及他們生活中的貿易或人生的貿易只是他們的衣服。曼寧（Manning）根據幾個世紀前的區域庫的豐富多樣性來慶祝計算機業務的創造力和主動性，並通過諸如此類收藏的圖片重新栩栩如生。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

1 End-to-end observability

This chapter covers

我們所說的可觀察性為什麼可觀察性重要可觀察性的端到端示例雲本地系統的挑戰以及可觀察性如何幫助

在雲本地環境中，例如AWS或本地基礎架構（例如Kubernetes群集）等公共雲產品，通常會處理許多活動部件。 這些零件範圍從基礎架構層，包括計算（例如VM或容器）和數據庫，到您擁有的應用程序代碼。

根據您的角色和環境，您可能負責難題中的任何數量。讓我們來看看一個具體的示例：在雲提供商中介紹無服務器的Kubernetes環境。在這種情況下，kubernetes控制平面和數據平面（工作節點）都為您提供了幫助，這意味著您可以在操作方面專注於應用程序代碼。

無論您負責哪個部分，您都想知道發生了什麼，以便您可以對，理想情況下，甚至可以主動管理諸如諸如

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

突然使用峰值（因為營銷部門在不告訴您的情況下推出了25%的訴訟），或者是由於第三方集成失敗並影響了您的應用程序。您擁有的組件的範圍或可以直接影響確定您應該關注的觀察性方面的範圍。

最重要的是，您不想盲目駕駛。在雲本地系統的背景下，這究竟是我們將以動手方式在本章中探討的內容。雖然重要的是要在行動中看到事情，但隨著我們的進展，我們還將嘗試通過包括定義在內的更正式的手段來捕捉概念的要旨。

本書假設您熟悉雲原生環境。通常，您希望找到微服務體系結構，這是大量相對短暫的組件共同努力提供功能。這包括雲提供商服務（我正在使用AWS在這裡演示這些想法）；集裝箱技術，包括Docker和Kubernetes；和功能 - 服務（FAAS）產品，尤其是AWS lambda。如果您想閱讀，這裡有一些建議：

Kubernetes in Action, Second Edition，由MarkoLuk a（Manning，2020）
AWS Lambda in Action by Danilo Poccia（Manning，2016）

此外，我推薦Jamie Riedesel（Manning，2021）的*Software Telemetry*，這是本書互補的，並為我們在本書中不會詳細研究的某些可觀察性方面提供了有用的深入研究。

在本書中，我們專注於雲本地環境。我們主要使用開源可觀察性工具，因此您可以在沒有許可費用的情況下嘗試所有內容。但是，重要的是要了解，儘管我們使用開源工具來顯示行動中的概念，但它們是普遍適用的。也就是說，在專業的環境中，您應始終考慮將零件或所有工具卸載到託管產品中，您所選擇的雲提供商擁有或同樣，可觀察到的供應商的產品，例如Datadog，Splunk，New Relic，New Relic，HoneyComb或Dynatrace。 在進入雲本地環境以及在這種情況下的可觀察性含義之前，讓我們退後一步，從概念層面看一下。

1.1 What is observability?

什麼是可觀察性，為什麼要關心？當我們說*observability*時，我們的意思是嘗試通過測量外部可用的數據來理解內部系統狀態。通常，我們這樣做是為了採取行動。

在對可觀察性進行更正式的定義之前，讓我們回顧一下整本書中將使用的一些核心概念：

System - system under observation（suo）的short。 這是您關心並負責的雲本地平台（以及在其上運行的應用程序）。 *Signals* - 可以從系統外部觀察到的信息。有不同的信號類型（最常見的是日誌，指標和痕跡），並且它們是由源生成的。第2章詳細介紹了信號類型。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Sources - 基礎結構和應用程序層的部分，例如微服務，設備，數據庫，消息隊列或操作系統。它們通常必須儀器發出信號。我們將在第3章中討論資源。即*Agents* - 可負責，用於收集信號，處理和路由。第4章致力於代理商及其使用。
Destinations - 出於不同的原因和用例，您在哪裡消耗信號。這些包括可視化（例如儀表板），警報，長期存儲（用於監管目的）和分析（為應用程序找到新用法）。我們將分別深入第5章和第6章的後端和前端目的地。 *Telemetry* - 從源，路由或通過代理來收集信號的過程，以及攝入到目的地的過程。

圖1.1為您提供了可觀察性的視覺描述。動機是從通過代理商來源代表的系統中收集信號，以通過人類或應用程序的方式消費，以理解和影響系統。

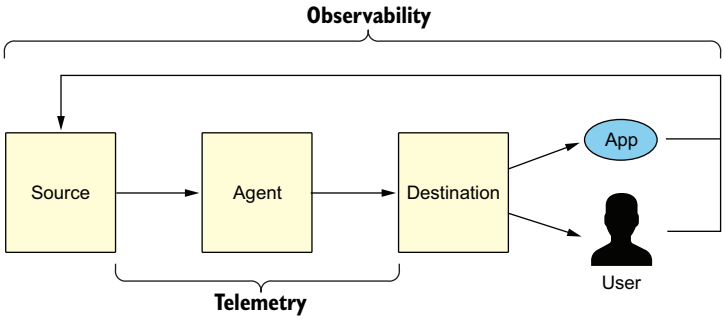


圖1.1可觀察性概述

從本質上講，可觀察性代表反饋循環。 例如，人類用戶可能會根據收集的信息重新啟動服務。在應用程序的情況下，這可能是群集自動製劑，該群集自動節目會根據測量的系統利用來添加工作人員節點。

可觀察性的最重要方面是提供可行的見解。只需在日誌行中顯示錯誤消息或在帶有精美圖形的儀表板中不夠。

定義*Observability*是基於觀察系統的信號連續生成和覆蓋可行的見解的能力，目的是影響系統。

可觀察性領域正在增長並涵蓋越來越多的領域，包括開發人員的可觀察性（我們將在第9章中介紹）和數據可觀察性。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

但是，您怎麼知道哪些信號是相關的？如何充分利用它們？在進入此主題之前，讓我們首先退後一步以設置場景，看看常見的可觀察性用例，並定義角色和任務。

1.2 Observability use cases

可觀察性是達到目的的手段。換句話說，當您要解決的一定挑戰或任務時，可觀察性支持您更快地完成該任務或更有效地管理上述挑戰。讓我們現在看一下常見用例，看看它們從中產生了什麼樣的要求：

Understanding the impact of code changes - 作為開發人員，您經常在代碼庫中添加新的feabure或修復錯誤。您如何理解這些代碼更改的影響？您需要哪些相關的數據點來評估（電位負面）效果，例如執行較慢或更多資源使用情況？

Understanding third-party dependencies - 作為開發人員，您可以使用在控制之外的事物，例如外部API（付款，位置服務等）。 您怎麼知道它們可用，健康和表演？

Measuring user experience (UX) - 作為開發人員，網站可靠性工程師（SRE）或操作員，您需要確保您的應用或服務響應迅速且可靠。您如何衡量這個？

Tracking health and performance - 作為操作員，您希望能夠掩蓋服務或應用程序的健康，其中包括整體正常運行時間，響應時間分配以及受部分或全面外的各方（例如，付費客戶，自由層的用戶，免費用戶或金絲雀帳戶）。

Blast radius exploration - 就像健康和UX一樣，在更基本的層面上，您可能希望了解任何直接應用或服務的爆炸半徑以及基礎平台（例如，在Kubernetes群集中運行的部署）。您如何確定您的應用程序中的錯誤還是在Kubernetes數據平面或基礎VM中存在錯誤？ *Optimizing a service* - 作為開發人員，您想進行優化，包括性能，資源使用情況，成本和響應時間。您如何衡量這些？

Increasing developer productivity - 作為服務所有者或工程經理，您對開發人員的生產力感興趣。現在，一套大屏幕和舒適的遊戲椅肯定會有所幫助，以及週一，星期二和星期五的無會議。但是，從長期（維護）角度來看，您如何圍繞生成的代碼來衡量開發人員的生產率呢？ *Auditing access and tracking compliance* - 作為操作員，您想跟踪誰可以訪問其他服務和客戶數據。您的系統如何捕獲權限和訪問權限，自動允許您警告未經授權的訪問，並在檢查時為您提供審計跟踪（例如，為了進行監管目的，可以說）？

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

這只是潛在的可觀察性用例的非常簡潔和壓縮的味道。如果您想了解更多有關可觀察性用例，信號和方法的信息，我強烈建議您通過Cloud Native Computing基金會的技術諮詢小組的可觀察性或更多非正式的技術建議小組來仔細閱讀“觀察性白皮書”（我很高興為; http://mng.bz/ eqjo）做出貢獻。 （術語*o11y*是指*observability*）。

> 可觀察性與監視
>
> *observability*只是*monitoring*的新流行語，還是這些不同的概念相互關聯？監視時，您正在查看系統的健康狀況（例如，您的平台或特定服務）。 可觀察性在概念上更廣泛。 儘管健康仍然很重要，但它還考慮了系統的變化，例如代碼更改的影響是什麼或外部依賴性對您的貢獻（例如，您的一項服務的響應時間的p90）。除了營銷術語外，您可以將監視作為*system health and performance-related observability*的速記，這就是我在本書中對待它的方式。

在我們的一些用例中，讓我們繼續在組織中扮演不同的角色。我們將討論他們的目標和任務，以更充分地理解並了解可觀察性如何幫助您團隊和更廣泛的組織中的不同人。

1.3 Roles and goals

如今，建造產品是一項團隊運動。除了小工具的情況下，不同的工作功能或角色的變化通常共同起作用，以實現基於軟件的產品。 軟件創建和操作過程中涉及的不同角色集中在應用程序生命週期的不同任務上：

Developers - 開發人員編寫代碼，測試和文檔，並製作諸如容器映像，二進製文件或腳本之類的工件（用於解釋的語言，例如Python），可用於部署。 可觀察性可以更快地迭代（通過日誌和痕跡查找錯誤），優化運行時行為（通過痕跡識別慢速代碼路徑），並支持操作（通過指標和日誌）。 *Release and test engineers* - 發行工程師可以擁有CI和/或CD管道，並且需要對構建，測試和部署過程的可見性。在這種情況下，一些組織具有專門的測試儀（或QA測試儀）角色，其要求與釋放工程師的要求相似。 *DevOps roles* - 一些團隊具有DevOps角色，這些角色是代碼部分負責的，通常是擁有自己的操作（在呼叫上），並建立和執行良好的實踐。 與開發人員有關可觀察性的類似考慮，此外，指標通常對儀表板或警報有用。一個著名的DevOps角色是SRE，他指導開發人員或支持其他DevOps角色，幫助他們實現某些操作目標（我們將在SLOS和SLIS的最後一章中回到這一目標）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Infrastructure and platform operations - 在不是雲本機的環境中，通常稱為*system administrators*或*sysadmins*。他們擁有平台（IAAS或容器），並提供（自助）訪問該平台以開發和DevOps角色。他們通常對低級信號感興趣，例如利用資源和與成本有關的信號。

Nonengineering roles - 產品所有者通常稱為*product (or project) man- ager*，並擁有一個功能，服務或整個應用程序，具體取決於範圍和大小。這些角色通常集中在最終用戶界定指標以及性能和可用性上。另一個非技術角色是商業利益持有人。 它們代表了指定要求的組織的一部分，通常專注於財務指標，例如使用，收入或出售單位。 對於此角色，技術指標通常是無關緊要的。 表1.1顯示了本節討論的示例可觀察性流。

表1.1每個角色的示例流

Role	Example flow
Developer	Uses traces to figure out which microservice along the request path is slow
Developer	Uses continuous profiling to speed up the hot path of a microservice
DevOps	Receives an alert about a component being down and then uses metrics plotted in a dashboard to understand the alert in context
DevOps	Uses metrics to narrow down components causing high latency; then jumps to traces to drill into a specific request path; and then looks at logs of one microservice, serving on that request path for details (correlation)
SRE	Tracks goals and their fulfillment, advising developers via dashboards
Platform operator	Uses a cluster-level dashboard to track autoscaling over time
Release engineer	Uses a dashboard to monitor the build progress of a microservice artifact (e.g., a container image) to, for example, prevent performance regressions
Product manager	Uses a CD dashboard to monitor deployment progress of microservices
Business stakeholder	Uses a dashboard to track weekly revenue and SLA violations

現在，讓我們看看一個雲本地系統，信號源和遙測的具體示例。

1.4 Example microservices app

我們將使用Weaveworks的微服務應用程序，Sock Shop（https：// microservices-demo.github.io/）來探索可觀察性。圖1.2顯示了其所有架構。 Sock Shop示例應用程序是一個使用不同語言和框架（Spring Boot，Go Kit和Node.js）構建的簡單演示應用程序，並使用打開的容器intiative intiative-Compliant容器圖像包裝。這使其可從集裝箱編排，例如Kubernetes，Docker組成，

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

和ECS以及FAASS，例如AWS Lambda。我們將在Kubernetes中顯示示例應用程序運行。

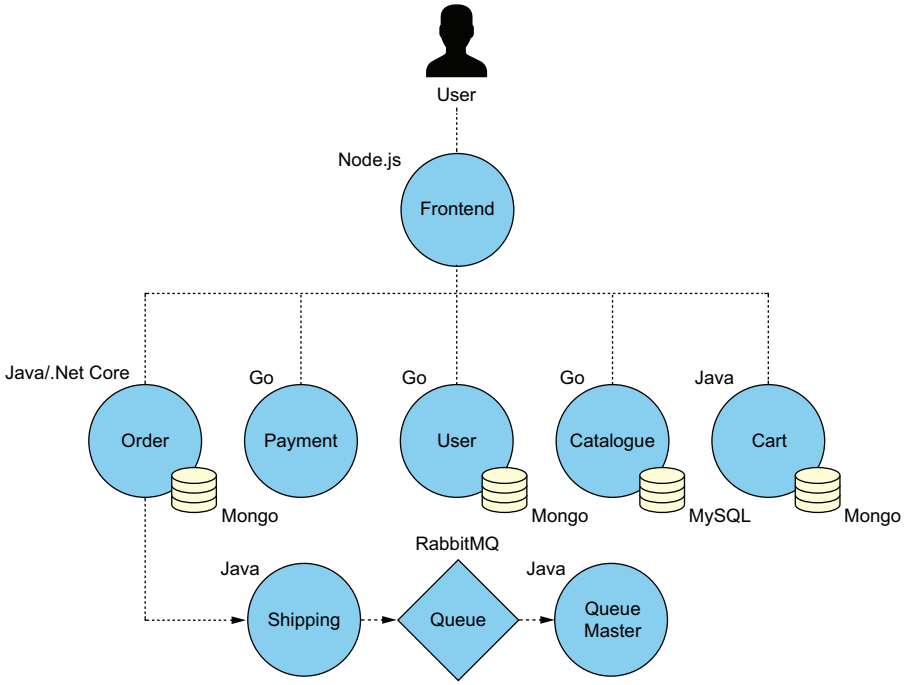


圖 ure 1.2我們用於可觀察性EXP的示例微服務應用程序 l

在整本書中提示，您會發現我使用教學設備來解釋沿日誌，指標和痕蹟的界限。 這純粹是為了構建內容並使您更容易學習和查找信息。 實際的現實更加混亂，因為項目和產品具有擴大範圍的趨勢。因此，在大多數情況下，明確的解釋，例如“這是用於日誌”或“將XYZ用於指標”，不適用於現實世界中。

要設置示例微服務應用程序，您需要一個Kubernetes群集。然後，按照以下列表中的步驟操作。

清單1.1本地可觀察性

克隆示例應用程序回購，並更改為kubernetes部署yaml文檔的目錄

```
$ git clone https://github.com/microservices-demo/microservices-demo.git &&
  \  cd microservices-demo/deploy/kubernetes/

$ kubectl create namespace sock-shop
```

Creates a Kubernetes namespace that acts as a logical home for our app

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

$ kubectl應用-f完整demo.yaml

$ kubectl -n襪子商店Port-Forward SVC/前端8888：80

創建所有必要的Kubernetes資源（部署，服務等），共同組成了該應用程序；如果您想密切關注它，請等到所有資源準備就緒，您可以（在其他終端會話中）使用Watch Kubectl -N Sock -Shop -Shop get Deplotion，POD，SVC命令。

所有POD都準備就緒後，您可以使用port-eward命令在本地計算機上訪問前端。

如果您成功設置了Sock Shop應用程序，則可以訪問瀏覽器中的入口點（前端服務）。前往http：// localhost：8888，您應該看到類似於圖1.3中所示的東西。

接下來，您要生成一些流量，因此使用用戶名用戶和密碼登錄（在右上角），然後在購物車中添加一些項目。現在，可觀察性呢？讓我們看看一個具體的示例到一端。

Sock Shop示例應用程序發出了各種信號。讓我們仔細研究一下兩種信號類型：日誌和指標，以探索確切的發射以及為給定角色引入可觀察性的必要條件。

就日誌信號類型而言，Sock Shop Services會發出各種日誌。 查看訂單服務的一個示例如下（輸出編輯為適合）：

$ kubectl日誌部署/訂單2021-10-25 14：33：10.649信息[bootstrap ,,,,,] 7 --- [main] S.C.A. AnnotationConfigapplicationContext：刷新org.springframework.context。 Annotation.AnnotationConfigapplicationContext@51521cc1：啟動日期[10月25日14:33:10 GMT 2021]; root of context hierarchy 2021-10-25 14:33:16.838 INFO [bootstrap,,,] 7 --- [main] tratio Delegate$BeanPostProcessorChecker : Bean 'configurationPropertiesRebinderAu toConfiguration' of type [class org.springframework.cloud.autoconfigure.Con figurationPropertiesRebinderaUto configuration $$ EnhancerBysbyspringcglib $$ B894F 39]不符合所有BeanPostProcessors處理的條件（對於Ex exame：不符合自動Proxying的資格）... ...

在這種情況下，源是訂單服務，信號類型是日誌。

注意幾乎每個人都使用日誌（與指標甚至痕跡相比），但是與其他信號類型相比，這種信號類型的標準化和互操作性要少得多。雖然OpentElemetry在出版時是穩定的，但對於指標而言幾乎是穩定的，但我們仍在社區工作，以實現與日誌的相似覆蓋範圍。

為了使日誌可行，以使開發人員修復錯誤，例如，我們需要以下內容：

代理（如fluent位）將日誌路由到目的地目的地，例如OpenSearch或CloudWatch

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

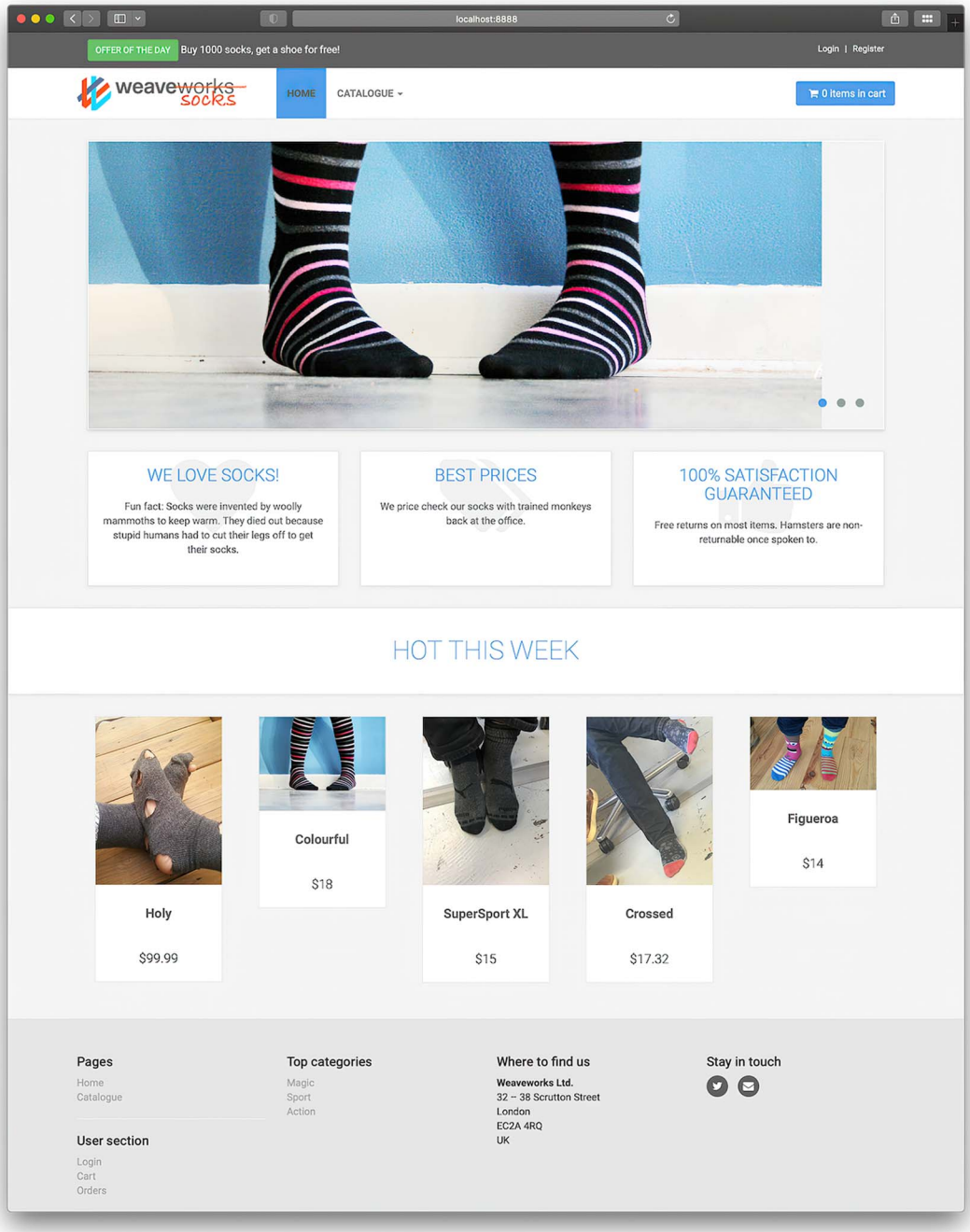


圖1.3 Sock Shop示例Microservices App Frontend的屏幕截圖

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

這些服務還以普羅米修斯博覽會格式公開指標（例如，查看我們較早地轉發給本地機器的前端服務）。以下內容編輯以適合：

```
$ curl localhost：8888/Metrics＃help Process_cpu_user_seconds_total總用戶CPU在幾秒鐘內花費的時間。
＃類型process_cpu_user_seconds_total counter進程
```

在這個cASE，源是前端服務，信號鍵 E是指標。

> 筆記
> 與客戶交談時，我注意到許多人面臨著同樣的挑戰：感覺需要收集和保留所有指標，因為“我們可能需要它們”。嘗試在攝入量中盡可能地修剪，並檢查是否實際使用了降落在目的地的指標。這些是一些問題，您可以問自己是否有必要收集特定指標：
>
> 該指標是否在警報中使用？它在儀表板上使用嗎？ 您能想到可以從指標中獲得見解的用例嗎？總體可以足夠嗎？ 痕跡在這裡是更好的選擇嗎？

為了使指標可行，以評估指標，例如，我們需要以下內容：

代理（例如Prometheus或Opentelemetry Collector）要刮擦指標目的地，例如Prometheus或用於長期存儲和聯邦，Cortex或Thanos

目前，我們不會考慮跡線，因為它們需要更多的設置（包括遙測方和目的地），但是簡而言之，您可以將痕跡視為提供跨服務執行的概述，包括成功處理請求的服務的持續時間和狀態。

我們現在已經接觸到日誌和指標。在本書的其餘部分深入了解雲的可觀察性之前，讓我們通過查看與雲本地系統時需要牢記的挑戰來結束本章，以及可觀察性如何幫助解決方案。

1.5 Challenges and how observability helps

雲本地環境及其上的應用程序面臨著幾個挑戰。在本小節中，我們討論了最緊迫的，並展示了可觀察性如何提供幫助。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

通常（忽略包裝和調度的細節，例如Kubernetes Pod與Lambda功能），雲本地系統具有以下特徵：

*They are distributed in nature.*大多數通信都沒有在進行中 - 甚至可能不會在同一台計算機上（例如，使用過程間通信[IPC]機制），但是通過網絡進行的。

Due to the distributed nature, not only the what *but also the*其中*is crucial.*想到為特定用戶服務的請求路徑的許多微服務。您希望能夠了解不斷到一端的情況，並能夠深入研究每個服務。

The volatility of the services is higher compared to a monolith running on, say, a VM. pods（隨之而來的是他們的IP地址）來來去去，每小時可能會將lambda函數的新版本推出，依此類推。

現在，讓我們看一下可觀察性如何幫助解決雲原生系統中發現的挑戰。使用雲原生系統，您不想盲目駕駛；具有自動化的方式來收集所有相關信號並將其用作決策的輸入（即可行的見解）是理想的。

1.5.1 Return on investment

要了解適當的投資水平（即時間和金錢），您需要首先了解成本：

Instrumentation costs - 包括開發人員的時間，這是一個持續的成本。

Signal retention - 這些成本（例如，日誌存儲）可以是一次性的，但通常屬於付款方式。 *Overhead of agents and instrumentation* - 這在服務本身使用的基礎上以計算和內存資源的使用方式表現出來。成本應在CPU和RAM利用率的低單位範圍內。例如，AWS Open-temetry Collector（http://mng.bz/n2j1）提供了跨不同信號類型的足蹟的報告。 *Network usage* - 這種類型的成本，例如出口流量，可能是可觀察性解決方案的重要成本驅動力。

投資回報到可觀察性的回報更難計算。 確定回報主要取決於目標的清晰度。我們將很快討論不同角色的目標。例如，您可以在儀器之前和之後進行給定的服務，可以測量平均恢復時間（mttr; https：//www.atlassian.com/incident-management/kpis/common-metrics）。但是，也有許多與人相關的面孔（例如，在呼叫壓力或工作 - 生活平衡）方面依靠收益。您可以在“以投資驅動的可觀察性”（https：// arxiv .org/abs/2303.13402）（我發表的一篇關於該主題的簡短論文）中閱讀有關該主題的更多信息。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

1.5.2 Signal correlation

不同的信號類型通常適用於不同的任務。另外，通常無法回答您可能遇到的所有問題。 我們將在第11章中詳細討論信號相關性；但是，我想在這里為您提供一個快速的動態示例。

假設您正在打電話並進行打電話（半夜）。您開啟筆記本電腦，查看收到警報中鏈接的Grafana儀表板。從儀表板（例如，繪圖Prometheus指標）中，您可以收集微服務應用程序中的某些服務器在過去20分鐘內的錯誤率提高。您有一個預感，這可能與最近的服務升級有關。您可以使用嵌入Prometheus指標中的示例性ID（http://mng.bz/d4jw）作為進入點，以更仔細地查看所涉及的服務，並使用分開的tracing tracing Tracing Tool jaeger jaeger。 在Jaeger中，您會看到跟踪持續時間分佈和錯誤代碼，然後深入到服務日誌中以驗證您的假設。

在這種情況下，您將不同的信號類型（度量與跡線與日誌）相關聯，並能夠快速有效地確認您的假設。一種簡單的形式是基於時間的，這是您在許多前端（例如Grafana）中獲得的東西，而沒有開箱即用的額外努力。這意味著您使用時間段查詢不同的後端，以手動查找告訴您整個故事的信號。

1.5.3 Portability

通過選擇開放標準和開源來避免鎖定。這樣可以確保您可以在不同的環境（例如雲提供商或本地基礎架構）中使用相同的機制和工具，而無需更改大部分設置。對於考試，如果您使用OpentElemetry來啟動應用程序，則可以從一個目的地切換到另一個目的地，而無需更改應用程序或代理。

Summary

雲本地系統，例如Kubernetes，會產生大量的遙測信號。 遙測信號的來源可以是您的代碼，API（例如AWS服務），數據庫等。 術語*telemetry*是指從源頭收集信號，通過代理進行操作並將其攝入可觀察到的目的地的過程，在那裡它們可以被人類和機器消耗。 通常，*observability*意味著從信號中獲得可行的見解，並影響並影響正在觀察的系統。 有許多可觀察性的用例，從減少MTTR到成本優化到提高開發人員的生產率。 可觀察性與執行其典型任務的不同角色（開發人員，DevOps，非技術角色等）有關。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

我們查看了一個雲本地應用程序的具體示例設置（Kubernetes中的容器iz Ized Microservices應用程序），其信號以及在這種情況下的可觀察性可能意味著什麼。 雲本地系統的挑戰包括它們的分佈性質和相關組件的體積。 可觀察性可以通過允許您盲目飛行來幫助應對雲原生系統中發現的挑戰。 了解可觀察性的成本和投資回報率很重要。 信號相關性很重要，因為它使您可以組合不同的信號類型來回答有關係統可能遇到的所有問題。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

2 Signal types

This chapter covers

哪種信號類型是何時以及如何使用特定信號類型每種信號類型的成本和收益如何確定哪些信號是相關的確定信號基線

在可觀察性的背景下，信號起著核心作用：這是我們決定的英特爾。在高級別上，我們可以區分自然界數字的信號（例如，服務調用數量）和攜帶需要人類解釋的文本有效載荷的信號（例如，日誌線）。

如第1章所討論的，觀察到的系統或僅*system*（例如，基於lambda函數的Kubernetes應用程序或無服務器應用程序），從各種來源發出signals。您作為人類或同樣的軟件，然後使用這些信號來理解系統並做出決定以影響它。系統和來源之間存在一對多關係。 您的系統通常涉及計算，存儲和網絡，這些都代表信號源。我們將在第3章中詳細介紹資源。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

在信號（和來源）方面，我們可以區分兩種情況：

諸如代碼之類的資源，您擁有並可以自行儀器（也就是說，確定要發射的信號和哪種信號）。 您不擁有的資源，這意味著任何信號將被預定義。在這種情況下，您的任務是選擇要消耗的信號。這可能是由於幾個原因： - 代碼是您的代碼（庫，軟件包，模塊等）的依賴性。 - 組件的源代碼不可用（例如，二進製文件）。 - 組件僅提供（網絡）API。您選擇的雲提供商的任何託管服務都屬於此類別。

無論您是否擁有代碼並可以決定要發射哪些信號類型或正在使用您不擁有的組件，在某個時間點，您都希望收集信號並將其發送到中心地點進行處理和存儲。這是遙測部分。

有了可觀察性，您想採取一種通過投資回報率（ROI）驅動的方法。換句話說，您不僅需要了解可用的不同信號，還需要每種信號類型的成本以及使您可以做什麼。

在本章中，我們將回顧最常見的信號類型，檢查如何儀器和收集每種信號，並討論這樣做的成本和收益。

2.1 Reference example

儘管在實踐中有幾個信號需要考慮，但最常用的三個是日誌，指標和痕跡。在本章中，我們將使用一個運行的參考示例，一個簡單的GO程序，稱為Echo服務，公開HTTP接口，以查看在操作中的三種常見信號類型。

更改為CH02/ base/ Directory，您可以在其中找到Main.GO程序2.1中顯示的程序。隨著我們的進展，我們將使用不同的信號來啟動參考示例，其目標是使用開源工具（包括Grafana，Prometheus和Jaeger）消費並與信號進行交互。

列表2.1示例GO程序要儀器

包裝主

```
import (
    "fmt"
    "log"
    "math/rand"
    "net/http"
    "time"
)

func handleEcho(w http.ResponseWriter, r *http.Request) {
    message := r.URL.Query().Get("message")
    if rand.Intn(100) > 90 {
```

Retrieves the payload to echo from the query parameter message, with the result of being able to interact with the service by issuing an HTTP GET to /echo?message

Simulates a 10% error case

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
        w.WriteHeader(http.StatusInternalServerError)
        return
    }
    fmt.Fprintf(w, message)
}

func main() {
    rand.Seed(time.Now().UnixNano())
    http.HandleFunc("/echo", handleEcho)
    log.Fatal(http.ListenAndServe(":4242", nil))
}
```

Return the exact payload we retrieved in 90% of the cases

Register handler for the /echo path

Launches the HTTP server that will serve on port 4242 on all available network interfaces

首先，讓我們構建Echo服務，如以下列表所示。

列出2.2構建我們的示例GO程序

```
$ go build -o echosvc。
```

構建可執行文件，然後在當前目錄中可用為./echosvc

現在，您可以在同一目錄中使用./echosvc運行服務。要通過其HTTP API調用它，您將使用以下代碼（在單獨的終端會話中）：

```
$ curl localhost：4242/echo？消息=測試

測試
```

默認情況下，我將使用Grafana作為目的地，因為它允許我們以統一的方式與廣泛的數據源進行可視化和相互作用。 這將使您可以通過開源免費複製所有內容。 現在，我們已經列出了參考示例，讓我們談談一種評估儀器成本的方法，然後才能移至儀器本身。

> 提示
> 要查看前進的信號，在Grafana中，您需要在Echo服務上加載一些負載。一種選項是直接從外殼執行以下操作（每秒調用端點）：
>
> ```
> 雖然是真的；做curl -s -o/dev/null localhost：4242/echo;睡眠1;完
> 畢
> ```
>
> 另外，您可以使用更奇特的東西，例如Salvo（https://github.com/tarekziade/salvo）等成熟測試工具，甚至是K6（https://k6.io/docs/testing-guides-guides-guides-guides-guides/api-load-testing/）。

2.2 Assessing instrumentation costs

為了評估每種信號類型的儀器成本，我們將使用一個簡單的度量：*business logic to instrumentation*（b2i）比率。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

B2i比
要計算B2I比率，請在添加儀器之前確定代碼線（LOC）的線數（添加用於發射信號類型的信號的代碼），然後在儀器後確定LOC。那時B2i的比率

b2i = loc_after_instrumentation/loc_before_instrumentation

在理想的世界中，B2I比率為1，代表代碼中的零儀器成本。但是，實際上，您專用於儀器的LOC越多，B2i比率就越高。例如，如果您的代碼具有3800個LOC，並且您添加了400個LOC以供儀器（例如，發射日誌和指標），則最終以（3800 + 400） / 3800的B2I比為1.105。

關於B2i，有兩件事值得一提：

由您決定是否在您的B2I比率計算中考慮是否考慮諸如Open-Telemetry SDK（https://opentelemetry.io/docs/instrumentation/）之類的遙測庫。從務實的角度來看，專注於自己的代碼（不包括這些依賴性）可能是最有意義的。這就是我在本章的其餘部分中要做的。 B2i比是一個簡單的措施，絕不應該作為您的代碼儀器效果的指標，或者告訴您您的代碼是否為*over-instrumented*。古哈特定律（請參閱Strathern and Marilyn，1997年，“提高評級：大英大學系統的審計”，*European Review*，John Wiley＆Sons）指出，當措施成為目標時，它就會停止成為一種良好的措施，這將適用於這裡。在回歸驗證中，將B2I比作為粗略的指南，而不是絕對的北極星。

隨著理論和概述，讓我們現在回顧三種基本信號類型：日誌，指標和痕跡。在本章的其餘部分中，我們將為每種信號類型進行以下操作：

1定義信號特徵和語義。 2討論如何啟動我們的示例迴聲服務。 3設置遙測設置以收集和路由信號。 4分析成本（就資源使用和努力而言）以及信號的收益。 5解釋各自信號如何有助於可觀察性解決方案。

2.3 Logs

讓我們從專注於日誌開始。日誌是具有捕獲事件的文本有效載荷的信號。 *event*是一個傘術語，涉及代碼中發生的任何事物，無論是在快樂的路徑上（完整的上傳）還是意外的或錯誤的行為（“無法將物品存儲在購物籃中”）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

日誌是時間戳，應進行結構，理想情況下包括一些可用於與其他信號相關的上下文（例如標籤）。通常，人類通過直接搜索事件或間接地通過相關性來進行記錄。

請注意，有很多方法可以參考單個日誌事件，包括日誌行，日誌項目，日誌條目，也許還有其他我還沒有遇到的其他日誌事件。為了使事情變得簡單，我們將在此處始終使用*event*一詞，但是這些術語總體上可以互換。

在日誌的上下文中，我們通常區分*levels*，這可以代表日誌事件的性質。 隨著時間的流逝，不同的編程語言，庫和框架作品已經建立了慣例； 但是，以下是一些常見語義：

與開發或修復錯誤相關的細粒度日誌事件（調試）生產型日誌事件，包括以下內容： - 預期行為（通常是信息） - 意外行為，例如警告或錯誤

有幾種可用的日誌標準和格式，包括面向線路的格式，例如Syslog RFC 5424（https://tools.ietf.org/html/rfc5424）和apache com- com- com- com- nog格式（clf; http:/ http://mng.bz/lwg8）; JSON日誌格式（大多數現代庫支持）； 和二進制格式，例如日記文件格式（http：// mng .bz/bmw8），可在SystemD的上下文中找到。要了解更多信息，請查看Graylog的日誌格式指南（http://mng.bz/d1no）。 在日誌背景下的一些良好實踐包括以下內容：

考慮將其記錄的內容，目的是提供可行的見解（上下文是國王）。 登錄到stdout而不是固定位置，例如文件。這使日誌代理可以收集和路由您的日誌事件。另請參見12個Factor方法（https://12factor.net/logs）中的條目。 使用標籤來增強日誌事件的價值。例如，您可以使用組件名稱或類型，源的種類或操作類型作為標籤，以幫助過濾事件。

提示，如果您想知道如何“未來證明”圍繞日誌的策略，我建議您閱讀本·西格曼（Ben Sigelman）（Lightstep的共同點（以Opentelemetry，Google Dapper和Monarch）的工作而聞名，2001年Twitter線程，“為什麼追踪可能會（幾乎可以替代）所有記錄”（http://mng.bz/ rass）通過儀器登錄工作量的首當其衝，以介紹傳輸本身（而不是記錄基礎架構或其他資源）。這是有道理的，至少對於交易設置中的日誌數量的大量用例，我想鼓勵您考慮這種思維並將其應用於您的環境，如果您可以這樣做。自然，許多環境比綠色田地更多

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

頻譜的結束，因此建立以追踪為中心的方法（而不是以日誌為中心）的方法可能很困難或需要大量時間和精力。

2.3.1 Instrumentation

現在，讓我們查看登錄行動。為此，我們可以進行迴聲服務，以便發出日誌事件。以下列表顯示了我們帶有日誌的Echo服務工具。

列表2.3示例GO程序，儀器發射結構化日誌

```
package main

import (
    "fmt"
    "math/rand"
    "net/http"
    "time"

    log "github.com/sirupsen/logrus"
)

func handleEcho(w http.ResponseWriter, r *http.Request) {
    message := r.URL.Query().Get("message")
    log.WithFields(log.Fields{
        "service": "echo",
    }).Info("Got input ", message)
    if rand.Intn(100) > 90 {
        log.WithFields(log.Fields{
            "service": "echo",
        }).Error("Something really bad happened :(")
        w.WriteHeader(500)
        return
    }
    fmt.Fprintf(w, message)
}

func main() {
    rand.Seed(time.Now().UnixNano())
    log.SetFormatter(&log.JSONFormatter{})
    http.HandleFunc("/echo", handleEcho)
    log.Fatal(http.ListenAndServe(":4242", nil))
}
```

No matter what, always emit an INFO log line here that confirms the input value we got via the query parameter message. Note the "service": "echo" in the line above (and again in the error case below): this provides a context for the event, allowing you to pinpoint the service.

Emit an ERROR log line in error case.

Configure the logger to emit log lines formatted in JSON. Also, note that the logger we're using here, logrus (https://github.com/Sirupsen/ logrus), defaults to output to stderr, something we have to consider when we consume the logs, later.

要構建服務，您還需要通過稱為go.mod的文件來定義其依賴項（如下列表所示）。

列表2.4示例GO程序依賴項文件go.mod

模塊Co11yia/ch2/logs

走1.20

需要github.com/sirupsen/logrus v1.8.1

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在，要拉動迴聲服務依賴項，請運行以下命令：

$ GO MOD供應商

com 如列表中所述，通過執行GO構建來完成構建 ING 2.2。

2.3.2 *Telemetry*

現在，我們可以在本地執行ECHO服務，並將其執行過程中發出的日誌事件發送到某些目的地，例如屏幕上並進入文件：

```
$ ./echosvc 2>&1 | \
  tee echo.log
```

Execute the echosvc and redirect its stderr output to stdout.

Use the tee command to show the log lines on the terminal, at the same time appending the log lines to a file called echo.log. We will use this file as an input to ship to a destination in the next step.

接下來，我們將使用Grafana Labs提供的一些開源工具來路由和收集日誌。 我們以Promtail（http：//mng .bz/v1j0）捕獲迴聲服務的日誌，並將其運送到Loki（https://grafana.com/docs/loki/latest/）（作為Desti-Nation）（作為Grafana的數據源）。 然後，使用Grafana作為前端，您可以根據需要查看並查詢日誌。

要設置我們的日誌場景，請使用以下列表中顯示的Docker組成的清單文件。它啟動了所需的一切，從目的地到前端。

使用docker-compose.yaml列表2.5站立起步，loki和grafana

```
version: "3.5"
networks:
  loki:
services:
  loki:
    image: grafana/loki:2.4.0
    ports:
      - "3100:3100"
    command: -config.file=/etc/loki/local-config.yaml
    networks:
      - loki
  promtail:
    image: grafana/promtail:2.4.0
    volumes:
      - ./promtail-config.yaml:/etc/promtail/config.yml
      - ./echo.log:/var/log/echo.log
    command: -config.file=/etc/promtail/config.yml
    networks:
      - loki
  grafana:
    image: grafana/grafana:latest
    ports:
      - "3000:3000"
    networks:
      - loki
```

Defines the location of the Promtail config file. We map the host local file promtail-config.yaml into the container to /etc/promtail/config.yml, where Promtail expects to find it.

Maps the host local file echo.log (which our echo services logs to) into the container to /var/log/echo.log. We will use that in listing 2.6 to define the log processing and routing pipeline.

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

提示在使用Docker撰寫本地開發或測試應用程序時肯定很方便，對於生產設置而言，Kubernetes更有用。例如，您可以通過Helm Charts安裝Promtail，Loki和Grafana堆棧，也可以使用Tanka（https://tanka.dev/）安裝。

現在，我們已經設置了通過我們的Docker組成文件（列表2.5）啟動日誌目的地（LOKI），遙測代理（Promtail）和前端（Grafana）的框架（grafana），我們還需要一件事：配置ProPARTAIL，我們的日誌路由器/ Ingester。 Docker構成了清單參考文獻（在第15行中）此Promtail Configucration文件，我們稱之為Promtail-config.yaml，如以下列表所示。

列表2.6 Promtail配置文件Promtail-config.yaml

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://loki:3100/loki/api/v1/push

scrape_configs:
- job_name: app
  static_configs:
  - targets:
      - localhost
    labels:
      job: echosvc
      __path__: /var/log/echo.log
 pipeline_stages:
  - json:
     expressions:
        level: level
        payload: msg
        timestamp: time
        service: service
  - labels:
     level:
      service:
```

This defines the input for Promtail. Here, we tell it to ingest logs from a file called /var/log/echo.log (which we mounted from the host via a volume in line 16 of the Docker Compose manifest).

This is the first part of the Promtail pipeline (http://mng.bz/x48e). It maps the input JSON fields that our echo service emits (e.g., service or level, the right-hand side here, the values) to labels we want to filter for in Loki; for example, service or level (the same names here, but you can, and often must, normalize keys, and that's a good place to do this).

Here, we define the labels we want to index or expose.

假設您將列表2.5的內容存儲在名為docker-compose.yaml的文件中，那麼使用以下代碼來添加堆棧是很簡單的：

$ docker-compose -f docker-compose.yaml up創建網絡“ logs_loki”，默認驅動程序拉動loki（grafana/loki/loki：2.4.0：2.4.0）... 2.4.0 ...

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

創建logs_promtail_1 ...完成創建logs_grafana_1 ...完成附加到logs_loki_1，logs_grafana_1，logs_promtail_1 ...

現在，您可以轉到http：// localhost：3000/在瀏覽器中登錄Grafana。使用以下默認值（輸入憑據後，請確保跳過密碼更改請求）：

用戶名：管理員密
碼：管理員

接下來，配置LOKI數據源，如圖2.1所示（請注意，屏幕截圖已被編輯，以便它專注於相關部分；實際上，您會在此屏幕上看到更多選項）。您需要在此步驟中設置的唯一配置是數據源URL，即http：// loki：3100，這要歸功於我們的Docker Compose Setup。

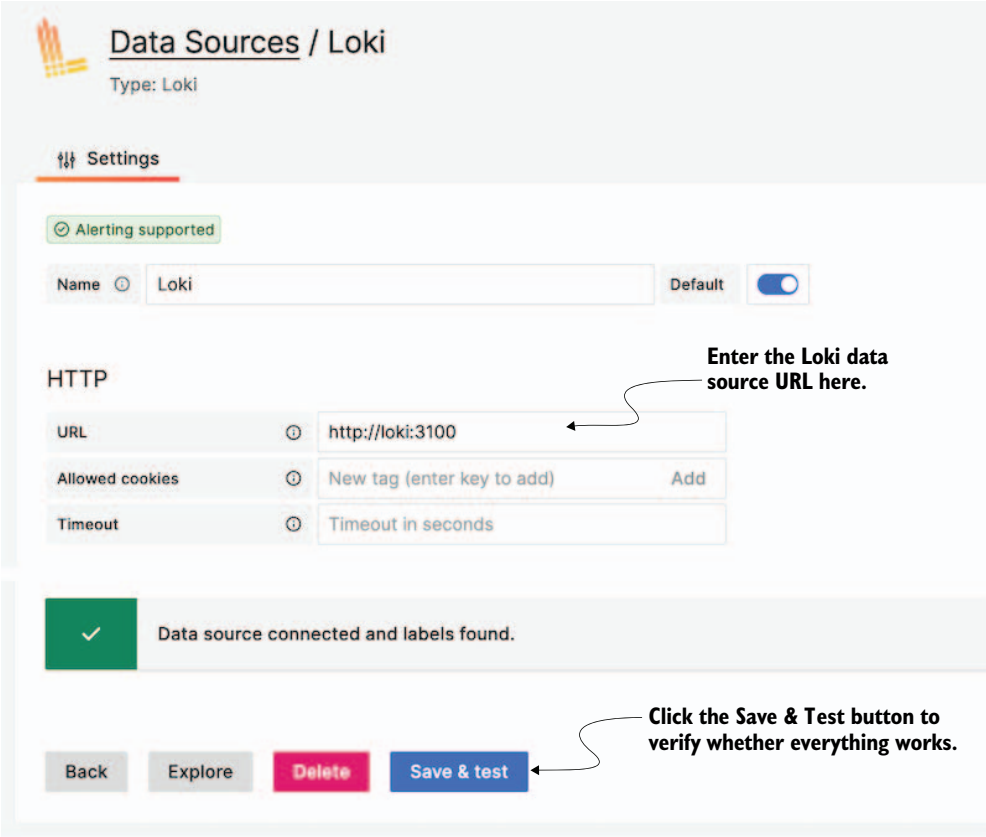


圖2.1在Grafana中添加和配置LOKI數據源

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

現在，您可以轉到左手菜單中的“探索”選項卡，看起來像是指南針。選擇loki作為數據源（如果尚未設置為默認值），然後在標籤過濾器下，從下拉框框中選擇“服務為標籤”，然後迴聲作為值。如果您選擇一定的時間範圍，則應看到類似於圖2.2所示的內容。

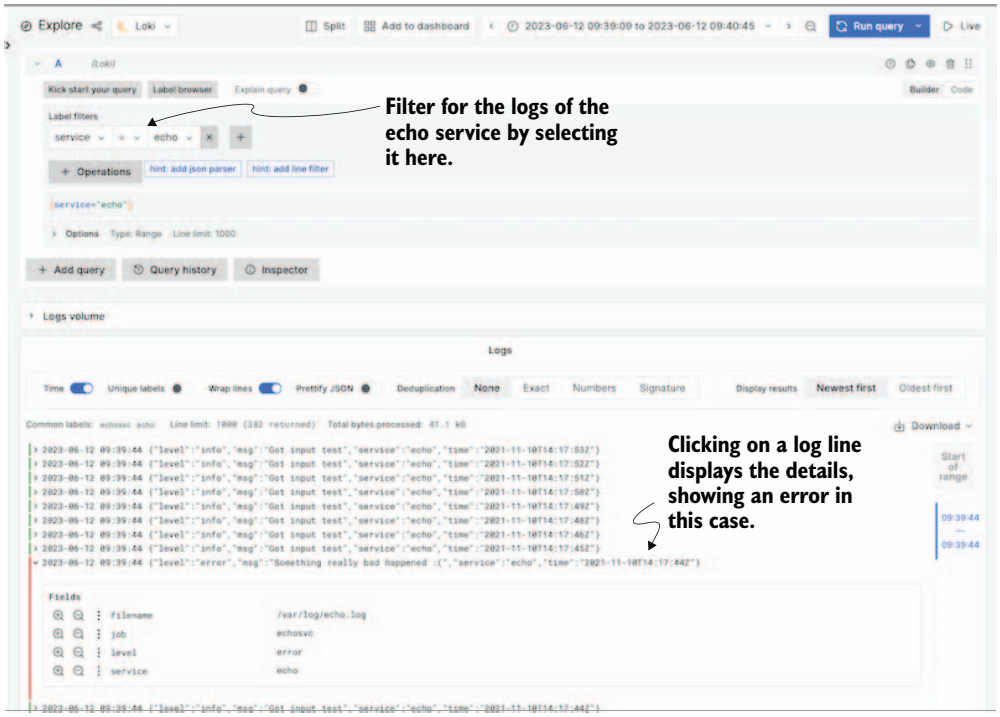


圖2.2探索Loki中的日誌

LOKI數據源探索視圖快速易於過濾某些日誌級別。例如，您可以添加另一個標籤過濾器，例如級別=“錯誤”，以顯示錯誤日誌行。

> 提示要查看正在運行的內容，請使用Docker-Compose Top（在包含Docker-compose.yaml文件的目錄中）或Docker PS（任何地方）。此外，要拆除所有內容，再次卸下所有容器，請使用Docker-opsose -f docker-compose.yaml down命令。

這樣，我們已經完成了實用的日誌場景。現在，我們可以繼續討論日誌所帶來的值以及關於日誌的成本（在資源和努力方面）的樣子。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

2.3.3 Costs and benefits

在本小節中，我們將仔細研究日誌信號類型的成本和收益。表2.1顯示了成本的概述。

表2.1日誌成本概述

	Overhead	Considerations
Instrumentation	Small and simple to add a log line	Widely adopted and supported by programming languages
Agent	Some runtime, lots of options, and few standards for formats (syslog, OTLP)	A runtime footprint, ingestion performance, reliability, and security
Destination	Long-term storage can be expensive.	Data temperature (access recency or frequency) can guide retention

原木的儀器成本

原木的成本相對較低，用於儀器，理解並得到廣泛支持。 所有編程語言及其庫允許用戶發出日誌。添加日誌線條並沒有遭受可讀性和可維護性的影響。

從開發人員的角度來看，讓我們看一下儀器成本。 列表2.1中的原始代碼具有24行代碼，並且列表2.3中帶有日誌的代碼具有32行代碼。將兩個LOC數字一起計算，我們可以計算B2I比率並達到1.33的值，這還不錯。

日誌的遙測成本

收集和路由日誌與中心位置相關，這通常是在分佈式系統中所必需的。根據您在此處使用的選項，可能會導致攝入（網絡流量）成本。

選擇某個代理時，請問自己以下 ：

運行時足跡是什麼（CPU和內存消耗）？ 攝入表現是什麼？例如，代理可以處理多少個日誌行？ 就可靠性而言，是否有保證每條日誌線（如果這是必需的）？ 從電線上的加密（TLS）到多種支持，是否有適當的安全性？

提示，如果您不確定如何測量日誌代理性能，請查看Phil Wilkins *Logging in Action*的第7章（Manning，2022; https：// www.manning.com/books/books/logging-in-in-action）。

以下是日誌路由器的一些示例：

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

雲本機計算基礎（CNCF）項目的Fluentd（https：//github.com/fluent/fluentd）和fluent bit（https://github.com/fluent/ fluent-bit）。 CNCF項目OpentElemetry提供了一個稱為Collector（http://mng.bz/aojk）的實驗日誌路由器。 雲提供商 - 特定代理商，例如Amazon CloudWatch Agents（http：//mng.bz/zqjo），Azure Monitor Agents（http://mng.bz/2dnd）或Google Cloud Cloud Suite Suite Suite Suite Agents（https://cloud.google.com.com.com.com/monitoring/agententer/agententer/agententer/agemitoring/agentering/agentering/agententer/）。 專有代理，例如Elastic的LogStash（https://github.com/elastic/logstash）和filebeat（http://mng.bz/1qev），dataDog代理（https：////// procks .datadoghq.com/agent/agent/agent the dataDog代理。

為了避免在此階段以及目的地中的極高成本，您可能希望在這裡修剪原木。您可能要考慮的一些修剪策略包括

Sampling - 也就是說，基於啟發式*Filtering based on type*刪除一定比例的事件 - 例如，在prod中丟棄調試級別日誌

有商業產品，例如CRIBL（https://cribl.com/），可以幫助跨提供商在大規模上定義和應用修剪策略。

日誌的目的地成本

一旦來自不同微服務的日誌到達中心位置（目的地，例如Splunk，CloudWatch，Open Search或Datadog），主要成本類型就是存儲。在大規模上，日誌可能非常昂貴，因此通常，您不想永遠將所有內容保留在周圍。最好根據*data temperature*來確定日誌的保留期：

Hot data - 您的工作集 - 也就是說，您在當前時間範圍內使用的日誌來進行故障排除或開發。它必須容易訪問，並且可以快速查詢。通常，幾天到幾週。 *Warm data* - 您可能需要但沒有必要立即加速或可查詢。這裡的保留期可能是幾個月甚至幾年。 *Cold data* - 所有您不主動使用的（日誌）數據。出於監管原因，您可能需要將這些日誌線保持在周圍。最好將這種日誌數據存儲在廉價，可靠和耐用的對象存儲中，例如在Amazon S3冰川中。訪問數據可能需要數分鐘或幾個小時，但是鑑於您很少使用它，這是您可以忍受的權衡。

原木的好處

現在您已經了解了日誌的成本，讓我們簡要地談論他們的收益。首先，日誌被普遍採用和理解。每個開發人員都知道如何發射日誌線，並且在每種編程語言中都有對日誌的絕妙支持，

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

通常在標準庫或更廣泛的生態系統中（第三方軟件包，模式或庫）。

日誌的第二個主要好處是，由於其文字性質，人類通常可以解釋含義。但是，開發人員仍然需要能夠編寫可操作的日誌線（https://logz.io/blog/logging-best-practices/），例如格式化，上下文等。除此之外，您需要的只是找到日誌行，您可以解釋它。找到相關日誌線的方法有兩種：

傳統上，每個日誌線（或日誌項目）都是*full-text indexed*。 Elasticsearch或OpenSearch等系統以及許多商業產品採用這種方法。這很棒，因為您可以搜索所有內容。但是，它具有可擴展性和成本挑戰。 最近，*label-based indexing*變得流行。 您只需要索引提供上下文的標籤（或標籤）（例如，這是來自Service Foo，它是關於和誤差條件）。這種方法最受歡迎的例子是Grafana Loki（https：//grafana .com/oss/loki/），靈感來自普羅米修斯。

這樣，我們已經達到了日誌成本和收益討論的終結，因此讓我們繼續在可觀察性的情況下繼續使用日誌的用例。

2.3.4 Observability with logs

我們該如何處理日誌，在動手部分中與觀察性有關的情況如何看待？

作為開發人員，您可以使用日誌

用於調試現有代碼庫（您擁有源代碼或可以對其進行反向工程）。 進行測試（例如，浸泡測試，性能測試或混亂工程），看看您的微服務如何分解。

此外，在DevOps或SRE角色中，日誌

幫助確定負責錯誤的服務和代碼，以（a）停止出血（重新啟動或離線），以及（b）在已確定的代碼中解決問題。 提供上下文（例如，確定誰在何時進行哪些更改）以及解決與監管或安全有關的問題。 log4j漏洞CVE-2021-44228（https://nvd.nist.gov/vuln/detail/detail/cve-2021-44228）和CVE-2021-45046（httttps：//nvd.nist.nist.gov/vuln/dettps：/wee n.nist.guln/deteet anty we 2021年，是安全動機用例的好示例。

2.4 Metrics

指標是數值信號，通常是按常規時間間隔採樣的。它們通常由名稱，值和一些元數據組成。名稱標識了度量標準（例如，

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

溫度），值是我們感興趣的觀察值（例如2.5），元數據提供了上下文（例如，與位置相關的上下文中的Room_42）。 觀察的單位可以用名稱編碼或作為元數據提供。最後但並非最不重要的一點是，該度量是時間戳，要么是由公制的生產商或Ingest（服務器端）明確的。

傳統上，語義通常會以層次的方式進行編碼，但是隨著Kubernetes和Prometheus的出現，基於標籤的，平面的組織計劃變得更加主流。 這種基於標籤的方案更加靈活和擴展。以下示例說明了區別：

```
room_42.sensor_window.temperature 2.5
```

A metric using a hierarchical scheme

溫度[位置=“ room_42”，傳感器=“窗口”] 2.5

使用基於標籤的方案相同的度量。如果一個人想在基於標籤的方案中添加另一個方面，例如位置，那麼添加新標籤（例如，位置=“ top”）就像是層次結構方案一樣容易的，一個人需要重新考慮整個名稱。

我們通常區分低級或系統指標以及應用程序級別或業務指標。前者是資源使用指標，例如CPU利用率或配置的主內存，後者包括每秒處理的請求或每個用戶產生的收入之類的內容。通常，通常可以自動收集低級指標，而應用級指標則需要人類來協助定義和暴露。

指標的另一個方面再次由普羅米修斯（Prometheus）普羅德（Prometheus）推廣，是指標的pull-versus-push方法。 要點是，在一個雲本地設置中，人們處理了短暫壽命的容器或功能，發現是更好的集中式的，並且中央實體會收集指標（或者像Prometheus的人們所說的那樣，從技術上講，它意味著將HTTP API端口端口端口召集而不是將應用程序推動到中央度量中。

在確定要考慮哪些指標的指標時，社區提出了幾種方法和建議。以下是最值得注意的：

The utilization, saturation, errors (USE) method - 由Brendan Gregg創建（請參閱https://www.brendangregg.com/usemethod.html），此方法非常適合低級度量標準，並且經常用於性能工程和根本原因分析。 *The rate, errors, duration (RED) method* - 由湯姆·威爾基（Tom Wilkie）創建（請參閱http：//mng.bz/pzj9），此方法重點關注所服務的請求數，失敗請求的數量以及請求需要多長時間。這對於應用程序級別的情況很有用。 *The four golden signals of monitoring* - 由Google工程師建立（請參閱http://mng.bz/jgj0），這四個信號是延遲，流量，錯誤和飽和度，就像紅色一樣，使用的是混合使用。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

從消費的角度來看，我們可以說經常（在儀表板中）繪製指標，或在監視和分析的背景下用作警報的觸發器。但是指標也很有用，實際上是用作程序的輸入。例如，Kubernetes中的水平POD Autoscaler（HPA）可能會根據某些自定義度量標準（例如服務處理的請求數量）向上或向下擴展副本的數量。

2.4.1 Instrumentation

讓我們看看如何將迴聲服務擴展到發射指標。列表2.7顯示了帶有指標的Echo服務工具。確切地說，我們正在使用Prometheus GO庫進行儀器。請注意，列表2.7僅顯示相關的片段，而不是完整的GO代碼，該片段可通過代碼/CH02/CH02/MENRICS/MAIN.GO（http://mng.bz/wvwq）中的代碼回購提供。

清單2.7示例GO程序，儀器發出Prometheus指標

```
func handleEcho(w http.ResponseWriter, r *http.Request) {
    message := r.URL.Query().Get("message")
    log.WithFields(log.Fields{
        "service": "echo",
    }).Info("Got input ", message)
    if rand.Intn(100) > 90 {
        log.WithFields(log.Fields{
            "service": "echo",
        }).Error("Something really bad happened :(")
       invokes.WithLabelValues(strconv.Itoa(http.StatusInternalServerError)).Inc()
        w.WriteHeader(500)
        return
    }
    invokes.WithLabelValues(strconv.Itoa(http.StatusOK)).Inc()
    fmt.Fprintf(w, message)
}
var (
    registry = prometheus.NewRegistry()
    invokes  = promauto.NewCounterVec(
        prometheus.CounterOpts{
            Name: "echo_total",
            Help: "Total invocations of the echo service endpoint.",
        },
        []string{"http_status_code"},
    )
)
func main() {
    rand.Seed(time.Now().UnixNano())
    registry.MustRegister(invokes)
    log.SetFormatter(&log.JSONFormatter{})
    http.HandleFunc("/echo", handleEcho)
    http.Handle("/metrics", promhttp.HandlerFor(
        registry,
        promhttp.HandlerOpts{
            EnableOpenMetrics: true,
```

Increments our metric with the context-specific label carrying the HTTP status code

Sets up the Prometheus registry

Defines our metric

Registers our metric

Defines the /metrics endpoint handler

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

））log.Fatal（http.listenandserve（“：4242”，nil））}}}

在這一點上，您知道如何啟動Echo服務來發出指標，但是這些指標如何到達可以消費它們的目的地？ 讓我們看看下一節。

2.4.2 Telemetry

現在，我們可以在本地執行Echo服務並揭示Pro-Metheus指標：

$。 /echosvc

我們可以使用Prometheus（http：// prometheus .io/）刮擦迴聲服務的指標，這又可以在Grafana中充當數據源。要設置目標，請使用下一個列表中顯示的Docker撰寫清單文件（引用列表2.9）來設置Prometheus和Grafana。

列表ING 2.8使用docker-compose.yaml站起來站起來 和刮擦

版本：“ 3.5”網絡：PROM：服務：Prometheus：圖片：Prom/Prometheus：最新端口： - “ 9090：9090”卷： - ./prom-config.yaml：//etc/prometheus/prometheus/prometheus/prometheus.yml networks：-promegrafana：-promegrafana： - prom grafana： - prom grafana：-grafana/grafana/grafana/grafana：3000：3000：3000：33000：33000：330：330：33 000：33 000：33 000：“”最新：

就像我們在日誌上看到的那樣，我們還需要為指標配置遙測代理。在這種情況下，我們通過定義Prometheus scrape配置來做到這一點，如以下列表中的Prom- config.yaml文件所示。

列表2.9 Prometheus配置文件Prom-Config.yaml

全局：scrape_interval：5s estaruitu
on_interval：5s

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
external_labels：monitor：'lab'scrape_configs： - job_name：'p
rometheus'static_configs： - 目標：['localhost：9090'] - job_na
me：'lab'static_configs： - targets： - targets： -
```

接下來，使用http：// prometheus：9090/作為URL，添加Prometheus作為Grafana中的數據源，如圖2.3所示。

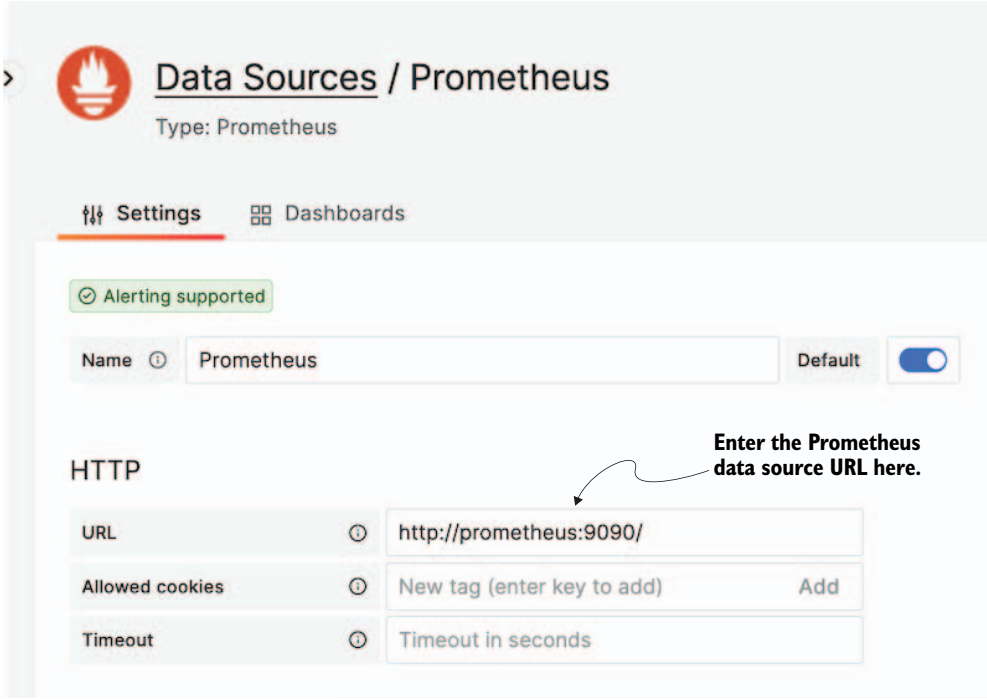


圖2.3在Grafana中添加和配置Prometheus數據源

現在，您可以再次使用“ Explore”選項卡來消耗ECHO服務的指標。選擇Prometheus作為數據源，如果輸入以下Promql（http://mng.bz/ qrgw）查詢，則應該看到與圖2.4中所示的內容相似，該查詢代表我們服務的錯誤率：

```
sum（速率（echo_total {http_status_code =“ 500”} [5m]）） /sum（r
ese（echo_total [5m]）） * 100
```

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

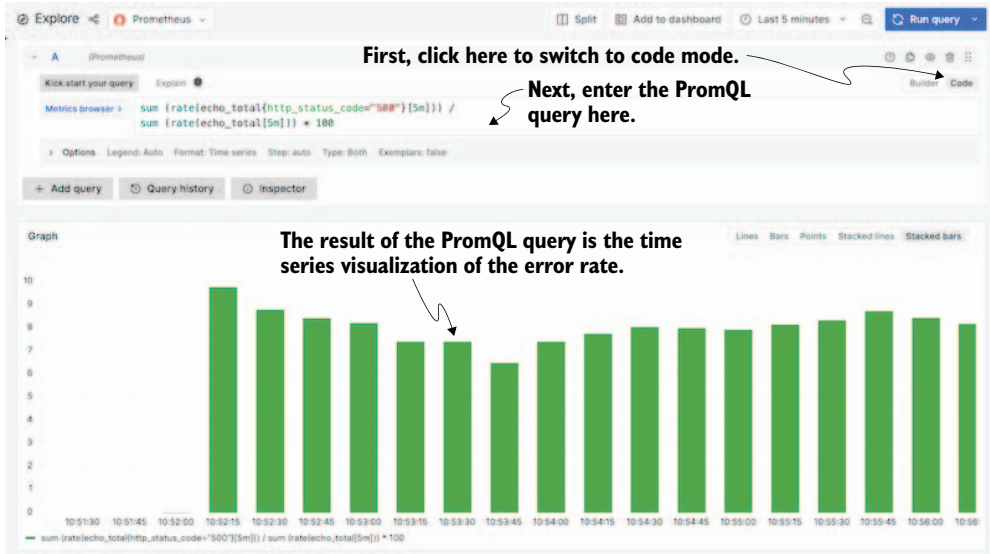


圖2.4與Prometheus探索指標

前面的promql查詢是捕獲錯誤率的規範方法。通過將錯誤（HTTP狀態代碼500）劃分為所有數據點，您就可以靈活但完整的捕獲錯誤。在這種情況下，錯誤率是一個較低的單數率，平均應該為10%，因為這是我們在列表2.7中配置的服務。接下來，讓我們討論指標的利弊。

2.4.3 Costs and benefits

現在，我們仔細研究了指標信號類型的成本和收益。表2.2顯示了與指標相關的成本的概述。

表2.2指標成本概述

	Overhead	Considerations
Instrumentation	Auto-instrumentation, if possible	Increasingly supported in programming languages
Agent	Prometheus, OpenTelemetry collector, and OpenMetrics as emerging standard	What to collect and autopruning
Destination	Cardinality, long-term storage, and federation	What to ingest and data temperature for retention

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

指標的儀器成本

關於用指標的儀器代碼成本，我們可以區分兩種情況：

使用自動啟示通常需要很少的精力，例如包括圖書館或配置它（例如，與Prometheus [http://mng.bz/7dr9]; Open-Telemetry [https://opentelemetry.io.io.io/docs/instrumentation/]; Open-Telemetry [https://opentelemetry.io.io.io.io.io.io and instrumentation/]; 手動儀器比自動啟示需要更多的努力，但還提供了更多的控制，您要在哪種管理方面公開哪些指標。通常，這些儀器應該是一次性的，對於一類會議（業務級指標），這種儀器是唯一可行的儀器。

該代碼的可讀性及其可維護性可能會帶有手動儀器。

現在，讓我們從開發人員的角度看一下儀器成本：列表2.1中的原始代碼具有24行代碼，並且列表列表2.3中使用指標的代碼具有56行代碼總體。為了隔離指標 - 導致儀器成本與總（日誌+指標）儀器成本相關，我們需要從由原木引起的儀器中減去8行代碼。然後，我們達到48行代碼的值。

將這兩個LOC數量列為一起，我們可以計算B2I比率，並以2為2的指標到達。該B2i值略高於我們使用的日誌（1.33），但仍然不令人擔憂。

如果您想了解有關指標工具的良好實踐的更多信息，我建議您仔細閱讀以下資源：

CNCFSIG儀器Kubernetes（http：//mng.bz/mver）的指南。 CNCF標籤可觀察性白皮書輸入帶有開發人員指南的有關如何實施新指標的指南（https://github.com/cncf/tag-observability/ esporce/18）。

指標的遙測成本

指標的代理的考慮因素與我們討論的日誌相似。此外，還存在圍繞基數的挑戰，也就是說，在度量方面可能有可能增加成本的增加的數量。

指標代理的示例包括以下內容：

CNCF Prometheus, especially in agent mode (https://prometheus.io/blog/ 2021/11/16/agent/) The CNCF OpenTelemetry collector (https://opentelemetry.io/docs/collector/) and, more specifically, collectors from OpenTelemetry distributions, such as the AWS Distro for OpenTelemetry收藏家（http://mng.bz/5wbb）

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

指標的目的地成本

指標最具挑戰性的方面之一是我們所說的*cardinality explosion*，我們將在第5章中詳細討論。這在高層次上意味著什麼？好吧，在許多情況下，度量的維度僅具有一組可能的值。例如，http狀態代碼的數量（https://httpstatuses.com/）可能會被限制在幾個：

myApp_http_requests_total {代碼=“ 200”，處理程序=“/helloworld”} - > 42 myApp_http_re
quests_total {code {code =

這不會引起任何問題，因為從存儲和查詢角度來看，時間序列數據庫可以很好地處理它。但是，如果您決定將用戶ID編碼為如下標籤怎麼辦？

myApp_http_requests_total {code =“ 200”，處理程序=“/ex1”，uid =“ 9876”} - > 567

恭喜！ 您剛剛引入了基數爆炸的潛力。為什麼？好吧，成像您的服務變得流行，您最終不得不處理數万甚至數百萬用戶。

UID標籤的值可能與用戶一樣多，對於時間序列數據庫來說，在存儲和查詢性能方面都非常昂貴（如果可能的話，通常會看到您的查詢在這種情況下會變得較慢）。實際上只有一個解決方案：保持尺寸控制。

目的地的另一個成本來源是有很多來源。假設您有一群Kubernetes集群，並想集中查詢指標。您需要考慮如何進行長期存儲和聯合會（http://mng.bz/6ddo）的指標，這是我們將在第5章中更詳細地詳細介紹的。

指標的好處

按照指標的本質，它們的數值可以實現各種自動化。例如，您不必通過查看儀表板來手動跟踪度量標準，而是定義條件（例如，“至少超過此值3分鐘”）並在情況下獲取通知。 這種自動化通常稱為*alerting*。

然後，建立基線的地方很廣。 （“ normally”的度量如何 - 範圍，最大和最小的？同樣，我們在本書後面更詳細地返回此主題。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

消費指標的流行目的地包括以下內容：

Frontends, such as low-level/debug offerings, including the Prometheus UI or PromLens (https://github.com/promlabs/promlens-public) Universal data sources frontends, such as Grafana (https://github.com/ grafana/grafana) Commercial or integrated offerings, including CloudWatch, New Relic, Data- dog, and so on 對象存儲，例如S3存儲桶，用於以後分析

這樣，我們已經達到了指標成本和收益討論的終結，因此讓我們繼續使用指標用例以進行觀察。

2.4.4 Observability with metrics

我們該如何處理指標，我們剛剛看到的與可觀察性有關？在DevOps或SRE角色中，我們可以執行以下操作：

Establish baselines for signals, based on metrics - further，您可以通過警報定義異常和期望，並在超過某些值的情況下獲取通知（這更像是指標的傳統監視用例，因為這些問題已經在預先知道的情況下知道）。

Track metrics over time and study histograms - 這可以幫助能力計劃或可伸縮性對話。

Define operational and tactical goals - 通常，這稱為*service level objective* (slo)（例如，“在一周的時間內，少於12%的錯誤服務”）。為了評估是否已達到目標，人們使用一個測量值（通常稱為*ser- vice level indicator*（SLI）），例如錯誤率（錯誤比率與總體請求的比率）。第10章致力於SLO，因此，如果您想了解更多信息，請隨時跳過。

現在，讓我們看一下如何處理痕跡。

2.5 Traces

當您聽到*traces*一詞時，這可能意味著您必須首先消除事物的歧義 - 也就是說，請務必詢問某人在參考痕跡時的真正含義。通常，我們可以區分兩種情況：本地或分佈式：

在本地“痕跡”的情況下，更廣泛使用和建立的術語是*pro- files*。我們使用*tracing*和*sampling*（關於配置文件）互換。這與在單個計算機的上下文中捕獲過程的某些特徵有關，例如內核級事件或用戶空間功能呼叫以及執行時間和資源使用情況，包括CPU和MEM-ORY，例如在使用Strace查看SyScalls的情況下。如果您走得更遠

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

並在一段時間內進行此抽樣，您最終將獲得所謂的*continuous profiling*（cp;

分佈式軌跡用於在分佈式系統（例如容器化的微服務）的上下文中沿請求路徑的多個微服務傳播上下文。 請求完成後，您可以將單獨的服務執行拼接在一起，經常在瀑布視圖中可視化或顯示服務調用的圖形視圖，通常稱為*service map*。

請注意術語*waterfall visualization*既不是新的，也不是分佈式跟踪的獨特之處。如果您查看可用的瀏覽器開發人員工具，您會發現它使用類似的表示形式顯示瀏覽器花費多長時間（http://mng.bz/o1od），例如dom elee-ments處理或與網絡相關的操作，例如下載樣式文件或圖像。

現在，讓我們高級查看分佈式痕跡，但是如果您對詳細信息感興趣，則可以跳到第8章，我們將深入研究該主題。簡而言之，分佈式跟踪是關於捕獲和可視化請求如何通過微服務移動，可能跨不同的機器移動，並涉及諸如HTTP或GRPC之類的不同協議。在這種情況下，您經常會遇到兩個術語

Trace - 代表請求通過參與的微服務進行端到端處理。痕跡可以具有多個跨度。
Span - 沿請求路徑中的一個特定服務中的執行，將根跨度作為跟踪和子女的第一個跨度作為子分類（嵌套）。

但是這是如何工作的呢？當請求進入系統時，會生成唯一的標識符（跟踪ID）。然後，根據協議，用於跨越http或grpc的服務，此跟踪ID以及一些有效載荷將發送給孩子們。 這就是我們在網絡工作設置中的過程中所說的*context propagation*。

分佈式的跟踪解決方案抽象瞭如何實現傳播的技術細節，並照顧將所有跨度拼接在一起，並將它們交付到中心位置，然後您可以查看，查詢和分析痕跡。

將分佈式痕跡與日誌提供的相結合會產生一些有趣的見解：

您可以查找活動的跟踪上下文，快速查明有趣的來源。 如果將日誌事件附加到跨度，則可以直接從水秋/定時可視化到組件的相關日誌消息。 這

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

意味著您不必搜索日誌消息，並且可以使用更少的索引開銷來逃脫。

現在，讓我們看看如何將這些知識付諸實踐，從在我們的示例應用程序中發出跨度開始。

2.5.1 Instrumentation

列表2.10顯示了在帶有痕蹟的儀器中，我們的迴聲服務的相關行。請注意，我們在這裡使用OpenTelemetry來發出跨度，我們將在第4章中進行更詳細的討論。

清單2.10示例GO程序，儀器發出跡線

```
func handleEcho(w http.ResponseWriter, r *http.Request) {
    message := r.URL.Query().Get("message")
    log.WithFields(log.Fields{
        "service": "echo",
    }).Info("Got input ", message)
    ctx, cancel := context.WithCancel(context.Background())
    defer cancel()
    tr := otel.Tracer("http-api")
    _, span := tr.Start(ctx, "echo")
    defer span.End()
    if rand.Intn(100) > 90 {
        log.WithFields(log.Fields{
            "service": "echo",
        }).Error("Something really bad happened :(")
        invokes.WithLabelValues(
            strconv.Itoa(http.StatusInternalServerError)).Inc()
        span.SetAttributes(
            attribute.Key("http-status-code").String(
                strconv.Itoa(http.StatusInternalServerError)))
        w.WriteHeader(500)
        return
    }
    invokes.WithLabelValues(
        strconv.Itoa(http.StatusOK)).Inc()
    span.SetAttributes(
        attribute.Key("http-status-code").String(
            strconv.Itoa(http.StatusOK)))
    fmt.Fprintf(w, message)
}
```

Starts a new span when invoked

Sets the error code as an attribute (signal metadata) of the span

Sets success as an attribute (signal metadata) of the span

func main（）{rand.seed（time.now（）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

http.handle（“//指標”，promhttp.handlerfor（註冊表，promhttp.handleropts {enableOpenMetric
s：true，}，））tp，err：= tracerProvider（= tracerProvider（ log.fatal（err）} otel.setTracerPro
vider（tp）log.fatal（http.listenandserve（“：4242”，nil））}}

設置Jaeger的OpenTelem
etry Config作為跟踪提
供商

現在，讓我們開始服務並生成一些跨度。

2.5.2 Telemetry

現在，我們可以在本地執行Echo服務，並像這樣攝入痕跡：

$。 /echosvc

我們可以使用Jaeger（https：//www.jaegertracing .io/）攝入迴聲服務的跟踪，然後我們將其用作Grafana的數據源。

要設置目的地，請使用docker撰寫的清單中所示的清單文件來設置Jaeger和Grafana。

使用Docker-Compose.yaml列表2.11站起來

版本：“ 3.5”網絡：Jaeger：服務：Jaeger：圖片：Jaeger
Tracing/All -In -In -In -In -In -In -In -In -In -In -In -In -In -in -
In -In -in -in -in -In -in -in -in -in -of：“ 6831：6831/udp”
- “ 16686：16686：16686” - “ 14268：14268：14268”
網絡：

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

接下來，使用http：// jaeger：16686作為URL添加Jaeger作為Grafana中的數據源。

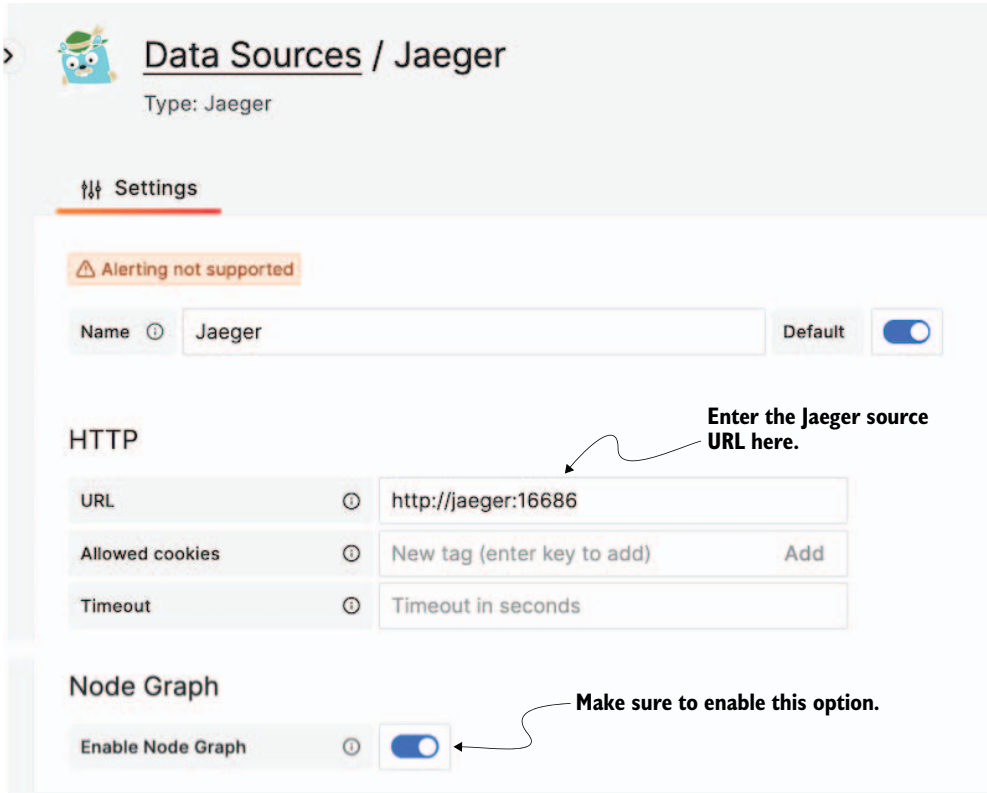


圖2.5在Grafana中添加和配置Jaeger數據源

現在，您可以再次使用“探索”選項卡。選擇Jaeger作為數據源；切換到搜索選項卡（在查詢類型字段中）；選擇echosvc;而且，如果您通過單擊跟踪ID選擇其中一個跟踪，則應該看到與圖2.6中所示的內容（在跟踪視圖中，右側）。 在下一個小節中，我們將查看痕蹟的成本和收益。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

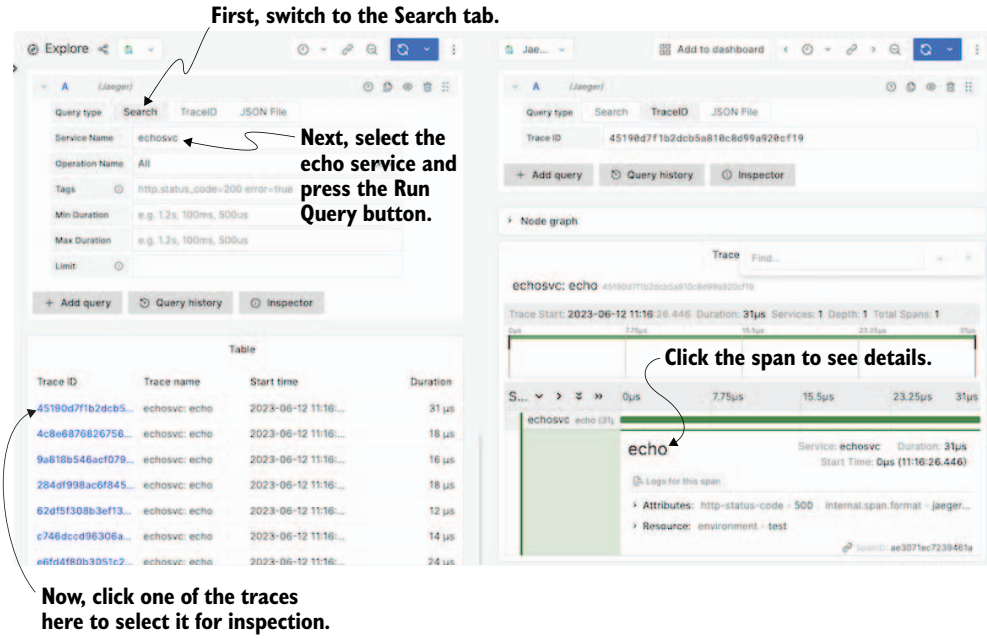


圖2.6使用Jaeger探索痕跡：詳細的跟踪視圖

2.5.3 Costs and benefits

讓我們仔細研究痕跡信號類型的成本和收益。表2.3列出了痕蹟的成本概述。

表2.3跡線成本概述

	Overhead	Considerations
Instrumentation	Some effort and auto-instrumentation	OpenTelemetry uptake and legacy
Agent	Sampling strategy and latency overhead	Use OpenTelemetry collector
Destination	Dedicated storage layer and query and analytics	Lots of commercial offerings and some OSS (e.g., Tempo and Jaeger)

軌蹟的儀器成本

從開發人員的角度來看，跡線的儀器成本如下：列表2.1中的原始代碼具有24行代碼，並且列表2.10中帶有跡線的代碼總體上有99行代碼。

現在，為了將引起跡線的儀器成本隔離為總（日誌+指標+軌跡）儀器成本，我們需要從被原木引起的儀器以及24個LOC中減去8個LOC，該LOC儀器儀器得到了貢獻。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

我們僅用於痕跡儀表儀的值為67個位置。將兩個LOC數在一起，我們可以計算B2I比率並達到2.79的值，僅是原木的兩倍（1.33），並且略高於單獨的指標儀器（2）。

痕蹟的遙測成本

一個主要的考慮因素（此主題也可能適合以前的“儀器”部分，但我選擇在這裡談論它）是痕蹟的採樣。我們將在第8章中更詳細地討論此主題。

在分佈式跟踪的背景下，採樣意味著捕獲或丟棄源發射的特定跟踪。我們通常區分

Head-based (sometimes called up-front) sampling - 代理丟棄了固定的痕跡。 當您專注於網絡成本時，這是確定性的，並且有所幫助；但是，您可能會錯過有趣的痕跡（異常，錯誤等）。 *Tail-based sampling* - 這個想法是將採樣決定推遲到何時完成。但是，您怎麼知道踪跡何時完成（Stragglers）？它通常要求代理商保持更多狀態，並可以導致更複雜的設置以及更高的資源使用情況（網絡+存儲成本），該設置與更簡單的前後採樣決策相關聯。

除了我們已經看到日誌和指標的運行時開銷（資源消耗）外，在痕蹟的情況下，通信路徑中還有一個很小的延遲開銷。這是由於以下事實：上下文傳播耗費了來源和電線上的時間和帶寬。

跡線的目的地成本

一般使用數據存儲中的分佈式跟踪解決方案使您可以處理稀疏的時間戳數據。通常，需要索引痕跡，以便您可以搜索或過濾特定的服務或可以按持續時間訂購。這通常意味著存儲（後端）需要能夠為某些跨度特徵（例如服務名稱，標籤和其他元數據（例如HTTP方法）索引。

野外有不同的方法來實現痕量存儲和檢索，可以按以下方式分組：

專用數據存儲，例如ClickHouse（https://clickhouse.com/）通用purpose nosql數據存儲，例如cassandra，hbase或Elastic-search/opensearch，這些數據存儲，這些數據存儲，這些數據存儲，與域特定的模式和邏輯一起使用，以適應pockertiqu offate cousptate couse outier ofisier out-based-base-based-based-based-based-based-based-niger s3 buckets in Ingers in Ingef the Ingrist-Ingrigest-ef.與Grafana節奏一樣

追踪中後端的成本（例如指標）通常屬於三個桶中的一個或多個：攝入（插入有效載荷的痕跡），處理（有時稱為*scanning*或*traces retrieved*）和長期存儲成本（保留策略）。我們將在第5章中詳細討論後端。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

實際的總成本取決於用例和您的政策，因此不可能提出通常適用的有意義的指導。以一種情況為例，您將所有開發人員都能訪問分佈式跟踪解決方案，而只有隻有操作或電話就能從中受益的情況。

痕蹟的好處

在微服務設置中，您不應將分佈式跟踪視為可選的。您可以獲得服務的動態高級呼叫圖/服務映射，內容涉及您的服務實際交流方式，並且可以鑽入請求路徑，而無需通過日誌事件手動（基於時間戳記）的服務呼叫。

流行的分佈式跟踪前端包括以下內容：

Zipkin（https://zipkin.io/）內置UIjaeger（https://www.jaegertracing.io/）indord-indook indend-indosent undendent in uii內置UI。

有一系列的商業（一些SaaS和一些OSS）產品，可以將後端/存儲和前端結合在一起，涵蓋了其他信號類型的痕跡。我們將在第6章中深入研究該主題。這樣，我們完成了痕跡成本和收益的劃分。現在，讓我們看一下在可觀察性的情況下的痕蹟的用例。

2.5.4 Observability with traces

我們可以如何處理痕跡，我們剛剛看到的與可觀察性有關？在開發人員或QA/測試角色中，

開發人員可以使用跨度來識別慢速服務並根據此信息優化請求拉登。 質量保證員工可以使用痕跡來解決DEV/測試或預生產的環境，以捕獲性能回歸。 開發人員和測試人員受益於調試情況下的分佈式痕跡，例如，當客戶報告問題和實時（生產）工作負載時。

在Devops或SRE角色中，

基於警報或報告問題的客戶，電話通常會發現他們擔任第一響應者的角色。他們需要能夠搜索並識別可能是問題原因的服務或組件。 對於SRE，為了評估整體或組件健康，跟踪聚集物（例如P90或P95值）用作指標或輸入數據，以提出有關開發人員的建議。

這樣，我們已經完成了信號類型和用例的調查。接下來，讓我們繼續進行一些通常適用的好實踐。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

2.6 Selecting signals

收集信號時，請確保考慮以下內容：

為什麼要捕獲此信號？從投資回報率的角度來看，信號是否產生可起作用的見解，還是有助於噪聲（更糟糕的是，在不提供任何好處的情況下，潛在的可能會導致網絡和與存儲相關的成本）？ 我應該存儲信號多長時間（保留期）？這主要取決於手頭的任務（例如，故障排除，呼叫或優化），但也可能有外部因素會影響決定保留多長時間的決定。例如，在金融行業中，您可能需要在一定時間內保留跨行動日誌，以便遵守支付卡行業數據安全標準（PCI DSS）。 對於數值信號，例如指標，“正常”值（或值範圍）是什麼？這通常用作基線。您可以考慮使用機器學習技術自動化建立基線的程序。在這種情況下，如果信號不在正常範圍內，則您的朋友是異常檢測（https：//github.com/aicoe/prometheus-anomaly-detector）。 當心信號ho積；僅僅因為您可能需要一天（沒有策略）可能需要它們，圍繞所有指標或日誌，可能會昂貴，並且會對ROI產生負面影響。 跨信號類型（例如Loki，Prometheus和Parca）始終如一的標籤。

通過對選擇信號的良好實踐和考慮因素的一般集合，我們已經達到了本章的結尾。

Summary

為了產生可行的見解，為了可觀察到性，我們依靠不同的遙測信號，最重要的是日誌，指標和痕跡。 您需要根據其實用性和成本來決定要收集，存儲和使用的信號。 日誌是帶有文本有效載荷的信號，該信號捕獲了通常由人類消耗的事件。 理想情況下，日誌應該是結構化的（例如，在JSON中表達）並帶有模式，但實際上，由於使用了許多格式和協議，情況可能會凌亂。 指標是數值信號，可用於自動化（例如，用於儀表板和警報）。 Prometheus格式已成為雲原生系統中的標準。 分佈式軌跡是捕獲和可視化請求如何通過微服務移動的信號，顯示服務持續時間和成功狀態。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

所有信號都帶有成本和收益，從儀器工作到遙測代理到目的地。這是您需要在ROI的背景下逐案（工作量）考慮的事情。 您已經看到了從示例應用程序的上下文中的儀器到通過代理到目的地的路由的儀器中的信號類型。 我們回顧了選擇哪些信號以及以給定的用例或任務（例如故障排除或優化）來揭示哪些信號的缺點和良好的實踐。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

3 Sources

This chapter covers

信號源的含義存在的來源類型以及何時選擇哪種源從選擇任務的正確來源獲得可行的見解如何處理您擁有的代碼以及如何以及何時進行儀器理解供應鏈可見性

從可觀察的角度來看，觀察系統（SUO）由*sources*的集合組成。這些來源發出的遙測信號，例如日誌，會議和痕跡，我們在第2章中討論了這些信號，哪些構成了人類和機器的可行見解的基礎。

在雲本地系統的背景下，事物往往會快速移動 - 新開源項目啟動，提供商為其產品添加功能，並且云提供商推出了新服務。正是由於這個原因（即，在雲本地系統中找到的迭代速度），我們不會詳細介紹特定產品，其API或它們的功能。什麼是重要的，我想集中的東西

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

ON是使您能夠了解哪些來源可供您使用，以及您可能希望選擇哪些來源。

在本章中，我們將使用簡單且廣泛使用的Trichot-Omy對資源進行分類：計算，存儲和網絡。在高水平上，這看起來如下：

Compute—Runtime environments, such as a virtual machine (VM), including those that are general purpose and language specific, like the Java VM, as well as the compute unit, like a microservice running in a container or a function supported by AWS Lambda
Storage—Relational databases (PostgreSQL, MySQL, etc.), NoSQL data stores (Redis, mongodb，dynamodb等），文件系統（本地和網絡，如nfs）以及對象存儲（例如S3）*Network* - NETWork Interfaces;網關；負載平衡器；虛擬專用網絡（VPN）；以及整個孤立的雲提供商網絡，例如虛擬私有云（VPC）

在本章中，我們將涵蓋各種技術。 目標不是詳細介紹每個源的每種信號類型，而是為您提供概述和一般性的想法以及有關在何處了解每個源的信息的信息。

3.1 Selecting sources

請記住，可觀察性的目的是獲得可行的見解，首先要了解要考慮哪些來源和信號。從好的方面來說，我們選擇的每個來源都可以直接使用或與其他信號相關聯。另一方面，我們考慮的每個來源都需要投資：從儀器成本到攝入信號的代理到目的地的存儲成本。

在來源的背景下要考慮的一件事是哪種來源與手頭的任務或角色有關。 選擇信號源的有效策略是按照或用例基礎向後工作。例如，手頭的任務可能是基礎架構操作的呼叫，您可能主要關心所有CPU，RAM和文件系統利用率。另一個示例可能是改善API請求作為開發人員的延遲，您需要可見性與網絡相關的來源。

您不僅必須考慮選擇哪些來源，還必須考慮哪些信號來自您使用的來源以及在何種程度上。 例如，考慮使用無服務器的Kubernetes產品的情況，在其中為您管理控制平面和數據平面。佔用ETCD，在此上下文中，Kubernetes控制平面保持其狀態：鑑於您無法訪問它，因此您無法更改設置中的任何內容（例如，重新平衡或比例比例等）。由於這種不變性，詳細的指標並不那麼有價值，而整體健康指標則為。

但是，您需要考慮一些權衡；例如，雖然您可能無法控制自動化的發生方式，但可見度很高

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

發生可以幫助故障排除。至少，它應該幫助您確定問題是在提供商方面還是在您的應用程序中。

我們可以區分您擁有的資源（因此，您可以自由樂器）以及您不擁有的來源。通常，您不擁有的來源（因此，也無法樂器）佔據了大多數來源。從計算運行時到存儲時間和網絡，再到您使用並依賴於您自己的代碼的庫以及其他服務。

好的，那是足夠的理論。現在，讓我們從相關的來源開始，讓我們弄髒。

3.2 Compute-related sources

本節審查了與計算相關的來源。我們將查看不同類型的計算，其主要特徵來自觀察性角度，以及哪些信號與哪些角色相關。

3.2.1 Basics

在雲本地環境的背景下，我們通常在計算時面臨多種選擇。 最終，我們關心可以交付給業務利益相關者的功能。因此，在理想的世界中，我們將能夠專注於企業核心的應用程序級別，並將繁重的工作（例如，安全補丁和操作系統升級，通常不是業務的核心）外包給我們選擇的服務提供商。

但是，有時候，我們想要或必須擁有比我們可能想要的更大部分的充分理由 - 從合規性要求到控制計算的某些方面的需求（例如GPU-或I/O相關）。在您的情況下，無論基礎架構和應用程序級別之間的界限何在，對責任（例如，擁有升級或安全方面）的良好理解是成功進行雲本地操作的基礎。

根據您的角色（操作，開發人員，建築師，發布工程師等），您可能會以不同的方式查看計算。一些常見的包括以下內容：

部署單位（VM，容器或功能）以服務器為中心與無服務器計算單片應用程序與以服務為導向的應用程序（微服務）

圖3.1顯示了使用一些常見的雲本地示例設置的計算頻譜。每個設置中的未陰影框代表相應的基礎架構共同組件和應用程序級組件的陰影框。

注意實際上，在任何給定的生產環境中，您都會遇到計算的組合。這可能是由於技術原因（例如，轉移到雲的遷移以及服務的容器化），也可能是由於需要集成不同的檔案的商業原因，例如收購或合併。 不管是什麼原因，這意味著您幾乎

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

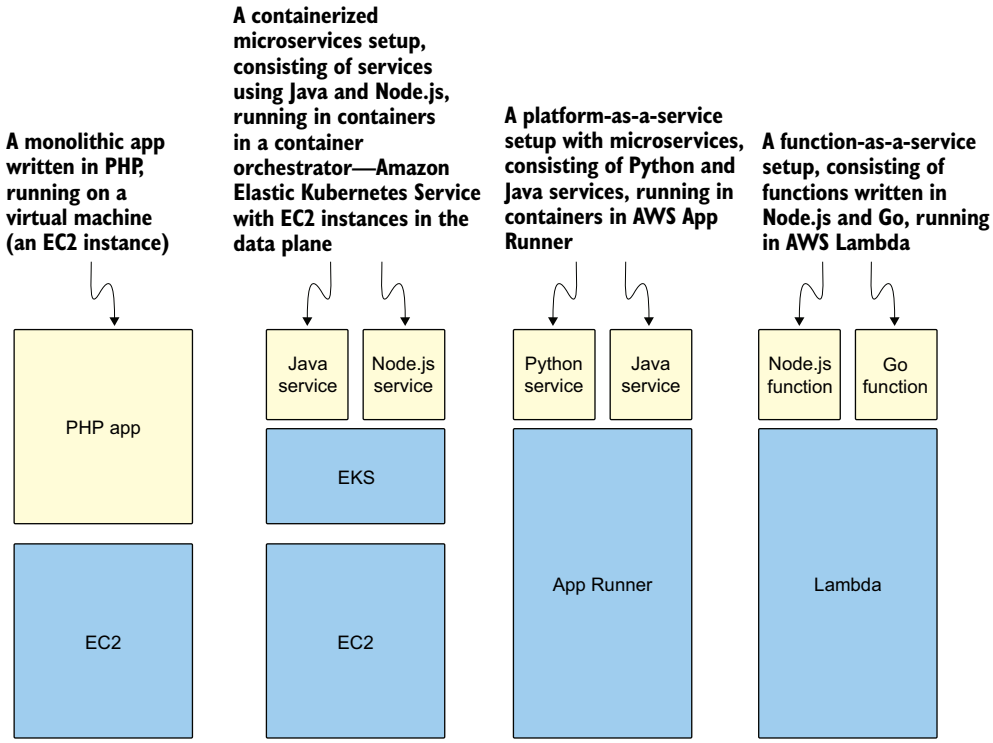


圖3.1計算，基礎架構和應用程序級別

始終必須準備好處理各種來源，看起來像是一種好的工具或方法，例如監視整體可能不適合容器。

表3.1概述了跨計算相關源的信號的覆蓋範圍。

表3.1與計算相關來源的可觀察性準備

Signal	Coverage	Example sources
Logs	All sources	EC2 instance, Docker daemon, and Lambda function
Metrics	Many sources	EC2 instance and Kubernetes metrics pipeline
Traces	Some sources	Node.js function in Lambda

現在，讓我們仔細研究與信號有關的具體計算相關來源及其特徵。 VM是一種已建立且廣泛使用的方法，可以與基礎（物理）機器上的其他租戶進行某些隔離保證。 還有特定於語言的VM，例如

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Java虛擬機（JVM）。 JVM代表一個豐富的信號源，暴露了日誌（https://openjdk.java.net/jeps/158）以及許多指標，從heap mem- ory用法到線程，您都有許多工具，您可以使用許多工具來監視JVM performance（https：https://sematext.com/gyidext.com/guides/gyide/gyiester/java/java/gyide/java/java-mja-mondiestav a。

如今，大多數Linux系統（裸機和VM）的共同點是它們正在使用SystemD，因此，讓我們查看某些方面通常適用於用戶可以訪問操作系統的系統。 SystemD不依賴文本日誌，而是具有自己的子系統，稱為JournalCtl，該子系統使用二進制格式存儲日誌，其優勢是訪問更快，並且存儲比文本更緊湊。但是，這是由於傳統基於文本的工具（例如，尾巴）不再起作用的缺點。

Journal Cttl工具支持豐富的接口，允許進行交互式和單次式查詢。例如，要顯示過去12小時的SystemD單元示例的日誌，

journalctl-“ 12小時前” -U示例。 e

在編程語言中記錄庫和框架時，廣泛採用的概念是*log levels*的概念，代表某個信息的嚴重或顆粒狀。例如，使用JournalCtl，您可以使用-priority = 3將輸出限制為錯誤（或更高優先級）的日誌。

現在，我們將VM作為基礎架構的一部分，並在其他計算相關來源（例如Kubernetes）的背景下涵蓋特定的字符。 對於您可以訪問VM的所有情況，即除了無服務器的環境外，通常都可以查看和查詢VM的系統日誌。讓我們來看看一個具體的示例，並在其中，是對日誌的專有解決方案：考慮一個像隨後的EC2實例（請注意，輸出已編輯以適合；相關部分被註釋）：

Details about the boot process of the VM, including supported hardware

```
$ aws ec2 get-console-output --instance-id i-0c87223xxxxx --output text
i-0c87223xxxxx     [    0.000000] Linux version 5.8.0-1041-aws
(buildd@lcy01-amd64-017) (gcc (Ubuntu 10.3.0-1ubuntu1~20.10) 10.3.0,
 GNU ld (GNU Binutils for Ubuntu) 2.35.1)
 #43-Ubuntu SMP Wed Jul 14 14:49:43 UTC 2021 (Ubuntu 5.8.0-1041.43-aws 5.8.18)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.8.0-1041-aws
  root=PARTUUID=fd4c3fdd-01 ro console=tty1 console=ttyS0
  nvme_core.io_timeout=4294967295 panic=-1
[    0.000000] KERNEL supported cpus:
...
Welcome to Ubuntu 20.10!
[    8.645187] systemd[1]: Set hostname to <ubuntu>.
[    8.650813] systemd[1]: Initializing machine ID from KVM UUID.
[    8.655158] systemd[1]: Installed transient /etc/machine-id file.
...
[  OK  ] Finished Execute cloud user/final scripts.
[  OK  ] Reached target Cloud-init target.
```

Here, systemd kicks in and takes over the VM initialization.

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

在基礎架構或平台操作員角色中，您會發現使用VM級信號，即用於故障排除或計劃。開發人員可能並不總是可以訪問該領域的來源，而JVM可能是一個潛在的例外。借助基礎知識，讓我們首先將注意力轉移到集裝箱環境上。

3.2.2 Containers

*Containers*是提供應用程序級依賴性管理的方便抽象。 也就是說，容器利用Linux內核功能，包括用於隔離的名稱空間（例如，過程，網絡堆棧或文件系統安裝點）和CGroups用於資源使用管理。容器以簡單可用的API包裝這些內核功能，使您可以啟動具有某些隔離保證的應用程序。它們非常適合將您的應用程序擁有的所有依賴項（從編程語言庫到操作系統級別的資源再到環境變量）中的所有依賴項，稱為*container image*，並在主機上運行。對於容器主機，適用VM或裸機實例的一般觀察結果，即通常取決於事件的日記帳，將JournalCtl作為其主要接口。

具體而言，對於容器，信號可以來自容器運行時，例如Docker或Containerd本身。另一方面，信號來自運行容器，即應用程序級信號。

對於日誌，不同的容器運行時間提供不同級別的支持：

> Docker守護程序（https://docs.docker.com/config/daemon/#read-the-logs）具有一個完整的解決方案，具體取決於主機環境，通常可以通過日記帳會議或在某個文件系統位置（例如under/var/log/log/log/log/）提供日誌。 在容器中，假設是使用它的環境（例如Kubernetes）照顧日誌管理。換句話說，它不支持自己的端到端解決方案。 但是，如果您將集裝箱作為系統單元運行，JournalD將捕獲輸出，為您提供一些基本信息。 kubernetes特定的容器運行時接口（CRI; http://mng.bz/ 0kaj）是平台不可思議的，並且不直接支持日誌記錄。取而代之的是，CRI將混凝土日誌支持定義為您使用的容器運行時，例如容器（請參見上一個）。

關於來自應用程序級別的容器日誌，我們必須退後一步。十二個因子應用程序考慮處理日誌（https://12factor.net/logs）作為事件流，鼓勵應用程序登錄stdout，並且不關心日誌路由。如今，您還可以找到這種“登錄到分”的立場。 Docker對容器日誌（http://mng.bz/ke8x）有充分的支持，使您能夠通過日誌驅動程序（例如JournalD，syslog和fluentd）以及雲提供商 - 特定的特定格式來管理日誌輸入和輸出。例如，要將日誌從Docker容器直接發送到Amazon CloudWatch，您將像以下方式配置日誌驅動程序：

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
docker run -log-driver = awslogs \ -log-opt awslogs-region = eu-west-1
\ - log-opt awslogs-awslogs-group = docker-test \ -log-opt awslogs-opt
awslogs-opt awslogs-opt awslogs-cereate-ewslogs-createe-group {v4
```

如果您想了解有關如何與容器一起使用日誌的更多信息，我建議您閱讀Phil Wilkins的*Logging in Action*（Manning，2022; https：//wwwww.manning.com/ books/logging-in-action，詳細介紹此主題。

對於指標，有一些可用於容器的選項：

cudvisor（http://mng.bz/9d4j）是一種相當流行且廣泛使用的解決方案，該解決方案與Prometheus一起使用。 telegraf（http://mng.bz/wzrg）支持Docker插件（https：//github .com/influxdata/telegraf/telegraf/tree/tree/master/master/master/plugins/inputs/inputs/docker），該插件會產生豐富的元素，包括集裝箱統計數據，記憶，記憶，CPU USAGE，CPU USAGE和WENTEKAL-ERNEWER-CLESSECE，和網絡相關的Metrics。 docker提供了一個本機（https://docs.docker.com/config/daemon/prometheus/）Prometheus集成，該集成僅限於運行時（也就是說，在寫作時不涵蓋應用程序）。

具體來說，如果您使用的是Docker守護程序（https://docs.docker.com/config/ daemon/），則可以選擇通過其套接字獲得（低級別）訪問指標，如以下代碼所示。 編輯捲曲命令的輸出以顯示一些示例統計信息；它是以JSON格式的，因此您可以通過使用JQ直接查詢它：

啟動一個長期運行的容器 - 在這種情況下是Nginx服務器。請注意，容器ID 82CEA007263254659534E47CFB1FB13B951F3426F6D91A1A1A16246143480F18A217718這裡，我們需要在下一步中查詢衡量指標。使用Curl查詢Unix套接字/var/run/docker.sock。 Docker守護程序有助於提供諸如啟動或停止容器之類的服務。我們需要提供我們想要的集裝箱ID，以便擁有指標的指標，或者至少一個唯一解決容器的零件（在我們的情況下，82CEA0072632）。

```
$ sudo docker run -d -rm nginx 82CEA0072632542659534E47CFB1FB1FB1FB1E3B951F3426F6D91A1624
6146143480F18A2177718 $ curl -unix-socket /var/run/docker.sock “http：//localhost/v1.40/containers/82cea0072632/stats？stream = false false”
1... “system_cpu_usage”：18084207160000000，“在線”：1，“rss”：1892352，“rss_huge”：0，
“total_active_anon”：1892352，...“total_active_file”：0，“total_cache”：0，“total_dirty”：t
otal_dirty“：0，}
```

表3.2顯示了一些示例指標（來自cadvisor），您通常可以從容器及其運行時找到這些指標。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

表3.2容器指標的示例

Category	Example metric in cAdvisor
CPU	`container_cpu_usage_seconds_total` (cumulative CPU time consumed) and `container_cpu_load_average_10s` (container CPU load average over the past 10 seconds)
Memory	`container_memory_max_usage_bytes` (maximum app memory usage) and `machine_memory_bytes` (memory installed on the host)
File system	`container_file_descriptors` (open file descriptors for the container)
Network	`container_network_receive_packets_total` (cumulative count of packets received)

作為開發人員，您希望可以訪問應用程序級日誌，並且您可能會發現某些指標有用，因為它可以根據資源使用或進行故障排除來優化您的應用程序。對於操作員，使用一種靈活的配置日誌路由的方法很重要（要求發送某些類型的日誌和/或保留的何處的要求可能會發生變化），並且能夠使用該指標來確保主機的可靠和有效使用。 現在，我們對容器運行時和應用程序基本信號源有了一個了解，讓我們繼續前進如何使用容器的規模使用容器：一個容器編排器，而Kubernetes是最受歡迎和最廣泛使用的示例。

3.2.3 Kubernetes

從可觀察性的角度來看，Kubernetes既高級又有限。為了理解這一說法，讓我們退後一步：一方面，Kubernetes定義了幾個API並暴露了一系列信號；另一方面，它假定您要照顧遙測和處理零件。 也就是說，如果沒有關於如何收集，路線，存儲和查詢Kubernetes提供的信號的全面策略，那麼幾乎沒有什麼可以使用的供您消費信號。

圖3.2提供了Kubernetes中可用的信號源的高級概述（https://kubernetes.io/docs/concepts/concepts/architecture/），包括在控制平面以及數據平面。

Kubernetes控制平面管理集群中所有資源的狀態，例如工作負載，名稱空間，節點等，並允許您查詢和更改管理中的資源狀態。在數據平面中，幾個競爭力共同啟動，健康檢查，報告和終止工作負載（包含在豆莢中）。請注意，雖然Kubernetes本身還帶來了工作負載的承諾，但低級位（從負載平衡器到網絡存儲）可以並且確實在包括API和格式在內的雲提供商之間確實有所不同。

日誌和指標信號類型已經在kubernetes中得到很好的支持，日誌（http://mng.bz/8rvk）在各個地方和社區中分佈更多。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

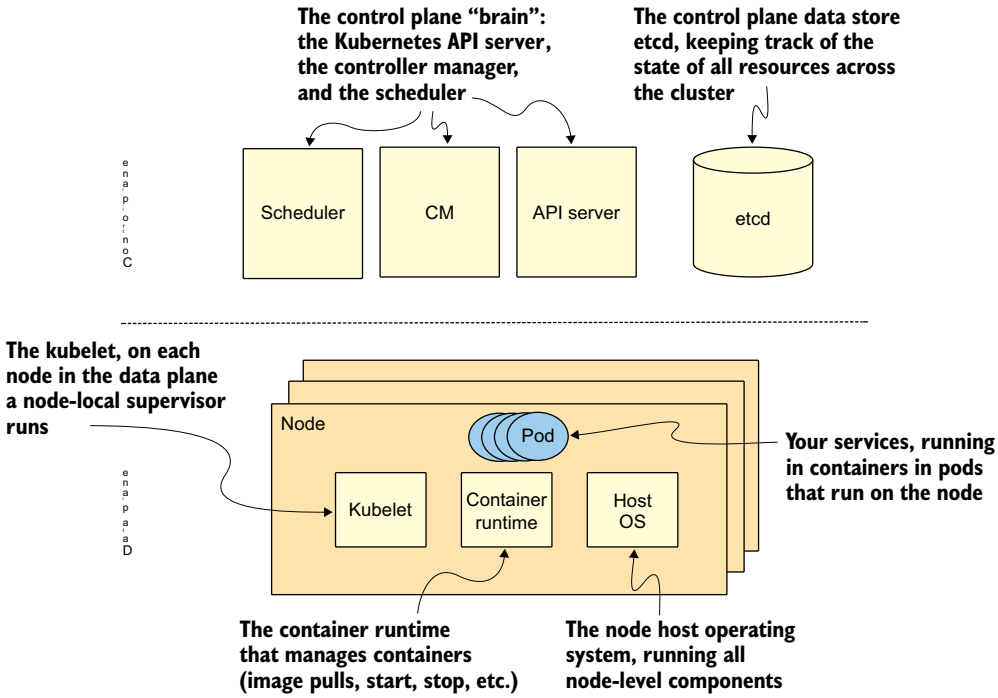


圖3.2 Kubernetes中的信號源

增加對軌蹟的支持。 讓我們仔細研究Kubernetes上下文中不同信號類型和來源。

關於Kubernetes中的日誌，我們可以在高級別上區分以下幾個：

從控制平面中的系統日誌（http://mng.bz/eqzq），包括API服務器，Kubelet和Container Runtimes。 您選擇的應用程序日誌，基於記錄策略（http://mng.bz/n2de）（您選擇的節點本地或基於中央搜索服務的路由，例如OpenSearch）。

根據所使用的容器運行時，可以使用不同的日誌管理功能，作為後備，您可以依賴於https://github.com/kubernetes/klog（kubernetes特定的Goggging Loggging庫）。 Kubernetes也不配備內置解決方案以進行日誌旋轉，因此這是另一個決定點，尤其是關於可移植性。

對於應用程序日誌，Kubernetes支持了一種簡單而有效的方法來捕獲我們已經從容器中的部分中知道的日誌：Stdout和STDERR。可以通過kubectl logs命令獲得轉換日誌； 但是，由於其限制（例如保留期），這些通常對於生產使用不可行。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

在開發或測試的背景下，Kubernetes提供的這些瞬時日誌非常有用，您可以從下面的示例中說出（請注意，已編輯了輸出）：

通過Kubectl日誌部署，我們要求在部署控制下所有POD的集體日誌。在我們的示例中，部署稱為前端，我們特別對容器HO11Y的日誌感興趣。最後，我們用頭限制輸出（顯示的日誌線）。

$ kubectl日誌部署/前端ho11y -f | head -4 { “ level” ： “ infor” ， “ msg” ： “使用http：//d...0” ， “ time” ， “ time” ： “ 2022-03-25T11：”

{ “ Level” ： “ info” ， “ msg” ： “使用http：//d....1” ， “時間” ： “ 2022-03-25T11：35：35：51Z” } { “ level” ： “ infor” ： “ info” ， “ msg” ， “ msg” ： “使用adot：4317” ，time：4317 “ ， ” time” ， “ 2022-25-03-25-25 time” ： “ 2022-25-111：” 35：33-35：35：351：351： “ 33 33 ” 。 { “ Level” ： “ Info” ， “ MSG” ： “ Init inst ... Done” ， “ Time” ： “ 2022-03-25T11：35：51Z” }

HO11Y容器中的工作量或應用程序級日誌

為了能夠查找工作負載的日誌，您需要製定一個策略或解決方案，該策略或解決方案能夠從各種豆莢中收集日誌並存儲在中心位置。假設您在數據平面中使用AWS Fargate的Amazon EKS，請看一下混凝土設置。在這種情況下，將日誌記錄（http://mng.bz/d4gy）設置縮小為使用kubernetes configmap資源作為IN接口，以配置流利的位作為ships logs logs logs logs logs logs logs的cloudWatch。看起來如下（輸出縮短）：

```
$ kubectl -n aws-observability \
          get configmap/aws-logging -o yaml
apiVersion: v1
kind: ConfigMap
data:
  output.conf: |
   [OUTPUT]
       Name cloudwatch_logs
       Match *
       region eu-west-1
      log_group_name ho11y-fluentbit
      log_stream_prefix fargate-
...
```

Gets the content of the resource aws-logging and renders it as YAML

The Fluent Bit configuration to route application logs to CloudWatch

Specifies the AWS region to use

Specifies the log group name to send the logs to (will be created if it doesn’t exist)

Adds fargate- to the beginning of each line (for easier filtering)

為了從命令行中消耗日誌，您可以使用AWS日誌獲取got-log-gest-log-group-name ho11y-fluentbit…命令或AWS控制台，如圖3.3所示。

有許多選項可供路由和消耗Kubernetes日誌，例如使用Elasticsearch和Kibana（http://mng.bz/lwkj）的流行端到端設置，或基於OpenSearch的變體，以及像Grafana Loki這樣的新人，我們將詳細介紹第5章中的詳細信息。

如圖3.2所示，在Kubernetes控制平面中，有幾個組合（例如API服務器或ETCD）產生日誌。 Kubernetes控制平面中日誌的另一個來源是節點級主管（Kubelet），通常可通過

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

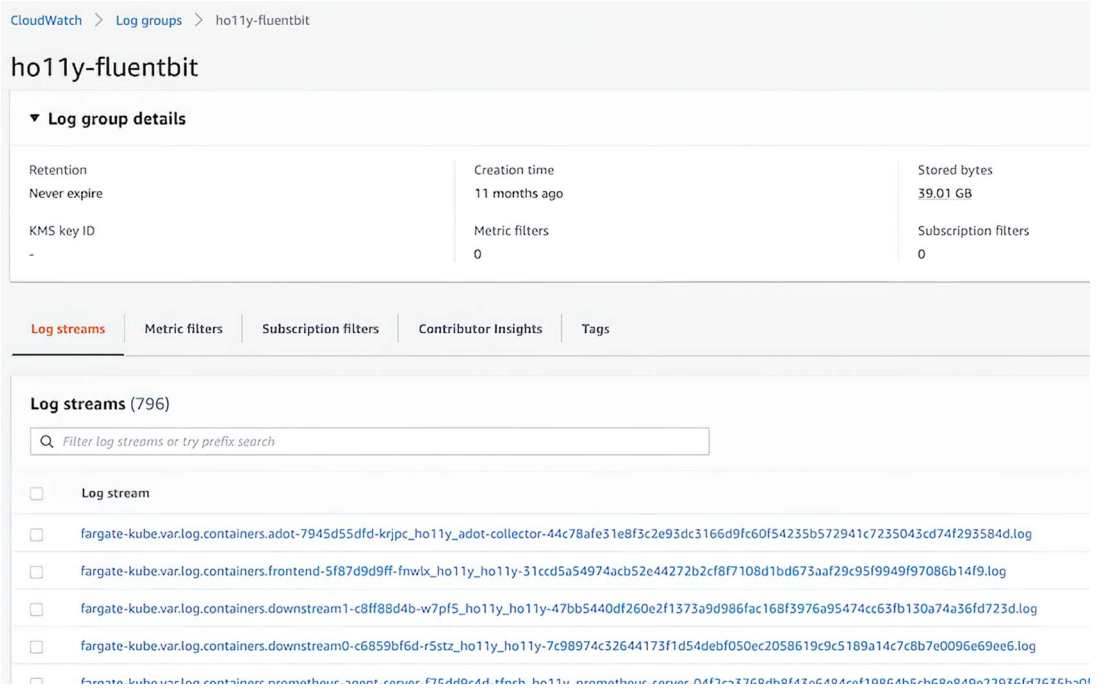


圖3.3在CloudWatch控制台中消費Kubernetes應用程序級日誌

雜誌。在這裡，考慮是使用託管服務還是親自運行Kubernetes，這是最重要的。

對於託管服務，控制平面日誌具有更大的信息性質，因為您通常無法訪問控制組件（與雲提供商支持團隊進行交互時可能有用）：

在Amazon EKS中，控制平面日誌（http://mng.bz/bmov）是可配置的，默認情況下是在CloudWatch中降落。 在Azure AKS中，可用於控制平面組件的幾個資源日誌（http://mng.bz/d1gd）。 在Google GKE中，您可以同時管理系統和應用程序日誌（http://mng.bz/ rwng），而前者包括控制平面組件。

審核日誌（http://mng.bz/v1zx）是Kubernetes中日誌的另一個來源，可捕獲用戶生成的活動以及與控制平面相關的活動。這些審核日誌提供了誰發布了每次更改以及集群身份驗證事件的詳細記錄，並且對於受監管的工作負載和環境通常是相關的，甚至是必需的。

對於指標，Kubernetes使用Prometheus博覽會格式（http：//mng .bz/x4v8）發射指標，該指標定義在指標管道（http://mng.bz/aopp）中，這是控制平面的一部分。與日誌相反，定義和管理

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

指標是集中的。您可以使用kubectl get -raw命令直接以Prometheus exposition格式查詢這些原始指標，以進行調試。以下代碼提供了一個示例（編輯為適合）：

訪問 /指標端點以獲取原始指標，用GREP過濾到具有APISER
VER的指標，並用頭限制為五行。

```
$ kubectl get --raw /metrics | grep --color=auto apiserver | head -5   # HELP apiserver_admission_controller_a
dmission_duration_seconds [ALPHA]   Admission controller latency histogram in seconds, identified by name
and broken out for each operation and API resource and type (validate   or admit). # 類型Apiserver_Admission
_controller_Admission_duration_seconds直方圖
```

```
Apiserver_Admission_controller_Admistration_duration_seconds_bucket {name =“證書”，
“ cuttranateApproval”，操作=“ update”，拒絕= false，type = validate =“ validate”，l
e =“ 0.005” 0.005“} 9.72” apiserver_admission_controller_admission_duration_seconds_b
ucket{ name="CertificateApproval",operation="UPDATE",rejected="false", type="validate",le=
"0.025"} 9.729568e+06 Apiserver_Admission_Controller_Admissim_duration_seconds_bucket
{name =“證書”“證書”，操作=“更新”，拒絕=“ false”，false =
```

此處顯示的指標，Apiserver_Admission_Controller_Admission_duration_seconds
，是控制平面組件（接收控制器）的直方圖類型。

軌跡是Kubernetes中最近添加的信號類型。它使用opentelemetry（http://mng.bz/zq6m）來捕獲和揭示API服務器的痕跡（在編寫時）。

3.2.4 Serverless compute

在本節中，我們在無服務器計算產品的背景下查看信號源。與親自管理服務器相比，這些無服務器的計算環境通常是從操作的角度來看的。 但是，在某些情況下，您無法使用它們。其中一些情況包括具有控制某些低級設置的要求，例如Linux內核模塊或硬件配置，出於性能原因。 另一類用例來自監管要求，如果沒有金融業或醫療保健行業工作量的某些證書。

但是，如果可以使用無服務器的計算來進行工作負載，則應強烈考慮。首先，讓我們討論無服務器計算的一般特徵以及環境對可觀察性方面的影響。

否（訪問）服務器

根據定義，*serverless*意味著您不與服務器打交道，也就是說，您不提供或維護服務器。自然地，這些服務器仍然存在於您的保護介紹中，但是現在，維護它們是他們的責任。一個含義是您通常沒有對服務器的SSH訪問權限，因此，上一節中與JournalCtl相關的許多語句不適用。相比之下，您正在處理

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

與API一起，您可以消耗日誌或指標和提供商控制台 - 允許在瀏覽器中查看相同的數量。

沒有補丁

從操作的角度來看，您要在無服務器計算提供商處採購維護的重要部分是安全補丁程序，通常是在操作系統和（語言）運行時級別上。這意味著，雖然您仍然想跟踪漏洞（尤其是在應用程序級別上），但除了測試外，您的末端需要更少的工作。

自動化

無服務器計算中的一個突出的期望是自動化。這意味著垂直（使用或多或少具有CPU/RAM的VM）或水平（添加或刪除相同的機器），以更改計算能力以適應更換負載。無服務器計算使您可以處理自動升級，而不必擔心縮放模型的詳細信息，從而減少了您需要攝入的指標數量。

你去付款

通常，無服務器帶有您的付費模型。換句話說，如果您不使用能力，理論上應該不需要為此付費。這是有效的（例如，諸如Kubernetes之類的編排者的固定費用），計費粒度也可能是一個因素。這裡的含義是您需要確保以自動化方式（包括警報）跟踪成本，尤其是如果您將自動化考慮在內。

從可觀察性的角度來看，一般來說，無服務器環境意味著要減少努力，因為大部分操作工作都將其外包給了該過程。潛在的缺點是缺乏可見性 - 如果OS或VM信號未明確暴露，則無法訪問它。

容器與功能

大致說，無服務器的計算產品以兩種口味出現：以容器為中心和功能作為服務（FAAS）。讓我們快速看一下這個空間中主要的雲領域提供的東西。

亞馬遜Web服務（AWS）中無容器中心的無服務器產品是AWS Fargate，您可以在兩個容器樂團的上下文中用作數據平面，要么在AWS Fargate（http://mng.bz/rxrj）上的Amazon ECS中使用，要么在AWS Fargate（HAP）eks fargate（http：httppate）上（根據您使用的編排，有不同的信號。例如，Fargate上的EKS通過CloudWatch提供了基於位的基於位的日誌路由解決方案（http://mng.bz/d4gy）以及度量量（http://mng.bz/1qz1），以及Fargate上的ECS以及Fargate上的ECS提供了監視（http://mng.bz.bz/mngz.bz.bz.bz.bzby）。

AWS Lambda（http://mng.bz/jgwk）是FAAS無服務器的計算產品。如圖3.4所示，它提供了包括AWS X-Ray（http://mng.bz/wv12）的統一和全面的可觀察性（http：// mng.bz/j129）方法。 在這個

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

上下文，您也應該考慮如何在Amazon CloudWatch日誌成本（http://mng.bz/qrvj）上節省。

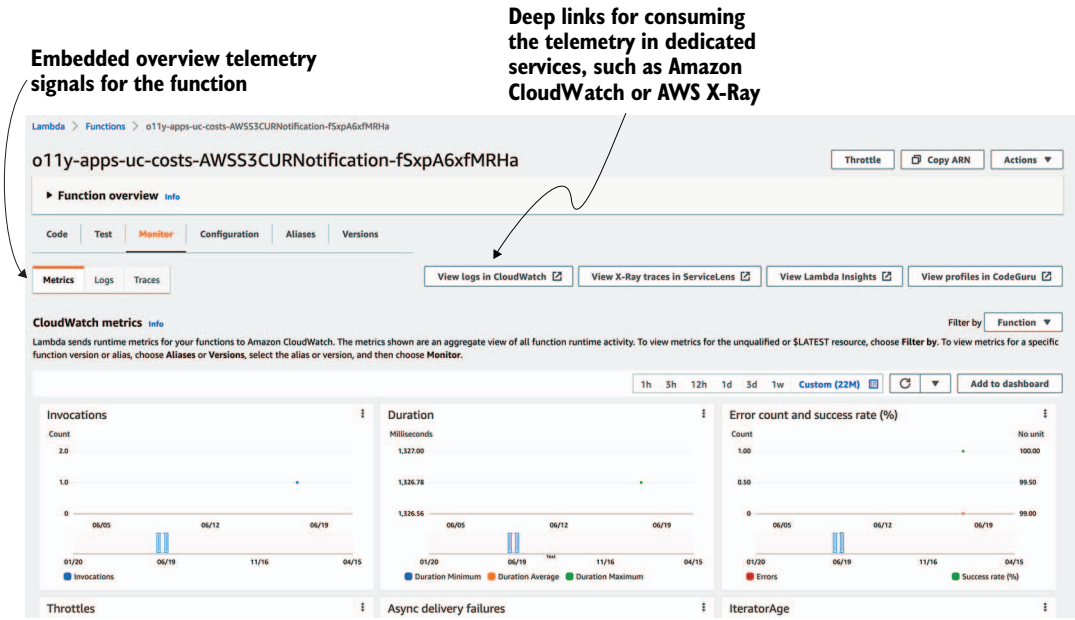


圖3.4 AWS LAMBDA監視控制台

Microsoft的Cloud中有許多無服務器的計算產品，最突出的Azure容器實例（http://mng.bz/7dge），其中具有Azure Monitor（http://mng.bz/mvl4），涵蓋了CPU，記憶和網絡用途以及全面的日誌管理。 此外，還有具有集成可觀察性（http://mng.bz/6daa）的Azure ConnerAner應用程序（http://mng.bz/5wma）。 Azure功能（http://mng.bz/o1dp）是FAAS的產品。它與Azure Application Insights（http://mng.bz/nwb2）集成在一起，以獲取日誌和met-rics以及基於OpentElemetry的跟踪支持。

Google Cloud Run（https://cloud.google.com/run/）是Google Cloud的Container-Centric無服務器產品，可針對無狀態服務進行了優化。 它附帶了Google Cloud的操作套件（https://cloud.google.com/products/operations--以前是StackDriver）集成，支持日誌，指標，痕跡和配置文件。 Goo-Gle Cloud功能（https://cloud.google.com/functions/）是Google的FAAS提供，再次使用Google Cloud的操作套件以可觀察到。 這樣，我們總結了與計算相關的來源，並以各種形式進行聲明。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

3.3 Storage-related sources

接下來，我們將仔細研究與存儲相關的來源。這些來源範圍從關係數據庫到文件系統。 通常，由於很少有人在此類別中編寫自己的存儲系統，因此主要是在確定哪些信號，以什麼形式以及在什麼時間內（保留）（保留）。

3.3.1 Relational databases and NoSQL data stores

任何有用的實際應用程序通常都具有某種狀態。在雲本地系統的背景下，在微服務中使用了幾種不同的關係數據庫管理系統（RDBM），例如PostgreSQL和MySQL，以及NOSQL數據存儲（例如DynameroDB，Redis和MongoDB），取決於需求，例如強大的一致性，可擴展性，訪問模式和訪問模式。

無論數據庫的類型如何，從用戶的角度來看，要回答的最重要問題是處理查詢的速度。從使用索引到緩存，使用了幾種技術數據基礎來加快查詢處理。通常，您希望將數據庫視為黑匣子，除非您的角色專門針對此部分，例如數據庫管理員。要在高級別上了解有關相關指標的更多信息，請查看OMED Habib（http://mng.bz/vn24）的文章“在企業應用程序中監視企業應用程序中的前6個數據庫性能指標”。

關係數據庫管理系統

在RDBMS上下文中發現的最常見的信號類型是日誌和METICS，痕跡也越來越可用。捕獲數據庫查詢METIC，例如最頻繁或最慢的查詢以及數據庫性能（例如，緩存和會話），為操作提供了基礎。

High Performance MySQL，Fourth Edition，Silvia Botros和Jeremy Tinley（http：//mng.bz/4dyj）是了解MySQL提供的信號類型的絕佳資源。它也可以用作將相關信號推廣到其他RDBMS的基礎，例如Maria db（https://mariadb.com/kb/en/server-monitoring-logs/）。 讓我們來看一個具體的示例：在圖3.5中，您可以使用Grafana作為前端看到亞馬遜紅移式監視。

您可以自己收集相關的數據庫指標（例如，流行的RDBMS gresql）：

將Prometheus與PostgreSQL Server Exporter（http://mng.bz/qp1v）使用專用工具，例如PGMonitor（https://github.com/crunchydata/ pgmonitor）

處理RDBMS性能指標的另一個選項是將此任務外包給供應商，即諸如DynAtrace（http://mng.bz/xnqa），datpps://wwwwwwwwwwwwwwwwwwwwwww.datadoghq.com.com/blog/postgresgresql-monoricorifice/ewricoriac，datpps://mng.bz/xnqa（http://mng.bz/xnqa）（http://mng.bz/yq6d）。另外，您可以使用AWS使用AWS（請參見圖3.5），Azure SQL數據庫（http://mng.bz/mbab）或云SQL Insights（http://mng.bz/a1ym）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

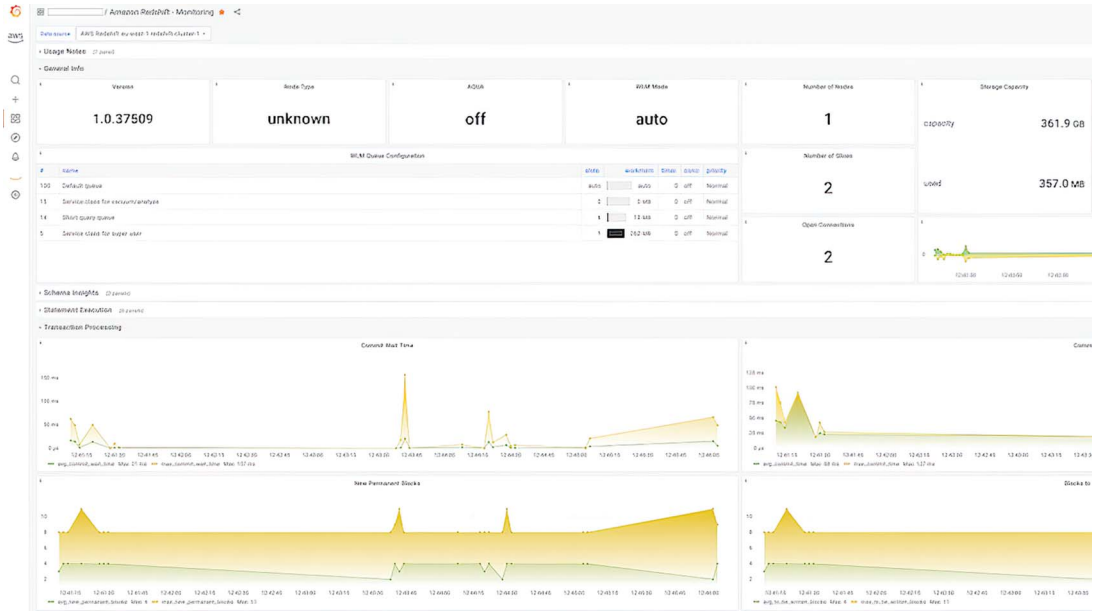


圖3.5驚奇 關於使用CloudWatch作為數據酸的紅移性能監視 圖中的內容

使用SQL，一種聲明性的語言，您可以說明所需的操作是什麼，而不是如何執行它們。執行您的SQL查詢的RDBM具有一個專用組件，稱為*query planner*（https://sqlite.org/queryplanner.html），該組件負責優化SQL查詢，對索引和其他因素進行考慮。 您可以通過PostgreSQL（https://explain.depesz.com/s/q6sq）中的“ dumply語句”中的查詢計劃的詳細信息了解。

nosql
NOSQL數據存儲的時間不及關係數據庫（在RDBMS的情況下，與50 +年相比，少於20年）。而對於日誌，適用於RDBMS的同一考慮：對於性能指標，我們面臨著不同的特徵。 NOSQL背景下的重點在於查詢計劃，因為缺乏關係意味著查詢計劃的重要性較小。 在NOSQL上下文中最重要的是訪問模式以及存儲層中發生的事情。

鑑於NOSQL數據存儲的廣度（從鍵 - 價值和文檔數據存儲，例如redis，mongodb和dynamoDB，到圖形數據庫，例如Neo4J或Amazon Neptune，到搜索引擎，包括Elasticsearch/OpenSearch），包括Elasticsearch/OpenSearch），我們不會在這裡詳細介紹一些詳細信息，但是一些示例可能會強調良好的實踐：

David de Torres Huerta（https：//sysdig.com/blog/redis-prometheus/）啟用的“如何用Prometheus監視Redis”）“監視Mongodb”（http://mng.bz/pprk）

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Jean-Mathieu Saponaro（https：//www.datadoghq.com/blog/top-dynamodb-performance-metrics/）of Jean-Mathieu saponaro（https：//www.datadoghq.com/bergysearch https：//www.datadoghq com.com/）

接下來，我們將介紹兩個通用但簡單的存儲系統：文件系統和對象存儲。

3.3.2 *File systems and object stores*

現在，我們將轉到兩個與存儲相關的資源，這些資源通常用於雲本機系統，通常用於二進制大型對象（BLOB），例如圖像，視頻或文檔（PDFS，Word Documents等）。雲本地上下文中本地文件系統的重要用例（例如Linux Ext4或窗口NTFS）是刮擦空間或換句話說，用於管理臨時文件。網絡文件系統，例如NFS（https://datatracker.ietf.org/doc/rfc3530/），並且對象存儲通常用於長期存儲，因為它們與本地附著的非ephemeral stor-ege相反（即，生命範圍都超出了實例，或者超越了實例，或者遠遠超出了實例，或者是contuner contuner或contuner funconter infocter nimecter infocter nimecter infocter nocter nocter nocter nocter nocter nocter funconter nimenternemence nocter funcormence）。

從應用程序日誌到系統日誌的各種日誌的中心位置，在Linux中是 /var /log Directory，並且用於管理本地日誌的主要工具是越來越多的JournalCtl。 我在此處強調*local*，因為日記cttl（按設計）不支持運輸登錄機器，因此將其作為操作員的練習。當我們談論從代理商和後端目的地的觀點上談論日誌運輸時，我們將在第4章和第5章中重新審視此問題。

從基礎架構操作員的角度來看（例如，Kubernetes平台管理員），最重要的文件系統信號是根設備上留下的可用空間量，並且可能會發生各種壞事（https://sre.google/workbook/workbook/eliminating-toil/）在Linux系統上檢查可用磁盤空間的一種簡單方法是DF（https://man7.org/linux/man-pages/man-pages/man1/df.1.html）命令。以以下示例：

Uses the -h option to print sizes in powers of 1024—that is, in a more human-readable format

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       7.7G  3.7G  4.1G  47% /
tmpfs           3.8G     0  3.8G   0% /dev/shm
tmpfs           1.5G  904K  1.5G   1% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           4.0M     0  4.0M   0% /sys/fs/cgroup
tmpfs           767M  4.0K  767M   1% /run/user/1000
```

We can see here that a little less than half of the disk space (47%) is used up on this machine.

您可以通過低級工具（例如Iostat和nfsStat）收集進一步的相關文件系統指標，或者使用諸如Prometheus（https：//prometheus.io/docs/guides/guides/guides/guides/guides/guides/node-exporter/）或使用OpenTElemetry，opentelemetry，host the host Metrics接收器（https node/node-exporter/）等更高級別的工具和代理。

對於雲本地應用程序，對象存儲是工作馬，幾乎用於從小文件（JSON）到大斑點（例如圖像）的任何內容。 通常，這些

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

對象存儲提供一個簡單的HTTP API來存儲和列出對象。不同的雲提供者提供了來自對象存儲的一系列信號，通常至少是日誌和指標，例如Amazon S3（http://mng.bz/y1xz），Azure Blob存儲（http://mng.bz/gyld）和Google Cloud Cloud Storage（http://mng.bz/gyld）

現在，我們已經完成了與存儲相關資源的簡要審查，該資源主要是基礎架構運營商相關的，對於開發人員而言是次要的。現在，無論您的角色是什麼：網絡及其信號。

3.4 Network-related sources

在本節中，我們將研究各種與網絡相關的來源，從低級相互之間到虛擬網絡。 與存儲一樣，這些來源主要由其他人（例如，您選擇的雲提供商）擁有，並且從本質上講是大量資源，這意味著您必須處理很多噪音。

3.4.1 Network interfaces

物理網絡接口或虛擬接口（例如回環接口）構成了與網絡相關源的最低層。 根據環境（認為無服務器數據平面或FAA），您可能可以或可能無法直接訪問這些相互之間的面孔。 由*direct access*，我的意思是能夠使用命令（http：// mng .bz/zxo1）等命令來查詢接口和路由信息。

為了證明直接訪問網絡之間的信息之間的區別，而在情況並非如此，請想像您擁有一個EC2實例（例如，作為Kubernetes群集數據平面的一部分），您可以將其SSH進入，然後將其ssh添加到可愛（擬合輸出）：

使用IP命令，列表網絡接口，其配置和狀態。

```
$ ip鏈接顯示1：lo：<循環返回，up，lower_up > mtu 65536 qdisc noque noque noque state未知模式默認組
默認組1000 link/loopback 00：00：00：00：00：00：00：00：00：00：00：00 BRD 00：00：00：00：00：00
：00：00：00：00：00：00： <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc mq state UP
mode     DEFAULT group default qlen 1000     link/ether 0a:78:c2:54:1f:39 brd ff:ff:ff:ff:ff:ff     altname enp0s5 3
: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state     DOWN mode D
EFAULT group default     link/ether 02:42:18:7d:57:00 brd ff:ff:ff:ff:ff:ff
```

```
$ IP路線
default via 172.31.32.1 dev ens5 proto dhcp src 172.31.40.108 metric 100 172.17.0.0/16 dev docker0 proto k
ernel scope link src 172.17.0.1 linkdown 172.31.32.0/20 dev ens5 proto kernel scope link src 172.31.40.108 1
72.31.32.1 dev ens5原始DHCP範圍鏈接SRC 172.31.40.108公制100
```

Using the ip command, list routing information, such as the default gateway.

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

在VM或物理服務器的情況下，在雲函數或Lambda功能的情況下，您確實可以了解網絡接口（配置和動態信息）時，您卻沒有。

3.4.2 Higher-level network sources

與網絡界面相反，高級網絡源（例如負載平衡器）通常可以直接訪問相關信號，最重要的是指標（與健康和績效相關）。 L3/4和L7（即HTTP級別）的負載平衡器通常提供大量信號，例如，Amazon Application Load Balancer（Alb; http：//mng.bz/0KVJ）支持日誌，元數據和痕跡。有關具體示例，請參見圖3.6，該圖是從開源可觀察性演示（https://github.com/observe-k8s/observe-k8s-demo）得出的。 Lambda功能前面的Amazon API網關（http://mng.bz/9dpj）也是如此。在這種情況下，一個重要的方面是後端的健康以及如何確保它們已經升起，因此，可以將流量路由到它們。 例如，來自ALB（http://mng.bz/wz4g）的HTTPCODE_ELB_504_COUNT代碼表明，後端資源沒有時間響應，或者如圖3.6所示，與var- ious使用相關的指標所示。 此外，您應該考慮某些編排者，例如Kubernetes，引入網絡摘要（http://mng.bz/8rek)-） - 用於審查，一種入口資源，您可以管理群集中的高級網絡資源，以及可能與摘要有關的信號以及可能不會受到摘要的信號。

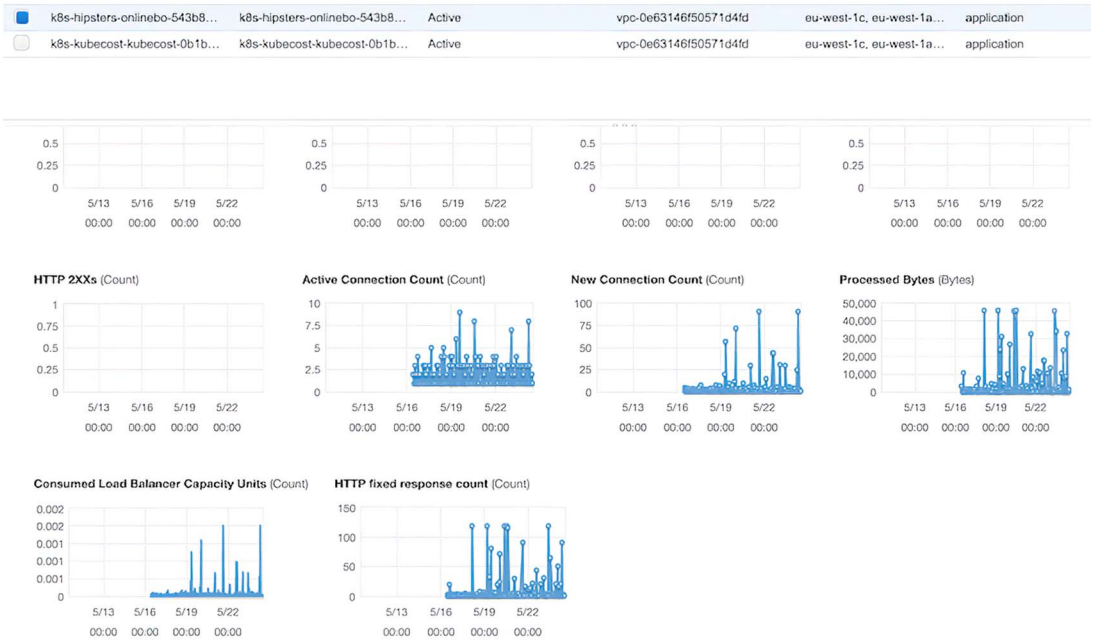


圖 ure 3.6 Alb-Monitoring控制台的屏幕截圖，向我顯示 技巧

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

網絡空間中的另一個高級來源是虛擬專用網絡（VPN），或者由雲提供商直接提供虛擬私有云（VPC）。 對於示例源，您可以將Amazon VPC流量日誌（http://mng.bz/eqnq）使用，該量提供了有關與安全相關的用例提供有關IP流量進入和來自VPC中網絡接口的詳細信息。 在圖3.7中，您看到了導出到S3存儲桶並使用achena查詢的VPC流日誌的示例描述（http://mng.bz/6n9d）。

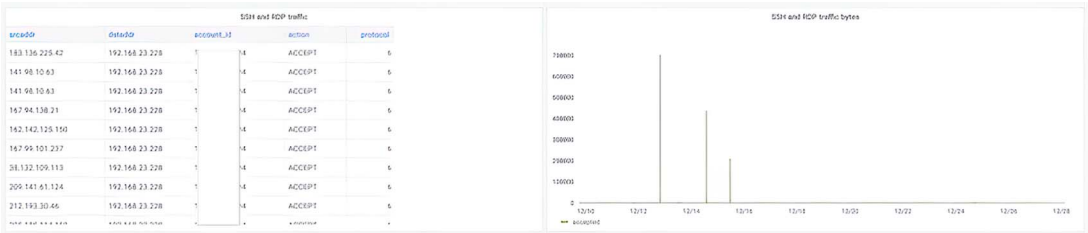


圖3.7 VPC流量日誌呈現在Grafana中的屏幕截圖，顯示SSH和RDP流量

這樣，我們已經完成了與基礎架構相關的資源。現在，我們可以轉到您通常可以控制更多控制的源類型：您的代碼。

3.5 Your code

現在，我們切換齒輪並專注於您擁有的代碼和可以儀器。有兩個方面：

> *The (business) code you write yourself* - 這是指從頭開始開發新功能或在您自己的代碼中修復錯誤。 *The code you depend on in your code* - 其中包括編程語言standard庫或第三方庫。這些庫通常是抽象的低級操作或共同功能，包括存儲和網絡訪問。

對於第二種情況，您的依賴性可能在某些情況下（例如，監管要求和業務原因）決定終點擁有它。例如，您可以決定維護開源項目的私人叉子，因此您可以更快地迭代或進行快速修復。

此外，考慮供應鏈方面很重要，您應該考慮到哪個（第三方）依賴關係，從它們來的來源等等。 要了解依賴關係的來源，您可以使用Backstage（https://github.com/backstage/backstage）和OpenlineAge（https：//pothlineage.io/）之類的工具。

在這種情況下，我們經常討論開發人員或開發可觀察性（例如，考慮和使用CI/CD管道作為信號源的實踐[http://mng.bz/ d4ey]）以及在此空間中新興的專用工具，例如什麼digma

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

(https://github.com/digma-ai/digma)和sprkl（https://sprkl.dev/）Envision。 總體方向是通過使用各種信號提供對代碼更改及其對應用程序的影響的見解，從而提高開發人員的生產力。

3.5.1 Instrumentation

我們已經介紹了第2章對日誌，指標和痕跡詳細介紹您的代碼的成本和收益。 根據您使用的編程語言（即解釋的與編譯），您面臨著不同的發射多種信號的努力。在可能的情況下，您應該使用自動啟動，通常是一個好主意，以opentelemetry進行標準化，以避免在切換信號目的地時重新啟動工作。我們將在第4章中更詳細地介紹這一點。

3.5.2 Proxy sources

如果您不擁有代碼，但仍然想發出一些信號，則可以使用代理資源。例如，在用Prometheus抓取指標的背景下考慮以下案例：

> 您想在混音中添加來自郵件隊列（例如Kafka或IoT設備）的指標，因此您可以運行Prometheus可以刮擦的眾多出口商之一（http://mng.bz/lwdj）。 在您擁有代碼（例如，在批處理作業或lambda功能的背景下），可能會出現另一種情況，這是一個短期運行的過程。 另一方面，Pro-Metheus假定它可以隨時刮擦。在這種情況下，您將使用PushGateway（https://github.com/prometheus/pushgateway）來補償阻抗不匹配。您可以通過Kubecon EU 2022談論“短暫的指標：監視短期或無服務器的工作”，由BartekP otka和Saswata Mukherjee提供，可通過YouTube（https:///wwwwwwww.youtube.com/watch？

代理也可以是一個適配器（例如，從日誌文件到STDOUT），以在雲本地設置中兼容和可用，以使某些類型的類型（例如日誌）。

這樣，我們已經完成了雲本地設置中的來源的討論，從計算到存儲再到網絡再到您自己的代碼，現在我們繼續談論信號如何到達各種目的地以供消費。簡而言之，我們轉向代理商。

Summary

來源是在觀察到的系統中發射遙測signal的任何內容，例如日誌，指標，痕跡和剖面。 如果您擁有一個源（例如，您編寫的應用程序代碼，以一個符合人運行），則可以儀器；否則，您需要圍繞它來處理源的所有者發出的內容。 我們可以區分來源類型：計算，存儲和網絡。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

由於其短暫的性質，容器和功能可能具有挑戰性的來源類型。 從可觀察性的角度來看，無服務器的環境意味著更少的精力，但控制量也較少。 與存儲相關的來源帶有廣泛的日誌；指標支持；而且，在較小程度上是痕跡，但是分析可用。 對於RDBMS，查詢計劃（分析）至關重要。對於NOSQL數據存儲，訪問模式和存儲佈局是要注意的最重要的領域。 對於文件系統，關注可用的磁盤空間（對於操作員）是最重要的。 由於事件的大量事件，與網絡相關的來源可能很嘈雜。通過源，目的地或應用程序進行過濾可能具有挑戰性。 高級網絡來源（例如負載平衡器）通常比NIC等低級來源更有價值，更有價值。 在啟動代碼時，請注意供應鏈問題（依賴關係）。 對於軌跡和指標，請始終使用自動啟示作為基準，然後根據ROI在哪里以及手動儀器進行決定。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

4 Agents and instrumentation

This chapter covers

代理和儀器的含義存在哪種類型的代理，哪種信號類型opentelemetry是什麼以及如何從中受益選擇代理的標準

在本章中，我們將重點關注如何從第3章中引起的源頭獲得的信號到目的地，我們將重點關注第5章和第6章。由於可觀察性是關於從信號中獲得可行的見解，從觀察到的系統中獲得了可行的見解，而沒有代理和工具，因此幾乎沒有觀察到的觀察。

簡而言之，在本章中，我們將學習如何儀器代碼（並自動執行該任務）以及選擇和部署收集，匯總，過濾，下樣本，編輯和路由日誌，指標和痕蹟的代理。

遙測行業在撰寫構造轉變中。這種從供應商特異性或特定於信號的儀器和試劑轉變為名為Opentelemetry的行業標準。與OpenTelemetry相反，我們將本書的文本視為供應商特異性和信號特異性藥物作為傳播劑。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

從AWS到Azure再到Google Cloud，以及可觀察性的雲提供商，從DataDog到Splunk到新遺物，再到LightStep，再到Dynatrace再到Honeycomb再到Grafana Labs，都決定在OpenTelemetry後面組裝。這有效地意味著遙測行業已決定制定遙測餐桌，對其進行調整，並在目的地上競爭（存儲和查詢以及前端）。

OpentElemetry按順序涵蓋，在寫作，痕跡，會議和日誌時涵蓋。 展望未來，社區決定還涵蓋OpentElemetry中的個人資料（在一個過程中的資源使用和時機）。另請參閱第2章中的討論。

由於日誌在2022年仍在進行中，因此我將使用特定於供應商或信號特定的設置來演示本章中的日誌。 另一方面，對於指標和痕跡，我將向您展示如何從Opentelemetry及其代理人（收藏家）的開放式和供應商 - 敏捷的standard中受益。

4.1 Log routers

首先，我們將查看日誌路由器。這些代理允許您從各種來源收集日誌，即在系統級別或您的應用程序上，並將其發送到中央查找位置。在這種情況下，您經常還聽到有關日誌轉發，日誌集合和日誌聚合的信息。 但是，為了使事情變得簡單，我們將掩蓋細節，並將所有這些都作為*log routers*將所有這些混合在一起。

在撰寫本文時（2023年），供應商特定的代理，例如Splunk的通用偽造者或DataDog代理，以及信號特異性藥物，包括Fluentd/fluentd/fluentD或廣泛使用的logstash。期望這會改變並轉變為OpentElemetry，向前發展。

我們將區分系統日誌和應用程序日誌。系統日誌來自但不限於以下資源（有關詳細信息，請參閱第2章中的討論）：

The operating system (e.g., in Linux, from /var/log and journalctl) The Kubernetes control plane (http://mng.bz/BmWv) logs (including from the API server, etcd, and controller manager) and the worker node logs (http://mng.bz/d10D), such as from the kubelet or the container runtime User activity and API用法日誌，就像AWS CloudTrail（https://aws.amazon.com/cloudtrail/）提供的API使用日誌基礎架構自動化日誌（例如，來自Hashicorp Terraform或AWS CloudFormation）

另一方面，應用程序日誌是您在代碼中發出的。您最終可能會使用不同的代理進行系統日誌和應用程序日誌；但是，如果可能的話，您應該嘗試通過選擇以下各節中討論的方法來標準化。這主要是出於操作原因，因為這意味著要修補的較少（例如，考慮到Log4J挑戰）和升級。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

NOTE In case you’re not familiar with the Log4j challenges I mention here (or maybe you’ve forgotten about them), CVE-2021-44228 (https://nvd.nist.gov/vuln/detail/CVE-2021-44228), informally known as Log4Shell, was a zero-day vulnerability in Log4j, discovered in November 2021. It caused a lot of work (to apply the安全補丁）在許多地方，從內部IT到雲提供商，並作為這種事情可能產生的操作影響的示例。

從API和模型的角度來看，日誌是穩定的，但在OpentElemetry中，直到2024年中期，直到整個SDK和分佈都受支持。這意味著您將需要臨時策略，直到最終可以遷移到OpentElemetry為止。

好的，那是足夠的理論。 讓我們看看在行動中的日誌路由。 首先，在容器的背景下是一個流行的選擇：Fluentd和Fluent Pit。

4.1.1 Fluentd and Fluent Bit

CNCF Projects Fluentd（https://www.fluentd.org/）和流利位（https：//flyentbit.io/）是在記錄域中啟動的流行雲代理。最初，只有Fluentd（用Ruby和C編寫），提供了一個豐富的生態系統（帶有1000多個插件），但為代理提供了相對較大的內存足跡（數十個MB）。

在2014年，Fluentd背後的人們Breasure Data背後承認需要專門針對約束環境（容器，IoT）的輕質日誌路由器。這種見解導致了流利位的發展（用C編寫），並具有數百個KB的內存足跡，而插件的插件比Fluentd（小於100）。

在寫作時，流利位將其範圍從僅日誌擴展到其他信號類型，例如指標和跟踪。有關詳細信息，請查看GitHub問題“ Fluent Bit V2.0即將到來”（https://github.com/fluent/fluent/fluent-bit/discussions/5993）。但是，在本節中，我們將重點關註記錄方面，因為飛行中仍然有許多WIP項目。讓我們看看容器化設置中的基本用法示例，並在此過程中討論一些屬性。

為了演示日誌路由，我們正在使用Docker Compose（https://opensearch.org/docs/latest/latest/opensearch/install/docker/）設置的OpenSearch。現在不必擔心opensearch暫時；我們將在第5章中詳細介紹它。只需將其視為可以將其發送到的地點日誌，並且您可以使用它來查找，搜索和查詢日誌行。例如，您可能對特定微服務（例如最後24小時）內的特定微服務的所有錯誤級日誌線感興趣。

在圖4.1中，您可以看到我們的示例記錄管道 - 我們使用的是HO11Y（http://mng.bz/rwxg）作為在Docker容器中運行的日誌信號源。該容器正在為其HTTP API露出端口8765，並使用Fluentd Logging驅動程序將日誌線發送到Fluent Plit（主日誌路由器），將日誌線攝入到開放式搜索中，作為日誌後端目標，然後您可以在OpenSearch搜索儀表板中查詢並可視化。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

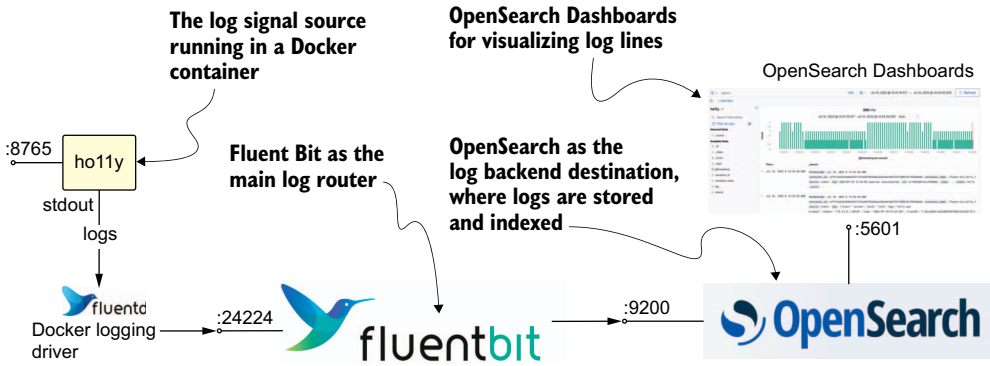


圖4.1使用Fluentd和Fluent位的示例記錄管道

繼續進行實施，我們再次使用Docker組合來站立管道，該管道在CH04/Fluent-Bit/Docker-compose.yaml文件中定義。讓我們逐步查看管道。 我在Docker中配置了HO11Y容器以使用Fluentd Logging驅動程序的方式組合文件（因為使用Fluent Bit作為驅動程序尚未直接支持那裡）。

列表4.1 Docker構成信號源的配置段Ho11y

```
ho11y:
    image: public.ecr.aws/mhausenblas/ho11y:stable
    ports:
      - 8765:8765
    logging:
      driver: fluentd
      options:
        tag: ho11y
        fluentd-address: localhost:8765
    depends_on:
      - fluent-bit
```

Exposes the ho11y HTTP API service port on port 8765 to make it accessible from the host machine (so we can invoke it with curl)

Configures the container to use the Fluentd logging driver

Defines that the ho11y container depends on the Fluent Bit log router, forcing the start-up sequence order

接下來是流利的位配置，如以下清單所示。如您所見，這是一個參數，即一個稱為fluent-bit.conf的文件，我們將在本章後面討論。

列表4.2 Docker構成配置段

```
fluent-bit:
    image: fluent/fluent-bit:1.9.8
    volumes:
      - ./fluent-bit.conf:/fluent-bit/etc/fluent-bit.conf
    ports:
      - "24224:24224"
      - "24224:24224/udp"
```

Defines where the Fluent Bit configuration (routing info) can be found

Makes sure we open port 24224 so that we can listen on incoming log lines from the Fluentd logging driver in Docker

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

最後但並非最不重要的一點是，讓我們看看如何在單個節點設置中以及前端/UI，OpenSearch dashboards配置日誌的後端OpenSearch（在以下列表中的Docker撰寫文件中）。

列表4.3 Docker組成的配置段用於OpenSearch（和儀表板）

```
opensearch-node1:
    image: opensearchproject/opensearch:2.1.0
    container_name: opensearch-node1
    environment:
      - cluster.name=opensearch-cluster
      - node.name=opensearch-node1
      - bootstrap.memory_lock=true
      - "OPENSEARCH_JAVA_OPTS=-Xms512m -Xmx512m"
      - "DISABLE_INSTALL_DEMO_CONFIG=true"
      - "DISABLE_SECURITY_PLUGIN=true"
      - "discovery.type=single-node"
    volumes:
      - opensearch-data1:/usr/share/opensearch/data
    ports:
      - 9200:9200
```

Exposes port 9200 so that we can ingest log lines (from Fluent Bit)

```
  opensearch-dashboards:
    image: opensearchproject/opensearch-dashboards:2.1.0
    container_name: opensearch-dashboards
    ports:
      - 5601:5601
    environment:
      - 'OPENSEARCH_HOSTS=["http://opensearch-node1:9200"]'
      - "DISABLE_SECURITY_DASHBOARDS_PLUGIN=true"
```

Exposes port 5601 to make OpenSearch Dashboards (frontend/UI) accessible from the host machine

Configures OpenSearch Dashboards (frontend)
使用OpenSearch（日誌的後端）

在這一點上，我們有效地實現了圖4.1所示的記錄管道。現在，讓我們仔細看看流利的位配置本身。 As depicted in list- ing 4.4, I’ m using the Forward (https://docs.fluentbit .io/manual/pipeline/inputs/ forward) input plug-in in Fluent Bit, taking in the log stream from Fluentd (from the Docker runtime) and, via the OpenSearch output section, configuring Flue nt Bit to route the log lines to OpenSearch.

列表4.4流利位配置

Here, we’re configuring the Forward plug-in capable of receiving log items from Fluentd (from the Docker log driver).

The Forward plug-in will listen on all network interfaces (that is, 0.0.0.0).

The default port for the Forward plug-in (that Fluentd connects to)

```
[INPUT]
    Name forward
    Listen 0.0.0.0
    Port 24224
```

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
[OUTPUT]
    Name opensearch
    Match **
    Host opensearch-node1
    Port 9200
    Index ho11y
    Suppress_Type_Name on
```

The destination configuration—in our case, the OpenSearch plug-in as the backend (downstream) for the log items

Here, we define that we, indeed, take any log item (we could be more selective here regarding which ones we're routing to OpenSearch, but for now, we use all of them).

The name of the index we want to create in OpenSearch

Tells the OpenSearch node that we don't have to set a type (since we don't care, in the context of this setup)

The hostname (which could also be a raw IP address) of the OpenSearch node

PHEW，這是很多配置！現在，是時候啟動管道並發電了一些日誌線了。首先，我們需要啟動流利的位 - opensearch組合：

```
docker-compose -f docker-compose.yaml up
```

在不同的終端會話中，像這樣啟動一個簡單的負載生成器（在bash shell中）這樣：

```
while true ; do \
  curl -s -o /dev/null -w "%{http_code}"
  ➥ localhost:8765 ; \
  sleep 1 ; \
done
```

Uses curl to invoke ho11y and generate log lines, discarding the output and only showing the HTTP status code returned

Pauses for 1 second and then repeats from the top

現在，您可以導航到http：//localhost：5601/。您應該看到OpenSearch儀表板的歡迎頁面（在Elk堆棧中 - Elasticsearch，Logstash和Kibana，等效組件將是Kibana）。單擊左上角（三條線）的漢堡圖標，然後選擇堆棧管理，然後進行索引貼。您應該看到類似圖4.2中所示的內容。

接下來，單擊“創建索引”模式，然後在標記為索引名稱的輸入框中，輸入HO11Y，然後單擊下一步。然後，在“時間字段”下拉列表中，選擇@timestamp，然後單擊“創建索引模式”按鈕。 您應該看到的結果類似於圖4.3中所示的結果。

現在，回到OpenSearch儀表板主頁，單擊左上角（三行）的漢堡圖標，然後單擊“發現”。 然後，您應該看到類似於圖4.4所示的內容。

現在，您可以使用Fluent Lot作為日誌路由器的運行日誌線，可供您使用。 從這一點開始，請隨意鑽入日誌線或使用Dash-Boards查詢語言（dql; https://opensearch.org/docs/latest/latest/dashboards/dql），一種域名詢問語言，一種在OpenSearch索引索引中找到數據，以在HO11Y logs中查詢特定的術語。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

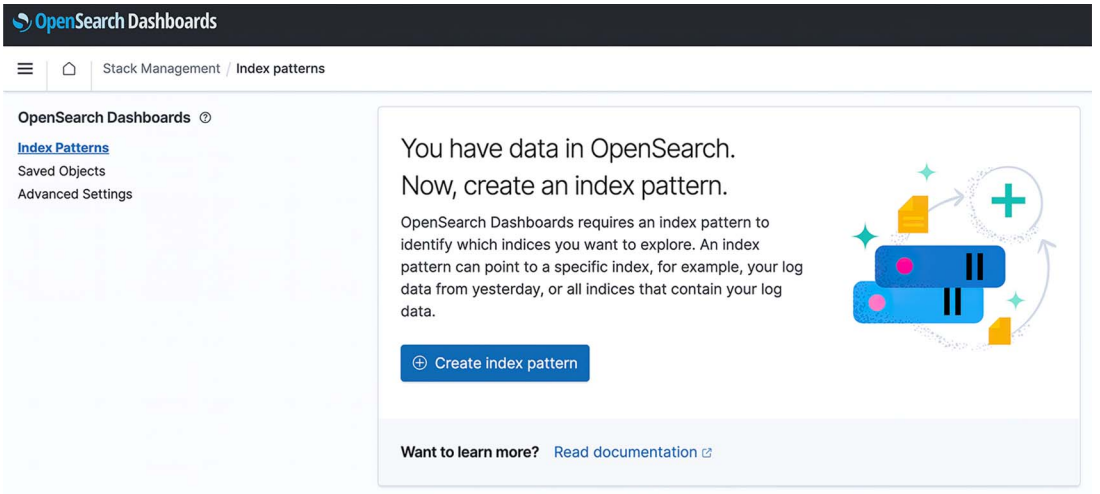


圖4.2在OpenSearch儀表板中創建索引模式

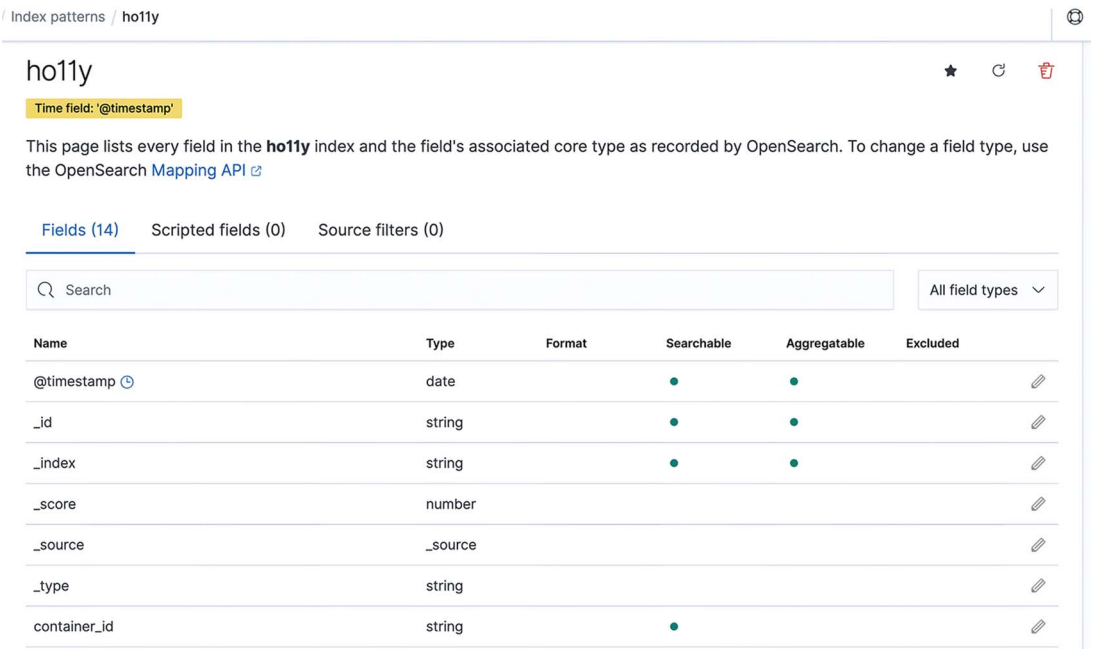


圖4.3 OpenSearch儀表板中產生的索引模式

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

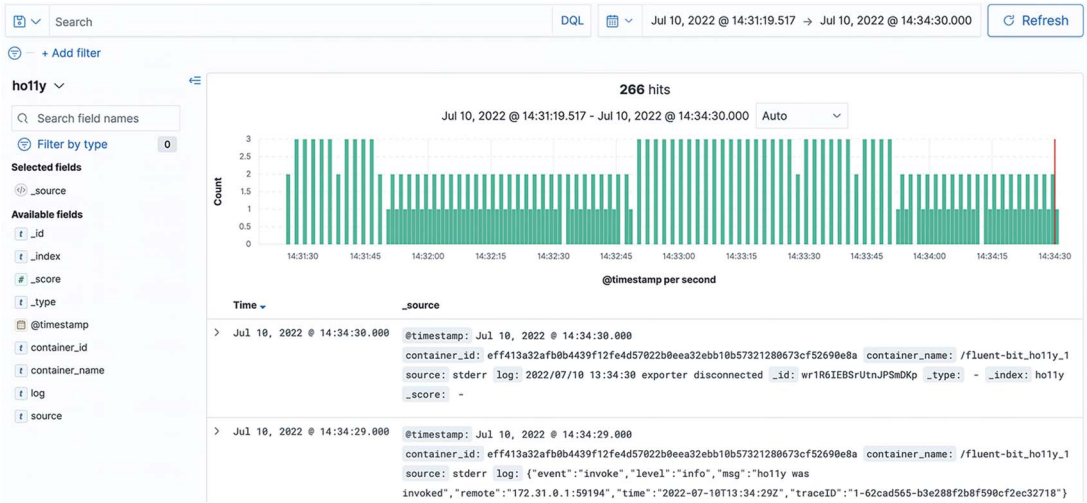


圖4.4 OpenSearch Dashboard的Discover View，我們選擇了我們的HO11Y索引

如果您有興趣了解在容器（尤其是Kubernetes）的背景下構建日誌路由管道的最佳詳細和最佳實踐，我強烈建議您研究菲爾·威爾金斯（Phil Wilkins）（2022; https; https; https; https; https; https; https; https; https; https; https; https; https; https; https：/wwwwwwwwwwwww.manning.com/ books/book/logging-in-action"中。

4.1.2 Other log routers

我們已經看到了流利的動作，但這不是我們唯一的選擇。確實，有許多商業和開源日誌路由器可用，例如：

the Beats（https://github.com/elastic/beats/tree/master）家族以及logstash（https://github.com/elastic/logstash），這是彈性堆棧的一部分。 DataPrepper（https://github.com/opensearch-project/data-prepper），一個AWS開源項目，允許進行狀態處理。 Syslog-ng（https://www.syslog-ng.com/products/open-source-log-management/）是一個系統實現，該實現默認情況下，默認情況下從SystemD Journal攝入消息，應用過濾器，並將其發送到文件。 Rsyslog（https://www.rsyslog.com/）是通過TCP/IP網絡路由和FIL- TRING LOG消息的開源工具，該工具被引入了Syslogd的替換。 GrayLog（https://www.graylog.org/）是一家日誌管理軟件公司，可提供日誌記錄解決方案，重點關注安全用例。 Cribl（https://cribl.io/）是一種具有強大攝入和過程能力的雲產品。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

請記住，隨著OpentElemetry的成熟和日誌繼續穩定，請確保您有一個遷移策略，以脫離您當前與OpenTelemetry Collector一起使用的日誌路由代理。現在讓我們繼續進行指標。

4.2 Metrics collection

與日誌相反，關於指標的景觀不那麼分散。在過去的幾年中（自2016年以來，隨著Kubernetes的興起的有效興起），主要參與者是Prometheus Project，它提出了指標的收集方式以及指標在電線上的代表方式 - *exposition format*。 首先，讓我們看看Prometheus，然後快速窺視大會空間中其他已建立的代理商。

注意，雖然普羅米修斯建立了基於拉動的集合（即代理通過HTTP端點默認為/指標查詢指標來源，但在OpenTelemetry的上下文中，情況更加細微。在那裡，入口部分仍然基於拉力。但是，出口部分（朝向後端目的地）通常是基於推動的。我們將在下一節有關OpenTelemetry的細節中獲取詳細信息。

4.2.1 Prometheus

Prometheus（https://prometheus.io/）是一個逐步的CNCF項目，與Kubernetes一起長大，在那裡也主要使用它。它涵蓋了指標的攝入，存儲以及查詢和聯邦界面，包括簡單的UI。此外，Prometheus項目的一部分是在提醒功能（單獨的二進制但緊密整合）。

在第2章中，我們進行了有關指標的一般討論。現在，讓我們更詳細地了解普羅米修斯如何做事。

Prometheus中的時間序列由其公制名稱和許多（可選的）鍵 - 值對，稱為*labels*。公制名稱捕獲了您要測量的一般功能。例如，API_REQUESTS_TOTAL可用於API調用總數。 另一方面，標籤代表普羅米修斯的維度數據模型。 假設您想使用API調用度量：所使用的HTTP方法以及URL路徑是什麼。您可以為此使用兩個標籤方法和路徑，例如：

API_RE QUESTS_TOTAL {方法=“ GET”，PATH =“/soming/fivder/t hing“}

普羅米修斯支持的指標有不同類型的指標，包括計數器，儀表和直方圖，而普羅米修斯（Prometheus）命名了指標和標籤的命名約定（https://prometheus.io/docs/practices/practices/naming/），但它並未定義或強制執行Seman-seman-tics-tics。實際上，普羅米修斯缺乏定義的語義不是一個巨大的問題，因為Prometheus提供了一個稱為*relabeling*（https：//prometheus .io/docs/docs/prometheus/prometheus/prometheus/prometheus/最新configuration/＃relabel_config）的功能，可以使您可以在naming上進行標準化。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

通過Relageling，您可以按目標添加，刪除或修改標籤。一個混亂的示例將是重命名標籤以標準化標籤名稱。 假設我們的來源將公開標籤處理程序下的URL路徑，但我們想在設置中使用路徑。在這種情況下，我們想進行以下重新標記：

api_requests_total {method =“ get”，處理程序=“/some/some/fivder/thing}

=>

api_requests_total {method =“ get”，path =“/some/list/fivder/thing}

您可以使用重新標籤規則來執行此操作，該規則可以在Prometheus Scrape Config（https://prometheus.io/blog/2015/06/01/advanced-service-discovery/#scrape-configurations-anderfigurations-and-relaburations-and-relabeling）中進行定義。

```
- source_labels: [handler]
  action: replace
  target_label: path
- regex: "handler"
  action: labeldrop
```

Matches the handler label name and adds the path label with the same label value

Once done, it drops the handler label name, leaving path as the one label capturing the URL path.

> 提示
> 如果您想進行交互嘗試重新標記，可以訪問https：//relabeler.promlabs .com/並使用Relabel規則（在左手Relabeling規則輸入框中），並將其作為右側（對象標籤）的輸入（對象）：
>
> 方法：“ get'handler：“/一些/有趣/事物”
>
> 現在，當您單擊“分析規則”按鈕時，您應該看到正在行動中的規則引擎，從而產生預期的結果。

此外，為了提高查詢性能，您可以在Prometheus Scrape配置中定義記錄規則（http：//mng.bz/v1vx）。此功能使我們能夠預先使用經常使用或昂貴的表達式，從而有效地將時間交易，從而將結果保存為新的時間序列。讓我們假設我們要使用錄製規則來捕獲我們的API調用示例的服務器端錯誤率（api_ request_total是計數器指標類型）：

The recording rule name

```
- name: api_invocation_server_error
  interval: 5m
  rules:
  - record: http_status_5xx:api_requests_total:ratio_rate_5m
    expr:
      sum(rate(api_requests_total{code=~"5.."}[5m]))
        /
        sum(rate(api_requests_total[5m]))
```

How often to evaluate the rule (here, every 5 minutes)

The name of the resulting metric (note the usage of :, which is, by convention, reserved for recording rule metrics)

The PromQL expression to evaluate at the current time, resulting in a new time series with the metric name defined by the preceding record value—http_status_5xx:api_requests_total:ratio_rate_5m, in our case

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

您可以通過兩種方式使用Prometheus（也就是說，您使用的是相同的二進製文件，具有不同的配置）：

PrometheusServer（默認模式）PrometheusAgent Mode（https://prometheus.io/blog/2021/11/16/agent/），優化用於刮擦和遠程閱讀–Write for Federation

我們在第2章中已經看到了Prometheus的行動，但是提醒您，Pro-Metheus的核心是其刮擦配置。在這裡，您可以定義Prometheus應該刮擦的內容（其目標），以及它應該如何找到目標（服務發現）以及（可選）記錄和警報規則。例如，要刮擦策略本身會暴露的指標（即自剪），您可以在scrape Config中使用以下內容：

```
global:
  scrape_interval: 30s

scrape_configs:
  - job_name: "prom"
    static_configs:
      - targets: ["localhost:9090"]
```

By default, Prometheus scrapes (calls) targets every 15 seconds, but here, we relax it to 30.

Prometheus adds the job name as a label (here, job=prom) to any metrics time series scraped. This is a provenance good practice and can aid correlation.

我們正在使用靜態服務發現方法（提供刮擦或調用的URL）。在這種情況下，由於Prometheus在$ host：9090/指標下公開了指標，因此我們將其用作目標密鑰的值。

請注意，刮擦配置仍然很重要，向前邁進，因為OpentEleme-旨在與Prometheus兼容，並允許您重複使用Scrape Configs（並進行較小的改編）。您將在本章的端到端示例中看到所有這些。

如果您想了解有關如何使用Prometheus以及它在內部工作方式的更多信息，我強烈建議您的最終參考：*Prometheus: Up & Running*，Julien Pivotto和Brian Brazil，*Second Edition*（o eyilly; http://mng.bz/x458）。

4.2.2 Other metrics agents

雖然普羅米修斯是雲本地生態系統中既定的指標平台，尤其是在Kubernetes環境中，但由於各種原因，您可能會遇到其他代理商，包括遺產使用情況，或者因為您的組織決定使用某個供應商或項目。這包括但不限於以下代理（按字母順序排列）：

Collectd (https://collectd.org/) - 在C中寫下，使用此代理，您可以有效地收集（操作）系統和應用程序性能指標。 幾個雲提供商和ISV提供了與CollectD的集成，例如Amazon CloudWatch，Splunk和Sumo Logic。

Grafana Agent (https://grafana.com/docs/agent/latest/) - 由Grafana Labs創建，該代理針對Prometheus指標進行了優化。 該代理的代碼被捐贈給Prometheus項目，並構成了代理模式的基礎

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

普羅米修斯，那裡。 在代理模式下，Prometheus停用了幾個功能，包括UI，警報和本地存儲。 *Graphite (https://grafana.com/oss/graphite/)* - 也在Grafana Labs傘下，該代理代表了上一代以機器以機器為中心監視的遙測的示例。
Nagios (https://www.nagios.org/) - 這是一種可尊敬的開源系統（包括用於監視機器和網絡的代理）。 *StatsD (https://github.com/statsd/statsd)* - 由Etsy開發，該代理是一個Node.js工具，具有客戶端 - 服務器模型，允許您收集和凝集應用程序指標並將指標發送到後端，例如Graphite。 *Telegraf (https://github.com/influxdata/telegraf)* - 從InfluxData開始，這是一個豐富而有力的代理，帶有許多插件，最初編寫了用於其InfluxDB時間序列的開源數據存儲，現在也支持OpentElemetry。

本節有意簡短，因為我認為重要的是要注意這個空間中的其他代理。但是，實際上，您可以預期，在大多數情況下，您將以一種或另一種形式與Prometheus打交道。如果您遇到了使用前面代理之一的腿部環境，則要確保有一條清晰的途徑可以遷移到Prometheus或Opentelemetry。

4.3 OpenTelemetry

OpentElemetry是一個孵化的雲本地計算基礎項目，該項目最初是由兩個項目合併的OpenCensus和Opentracing，該項目集中在分佈式軌跡上。 OpentElemetry從2019年開始擴大了指標和日誌的範圍。 該項目是僅次於Kubernetes的第二大CNCF項目，其核心是您可以使用的API，SDK和工具的集合來進行儀器，收集，處理，處理和攝入的遙測信號。

它提供了一個供應商 - 不可或缺的模型，用於發射痕跡，指標，日誌（以及向前看，還具有向後兼容性（通過接收器和老式的）以及通過專用協議的本機功能，還具有向後兼容性（通過接收器和實驗器）[https://github.com/open-telemetry/oteps/issues/139]）。 OpentElemetry包括對某些流行程序語言和運行時間的自動啟示支持，例如Java/jvm，.net，javascript/node.js和Python。如果您對開放遙測的許多組件的當前狀態感興趣，則可以

查看高級項目狀態（https://opentelemetry.io/status/）頁面。 查看細粒度的信息，因為該項目還跟踪實現的詳細遵守規範（http://mng.bz/aojp）。

請注意，除了規格（API和數據模型）外，OpentElemetry還具有一組語義，例如係統/運行時公制儀器的標準名稱（http://mng.bz/zqmm）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

在圖4.5中，您可以看到信號收集的OpenTelemetry概念，重點是相關。從底部開始移動到頂部，無花果顯示了從應用程序和基礎架構的信號流，最終通過OpenTelemetry Collector，最終攝入了後端目的地。

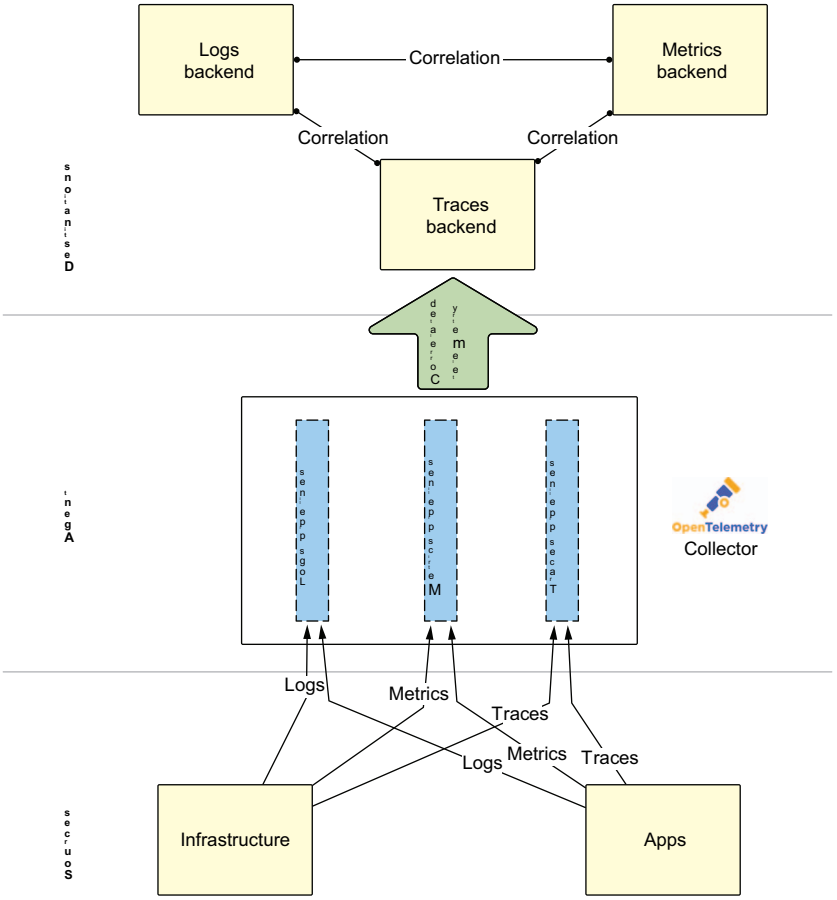


圖4.5 OpenTelemetry集合的概念（來自上游文檔; https://opentelemetry.io/docs/reference/reference/specification/logs/overview/）

在收集器中，在所謂的管道中進行了各種信號類型（目前的日誌，指標和痕跡），該管道專用於信號類型。 我們將在本章之後更詳細地討論該代理的內部工作，但是讓我們首先研究Opentelemetry House的應用程序代碼方面：儀器。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

4.3.1 Instrumentation

在您擁有的代碼的上下文中，您要解決的第一件事是使用OpentElemetry發出signals。OpentElemetry在撰寫本文時為11種編程語言提供了儀器支持。更具體地說，OpentElemetry支持C ++，.net，Erlang/Elixir，Go，Java，Java，JavaScript，PHP，Python，Python，Ruby，Ruby，Rust和Swift。

在OpentElemetry中，組織了儀器，以便用戶為每種編程語言獲得一個圖書館軟件開發套件，並可能支持自動儀器。某些語言通過單獨的存儲庫（例如Java）提供了這種語言，而其他語言（例如Ruby）可以維護支持手動和自動儀器的單個存儲庫。請記住，樂器不是免費的。即使在您不必更改代碼的最佳自動啟動情況下，也會對您的應用程序的性能產生很小的影響（http://mng.bz/rx5j）。

讓我們仔細研究OpentEleme-嘗試中不同的儀器方法。我們將從自動儀器開始，因為這是一個很好的起點，需要更少的精力。

自動儀器

使用自動儀器或簡單的自動啟動，在OpentElemetry中，所需的步驟如下：

1首先，啟用自動啟動。 您通常需要添加一個或多個依賴項。這些依賴項至少將添加OpenTelemetry API和SDK功能。 2然後，通過環境變量（和/或特定於語言的機制，例如Java中的系統屬性）來配置自動啟動。 您至少必須提供服務名稱來識別您的輔助服務，並選擇其他配置選項，包括諸如源配置，出口商配置和資源配置之類的內容。

OpenTelemetry（在寫作時）提供了為某些編程語言生成軌蹟的自動啟示，包括

Java自動啟動（https://github.com/open-telemetry/opentelemetry-java-instrumentation），它是OpentElemetry的一部分，在您包括的JAR文件中，您包括（使用-javaagent）。您可以以細粒度的方式（http://mng.bz/2dq8）配置此代理，從消毒數據庫查詢到捕獲預定義的HTTP標頭，該標題將動態地將JVM Bytecode注入了Punisure Java Frameworks的captemals。

JavaScript（https://github.com/open-telemetry/opentelemetry-js），也通過OpentElemetry Core正式支持，還允許YouT使用所有Java-Script-Script應用程序。請注意，還有其他特定於框架的SDK，例如Open-Telemetry Node SDK（http://mng.bz/1q41），可以啟用自動啟動。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

python（http://mng.bz/pz2y）也在自動啟動的正式支持語言中。 .NET（https：//opentelemetry.io/docs/instrumentation/net/automatic/）sup- ports auto-ports auto-instrumentation作為.NET Profiler，使用Monkey Patching在跑步時間內註入其他樂器。 go（https://github.com/keyval-dev/opentelemetry-go-instrumentation）是基於EBPF的用戶貢獻。嚴格來說，EBPF是Linux Syscall和虛擬機（VM），對內核版本和符號支持具有某些要求，使您可以將用戶空間功能推向Linux內核，並根據事件觸發它。在這種情況下，它意味著自動捕獲您的遙測信號。

展望未來，您可以預期為更多編程語言以及支持指標的自動啟示支持。 自動啟動是一個始終考慮的良好基準，因為您可以免費獲得一些見解。但是，它具有其限制性：自動啟示將該服務視為黑匣子，這意味著您不會從中獲得與商業邏輯相關的信號。為此，您需要使用手動儀器，所以讓我們看看它。

手動儀器

手動儀器意味著您修改自己的代碼以手動發射signals。不依賴語言所需的步驟是

1在您的代碼中導入opentelemetry，也就是說，對特定於語言的API和SDK進行依賴。 2對於跡線和指標，首先是創建一個提供商（通常有默認情況下的提供商），從而允許您創建一個實例（以及名稱和版本）。 3用於服務（與您可以儀器的庫相比），此外，配置了SDK，並提供適當的選項以導出您的信號。 4所有設置完成後，重點關注主要作業：創建事件（traces和tercric）。

注意日誌（http://mng.bz/jgrk）不遵循先前顯示的模式。相反，他們利用現有的設置；換句話說，儘管OpenTelemetry採用了指標和痕蹟的清潔式方法（通過指定新的API並提供在整個編程語言中的上述API的完整實現），但使用日誌的方法是不同的，因為重點是支持現有日誌和記錄庫。

手動和自動啟示的混合物有很多可能的用例和組合，如果您的確在設置的背景下尋求有關如何使用它的動手指導，您將在網絡上找到一系列出色的文章和教程。例如，“Opentelemetry和Python：完整的儀器指南”

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

（http://mng.bz/wvk2 ），“ auto-instrumenting node.js用opentelemetry進行應用”（http://mng.bz/qrpj），“用opentelemetry進行跟踪上下文繁殖” ＆代理示例”（https：// www .containiq.com/post/opentelemetry in-java）都是極好的資源。 接下來，我們將查看OpenTelemetry收集器 - “通用信號類型”代理或“統一遙測”代理。

4.3.2 Collector

OpentElemetry的代理稱為*collector*。 它包括組裝收集器組件的管道，例如接收器，處理器和出口商，如圖4.6所示。將管道視為一個組合組件的組合，這些組件定義瞭如何和位置從（接收器）獲取信號，在邊緣（通過處理器和連接器）在邊緣（轉換，批處理等）處理它們，並在其中攝入信號（出口商）中。

您可以在收集器中定義一個或多個管道。每個管道專用於信號類型（例如指標）。您也可以使用相同類型的多個管道，例如不同的環境或執行不同的策略。每個管道都獨立起作用，從來源收集信號並將其攝入後端。

與收集器交談的本機方法是通過OpentElemetry協議（OTLP; https://tinyurl.com/mngbz），這是一種通用遙測數據解散協議。 OTLP規範描述了下游來源，中間節點（如收集器）和後端目的地之間信號的編碼，傳輸和傳遞機制。

OTLP本身是由OpenTelemetry項目設計的，是對OpenCensus協議的增量改進，重點是交付的高可靠性和當數據傳遞時清晰可見性。此外，OTLP是L7負載 - 平衡器友好的，它支持了不平衡的傳輸流量的正確映射，以平衡傳出流量。此外，OTLP允許從desdeSinations一直回到來源的背壓信號傳導，從而使您能夠通過中間節點（例如Opentelemetry Collectors）構建可靠的Multihop delivery，每個節點具有潛在的處理能力。

從協議的角度來看，OTLP使用請求 - 響應樣式：Down-Stream客戶端發送請求，上游服務器以相應的響應回复。 OTLP非常靈活 - 您可以通過GRPC，HTTP和（向前）Websocket運行它。

在將來，我們可能生活在一個世界和所有事物的世界中，在撰寫本文時（即說話otlp）（即說話otlp），在撰寫本文時尚不如此。 opentelemetry社區找到了一種聰明的方式來幫助跳躍事物，從而啟動了一開始：通過介紹了眾多的批准，並且可以從現有的範圍中介紹一項逐漸讀取的經驗，並且可以讀取一定的經驗，並且可以讀取一份逐漸宣傳，並且可以讀取一項逐漸讀取的人（可以介紹），並且可以介紹一個逐漸讀取的人（可以介紹），並且可以介紹一份逐漸宣傳，並且可以介紹一個逐漸讀取的人（可以介紹），並且可以介紹一份經驗豐富的人，並且可以讀取現有的經驗。目的地的格式和協議。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

這是如何詳細工作的？現在，讓我們仔細查看圖4.6，以了解如何使用OpentElemetry收集器來定義信號類型的管道，以從下游填充信號並在上游攝入信號。

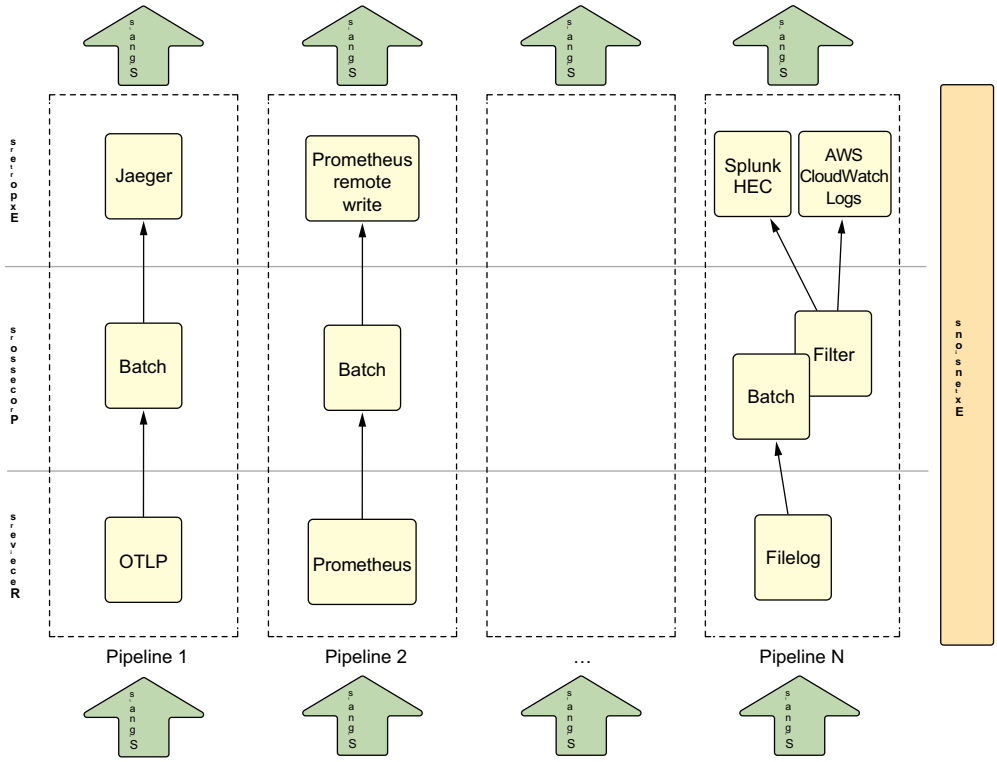


圖4.6 OpenTelemetry收集器及其組件

現在，我們將專注於收藏家中的管道。每個管道使用一個或多個組件：

在入學路徑中，從下游來源收集信號，一個或多個接收器可以來自兩個地方。 – For the native OTLP, from the opentelemetry-collector (http://mng.bz/ mVM4) repo – For all other receivers (Prometheus, Jaeger, Kafka, etc.), from the open- telemetry-collector-contrib (http://mng.bz/5wVa) repo The core manipulation component—one or more processors那可以來自兩個地方。 - 對於少數通用處理器（例如批處理），來自開放式遠程收費器（http://mng.bz/6dxa）repo

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

- 對於所有其他處理器（過濾器，變換，尾部採樣等），從Opentelemetry-Collector-Contrib（http://mng.bz/o1gp）repo進入出口路徑，將信號攝入到上游目的地，一個或多個可以來自兩個地方的出口商。 - 對於本機OTLP（和日誌記錄出口商），來自OpenTelemetry-收集器（http://mng.bz/nwa2）repo - 對於所有其他出口商（Prometheus Remote Write，lasticsearch，Elasticsearch，Elasticsearch，AWS X-ray等）

讓我們高度查看圖4.6的一些示例管道：

管道1是一個軌道管道，可以通過OTLP入站來收集痕跡，將其批量批量，然後將其攝入Jaeger實例。 管道2是一種指標管道，它通過Prometheus接收器入站刮擦指標，再次將它們批量批量，並使用Prometheus Remote Write API將它們攝入，例如，例如Thanos或Cortex。 管道N是一條日誌管道，它通過Filelog（http：//mng.bz/rjje）接收器收集日誌線，然後通過相應的出口商將日誌線再次批量批量，然後將其插入兩個不同的後端目的地。

In addition, the collector also offers several extensions (https://opentelemetry.io/ docs/collector/configuration/#extensions) to operate the agent, including health checks, service discovery, and rich telemetry (e.g., logs, metrics in Prometheus format, traces in OTLP, and profiles in pprof format).

提示有不同的方式來部署（https://opentelemetry.io/docs/col-lector/deployment/）收集器，通常在靠近工作負載（代理模式）和中心位置（網關模式）之間進行區分。在網關模式下運行收集器（有效地，負載平衡器背後的許多收集器）可以幫助實施策略，並支持圍繞訪問控制和身份驗證（包括API鍵和IAM角色）的關注點分離。

讓我們看一個具體的示例，該示例顯示了OpentElemetry收集器的作用，並照顧了指標和痕蹟的收集和路由； 我們的示例設置看起來像圖4.7中所示的內容。我們再次將HO11Y用作信號發生器，以Prometheus展示格式暴露指標，並以OTLP格式發射痕跡，而OpentElemetry Collector配置為刮擦痕跡並收集痕跡並將其攝入到各自的後端零件 - 用於Metrics和Jaeger的Prometheus。

現在，我們將再次使用Docker Compose構建圖4.7中所示的整體設置，因此，依次仔細查看每個組件。列表4.5顯示樣本應用程序HO11Y，生成指標和痕跡。特別注意

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

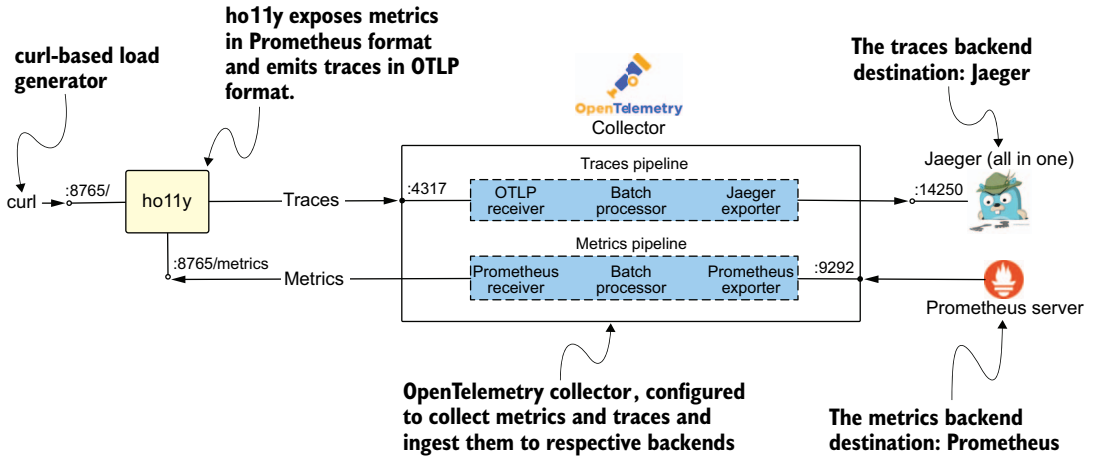


圖4.7示例軌跡和指標的OpenTelemetry管道

通過OTEL_EXPORTER_ OTLP_ENDPOINT環境變量連接到OpenTelemetry收集器的配置。

列表4.5 Docker構成HO11Y的配置片段

```
ho11y:
    image: public.ecr.aws/mhausenblas/ho11y:stable
    ports:
    - "8765:8765"
    environment:
    - "OTEL_EXPORTER_OTLP_ENDPOINT=otel-collector:4317"
    - "OTEL_RESOURCE_ATTRIB=ho11y"
    - "HO11Y_INJECT_FAILURE=enabled"
    depends_on:
    - otel-collector
```

The port for the HTTP API

Configures the downstream OpenTelemetry collector (for OTLP over gRPC)

列表4.6使用自定義配置（Otel-Collector-config.yaml）顯示OpenTelemetry收集器，我們稍後討論。我們將所有這些端口公開，以使它們可以從Docker外部訪問 - 也就是說，您可以通過此設置更輕鬆地從主機機器進行故障排除。

列表4.6 Docker構成opentelemetry收集器的配置段

```
otel-collector:
    image: otel/opentelemetry-collector-contrib:0.55.0
    command: ["--config=/conf/otel-collector-config.yaml"]
    volumes:
    - ./otel-config.yaml:/conf/otel-collector-config.yaml
    ports:
    - "4317:4317"
    - "8888:8888"
```

The collector config to use

The config file (mounted into the container from the host file system)

4317 is the OTLP receiver port.

8888 is the port the collector uses for metrics self-telemetry.

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
    - "9292:9292"
    - "13133"
    depends_on:
    - prometheus
    - jaeger
```

9292 is where we expose the Prometheus exporter.

13133 is the port used for collector health checks.

列表4.7顯示了配置為陸地指標的指標後端目的地的Prometheus，然後我們可以通過CLI查詢或使用內置UI查詢。 請注意，此處引用的Protheus scrape配置（Prom-Config.yaml）是我們將在本章稍後討論的內容（請參閱列表4.12）。

列表4.7 Docker組成的配置片段用於後端Prometheus

普羅米修斯：

```
    image: prom/prometheus:latest
    ports:
    - "9090:9090"
    volumes:
    - ./prom-config.yaml:/etc/prometheus/prometheus.yml
```

The Prometheus config to scrape the OpenTelemetry collector

最後但並非最不重要的一點是，列出4.8顯示了我們如何配置痕蹟的後端命運。在這裡要注意的兩件事是端口14250，我們將在Opentelemetry收集器中用作將痕跡發送到的目標，以及環境變量collector_otlp_enabled，我們（在撰寫本文時，為Jaeger的版本，我們在此處使用的Jaeger版本）仍然需要設置otlp，因此在收藏家和Jaeeger工作室之間進行了otlp的設置。

l iSting 4.8 Docker構成後端的配置段 Jaeger

```
j aeger:
    image: jaegertracing/all-in-one:1.36
    ports:
    - "5778:5778"
    - "14250:14250"
    - "16686:16686"
    environment:
    - "COLLECTOR_OTLP_ENABLED=true"
```

5778 is Jaeger's config interface port.

16686 is the port where Jaeger exposes its frontend UI.

14250是Jaeger接收Model.Proto的端口，該端口是GRPC的OTLP。

現在，我們知道了設置的所有組件，讓我們繼續進行Opentelemetry集合器和Prometheus的配置。要從示例應用程序收集指標和跟踪，我們將配置OpentElemetry收集器，其中包括兩個管道，一個用於指標，一個用於痕跡，如列表4.9至4.11中的顯示（將配置分解為接收器，出口商和實際服務配置），您可以在CH04/OTEL/OTEL/OTEL/OTEL/OTEL/OTEL/OTEL/OTELELEL-CONFIG.YATFIG.YAMAML FILE中找到。

讓我們從入口路徑和接收器開始。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

列表4.9 OpenTelemetry收集器配置，接收器

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: "0.0.0.0:4317"
  prometheus:
    config:
      scrape_configs:
       - job_name: "ho11y"
         static_configs:
         - targets: ["ho11y:8765"]
```

Enables and configures the OTLP over gRPC receiver (http://mng.bz/4Drj) we're using for ingesting traces. We're declaring here that it should listen on port 4317, the official, IANA-registered port for this scheme and protocol.

Here, we enable the Prometheus receiver (http://mng.bz/QP9v) and configure it so that it scrapes our sample app.

請注意，Docker Compose構成，以便以所述名稱可以觸及HO11Y應用程序。讓我們進入出口商的出口路徑。

列表4.10 OpenTelemetry Collector配置，出口商

```
exporters:
  logging:
    loglevel: debug
  prometheus:
    endpoint: "0.0.0.0:9292"
  jaeger:
    endpoint: "jaeger:14250"
    tls:
      insecure: true
```

We enable and configure the Prometheus exporter (http://mng.bz/XNla), turning our collector, effectively, in a pass-through.

Next up is the Jaeger gRPC Exporter (http://mng.bz/yQjd), which, as the name suggests, allows us to export (trace) data via gRPC to a Jaeger backend destination.

在我們繼續前進之前的一件事：還記得列表4.8，它顯示了Docker中的Jaeger配置，指出了端口14250？ 這正是我們在列表4.10中插入的內容，因此OpentElemetry Collector知道將痕跡發送到哪裡。

為了在此示例中保持簡單，我們使用Prometheus出口商允許Prometheus Server刮擦。在相關的餵養和長期存儲的生產設置中，您可能想在此處使用Protheus remote Wright Offorter（http://mng.bz/mbgb）。 這使您能夠將指標攝入後端，例如皮層，Thanos或託管服務，例如Prometheus的亞馬遜託管服務（https://aws.amazon.com/prometheus/）。

> 收集器配置中的提示，組件（接收器，處理器，出口商，連接器和管道）由類型[/name]格式的組件標識符定義。只要標識符是唯一的，您就可以多次定義給定類型的組件。例如，您可以有兩個指標管道，一個管道，一個稱為指標/開發的指標和一個指標/產品，具有不同的後端。

最後，我們可以使用所有這些接收器和出口商來定義我們的管道，如以下列表所示。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

列表4.11 OpenTelemetry Collector配置，服務（管道定義）

```
service:
  telemetry:
    logs:
      level: debug
    metrics:
      level: detailed
  pipelines:
   metrics:
      receivers: [prometheus]
      processors: [batch]
      exporters: [logging, prometheus]
    traces:
      receivers: [otlp]
      processors: [batch]
      exporters: [logging, jaeger]
```

A good practice in dev/test is to configure the collector telemetry to emit fine-grained agent logs and metrics (available via port 8888 on the collector).

This is where the pipeline definitions starts; you can have as many as you like.

Here, we define the metrics pipeline using the Prometheus receiver in the ingress path and the Prometheus exporter for egress.

We configure the traces pipeline that uses OTLP to ingest traces from ho11y and routes them to our downstream configured Jaeger instance.

警告如果您僅像以前的步驟一樣啟用和配置一個組件，例如接收器或導出器，但不要在管道定義中使用它們，那麼它就不存在組件。 收藏家只是忽略了它，沒有任何事情可以按照您的期望。這是我在野外見過的一個非常常見的坑，誠然，幾次遇到了自己。

另外，您可能已經註意到，在出口商部分中，有兩個條目。原因很簡單：通過包括記錄出口商，您可以直接在屏幕上看到正在發生的事情（例如，指標刮擦），因為所述出口商會向STDOUT發送信號。

To wrap up the configuration, let’s have a quick look at how downstream Prometheus (our metrics backend destination) is configured (listing 4.8): it scrapes itself, which is a good practice, and most importantly, for our overall setup to work, it scrapes the OpenTelemetry collector via otel-collector:9292 (this is where we told the Pro- metheus exporter to expose its endpoint).

清單4.1 2 Prometheus Server配置，刮擦OpenTelemetry Co Llector

```
global:
  scrape_interval: 5s
  evaluation_interval: 5s
scrape_configs:
  - job_name: "prometheus"
    static_configs:
    - targets: ["localhost:9090"]
  - job_name: "otel-collector"
    static_configs:
    - targets: ["otel-collector:9292"]
```

Self-scraping the Prometheus server

Scraping the Prometheus exporter in the OpenTelemetry collector

好吧，這是很多YAML代碼。現在，讓我們來運行一些事情。將一個ter-minal會議和CD啟動到CH04/Otel/，然後進行

```
docker-compose -f docker-compose.yaml up
```

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

請注意，這次，我將窮人的負載生成器（基於捲髮）包括在Docker組成的文件中，因此此時不需要進一步的步驟。

現在，終於是時候收穫我們艱苦的YAML爭吵的果實了，因此打開您選擇的瀏覽器，並輸入Prometheus的兩個URL（Localhost：9090）和Jaeger（Localhost）（Localhost）（Localhost），您應該看到一些事情。

切換到Prometheus並輸入Promql查詢

總和（http_status_code）（ho11y_total）

您應該看到（在“圖”選項卡上）類似於圖4.8中所示的內容，即由返回的HTTP狀態代碼分組的總示例應用程序調用（HTTP API調用）的總和（以平均方式配置為10%的錯誤率）。

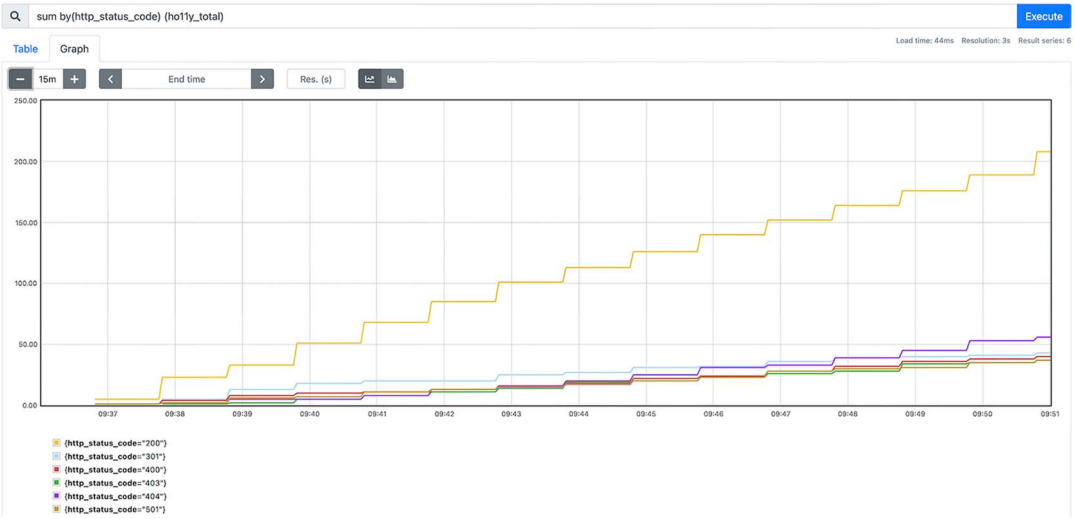


圖4.8行動中的Prometheus UI，顯示了隨著時間的流逝繪製的簡單HO11Y公制

轉到軌跡：在Jaeger中，您首先必須選擇服務。您可以通過從相應的下拉框中選擇它來做到這一點。此外，在標籤字段中，您將輸入

http.status_code = 403

這意味著您只對帶有此特定標籤的跡線感興趣 - 換句話說，捕獲事件的踪跡，其中HO11Y示例應用返回了HTTP狀態代碼403。現在，按左側導航中的查找軌跡按鈕，結果應類似於圖4.9中所示的。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

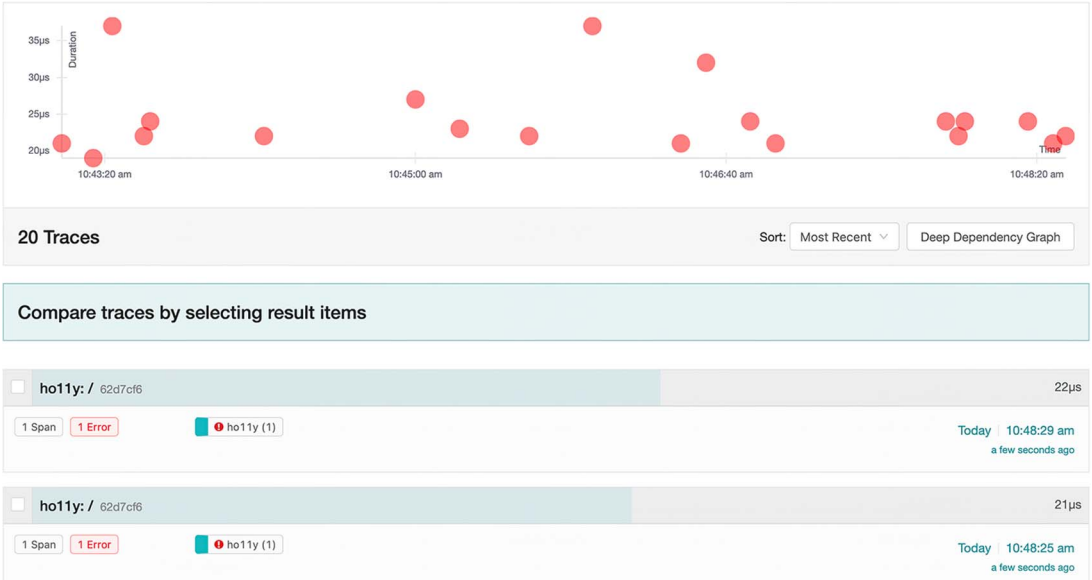


圖4.9 jaeger UI正在使用，顯示HTTP 403狀態代碼過濾的HO11Y痕跡

請記住，在Jaeger中查看單獨的跟踪就像在右側痕跡列表中單擊它一樣簡單，因此請繼續選擇隨機的踪跡。這應該使您了解類似於圖4.10所示的視圖。

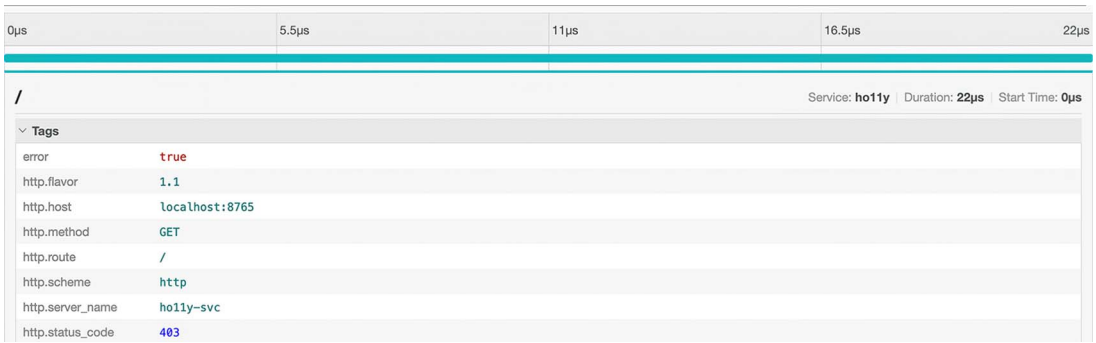


圖4.10在Jaeger中鑽入特定痕跡，顯示了所有可用的細節

總而言之，在此操作演練中，我們有

通過手動儀器產生的1個產生的痕跡和指標。 2配置了OpentElemetry集合器，以收集這些信號並將它們攝入兩個不同的後端目的地（用於指標的trace，thrace，for Trace，for trace和protheus）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

3查看了後端中的信號。在Prometheus中，我們通過HTTP響應代碼繪製了整體調用，在Jaeger中，我們查看了客戶端錯誤跨度（包含403 HTTP響應代碼）。

我們將在以後的章節中介紹與目的地合作的細節，但是在這裡，我們專注於OpentElemetry背景下的儀器和代理方面。

好的，那很多 - 堅持下去！ 我希望這個小小的開放式遙測步行能夠通過這個通用的遙測代理商給您帶來的第一個品味和對您可用的力量的想法。

提示在OpentElemetry分佈（http://mng.bz/a12m）時，您可以通過幾種方法來解決：您可以使用上游，預構建的分佈（http://mng.bz/gbnv），可以隨時隨地選擇特定的Cloud Could Provider or O.11y Vendor（E.11y Vendor）， [http://mng.bz/5wbb]），或者您可以決定使用項目提供的工具（http://mng.bz/5wvo）使用OpentElemetry Collector Builder（http://mng.bz/5wvo）滾動自己的分發。

如果您想深入研究OpenTelemetry，請了解其來自哪裡，整體願景是什麼以及如何在環境中應用它，我建議您仔細閱讀以下書籍：

Alex Boten的出色書，*Cloud Native Observability with OpenTelemetry*（https：//www.cloudnativeobservability.com/），為您提供有關Open-Telemetry和許多動手示例的更多上下文。 Daniel Gomez Blanco（Apess，2023; http://mng.bz/6dxe）by Daniel Gomez Blanco撰寫，這是一本很棒的學習書籍，但也可以幫助將Opentelemetry介紹給一個組織（基於他自己的經驗）。 Austin Parker和Ted Young（O’Reilly; http：// mng . bz/o1gz）的作者：這是該領域的最新書籍，有望成為參考書，兩位作者是OpentElemetry本身背後的動力。

要解決我們的代理評論，讓我們快速查看當前正在使用的既定代理商，並且將來可能會被OpentElemetry所取代。

4.4 Other agents

除了fluent forme for for for原木，指標的普羅米修斯和開放式遙測收集器作為通用遙測劑外，還有幾種已建立的且廣泛使用的代理。其中大多數是由雲提供商撰寫的，並由雲提供商分發，但此外，一些可觀察性供應商已經引入了代理，因此讓我們快速了解它們（適用相同的想法：在時間的填補度中，所有這些都可能會被OpenTelemetry或其收藏家替換）：

AmazonCloudWatch代理（https://github.com/aws/amazon-cloudwatch- agent/）可用於日誌和指標，支持AWS的專有嵌入式式指標格式（EMF; http; http://mng.bz/nwa5）。 CloudWatch代理已經

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

通過嵌入式收集器支持OpentElemetry，並已在OpentElemetry（http://mng.bz/g91q）上進行了替代。除了雲手錶代理外，AWS還通過其AWS發行版（https://aws.amazon.com/otel/）提供了符合OpentElemetry的收藏家。

Azure正在貶低其日誌分析代理和Telegraf代理，以支持新的Azure Monitor Agent（AMA; http://mng.bz/2dnd）。就像其他雲提供商一樣，Azure（從長遠來看）Opent Elemetry（http://mng.bz/vn0p）也是標準化的。 Google Cloud類似於Azure，Google Cloud正在遠離其基於收集的監控代理，而有利於新的OPS代理（https://cloud.google.com/monitoring/commoning/agent/agent/ops-agent），它使用流利的log lot來進行日誌收集和路由，以及opentelemetry collector for Metrics for Metrics。 dataDog的vector（https://vector.dev/docs/about/what-is-vector/）是一個互動的開源項目，在某種程度上，它在某種程度上與opentelemetry競爭了通用信號類型支持角度； 但是，它也支持開放遙測。如果您全都使用DataDog，這似乎是一個不錯的選擇。

現在，我們對存在哪些代理，其覆蓋範圍和用例有很好的了解，讓我們繼續討論您應該如何選擇代理的主題。

4.5 Selecting an agent

通常，您可以使用幾種選擇。重要的是要確保您選擇一個適合工作量，環境和組織能力的代理商。

消耗有關該主題的一般材料（例如博客文章和文章）沒有什麼壞處，請參見“ logstash，fluentd，fluent bit或vector？如何選擇正確的開放式日誌收集器”（http：//mng.bz/4drr）或“誰是贏家：比較矢量，fluent bit，fluent bit，fluent，fluentib cormess（htplyentd cormess）（httpspspspts formation forcepts formec.com/--雲/日誌收集器 - 績效基準-8C5218A08FEA），但是，您應該將它們用作靈感或起點，而不是“買家指南”。 換句話說，您很可能必鬚根據自己的需求和偏好進行自己的評估，以便以任何特定的順序選擇代理。

4.5.1 Security for and of the agent

我在AWS上工作的位置是最高優先級，因此我從此開始。重要的是要同時考慮安全*of*代理和安全*for*代理，我的意思是以下內容：

使用Security *for*代理，請問自己和/或供應商以下內容： - 誰處理CVE和安全補丁？ - 代理筆是測試的嗎？

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

- 您可以為工件（例如代理的容器圖像）獲得可再現的版本嗎？ - 代理商的供應鍊是否已知和/或記錄？ 使用Security *of*代理，嘗試確定以下內容： - 代理是否支持TLS/HTTPS和等效的加密交通？ - 您可以在代理機隊（例如憑證，訪問和過濾器）中應用政策嗎？ - 是否有適當處理敏感數據的機制，例如通過某些立法（例如在歐洲，通用數據保護法規（GDPR））所要求的諸如個人認同信息（PII）？

理想情況下，您將通過開源項目，ISV或Cloud Provister驅動代理商的清單來提前獲取此信息。

4.5.2 Agent performance and resource usage

代理性能（即可以處理的速度，通常通過吞吐量以及所使用的資源來測量的速度 - 是一個關鍵的評估標準。以下是此類性能相關報告的一些例子：

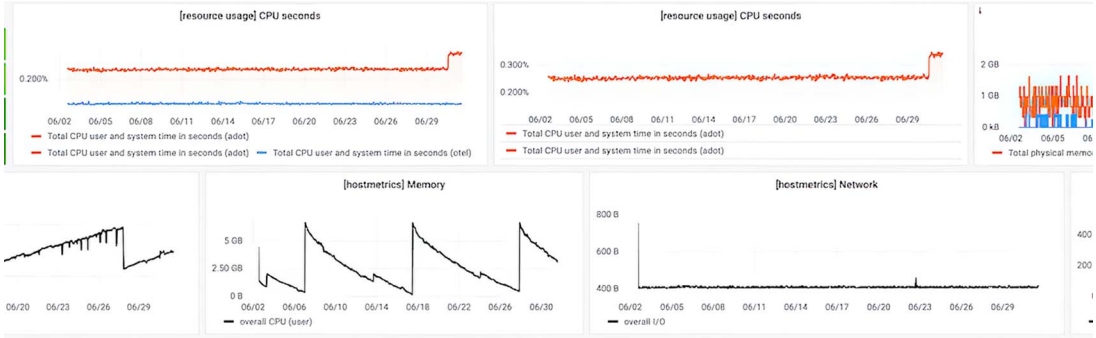
上游的“ OpenTelemetry Collector績效”文檔（http：// mng .bz/qp96）。 Opentelemetry收集器（ADOT）收集器性能報告（http://mng.bz/n2j1）的AWS發行版。

簡而言之，一個重要的標準是與代理商的資源足蹟有關。 例如，標準可能是代理商在各個階段（例如，閒置，平均或峰值）中消耗多少RAM和/或CPU來收集，突變和攝入信號。

除此之外，您希望能夠在多信號代理（例如OpentElemetry Collector）的情況下跨信號類型發生故障。圖4.11顯示了Grafana儀表板（https://ref.otel.help/otel-collector--ops/otel-collector-dashboard.json）的檢查屏幕捕獲（隨著時間的流逝）。

在資源使用情況下，您要確保代理使用的資源與您的足跡相比很少。不要忘記累積效應 - 也就是說，一個代理商可能只用幾千個代理商在整個車隊中耗盡了幾個MB，這可能總結到高結核病系列。如今，大多數代理都具有使您可以控制資源使用量的機制（例如，在OpentElemetry Collector的情況下，您可以使用內存限制器處理器來防止失位（OOM）方案。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>



f igure 4.11 opentelemetry收集器資源的Grafana儀表板U 聖人

4.5.3 *Agent nonfunctional requirements*

除功能方面外，重要的是要問自己以下幾點：

> 代理商開源，可擴展或由單個供應商或社區支持？ 誰看著觀察者，或者以不同的方式說，代理商遙測呢？在2023年（及以後），您不僅應該期望可以訪問代理日誌，而且還應獲得指標（理想情況下，以普羅米修斯格式）以及您的代理商的痕跡和文件。 代理是否能夠處理一種或多種信號類型？信號特異性代理意味著您需要多個代理才能進行360覆蓋。

非功能要求比我們討論的要多得多，但是這些問題再次構成了基礎知識，您可以選擇在選擇代理時使用組織要求或偏好擴展它們。

Summary

有兩種類型的代理：供應商特定的代理，例如DataDog代理，以及特定於信號的代理（通常是開源），例如用於日誌的流利位。 在本書中，我們將特定於供應商的特定和信號特異性代理視為傳統代理，與開源的opentelemetry收集器相比。 Opent Elemetry是一組規範，SDK，協議（OTLP）和代理。 opentelemetry允許您以供應商中立的方式收集，處理和攝入的日誌，指標，痕跡以及（將來）配置文件。 使用OpentElemetry，您會在信號來自（資源屬性）和遙測信號本身的位置上獲得富裕的元數據。 基於語義慣例，OpentElemetry使您能夠在後端實施有效的相關性。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

自動啟示是自動在不更改應用程序代碼的情況下自動發電遙測的重要基線方法。如果供應商或編程語言（在OpentElemetry中）支持自動儀器，請使用它。 在自動啟示無法提供所需的（業務邏輯）的地方，您需要使用OpenTelemetry SDK手動儀器。 使用OpentElemetry收集器來收集和路由軌跡和指標，並最終從任何源到任何目的地，從而形成通用式觸點代理。 代理需要安全，表現和精益（最少的資源使用率）。 選擇代理可能需要時間，可能需要一些努力，但從長遠來看值得投資。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

5 Backend destinations

This chapter covers

我們是後端目的地的意思以及存在哪些類型的目的地可處理日誌，指標和痕蹟的後端目標選項什麼時間序列數據存儲是什麼，您可以將它們用於引入柱狀數據存儲和格式及其用例選擇後端目的地的考慮因素

到目前為止，您應該知道存在哪些主要信號類型（請參閱第2章），它們來自何處（請參閱第3章），並且存在哪些選項進行收集和預處理所述信號（請參閱第4章）。在本章中，我們將深入研究後端目的地，代理商的目標或下沉。後端目的地或簡稱後端是您可觀察到的問題的真實源頭。它們允許您存儲和查詢信號，並使它們可用於前端目的地（請參閱第6章）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

為什麼我可以區分後端和前端？好吧，有一些示例完全屬於一個或另一個類別。例如，Grafana是一個純淨的前端目的地（有時也稱為*presentation stage*），該目的是可視化或警報信號的後端目的地。另一方面，雲本機計算基礎（CNCF）項目Cortex是一個純粹的後端目的地（除了基本的管理Web UI外，您只有API可以進行攝入和查詢指標）。

然後，有來自專用可觀察性供應商的產品，例如Splunk，Datadog，Dynatrace，honeycomb，New Relic，New Relic，ServiceNow的Lighstep，Cisco的AppDynamics和Sumo Logic，以及通常結合前端和後備，提供整合的經驗。 在這種情況下，您無法從後端使用前端（UI）。 開源空間中也有混合的情況，例如CNCF Jaeger（帶有攝入管道，但需要額外的存儲系統）。我們將在第6章中介紹這些統一或統一的目的地。

另一個特殊情況是雲流行產品的產品，尤其是Amazon Cloud-Watch，Azure Monitor（包括應用程序見解）和Google Cloud Cloud Operaimes Suite，它們涵蓋了廣泛的功能，並提供具有不同後端和前端的集成。

請注意，就像我在第1章中指出的那樣，關於為什麼我使用日誌，指標和痕跡邊界來提出主題，我正在將前端區分 - 主要是在概念層面上進行區分。 這應該有助於您的理解； 但是，到目前為止，您知道現實更加混亂，更複雜。

現在，讓我們開始。在本章中，我們將

請查看一些共同點和一般的後端術語審查日誌後端的選項和使用審查指標後端的選項和使用方式審核痕蹟的選項和使用痕跡後端討論柱狀數據存儲討論如何選擇後端。

準備好？讓我們潛入！

5.1 Backend destination terminology

後端是您所有可觀察性問題的真理之源，無論是人的詢問（根據預感，臨時）或一段代碼（想想一個警報，以表現出後端，以評估是否滿足某些條件）。在我們了解後端及其優缺點的不同類型之前，我們將首先退後一步，看看它們的共同特徵以及本章和書籍其餘部分所需的一些基本術語。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

通常，無論是有關性能還是成本，您都需要考慮三個主要部分（請參閱第5.6節）：

Ingestion - 信號如何到達後端。存在各種協議和格式，例如Opentelemetry的OTLP或Prometheus格式。 *Storage* - 通常，人們期望攝入的信號儲存較長的時間（幾週，幾個月甚至幾年），從而允許歷史查詢和數據分析。但這並不總是必須這樣，因為存在純記憶解決方案（例如CNCF Pixie）並具有其用途，例如實時調試，僅保留數小時的（揮發性）數據。 *Query* - 我們用於*asking a backend a ques- tion*的更精確和常見的術語是 *query the backend*。 *query*的含義從後端到後端可能會有所不同，並且可以從聲明性的樣式（例如SQL）到API調用，例如許多跟踪後端的情況。您還會經常遇到特定於域的語言（DSL），例如Promql，並且您會在本章中看到這些DSL的幾個示例。考慮到這一點，讓我們繼續進行各種日誌的第一個，可能是最廣泛的後端。

5.2 Backend destinations for logs

在本節中，我們將粗略地查看可用於攝入日誌並查詢的後端目的地。正如我們在第2章中討論的那樣，日誌是時間戳，典型地捕獲事件。擁有（希望結構化的）有效載荷；並且主要用於人類消費。大量不同的日誌類型和格式數量使有時很難弄清楚最佳後端是什麼。通常，這在很大程度上取決於用例和您的喜好，例如開源和打開標準，或者您可能更喜歡託管產品而不是DIY方法（或反過來）。話雖如此，當然，這也適用於指標和分佈式痕蹟的其餘後端，讓我們看看我們在這裡擁有的東西。

5.2.1 Cloud providers

至少每個雲提供商都有其產品中的服務，使您可以登陸日誌，更特別地（重點關註三大高度標準）：

AmazonCloudWatch Logs（http://mng.bz/yqjp）在日誌組中組織了各種日誌，這些日誌又包含一個或多個日誌流（考慮到由日誌事件組成的一個永無止境的事件）。它使用一種原則方式來攝取日誌並提供嵌入式指標格式（EMF; http://mng.bz/nwa5），該格式允許您將自定義指標與日誌嵌入。換句話說，與純的日誌格式相比，EMF更多的是以事件為中心（支持高飲食性）。您可以使用CloudWatch Log Insights，它提供了專門構建的查詢語言來查詢您的日誌。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

AzureMonitor Logs（http://mng.bz/mbgq）允許您從Azure中的受監視資源（從資源組到應用程序級別）收集，組織和分析日誌和性能數據。它在工作區中組織日誌，並用Kusto查詢語言（KQL）編寫的查詢，這與您在CloudWatch Logs Insights中找到的相似。 它具有一個有趣的功能，稱為*functions*，使您能夠重複使用查詢邏輯以及豐富的導出功能（從存儲到活動中心，基於各種目的地的規則）。 GoogleCloud Loggging https://cloud.google.com/logging/）是Goo- gle Cloud的操作套件的一部分，其中包括存儲，查詢和日誌路由。作為頂級單元，Google Cloud Logging使用日誌來組織條目。該日誌包括可能源於Google Cloud Services或第三方（平台級）以及應用程序的日誌條目（事件）。 在這種情況下脫穎而出的是，日誌路由的目的地的靈活性是：從存儲在雲儲物桶中的JSON文件（想想S3桶的等效）到大查詢（一個大規模的，大規模的，適當的SQL SQL分佈式查詢引擎，用於數據倉庫）到Pub/sub（Google sub的事件流）到可以與其他clucs supters-Bucks Supports-docts-with-with-with-with-with-with-with-with-with-with-whits-whits-whits-what suds-whits-what suds-what suds-what suds-whits clubs-時期。

讓我們看一個使用CloudWatch的具體日誌示例。檢查以下CloudWatch日誌見解查詢（其中CloudWatch報告了它掃描了3.7 GB的數據，涵蓋310萬條記錄）：

Selects fields @timestamp and @message for output; we expect to see a table with two columns as a result.

Sorts by time, descending, or more simply put, most recent event first

```
fields @timestamp, @message
| sort @timestamp desc
| limit 25
```

Only shows 25 results in the output

在上述CloudWatch日誌中使用限制25的警告查詢示例，我們指示CloudWatch將顯示的輸出限制為25，但是CloudWatch仍然需要掃描大量日誌，具體取決於所選的時間範圍。 時間範圍以及掃描數據的量，您應該始終考慮。一方面，查詢返回的查詢可能需要很長時間，最多需要幾分鐘，這遠非交互式用法。另一方面，您通常會收取掃描的數據量。

圖5.1顯示了前面的CloudWatch日誌的結果Insights查詢。通常，雲提供商將所有系統級日誌（有時稱為*platform logs*，例如syslog和journald，以及cloud-platform/api logs）將其置於自己的產品中，這使其成為一個很好的默認起點，尤其是對於所有非應用程序級logs。如果您很好奇如何將應用程序級的日誌路由到這些目的地，請參閱

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

第4章，我們在其中討論了代理商的選項和用法。接下來，讓我們看看這個領域的開源產品。

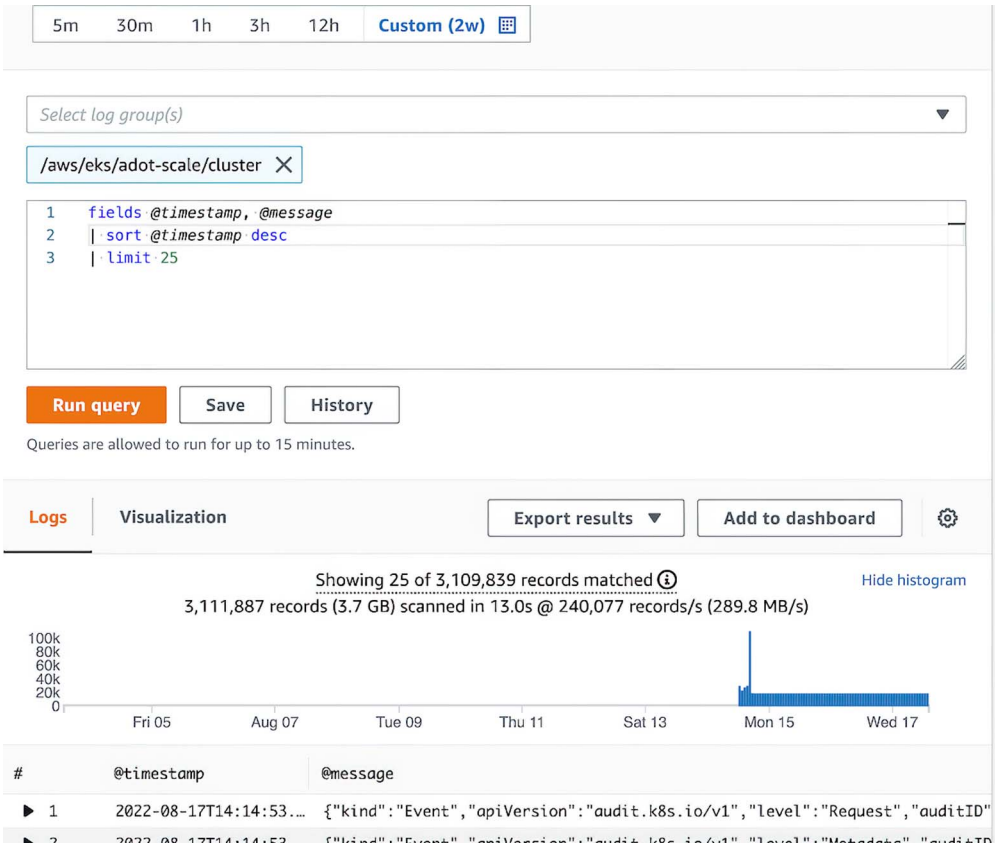


圖5.1 CloudWatch Logs Insights查詢結果顯示25個最近添加的日誌事件

5.2.2 Open source log backends

沒有那麼多廣泛使用的，基於開源的日誌後端可用（與指標和痕跡相吻合，儘管如您稍後會看到的那樣，日誌和跟踪後端具有許多特徵，並且後端通常會類似地存儲它們）：

Elasticsearch and OpenSearch - 我將Elasticsearch（https：//www.elastic .co/elasticsearch/）和OpenSearch（https://opensearch.org/）一起使用 - 簡短的ES/OS，即將前進，前進 - 實際上是兩個項目和產品的家族（託管服務）。 他們倆都擴展了Apache Lucene（https://lucene.apache.org/），這是一款用Java編寫的資深，全面的文本搜索引擎，使其易於用於信號攝入和

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

查詢，即使在分佈式設置中也是如此。 OpenSearch最初是由亞馬遜贊助的彈性搜索的叉子，並在2021年7月達到了其1.0版本的一般可用性（http：//mng .bz/a12j）。隨著時間的流逝，ES/OS代碼庫預計將出現，但從一開始就差異，每個項目都在分發每個項目。但是，ES和OS的事實是，它們是日誌優先的後端（用於對日誌的存儲和檢索進行了優化）。提醒我們，在第4章中，我們將Opensearch用作Fluent Bit Agent的目標。

我不會在這本書中詳細介紹ES/OS，因為有關該主題的資源可用，因此，如果您想深入研究，我建議您閱讀Mad-Husudhan Konda（2023; https; https：//wwwwww.manning.com/books/books/elasticsearchs/elasticsearch-in-in-in--in---in---in-------in--in------- v1}的出色第二版。

我們將在本章的後面跡線的上下文中回到ES/OS主題，因為這是一個信號類型ES/OS也決定涵蓋當今（也可以提供對指標的支持；但是，它不如其他兩種類型那樣充分吹牛）。

Grafana Loki (https://grafana.com/oss/loki/) - 這是根據AGPLV3許可發布的水平可擴展的，mul-Titenant的日誌聚合系統，其設計受Prometheus的啟發。它不會索引日誌本身的內容（如ES/OS DO），而是為攝入的日誌流的一組標籤。您已經在第2章中看到了Loki的行動，這肯定是可以仔細研究的事情。目前，很少有基於它的託管產品存在。 *ZincObserve (https://github.com/zinclabs/zincobserve)* - 這是您可以用於索引和查找日誌的搜索引擎，重點是降低日誌存儲成本。

現在，讓我們繼續進行該領域的商業產品。

5.2.3 Commercial offerings for log backends

為日誌後端提供了商業提供的空間，您將在所有這些方面找到一些共同的功能，例如付費攝入，強大的多端支持以及與其他信號源的廣泛內置集成，從應用程序到雲提供商產品。產品包括但不限於以下內容：

Splunk (https://www.splunk.com/) - 既定的日誌有能力，具有一系列企業特徵和強大的OpenteMetry承諾*Instana (https://www.instana.com/)* - 由IBM獲得，並提供許多相關的功能和集成> *SolarWinds (https://www.solarwinds.com/)* - 以前是獨立的saas產品，以及啟動的啟動．雲原生域中的產品（使用OpenSearch和Jaeger）

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

無論您在哪裡看，日誌總是在使用，因此問題不是*if*，而是*which*後端，您應該使用。我們將在本章後面的後期回到選擇標準，但是目前，您可以放心，您將需要日誌後端。這樣，您大致了解了日誌後端可以做什麼以及存在哪些選項，所以讓我們繼續進行指標。

5.3 Backend destinations for metrics

時間序列數據庫（TSDB）是指標類型的存儲和查詢signals的麵包和黃油。您可以使用TSDB來跟踪一系列不同的指標，從低級系統指標（例如容器的內存使用情況）到應用程序級別的指標，例如服務的調用數量。

什麼是TSDB，它如何工作？如圖5.2所示，主要索引是時間。 簿記單元是一個或多個時間序列的指標。 反過來，每個時間序列都會跟踪一個值，通常，您可以同時擁有許多這些時間序列（從某種意義上說，您可以使用它們

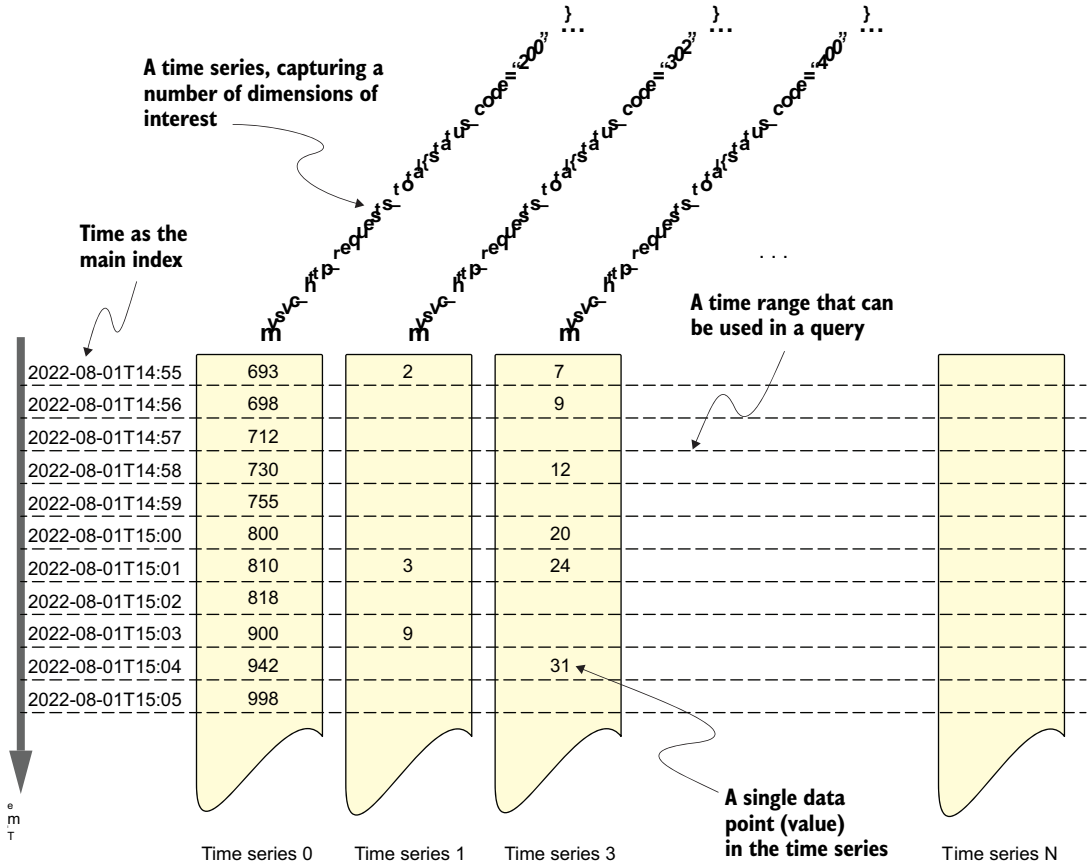


圖5.2時間序列數據庫概念，顯示mysvc_http_request_total metric的n時間序列

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

在查詢中，它們的價值很容易獲得）。與TSDB互動的典型方法是問：“在*x*到*y*的時間範圍內，與這些條件匹配的值或值聚集是什麼？”

TSDB中使用的主要索引是時間，通常分為相等的桶。為了使事情保持簡單，圖5.2使用微小的間隔，但通常，間隔可以在粒度上盡可能小至秒或毫秒。

每個時間序列都有一個唯一的名稱（例如，mysvc_http_request_total {status_ code = “ 400”}），捕獲了幾個感興趣的維度。 此處感興趣的維度是返回的HTTP狀態代碼（status_code），還有更多，例如HTTP URL路徑，HTTP方法等等，也可能是您要跟踪的內容。這個時間序列屬於類型計數器的指標。這意味著其值嚴格單調地增加，並捕獲了MySVC服務的HTTP API的總數。在這種情況下，時間段之間的淨變化將為您提供一段時間內的調用數量。 按時間和唯一時間序列解決的單個數據點表示值。在我們的情況下，如圖5.2所示，這意味著在2022-08-01T15：04時，TSDB總共看到了31個調用，導致了400 HTTP狀態代碼。

通常，您想查詢一個時間範圍，而不是時間的時間範圍。您可以通過指定開始（2022-08-01T14：58）和END（2022-08-08-01T15：01）來做到這一點。 例如，您可以詢問TSDB發生了多少個類型的調用；在我們的情況下，如果您對客戶端錯誤感興趣，則您會詢問在時間範圍內見證了多少個400 HTTP狀態代碼，答案為12（在範圍結束時，我們有24個，並且在開始時，我們有12個）。

基數爆炸

您可能會想知道，如果您開始在要跟踪的度量標準中添加尺寸，將會發生什麼，而指標是所有相關時間序列的傘術語。在此檢查中，假設您感興趣的維度是用戶ID。

與HTTP狀態代碼相比，該狀態代碼具有少量的閥門（從1xx到5xx），用戶ID可能具有許多值。如果您的應用程序成功，可能會為數十萬個用戶提供服務，這意味著TSDB必須攝入並存儲數十萬個時間序列（僅對於單個指標，並且您可能需要跟踪成千上萬的度量標準）。

我們稱之為*cardinality explosion*的數字可以乘以數量的數字可以乘以數量的許多不同值的想法，並且可以導致查詢時間緩慢，因為TSDB需要在給定時間範圍內檢查每個時間範圍的每個時間序列，即每個維度。系統屬性為某些類型的可觀察性用例製作了TSDB，而不是最佳擬合：至少，您應該意識到這一點，並避免在其基數方面添加您知道或懷疑“爆炸”的尺寸。

如果您想深入研究TSDB及其用例的主題，請考慮閱讀Ajay的文章“什麼是時間序列數據？定義和示例”

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Kulkarni，Ryan Booz和Attila Toth（https://www.timescale.com/blog/time-series-data/）。現在，讓我們通過查看可以用作TSDB後端的幾種開源和商業產品來切換齒輪並實用，尤其是在兼容兼容的情況下。 我們在第4章中介紹了Prometheus的基礎知識，在這裡不會重複它們。在本章中，我們將重點關注大規模生產用例，換句話說，當一個普羅米修斯的實例通常不再足夠時。讓我們再次查看雲提供商，開源產品和商業產品的選項。

注意Prometheus是根據設計而不是水平擴展的。 也就是說，您可以添加更多的RAM和本地連接的持久存儲，以垂直擴展Prometheus，以處理更多的指標或處理更長的時間段。您無法期望的是，在這種分佈式設置中為您管理多個Prometheus的多個副本，這意味著遙測數據是為您管理的。

如果您確實有用例，例如Kubernetes群集，您想從一個地方（聯邦）從許多機器（聯邦）中降落指標，或者您希望能夠查詢長時間的長時間，例如過去三年，以及一個Prometheus實例，您無法處理任務，您正在尋找長期存儲（LTS）解決方案。以下選項可以處理這些用例。

5.3.1 Cloud providers

至少每個雲提供商都有其產品中的服務，可讓您攝入一些開源度量以及基於普羅米修斯的指標，此外還可以攝取其本地產品。 以下是一些最突出的超級評分選項，可提供平台和應用程序攝入：

Amazon CloudWatch Metrics (http://mng.bz/gBne) - 從100多個AWS服務（vended Metrics）中劃分指標，並將其保存15個月，從而使您可以平均查詢最新的數據和歷史數據。您可以在某些條件下使用屬性查詢語言以及SQL（通過CloudWatch指標見解）來查詢指標以及警報（稱為警報）。 它還帶有儀表板，可以提供開箱即用的見解。 *Amazon Managed Service for Prometheus (http://mng.bz/e169)* - 一種使用CNCF皮層在其數據平面中使用CNCF皮層的Prometheus-兼容監視服務。您可以通過Prometheus Remote Write接口（http://mng.bz/ppe5）攝入指標，並使用promql查詢它。 *Amazon Timestream (http://mng.bz/Ox0K)* - 一個無服務器時間序列數據庫service for IoT和操作應用程序，提供基於SQL的查詢語言，並使您可以在更傳統的大數據用例中使用它。 *Azure Monitor Metrics (http://mng.bz/Y1Bo)* - 允許您從Azure中的受監控資源收集，組織和分析指標。它支持邏輯上小組時間序列的名稱空間，以及豐富的探索以及細粒度的存儲和保留選項。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Google Cloud Monitoring (https://cloud.google.com/monitoring) - Goo-Gle Cloud的操作套件的部分； 它使您可以從Google Cloud，AWS，託管正常運行時間探針和應用程序儀器中收集指標，事件和元數據。

Google Cloud Managed Service for Prometheus (http://mng.bz/GyaM) - 基於Prometheus的收集者作為Kubernetes中的守護程序，並僅通過將目標刮擦到共裂的節點上並將刮擦的指標推向中央數據存儲（Monarch）來確保可擴展性。 您可以使用PROMQL查詢指標，以獲取警報，例如，作為Grafana中的數據源。

圖5.3顯示了一個示例指標資源管理器輸出 - 在這裡，Metrics Explorer是Azure Monitor指標服務的一部分。接下來，讓我們看一下指標LTS和Federation的開源優惠。

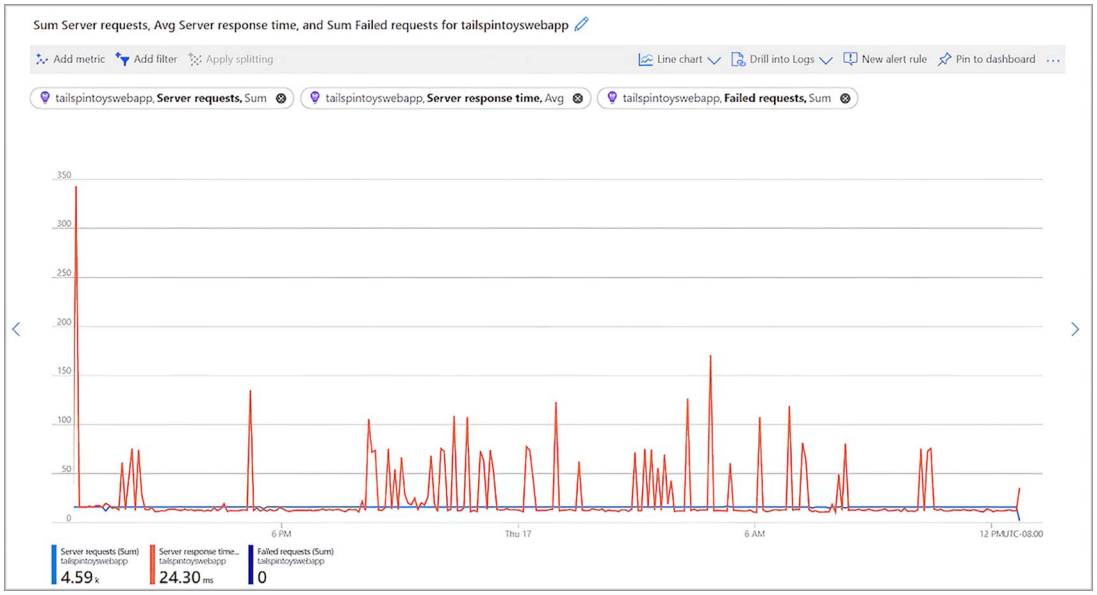


圖5 .3 Azure Metrics Explorer示例輸出（Azure Docs，http：//mng.b 來自/y1bo）

5.3.2 Open source metrics backends

在可用的選項方面，指標後端空間很擁擠，對您有好處，但是跟踪發展並做出最佳工作決定也可能具有挑戰性：

CNCF Cortex and Thanos - 兩個CNCF項目具有不同哲學的項目，以了解如何攝入和查詢Prometheus次數。 Cortex（https：// cortexmetrics .io/）用於Prometheus的Amazon託管服務；和Thanos（https：// thanos.io/），由不可能建立的，在多種情況下都使用

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

從Amadeus到eBay到SoundCloud再到Tencent的公司。 Cortex具有推動方法（使用Remote_write Prometheus界面），可以使Pro-Metheus艦隊攝入指標，而Thanos最初具有拉動方法（使用遠程_RED API），從而使一個人可以從許多Pro-Metheus設置（但是現在，它也支持推動推動）。

Clymene (https://clymene-project.github.io/) - 分佈式系統的時間序列數據和日誌集合平台，該平台允許在不同類型的數據庫中存儲信號，例如日誌或指標。它是一個相對較新的系統，但是在設計中，它是雲本地。

Grafana Mimir (https://grafana.com/oss/mimir/) - 根據AGPLV3許可證可用的Prometheus的水平可擴展，高度可用，多租金的長期存儲選項。它始於2022年，當時是CNCF Cortex Project的叉子，Grafana Labs增加了許多改進，也使其成為SaaS產品的一部分。 *Icinga (https://github.com/icinga/icinga2)* - 一個開源監視系統，檢查網絡資源的可用性，通知用戶中斷並生成用於報告的性能數據。您找不到在雲本地設置中廣泛使用的ICINGA，但是您可能仍然需要能夠與之互動。
InfluxDB (https://github.com/influxdata/influxdb) - 一個流行的開源時間序列平台，涵蓋了廣泛的功能，包括數據的離線過程，警報和探索，並具有良好的Prometheus兼容性。 *LinDB (https://github.com/lindb/lindb)* - 一種可擴展，高性能且高度可用的分佈式時間序列數據庫，該數據庫具有其自己的SQL樣查詢語言，稱為Linql。 *M3DB (https://m3db.io/)* - 從Uber起來，此時間序列數據存儲由Walmart，LinkedIn和許多其他數據存儲使用。現在，還可以通過ChronoSphere獲得它的縮放：https：//archive.fosdem .org/2020/scheple/event/m3db/），能夠處理數十億個指標（請參閱此FOSDEM 2020的談話：https：//archive.fosdem .org/2020/scheple/event/m3db/）。 *Nagios Core (https://github.com/NagiosEnterprises/nagioscore)* - 一個開源應用程序，使您能夠監視系統，網絡和基礎架構。 *Netdata*（https://github.com/netdata/netdata)-開源，高模式基礎架構監視和故障排除工具（代理和UI）。
Nightingale (https://github.com/ccfos/nightingale/blob/main/README_en.md) - 您可以用來包裝和增強程序，AlertManager和Grafana組合的雲本地監視系統。
OpenTSDB (https://github.com/OpenTSDB/opentsdb) - 在HBase頂部寫的分佈式可擴展的TSDB。 它的鼎盛時期是在2010年代中期，當時HBase被廣泛使用。
Promscale (https://www.timescale.com/promscale) - 基於PostgreSQL建立的Prometheus指標和OpenTelemetry Traces的統一可觀察性後端，而TSDB擴展名為TimesCaleDB（https://www.timescale.com/）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

QuestDB (https://github.com/bluestreak01/questdb) - 一個高性能的開源SQL數據庫，用於從金融服務到可觀察性的應用程序。它包括PostgreSQL線協議的端點； 使用InfuxDB線方案，高通量，模式攝入；以及查詢，批量進口和出口的REST API。

Zabbix (https://github.com/zabbix/zabbix) - 開源分佈式監控解決方案。 像此列表中的其他人一樣（Nagios，Icinga和OpentsDB都會想到），如今它在雲本地設置中並未廣泛使用，但您應該意識到這一點。

一些開源項目被包裝並提供為服務。讓我們現在繼續討論這個話題。

5.3.3 Commercial offerings for metrics backends

那裡 重新提供專門用於指標的產品，包括Follo 翼：

Chronosphere (https://chronosphere.io/) - 基本上是“ M3DB作為服務”。 他們在功能和OpenTelemetry兼容性方面取得了良好的進步。

VictoriaMetrics (https://github.com/VictoriaMetrics/VictoriaMetrics) - 一種使用集成的TSDB的監視解決方案，您可以使用獨立的或作為服務。雖然Prometheus合規性（https://github.com/prometheus/comliance）是在寫作時，是Victoriametrics的增長領域，但總體產品是雲原生和強大的。

在大多數（如果不是全部）中，案例（Kubernetes，ecs），您想要和需要指標，並與後端一起。根據Prometheus兼容性和特定的使用模式（例如，攝入，存儲或查詢分發）選擇您的後端。

5.4 Backend destinations for traces

現在，我們可以查看攝入和查詢分佈式痕蹟的各種選項。在這個空間中，您通常會發現對Opentelemetry的廣泛支持，儘管在某些地方（例如PHP）仍然需要Zipkin Sup-Port。

5.4.1 Cloud providers

就像日誌和指標一樣，如果您想攝入和使用跟踪：您選擇的雲提供商還可以提供一些東西：

AWS X-Ray (https://aws.amazon.com/xray/) - Amazon的託管分佈式追踪產品，它使您可以分析和調試應用程序。 它曾經是獨立的服務，現在正在集成到CloudWatch中。 X射線已在OpenTelemetry上進行了標準，並捕獲了許多計算和數據庫AWS服務以及Java，Node.js和.NET中編寫的應用程序的痕跡。

Distributed tracing in Azure (http://mng.bz/zXWw) - 使用SDK的應用程序洞察力的部分。現在，應用程序洞察力還支持通過OpentElemetry分佈式跟踪。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Cloud Trace (https://cloud.google.com/trace) - Google通過OpentElemetry提供了支持GO，Java，Node.js和Python的分佈式跟踪的產品。

圖5.4提供了Google Cloud Trace的示例輸出，這是一個複雜的分佈跟踪後端。它顯示了跡線的分析，特別是延遲分佈。接下來，我們將介紹開源跟踪解決方案。

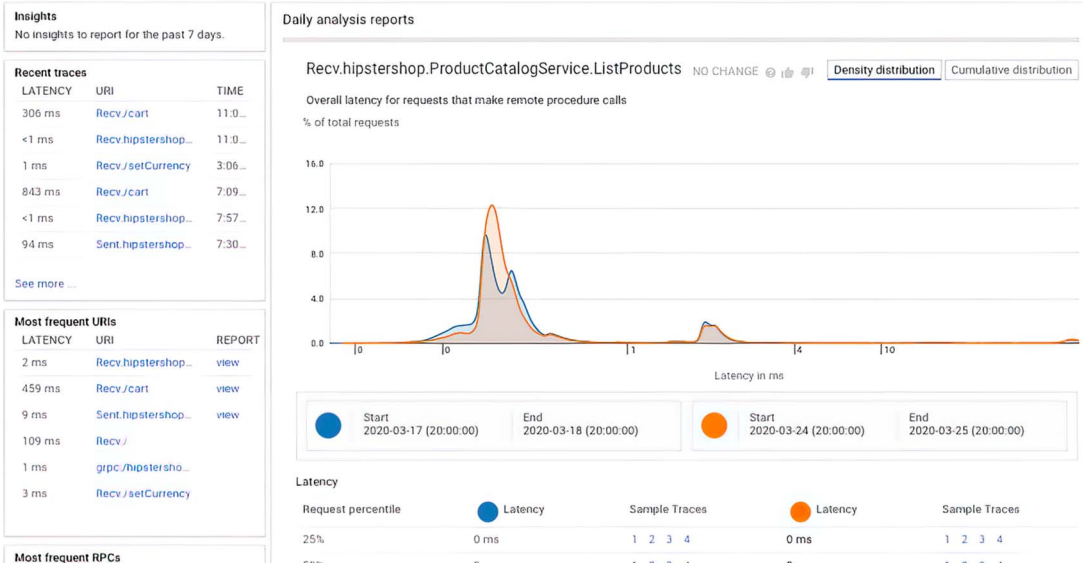


圖5.4在Google Cloud Trace中查看軌跡（Google Cloud Docs; https://cloud.google.com/trace/docs/ trace-overview）

5.4.2 Open source traces backends

開源跟踪解決方案沒有提供與標准後端一樣多的選項，但它們仍然提供一些選擇：

CNCF Jaeger and Zipkin - jaeger（https://github.com/jaegertracing/jaeger）和Zipkin（https://github.com/openzipkin/zipkin/zipkin）是由CNCF託管的。如今，Jaeger似乎是更積極的開發項目，但Zipkin仍然得到廣泛使用和支持。我建議您看看各自的社區（鬆弛和Github問題），並下定決心。我們將在第6章中介紹這兩個。即

Elasticsearch and OpenSearch - 這些選項已從日誌擴展到Sup-port處理跟踪。兩者都有出色的前端（Kibana和儀表板），我們將再次在第6章中覆蓋。

Grafana Tempo (https://grafana.com/oss/tempo/) - 根據AGPLV3許可證可用的分佈式跟踪後端。 它使用對象存儲（例如S3）

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

持久層，並與Grafana，Prometheus和Loki集成。它支持Jaeger，Zipkin和Opentelemetry攝入。

現在，讓我們繼續提供商業產品，與開源產品相比，這些產品非常豐富。這始於相關特徵，並一直到採樣和集成。

5.4.3 Commercial offerings for trace backends

最初稱為*application performance monitoring*（apm）供應商，如今，跟踪方面（或*observability solution*）通常突出顯示。我們將在第6章中詳細介紹這些產品。

儘管分佈式跟踪尚未達到與指標和日誌相同的採用水平，但對於雲本地設置而言，它越來越重要。即使您沒有圍繞它的短期策略，也考慮在較小的項目或產品中進行測試。 現在，我們將把齒輪轉移到討論一類後端，以幫助解決上述基數爆炸的挑戰：柱狀數據存儲和格式。

5.5 Columnar data stores

有許多所謂的基於柱狀存儲的引擎或簡單的數據存儲，您可以用作遙測數據的後端。*Columnar*是指數據在內存或磁盤上列出的方式，從而提高查詢性能。

面向行的數據庫，例如MySQL和PostgreSQL，將所有數據保留在記錄中，並在內存（或磁盤上）密切接近記錄。這意味著，在一般面向行的DBS中，非常適合在線交易處理（OLTP）用例（例如電子商務），您就可以使用記錄的更新。它們是針對閱讀和寫作排的優化，當您擁有許多具有潛在數據量的本性交易時，這是非常合適的，例如，在在線Web應用程序中插入或更新單個記錄時，就是這種情況（例如，將產品放入購物籃中，在社交媒體網站上關注某人等）。

面向列的或柱狀數據存儲（請參閱Mike Stonebraker等人，“ C-Store：面向列的DBMS”。第31屆VLDB會議的會議記錄，Trondheim，Trondheim，2005年；另請參見https：//people.brandeis.brandeis.edu/brandeis.edu/ eedu/ eede toctor auternecty comentery comentery commerationals to Plume toter comploty to Plume toters toters數據對於所謂的在線分析處理（OLAP）用例，一個不錯的選擇。 在這些情況下，您的交易很少，但正在處理大量查詢。寫入路徑是時間序列數據的攝入，例如，讀取路徑是在一個時間範圍內的臨時查詢。因此，這些柱狀數據存儲非常適合歷史（時間序列）數據，我們想從大型數據集（數十億行，可能是數據的數據）或換句話說，可以從大型數據集（數十億行）中提取可行的見解。讓我們仔細研究一下：圖5.5顯示了面向行和麵向列的存儲策略。

從邏輯佈局開始，列（垂直）和行（水平）上的表格視圖，特定值具有坐標cirj（例如，第三個值的值

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

列，第二行將由C2R1引用）。現在，當您存儲面向行的閥（圖5.5的左側）時，您會從第一行開始，然後繼續進行第二行，依此類推，從而導致存儲佈局（在磁盤或內存中）存儲了第一行，第二行，第二行，等到最後一行。在遺憾的是，當存儲以列的為導向的值（圖5.5的右側）時，您可以從第一列開始，然後移至第二列，依此類推，依此類推，導致存儲佈局（在磁盤上或內存中）存儲第一列，第二列，第二個col-umn等，等到最後一列，直到最後一列。

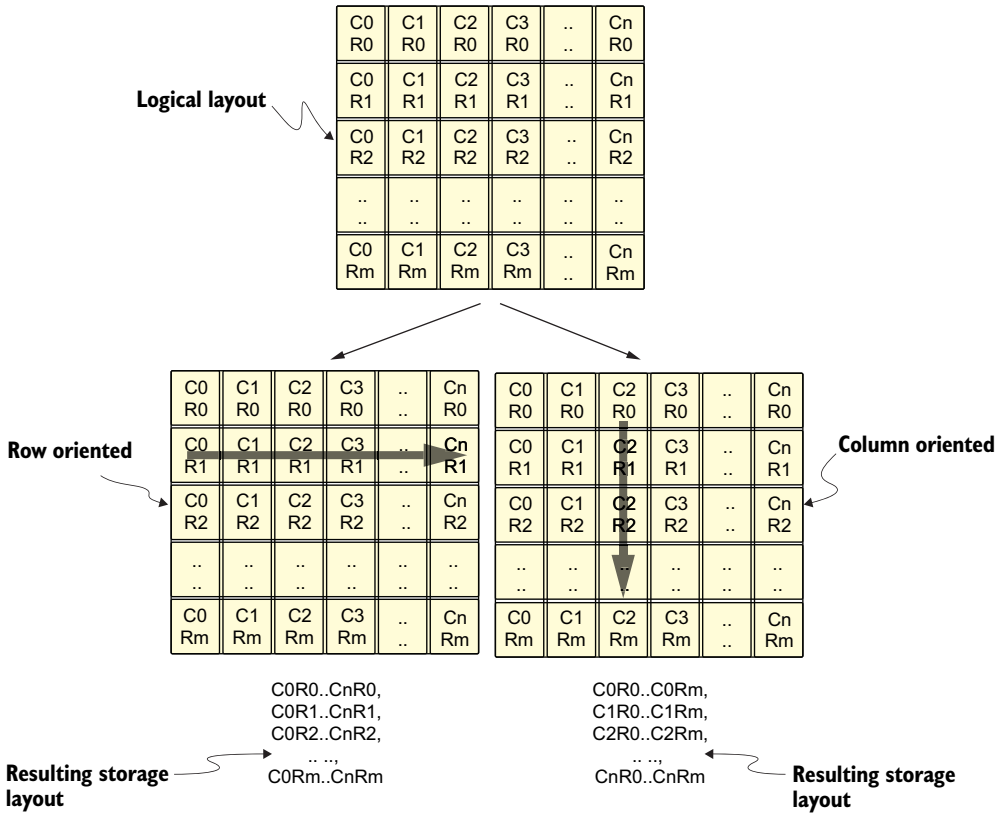


圖5.5面向行與列的存儲

您可能想知道，為什麼要打擾面向列的存儲？事實證明，Columnar存儲在OLAP用例中具有許多優勢，其中包括我們的可觀察性用例。

在可觀察性用例中，您可以具有許多感興趣的維度，從而導致較高的基數和許多列。對於任何給定的查詢，可能只有幾個與查詢有關的信息。柱狀數據存儲允許更快和更少的掃描，因為

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

需要閱讀的不必要的值較少（然後立即被丟棄，因為它們不會貢獻答案）。

此外，面向列的設計使您可以應用存儲優化的技術，例如差分存儲，例如在大猩猩TSDB的上下文中開發的XOR壓縮facebook（請參閱Tuomas Pelkonen等人，Gorilla：“Gorilla：快速，可縮放，能夠縮放，可縮放的，內存的時間序列”的數據庫。 http://www.vldb.org/pvldb/vol8/p1816-teller.pdf）and run-長度編碼（rle;請參閱D. Abadi，S。Madden，M。Ferreira，M。Ferreira，“集成了與方向性的數據庫系統中的綜合和執行”。 https：// doi .org/10.1145/1142473.1142548），產生較小的足跡和較少的資源使用情況。

除壓縮外，柱狀設計還可以實現矢量執行（每個CPU週期多個數據項）和在single機器的上下文中進行有效的分區，並分解複製以縮放縮放，使您可以回答亞秒時框架中數以平等的行。

使用此策略的柱狀數據存儲，柱狀格式或服務的突出示例包括以下內容：

Apache Cassandra (https://cassandra.apache.org/) - 在Java中寫入c ++變體，scylladb，也可用（https://www.scylladb.com/）。 *Apache Druid (https://druid.apache.org/)* - Airbnb，Alibaba，eBay，Lyft等的開源實時分析數據庫，具有可觀察性用例。您可以在kubernetes（http://mng.bz/0kdp）中運行druid。
Apache Pinot (https://pinot.apache.org/) - LinkedIn（http://mng.bz/keop）使用的開源實時分佈式OLAP數據存儲，Uber，Slack，Stripe和更多公司。
ClickHouse (https://clickhouse.com/) - 一個開源列數據庫，由Yandex於2008年開發。您可以將ClickHouse用於所有信號類型，無論是單台計算機還是在各種分佈式設置中（以Apache Zookeeper作為協調員的碎片和授權數據）。 在Kubernetes上運行它時，您應該考慮使用Altinity操作員進行ClickHouse（https://github.com/altinity/clickhouse-operator）。 Clickhouse的採用率超過了，包括Uber（Logs），Cloudflare和Signoz，在該塊中，它被用作O11Y解決方案中的構建塊。
FrostDB (https://github.com/polarsignals/frostdb) - 一個允許半結構化模式的柱狀數據庫，使用Apache Parquet進行存儲以及用於查詢的Apache Arrow，該數據庫由Polar Signals開發為開源。 *Snowflake (https://www.snowflake.com/)* - 作為服務提供的集成平台。 它具有物理分離但邏輯整合的存儲，計算和全球服務層。

如果您有興趣了解有關上述一些數據商店的實施詳細信息的更多信息，我建議閱讀出色的文章“比較

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Roman Leventov（http://mng.bz/9drx）的開源OLAP系統：Clickhouse，Druid和Pinot”。

在這種情況下，有趣的是，柱狀存儲不限於後端。例如，在OpentElemetry中，正在進行的工作以在收集器和協議中引入柱狀編碼（https://github.com/open-telemetry/oteps/pull/171），以加快處理並提高性能和可擴展性。

> 提示
> 在過去的十年中，柱狀數據存儲和數據庫，格式和開源程序的空間爆炸了。實際上，我的第一份行業工作（經過12年的研究）是在現已退休的Apache Drill Project的Hadoop啟動的背景下（這與Google Dremel的開源版本相似，Google的Dremel構成了Big Query的基礎）。
>
> 您可能感興趣的數據存儲有許多有趣的數據存儲，包括MonetDB，Hypertable，Apache Kudu和GreenPlum。 詳細介紹它們超出了本書的範圍。如果您想了解更多信息，我建議您閱讀馬丁·克萊普曼（Martin Kleppmann）的出色書 *Designing Data-Intensive Applications*（O’Reilly，2017; https：//dataintisgensing.net/）和*Fundamentals of Data Observability*（O’Reilly，2023; http：//mng.bz/jpqz）the and and and and and and and andy andy andy andy petrella。

在這麼多理論之後，我們應該採取一些行動！因此，讓我們深入研究一個開源柱數據存儲的示例，該數據存儲在過去幾年中獲得了很多吸引力：Clickhouse。

在這次演練中，您將（再次）使用HO11Y發電機，配置為通過Fluentd驅動器將其日誌發送到OpentElemetry收集器中，並使用Fluent Forward接收器（http://mng.bz/wzd4）和clickhouse Exporter（herm clickhouse Exporter（herm youss to Ingest to Ingest youss youss），請使用流利的前向接收器（http://mng.bz/wzd4）將其發送到OpentElemetry collector（http://mng.bz/wzd4）。使用SQL查詢日誌。

興奮的？我們走吧！

我們從下一個列表中顯示的Docker組成的配置開始。您知道以前示例中已經有大多數設置，因此我們將重點關注本設置的新部分。

清單5.1 Docker的片段構成了Clickhouse設置的配置

```
ho11y:
    image: public.ecr.aws/mhausenblas/ho11y:stable
    ports:
      - 8765:8765
    logging:
      driver: fluentd
      options:
        fluentd-address: "localhost:8006"
```

Here, we provide the Docker Fluentd driver with the address of the fluentforward receiver (part of the OpenTelemetry collector pipeline).

Otel-Cololector：
圖像：Otel/Opentelemetry-Coltector-Contrib：0.55.0

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
    command: ["--config=/conf/otel-collector-config.yaml"]
    volumes:
    - ./otel-config.yaml:/conf/otel-collector-config.yaml
    ports:
    - "8006:8006"

  clickhouse-server:
    image: clickhouse/clickhouse-server
    ports:
    - "8123:8123"
    - "9000:9000"
    ulimits:
      nproc: 65535
      nofile:
```

We need to expose the Fluent Forward receiver port here so that we can receive log lines from Docker.

This is where we set up ClickHouse: the port 8123 for the Web UI and port 9000, the default native TCP endpoint for clients (allowing the OpenTelemetry exporter to talk to it).

There are several low-level Linux settings, such as the number of processes and files allowed for ClickHouse that we're setting here.

有關ClickHouse配置和良好實踐的更多詳細信息，請參見Click-House操作文檔（https://clickhouse.com/docs/en/en/operations/settings），Spe-ceelle，ClickHouse Networking（http://mng.bz/eqyo）上的出色文章。現在，我們將繼續進行遙測代理設置。 以下列表顯示了我為ClickHouse示例組合的OpenTelemetry Collector配置。

列表5.2 ClickHouse設置的OpenTelemetry收集器配置

```
receivers:
  fluentforward:
    endpoint: 0.0.0.0:8006

processors:
  batch:
    timeout: 5s
    send_batch_size: 100

exporters:
  logging:
    loglevel: debug
  clickhouse:
    dsn: tcp://clickhouse-server:9000/default
    ttl_days: 3

service:
  telemetry:
    logs:
      level: debug
  pipelines:
    logs:
      receivers: [ fluentforward ]
      processors: [ batch ]
      exporters: [ logging, clickhouse ]
```

We configure the Fluent Forward receiver here to listen on all addresses on port 8006 for TCP connections. If all works, the Docker Fluentd driver will connect here.

Here, we set the batch size to 100 log lines. In practice, this would be much higher, likely in the thousands (depends on how long you want to wait until the logs are accessible in ClickHouse). I chose this low number to speed up things for you to see results.

This is the place where we set up the ClickHouse exporter, configuring it with the address of where the ClickHouse server is running. From the server perspective, our exporter is just another client talking TCP with it. The ttl_days setting limits the time data is kept around (not really important for our demo).

Finally, here, we assemble the pipeline using the components we configured previously.

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

ph！那又是雅姆爾又吵架了，所以讓我們開始使用它。在新的term-neal會話中，轉到CH05/ClickHouse/，然後執行（自獨立，不需要外部負載生成器）：

```
docker-compose -f docker-compose.yaml up
```

現在，您可以訪問http：// localhost：8123/play，這是Clickhouse的遊樂場，然後輸入以下SQL查詢（作為參考，在我的計算機上，需要一些毫秒來讀取約1,000行）：

```
SELECT
  Timestamp as ts,
  Body AS payload,
  JSONExtractString(Body, 'traceID') AS traceID
FROM otel_logs
WHERE
  match(payload, 'invoked') AND
  ts >= NOW() - INTERVAL 30 SECOND
```

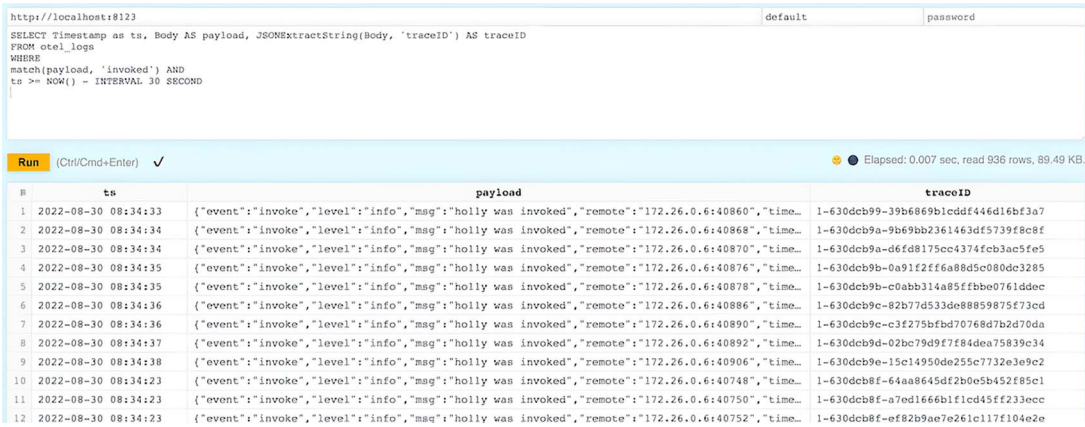
We pull out certain columns of interest, especially the associated trace ID.

We do a regex match against the body of the log message, requiring it to contain the string invoked.

We limit the output to the last 30 seconds.

The table the OpenTelemetry ClickHouse exporter set up for us to query.

SQL查詢執行的結果是您在圖5.6中看到的結果。注意輸出表上方右側區域中的查詢統計信息。



#	ts	payload	traceID
1	2022-08-30 08:34:33	{"event":"invoke","level":"info","msg":"holly was invoked","remote":"172.26.0.6:40860","time…	1-630dcb99-39b6869b1cddf446d16bf3a7
2	2022-08-30 08:34:34	{"event":"invoke","level":"info","msg":"holly was invoked","remote":"172.26.0.6:40868","time…	1-630dcb9a-9b69bb2361463df5739f8c8f
3	2022-08-30 08:34:34	{"event":"invoke","level":"info","msg":"holly was invoked","remote":"172.26.0.6:40870","time…	1-630dcb9a-d6fd8175cc4374fcb3ac5fe5
4	2022-08-30 08:34:35	{"event":"invoke","level":"info","msg":"holly was invoked","remote":"172.26.0.6:40876","time…	1-630dcb9b-0a91f2ff6a88d5c080dc3285
5	2022-08-30 08:34:35	{"event":"invoke","level":"info","msg":"holly was invoked","remote":"172.26.0.6:40878","time…	1-630dcb9b-c0abb314a85ffbbe0761ddec
6	2022-08-30 08:34:36	{"event":"invoke","level":"info","msg":"holly was invoked","remote":"172.26.0.6:40886","time…	1-630dcb9c-82b77d533de88859875f73cd
7	2022-08-30 08:34:36	{"event":"invoke","level":"info","msg":"holly was invoked","remote":"172.26.0.6:40890","time…	1-630dcb9c-c3f275bfbd70768d7b2d70da
8	2022-08-30 08:34:37	{"event":"invoke","level":"info","msg":"holly was invoked","remote":"172.26.0.6:40892","time…	1-630dcb9d-02bc79d9f7f84dea75839c34
9	2022-08-30 08:34:38	{"event":"invoke","level":"info","msg":"holly was invoked","remote":"172.26.0.6:40906","time…	1-630dcb9e-15c14950de255c7732e3e9c2
10	2022-08-30 08:34:23	{"event":"invoke","level":"info","msg":"holly was invoked","remote":"172.26.0.6:40748","time…	1-630dcb8f-64aa8645df2b0e5b452f85c1
11	2022-08-30 08:34:23	{"event":"invoke","level":"info","msg":"holly was invoked","remote":"172.26.0.6:40750","time…	1-630dcb8f-a7ed1666b1f1cd45ff233ecc
12	2022-08-30 08:34:23	{"event":"invoke","level":"info","msg":"holly was invoked","remote":"172.26.0.6:40752","time…	1-630dcb8f-ef82b9ae7e261c117f104e2e

圖5.6 E Clickhouse Web UI操場，執行我們的SQL查詢和Showin G結果

提示，如果您發現自己與Clickhouse一起工作，我建議您查看ClickCat（https://github.com/clickcat-project/clickcat），這是一個漂亮而簡單的用戶界面，可讓您在ClickHouse中搜索，探索和可視化數據。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

另一個（相關的）類別是分佈式查詢引擎，除其他外，它可以使用柱狀數據格式。這些包括

Prestodb（https://prestodb.io/），由Facebook（Meta）開放來源。

在本章的最後一部分中，我們將齒輪轉移到一個熱門話題中：在這些選擇中，如何最好地選擇一個（或更多）後端。

5.6 Selecting backend destinations

選擇後端可能很難。 為了讓您了解大量的選擇，請查看可觀察性和分析中的CNCF景觀（https：//landscape.cncf.io/card-mode?category?category?category = observability-and-and-Analysis）空間。從開源到SaaS，它具有許多不同的選擇。

現在，您已經對後端域中的各種產品有了一個了解，讓我們將注意力轉向選擇適合您需求的後端。我推薦從工作量和要求後退。另外，請注意，最終使用多種類型的背面一致的頻率並不少見。 例如，您可以將CNCF Cortex用於指標LTS和Federation以及OpenSearch以進行日誌和跟踪存儲和查詢。 最後，請記住，某些前端（例如Grafana）作為規範的示例，可讓您將各種後端的信號匯總在一起，有效地構建一個玻璃玻璃，以滿足您的可觀察性需求。我們將在第6章中詳細討論這一點。

我們在早期的章節中討論了投資回報率（ROI），並與後端討論了這一點至關重要。讓我們首先研究成本方面。

5.6.1 Costs

在投資回報率的情況下，後端通常需要投資，通常在長期到長期範圍內，即數年。這是由於數據具有重力，這意味著大多數數據所在的大部分都定義了您的主要環境（思考本地與雲或不同雲提供商之間）。在環境之間傳輸數據會花費時間和金錢，您需要避免不必要的出口和進出數據傳輸成本。

更具體地說，用於後端的貨幣投資可能會崩潰

Ingestion costs - 像OpentElemetry收集器一樣，攝入（或寫入）信號進入後端。例如，對於兼容Prometheus兼容的後端，這可以以攝入的百萬美元樣本進行衡量。

Storage costs - 攝入後，需要存儲信號以進行長期檢索。這些費用通常不是您賬單的最大貢獻者（通常是攝入或查詢）。

關於存儲，通常有用的財產是“保留期”。這樣，您的提供商表示您的信號將存儲多長時間並可用

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

用於回答查詢。這可以是靜態的或固定的屬性（例如一年），也可以是您可以配置的東西（可能以及像CNCF Thanos一樣已經提供的指標後端中的降採樣策略）。

Query costs - 使用信號回答您的O11y問題意味著您需要能夠查找它們，掃描並查詢它們； 這些成本可以在處理的樣品中表達，也可以以更大的量為導向（例如，GB掃描）。 了解查詢在何處以及如何發生（除了明顯的臨時查詢您作為人類的互動性能）至關重要。

例如，想想您定義警報或警報的情況（第7章），然後在常規時間段內評估條件，以執行一個或多個查詢。同樣，從大規模上講，這可以概括，我看到了不知道這一點的客戶的驚喜（不是很好的）。

除了這些可變成本外，還可以有固定的成本，例如人均許可成本；但是，這些是相當可預測的（與可變成本相比，例如攝入或查詢），並且更常見於前端或多合一解決方案。在後端成本的背景下，還有一個考慮因素 - 佔據了手頭的實際任務，例如故障排除或性能優化 - 可能存在監管要求，以使某些類型的信號（通常是日誌）在幾年內保持幾年。

注意圍繞遠程信號的存儲和保護的監管要求在受管制的垂直行業（例如金融工業或醫療保健）中尤其相關。您可能需要在獲得認證的背景下證明此功能，例如處理通常需要支付卡行業數據安全標準（PCI DSS）合規性或隱私要求的應用程序的情況，在系統和組織控制（SOC）的背景下。

這是在後端的背景下進行的一個快速詞與構建：除非您是可觀察性供應商（或云提供商），請考慮不要從頭開始構建自己的後端。建立可靠和可擴展的東西可能需要花費很多年，如果您擁有*N*創新令牌或工程週期，它們通常會更好地投資於您的核心業務，以及與競爭的區別，而不是構建下一個時代的數據商店。

我並不是要阻止您建立基於內部可觀察性平台，例如，將各種開源後端集成在一起，並將開源前端作為單個玻璃窗格。但是，您應該意識到總擁有成本（TCO），因此請準備好在做出決定時考慮工程團隊，支持，培訓和安全補丁（僅舉幾例）。接下來，讓我們繼續談論開源和開放標準的主題。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

5.6.2 Open standards

以公開標準為基礎是一個不錯的選擇：嘗試避免可通過專有API訪問的後端，或者至少在如何使這些專有API符合（行業）標準的情況下有一個策略。

例如，在我們的可觀察性考慮因素的背景下，您想在支票列表中包含的東西（至少在路線圖上）的支持（至少在路線圖上）。每個主要的雲提供商和ISV現在已經支持OpentElemetry，請參見項目主頁上的各自的供應商列表（https://opentelemetry.io/vendors/），詳細介紹OpenTElemetry Support（分佈和OTLP端點支持）。

在這方面，並非所有信號類型都是平等的：

日誌傾向於使用許多開放標準和格式。標準化是不切實際的，因此，諸如OpentElemetry之類的標准通常會限制它們以宣布從Syslog（例如Syslog）宣布標準的規範映射。 指標越來越多地以Prometheus博覽會格式或其繼任者Openmetrics（具有示例性支持）。 但是，請注意，有很多老式的東西，尤其是來自或非雲的本地設置中，它們仍在使用石墨格式，InfluxDB Line proto-col或statsd之類的東西。 如今，重要的是要確保符合OpentElemetry的踪跡。

接下來，我們將研究攝入路徑中的一個共同挑戰。

5.6.3 Back pressure

當查看大規模攝入信號時，您可能會遇到與以高速率產生（例如日誌線或跨度）來處理的挑戰。為了處理背壓（即不超載後端），您可以在源，代理和後端之間使用隊列。這使您可以將生產者側（信號源或代理）和消費者側（後端）解脫。

例如，您可以在apache kafka（https://kafka.apache.org/）中使用opentelemetry collector的出口路徑，該opentelemetry收集器（http://mng.bz/n2zn）組件中的pipeline中的組件（我們在第4章中都在conformory querory in Conformory quee in o. quartory of the quartory of the Mortory querory in o. quare querory in o.發佈到KAFKA經紀人中，例如，後端可以是通過lambda功能在計劃中填充的S3存儲桶。接下來，我們將介紹如何管理太多值。

5.6.4 Cardinality and queries

在本節中，我們在選擇後端時討論兩個相關問題：紅衣主教和查詢。我們已經在“標准後端”部分討論了基數爆炸。 我們的意思是指標的維度（例如，會話ID或用戶ID）

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

可能會承擔高度變化的價值，從而在攝取和存儲指標（成本）以及查詢它們時會引起問題。在查詢的背景下，高基數可能會導致緩慢或懸掛的查詢 - 也就是說，查詢可能會在返回結果之前超時。

解決此問題的最佳方法是避免使用METCRICS後端時較高的基數。通過避免它，我的意思是仔細控制您刮擦（或收集）的哪些指標，並從實際使用的內容中向後工作。例如，您的指導可能是只有在儀表板或警報中積極使用的指標才能“允許進入”。您可以通過在代理中實現此邏輯來推遲該決定，例如，通過您的一個管道中的OpenTelemetry處理器。

另外，如果您不能或不想限制自己，則可以使用可以處理大量指標的專用數據存儲，例如Clickhouse或商業產品。 例如，HoneyComb最初花費大量時間從頭開始構建一個專有的數據存儲，以用於高心電圖信號。 就他們而言，這是有道理的，因為這實際上是他們的商業模式。同樣，這很快提醒您在這種情況下購買與構建。

通常，查詢性能（後端可以作為對查詢的反應返回結果的速度）是一個重要的主題。但是，細節超出了該章節的範圍。它確實在某些系統中有時間 - 速度折衷；例如，如果您處於其基數限制範圍並進行“警報”樣式查詢，則Prometheus和InfuxdB是快速的表演者。另一方面，分佈式系統在短期內往往會較慢，但處理遠程查詢（即“告訴我我們在上一季度的表現”）要好得多。

如果您有興趣設置自己的查詢性能TSDB基準測試，例如，請查看Aliaksandr Valialkin在“高級熱電學TSDB基準：Victoriametrics與TimesCaledB與InfluxDB”（http：// mng .bz/d4mmn）中共享的。

關於查詢的方式以及電線格式，一些常用的查詢語言，模式和格式如下（請注意，我在這裡有意混合運輸模式和查詢語言，以使情況變得簡單）：

> 對於日誌信號類型： - 具有Elastic-favored Lucene的查詢語法 - Grafana Labs的logql（https://grafafana.com/grafana.com/docs/docs/loki/loki/latest/logql/） - （http:// mng.bz/bmj0） - 根據RFC 5424 - Apache的常見日誌格式（https:/ https:/ https://en.wikipedia.orgiia.org/common – nttps://datps://datps：//datatracker.ietf.org/doc/rfc5424/） （https://docs.nginx.com/nginx/admin-guide/ Monitoring/Logging）

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

- GrayLog擴展日誌格式（gelf; https：//go2docs.graylog.org/5-0/ get_in_log_data/gelf.html） - 如果您使用Windows Log schema（http://mng.bz/d1an），以作為您使用Windows Event extorer-common –c eff torge; https://docs.nxlog.co/userguide/integrate/cef-logging.html）

在時間序列數據庫中或用於指標信號類型中： - Promql（http：//mng.bz/qrgw），Prometheus查詢語言 - Prometheus exposition Formats（http://mng.bz/rw7b）和open- Metrics（htttpers.ioperics.ioce.ioce.io – ioperics.ioce.ioce.ioce.ioce.ioce.ioce.ioce.ioio – ioperics.io – ioper – ioperics.io – iio.ioioetrics.ioce。 （https://www.influxdata.com/products/flux/），獨立數據腳本和查詢語言

對於跟踪信號類型，尚無針對痕蹟的廣泛建立的查詢標準，但是Grafana Labs正在研究此主題。 同樣，在這種情況下，OpentElemetry OTLP作為默認的線格式被廣泛採用。 對於類型配置文件的信號，PPROF（https://github.com/google/pprof）被廣泛使用；另請參見第9章，我們將在其中深入研究該主題。

請注意，對於上述大多數信號類型（排除的配置文件，預計將在2023年提供），Open Telemetry OTLP（http://mng.bz/v1kp）已經或很快就會是執行數據交換的標準方法。 在大多數情況下，您想選擇一個支持（開放性的）行業標準的後端，例如OpentElemetry或OpenMetrics，而不是專有（僅），但也請參見有關此背景下預期警告的上一節。

通過此選擇指南，我們已經完成了有關後端的章節的結尾。接下來，我們將討論前端（例如Grafana）和多合一解決方案，例如Apache Skywalking。

Summary

後端是您的可觀察性問題的真理來源。 後端以持久和長期的方式存儲信號，支持各種查詢語言，通常是聲明性的。 對於日誌，指標和痕跡後端有各種產品。 您可以在雲提供商提供的內容，開源（和自行操作）選項以及各種提供商的託管產品之間進行選擇。 某些後端（例如，在OSS空間中進行開放搜索以及可觀察性供應商的大多數商業產品）可以處理多種信號類型；但是，實際上，每種信號類型的後端可能會出現不同的後端。 能夠從後端中的一種信號類型轉移到另一種信號類型（在另一個信號類型上）（主要不同）對於回答任意問題至關重要。 基數爆炸是公制維度可以承擔許多不同值（想想用戶ID）的挑戰。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

TSDB，例如Prometheus，是指標信號類型（數值聚集體）的絕佳選擇，直到您遇到基數爆炸為止。 將柱狀數據存儲和格式用於高心電圖信號和用例。 請確保您對要存儲的確切要存放的內容有一個很好的了解（例如，日誌versus指標）。 由於數據重力，仔細選擇您的後端至關重要（在傳輸數據時避免大量出口成本）。 在選擇後端時，請注意攝入，存儲和查詢階段的直接和間接成本。 選擇支持有關查詢語言的開放標準，並在專有的標準方面使用電線，以最大程度地減少鎖定性並最大化互操作性。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

6 Frontend destinations

This chapter covers

什麼是前端目的地，存在哪些選項什麼是多合一的目的地以及可用的選項如何選擇前端和全面選擇

在第5章中，我們查看了可觀察性後端。本質上，這些是特殊的時間序列數據庫（TSDB），具有TSDB擴展名的通用關係數據庫或能夠存儲日誌，指標和痕蹟的柱狀數據存儲。一旦您攝入了後端的信號，就可以使用後端回答可觀察性問題，通常稱為*querying*。此過程可能是一個聲明性的過程，例如SQL或Promql的情況，或者是當務之急，例如，您告訴traces後端通過根跨度ID查找跨度。

但是，信號的存儲和能夠查詢它們只是達到目的的手段。可觀察性是關於了解系統的理解並從收集和攝入的遙測信號中獲得可行的見解。因此，作為可觀察性解決方案的用戶，您真正想要的是提出臨時問題並得到（及時）答案。

理想情況下，無論您詢問涉及的遙測數據的詢問的時間或量的數量，都應該發生這種情況。例如，假設

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

您每天攝入10 GB的日誌數據和10億個痕跡，並保留一年。當您進行故障排除時，您不希望這個事實減慢您的速度（等待查詢時間或花幾分鐘返回），因為這會破壞您的流動，可能導致您忽略問題，例如找到較慢的錯誤或根本沒有。

您問這些（可觀察性）問題的環境是什麼？ 這些環境可以是前端和全面的。按設計，前端是純圖形用戶界面（GUI），可讓您配置一個或多個數據源，用於創建儀表板或警報，例如Grafana。另一方面，將後端和前端組合為一個不可分割的邏輯單元。換句話說，您不能將多合一的GUI用作獨立的GUI，而只能與提供商為其設計的後端結合使用。

那麼，您如何提出不同的問題，如何給您帶來結果呢？您已經猜到了：實際上，它是為實現這一目標而使用的前端和多合一目的地。您可能在可觀察性社區中遇到的術語是*a single pane of glass*。在本書的上下文中，我的意思是一個單個地方或應用程序，您可以使用它來消耗所有信號，查詢並關聯不同的信號類型。讓我們從可觀察性的前端參觀進入深處。

提示
真人秀表明，將玻璃的單個窗格作為最重要的要求或優先級（可能來自管理）通常不是最有效地使用時間和金錢。是否有標準化可以在一個地方消耗許多信號，即使不是大多數信號？是的。會有例外嗎？也是。換句話說：通過考慮本章中提出的一種解決方案之一，目的是為此而實現，但是在不容易（容易）可能的情況下允許靈活性。

請注意，我並不是建議您應該放棄在一個主要前端（例如Grafana）上標準化的想法（將所有數據放在一個地方，包括相關性的支持非常有幫助）。 但是，請準備好將某些信號或源沒有涵蓋該前端涵蓋的情況，要求您使用專用工具。

6.1 Frontends

我們將首先討論兩個最受歡迎的開源前端：Grafana和Kibana。接下來，我們將查看可觀察性空間中使用的其他開源產品和商業產品。

6.1.1 Grafana

如今，Grafana（https://github.com/grafana/grafana）是Grafana Labs的LGTM堆棧（https://github.com/grafana)- loki，loki，grafana，tempo和mimir和mimir和mimir- iT和最重要的資產。 Grafana最初是Kibana的叉子（http：//mng .bz/x47w），當Grafana Labs的創始人TorkelÖdegaard正在尋找一個關注時間序列數據並提供以圖形為中心的儀表板的UI。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Grafana提供了幾種資源，但最重要的三個是

Data sources - 代表您要在儀表板或警報中使用的後端*Dashboards* - 一種組織相關可視化的方式*Alerts* - 在發生某些事情時向您發送通知（請參閱第7章，請參閱第7章）

現在讓我們看看Grafana及其最高級別的資源。首先，我們需要站起來一個grafana實例。 我們使用以下Docker組成的文件來完成此操作，您現在可以使用該文件啟動，到現在為眾所周知的Command docker-compose -f docker- compose.yaml（確保您首先轉到CH06/Grafana/）。

列表6.1碼頭組成Grafana實例的配置

```
version: "3.5"
services:
  grafana:
    image: grafana/grafana-oss:9.1.4
    ports:
    - "3000:3000"
    volumes:
    - ./grafana.ini:/etc/grafana/grafana.ini
    - ./ds.yaml:/etc/grafana/provisioning/datasources/datasource.yaml
    depends_on:
    - prometheus

  prometheus:
    image: prom/prometheus:latest
    ports:
    - "9090:9090"
    volumes:
    - ./prom-config.yaml:/etc/prometheus/prometheus.yml
```

Provides a Grafana configuration file that simplifies login

Configures Prometheus as the default data source

我們在Docker組成的文件中引用的Grafana.ini文件允許我們避免登錄。請注意，在撰寫本文時，Grafana在版本9中；因此，概念和屏幕截圖引用了此版本。從Grafana 8到9（統一的警報）有一些根本的更改（即警報是全局的，而不是按儀表板為基礎），因此，在將屏幕上的輸出與您在此處看到的屏幕截圖進行比較時，請記住這一點。

讓我們從Grafana數據源（https://grafana.com/docs/grafana/latest/dataSources/，包括Amazon CloudWatch，Mysql，Prometheus和Jaeger，並不利於後端，您可以使用可視化或警報。（https://grafana.com/grafana/plugins/）可用，或者您可以創建自己的。

現在，訪問http：// localhost：3000/，然後在左側菜單下部的配置標籤中單擊齒輪圖標。選擇數據源菜單項。您應該看到預先配置的Prometheus數據源，當單擊“添加數據源”按鈕時，所有可用數據源的列表（包括已安裝的潛在數據源插件）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

您可以在Grafana Startup上預裝和預配置插件（數據源和可視化）。細節超出了本書的範圍，但是作為起點，這就是一個例子。假設您想預先安裝ClickHouse數據源插件（假設您在本地可用Grafana-clickhouse-ds二進制二進制），則使用Click-House自託管或從SaaS產品之一中運行。使用Docker Compose，您可以將以下幾行添加到docker-compose.yaml yaml在Grafana部分中表現出來：

```
...
    volumes:
    - ./ds.yaml:/etc/grafana/provisioning/datasources/datasource.yaml
    environment:
      GF_INSTALL_PLUGINS: grafana-clickhouse-ds
```

The binary of the plug-in, in your local filesystem

此外，您將在名為ds.yaml的文件中定義ClickHouse數據源（以下內容在Docker撰寫文件中使用）：

```
apiVersion: 1
datasources:
  - name: ClickHouse
    type: grafana-clickhouse-datasource
    jsonData:
      defaultDatabase: default
      port: 9000
      server: clickhouse
      username:
      tlsSkipVerify: true
      secureJsonData:
        password:
```

Defines the type of data source plug-in

Below, the data source plug-in-specific configuration

如果您現在進行Docker-compose，則自動添加和配置了ClickHouse數據源，很容易在Grafana中使用。

在有關如何配置插件的簡短繞行之後，讓我們看看具體的數據源。單擊“添加數據源”按鈕（我選擇了CloudWatch；請參見圖6.1，並指出我修剪了某些行，因此您將在此處看到更多行），您應該看到數據源 - 特定的配置選項，包括身份驗證，授權，以及您想要設置的各種默認或限制。

在此視圖中，您還可以定義是否應將數據源用作新儀表板或警報的默認數據源（默認切換在右上角），並且如果單擊“儀表板”選項卡，則通常（但不是總是）查找數據源的預定儀表板。您可以將這些預定義的儀表板按原樣或自己的儀表板的起點使用。最後，在底部行中，您會找到多個按鈕，並且使用“保存和測試”按鈕，Grafana執行模擬（數據源 - 依賴性）檢查，以查看是否可以使用提供的cretentions訪問數據源。

請注意，按下保存和測試按鈕時，Grafana將進行的測試通常非常基本。 認為這是必要但不夠的

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

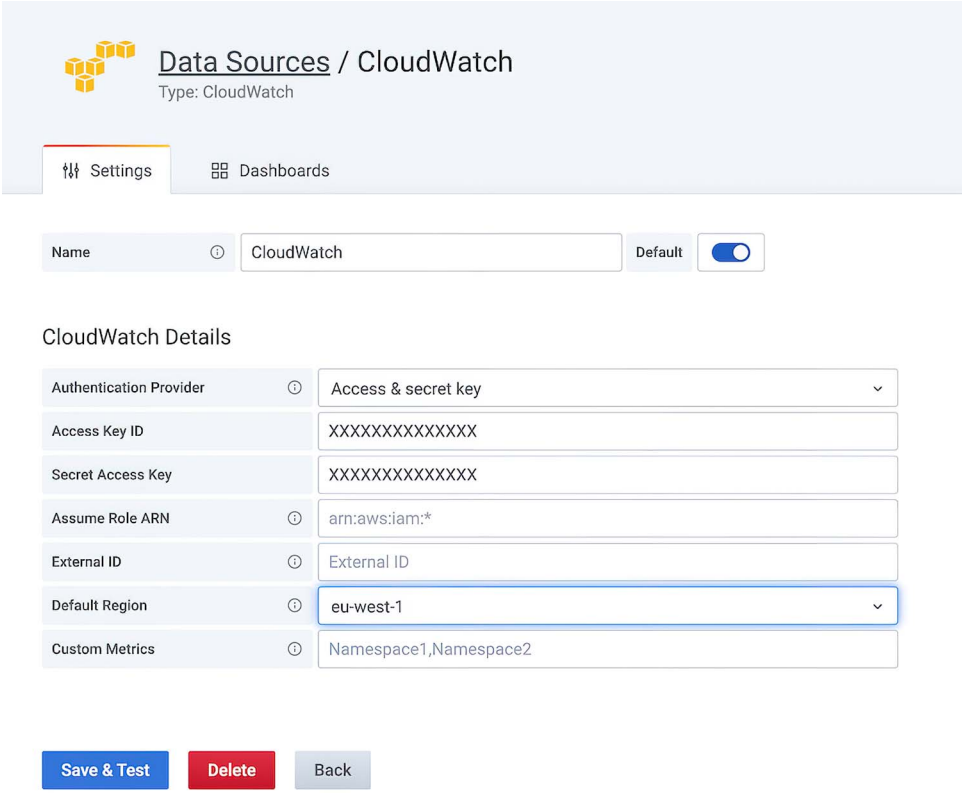


圖6.1示例Grafana數據源，顯示配置O Ptions

條件：如果它通過，您可以繼續前進（例如，使用探索來針對數據源發出測試查詢）。如果沒有通過，您需要檢查Connectiv-Ity以及是否可以訪問數據源，例如檢查IAM角色或API鍵。

Grafana Dashboard（https://grafana.com/docs/grafana/latest/dashboards/）組小組（https://grafana.com/docs/docs/docs/grafana/latest/panels/panels/）將熱圖到組織成（可折疊）行的條形圖。您可以在儀表板上使用變量（https：//grafana.com/docs/grafana/latest/latest/dashboards/variables/），使儀表板更加重複和靈活。

讓我們看看在行動中：首先，創建一個新的儀表板（單擊顯示帶有標題儀表板的四個正方形的菜單項），然後在那裡單擊新的儀表板。現在，在具有標籤儀表板設置的右上角圖標行中選擇齒輪符號。您現在應該看到儀表板設置；在那裡（在左側選項卡中），單擊變量。 在這裡，您可以通過單擊添加來創建一個新變量

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

可變按鈕。 讓我們在標籤字段中使用類型自定義和示例下拉列表進行名稱測試。這使我們能夠定義儀表板用戶可以從中選擇一個或多個預定義值的下拉列表。對於通過逗號選項隔開的閥門中的值，請使用200,301,404，結果將是您在圖6.2中看到的。

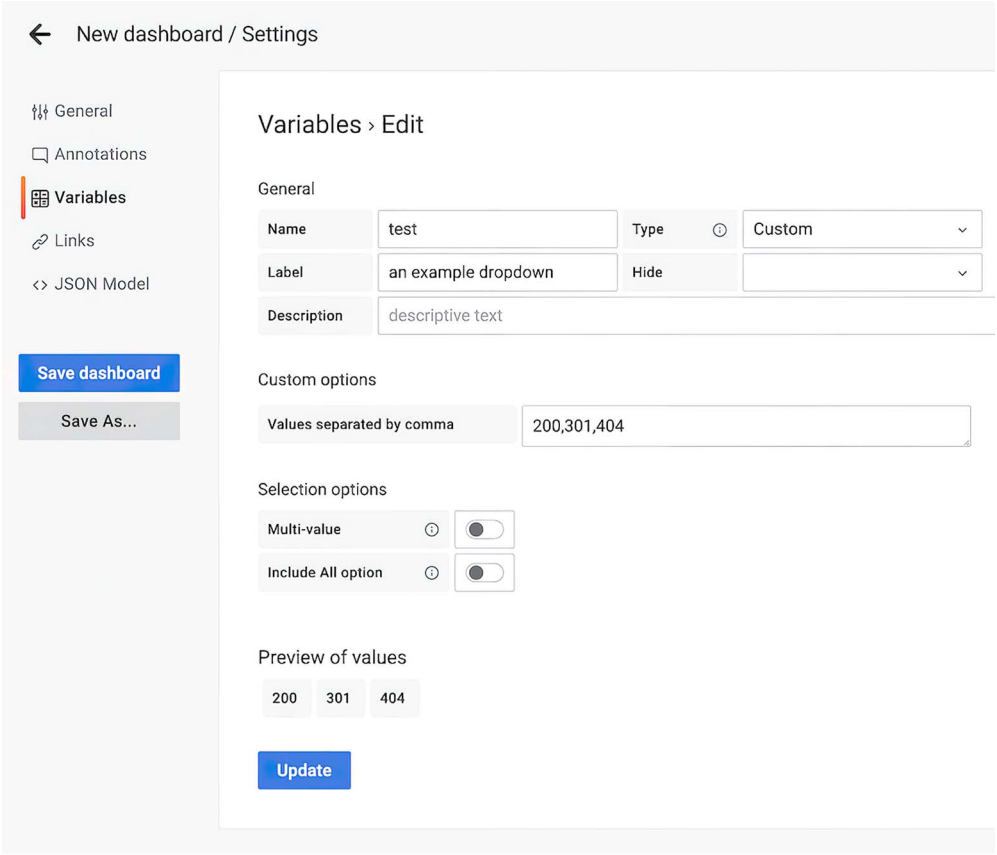


圖6.2示例的定義Grafana儀表板變量，帶有固定（預定義的）值

輸入後，單擊“更新”按鈕。現在，當您按ESC鍵或單擊左上角的左箭頭時，您將返回儀表板，並應看到定義測試變量的結果，類似於圖6.3中所示的結果。

通過Grafana版本9進行提示，請考慮使用支持的不同類型的變量（例如查詢，自定義和數據源）作為實現相關性的一種方式。從Grafana版本10開始，有一個專用的相關性

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

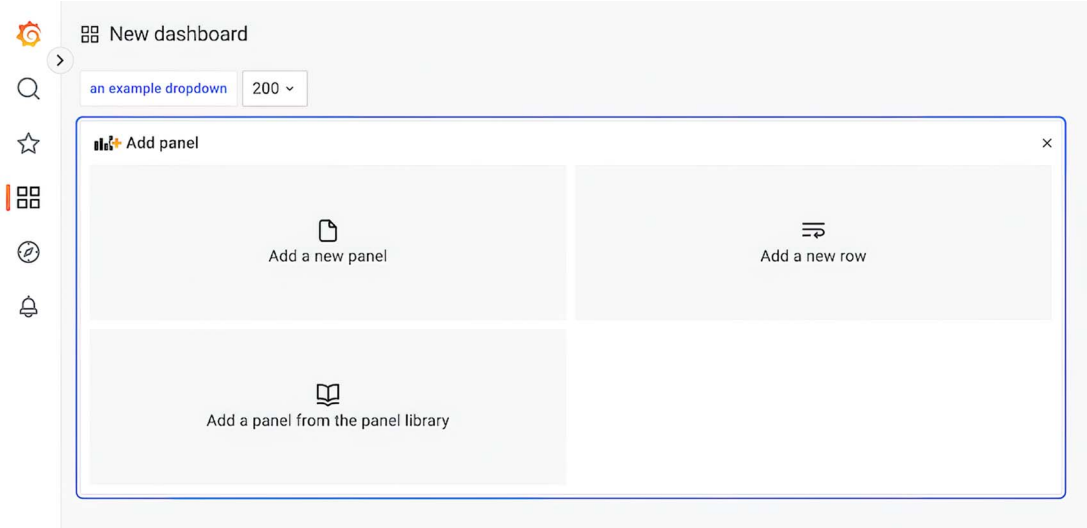


圖6.3我們的示例儀表板變量測試在儀表板上呈現

API，包括使用變量和轉換的UI支持（http://mng.bz/aopz）。

現在，您已經定義了變量測試，可以在儀表板查詢中使用它來參數化。讓我們做到這一點。首先，我們需要一個容納可視化的面板，因此，通過單擊添加新面板按鈕來創建一個面板。 由於我們將Prometheus定義為默認數據源，因此Grafana為我們選擇了合適的可視化（類型時間序列）。

那裡也已經有一個名為A的佔位符（空）查詢。這是您可以在查詢的右上方側切換的東西。 現在，在“指標瀏覽器輸入”字段中輸入以下參數化的promql查詢：

速率（Prometheus_http_requests_total {code =“ $ test”} [5M]）

此查詢使用我們先前定義的Grafana儀表板變量測試來過濾標籤值。換句話說，讓儀表板用戶選擇要查詢的HTTP狀態代碼（使用HTTP狀態代碼200作為默認值）。圖6.4顯示了儀表板編輯器中參數化查詢的結果（當您對其滿意時，在右上角的應用按鈕按下了應用按鈕並想嘗試儀表板）。

圖6.5顯示了我們promql查詢中使用的儀表板變量測試的結果，使您可以選擇HTTP狀態代碼。如果您選擇其他值（例如301），則面板很可能是空的，這是可以預期的，因為沒有這樣的標籤值

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

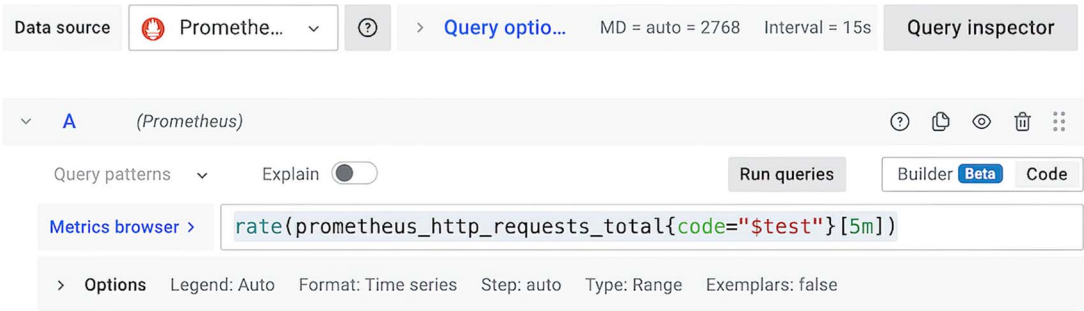


圖6.4 PROMQL查詢中使用的Grafana儀表板變量測試（面板編輯器）

存在（除非您除了在Prometheus之外做過某件事或事件不明智的事情）。

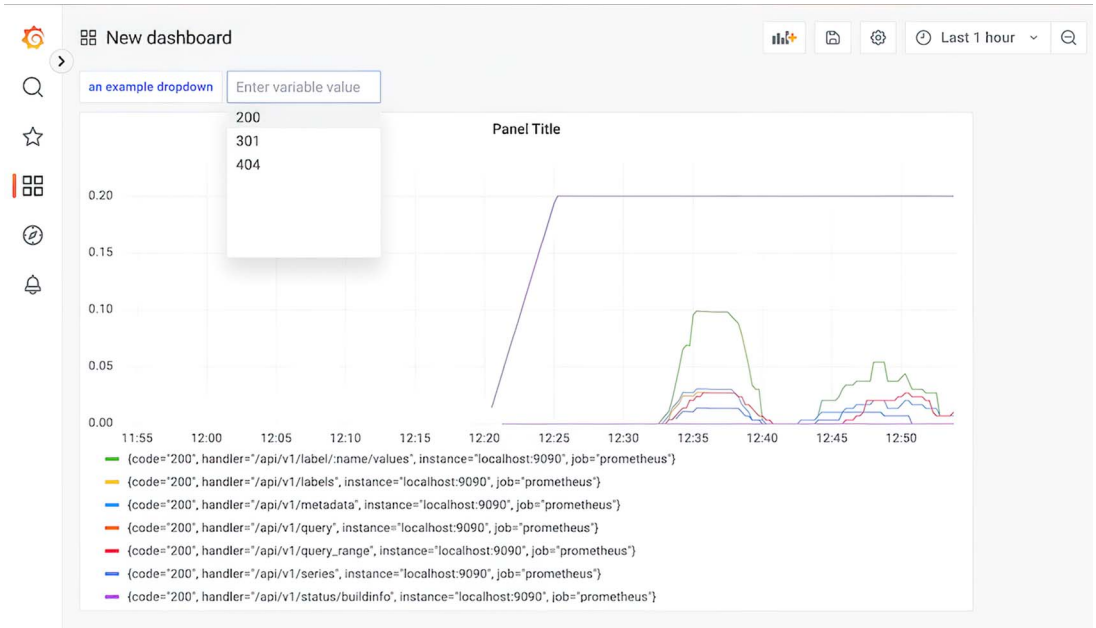


圖6.5使用Prometheus作為數據源的Grafana儀表板（帶有參數化的Promql查詢）

嘗試使用測試值或為處理程序標籤添加新變量的值。在這種情況下，請注意，從數據源選擇到自由文本輸入再到查詢變量的簡單檢查之外，還有一些Sveral變量類型。後者允許您使用特定數據源編寫查詢，並返回值列表。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

例如，您可以編寫一個promql查詢，該查詢填充了一個新處理程序變量的下拉列表，然後您可以在擴展參數化的Promql Query Query Prometheus_http_requests_total {code {code =“$ test”，handler =“$ handler”中使用。 對於針對數據源查詢以填充儀表板變量值的另一個示例，請考慮一個Kubernetes群集，您希望在其中為您的DASH-董事會用戶提供基於ProMQL查詢的可用名稱空間的選擇。

> 提示與數據源一樣，您可以直接，如前所述，直接使用儀表板，或使用Terraform（http://mng.bz/zq8p）。您可以通過參考或下載JSON定義來導入許多偉大的，社區維護的儀表板（https://grafana.com/grafana.com/grafana/dashboards/）。

警報（https://grafana.com/docs/grafana/latest/alerting/）是第三個核心Grafana資源。 Grafana版本9中引入的統一警報的主題已經進行了根本的更改。我們不會在本章中討論警報，而是深入研究第7章中的主題。

您不必自己操作Grafana。如果您希望將升級，安全補丁和訪問控件的繁重外包，那麼有很多選擇（在出版時）：

> 亞馬遜託管Grafana（https://aws.amazon.com/grafana/）自2020年底以來就已經存在，並於2021年9月進入一般可用性（GA）。它帶來了對AWS SSO和SAMLV2的支持，使您可以輕鬆地向組織中的任何用戶訪問儀表板。 Azure Managed Grafana（http://mng.bz/rxyk）自2022年8月以來一直在GA中，支持Grafana版本9功能。 您可以將託管的Grafana用作Grafana Labs的Grafana Cloud（https：//grafana.com/products/cloud/）產品的一部分，以及其他一系列基於OSS的工具，包括grafana loki for Logs for Logs for Logs和Grafana tempo。 Alibaba Cloud為Grafana（https：//www.alibabacloud .com/product/grafana）提供了託管服務，該服務可與包括DingTalk和Wecom在內的協作工具集成。

好的，那很有趣。接下來是Kibana（及其OpenSearch變體儀表板）。

6.1.2 *Kibana and OpenSearch Dashboards*

We discussed Elasticsearch and OpenSearch (ES/OS) as backends in chapter 5. In this context, you have the choice between Kibana (https://www.elastic.co/kibana/), for Elasticsearch and OpenSearch Dashboards (https://opensearch.org/docs/latest/ dashboards/index/) for OpenSearch as the respective frontends.

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

由於OpenSearch儀表板是Kibana的叉子，因此正如人們所期望的那樣，有許多常見的概念和功能，而不是考慮許可的差異。鑑於兩個項目的性質和動態以及基於它們的託管產品的性質和動態，我不會在此處詳細介紹Kibana或OpenSearch儀表板，但是我將為您指出許多起點，如果您希望更多地了解其中之一：

在第4章中，我們使用OpenSearch和OpenSearch儀表板作為後端和前端目的地，以使用Fluent Bit Agent進行日誌攝入和查詢。 通過logz.io博客，可以提供一個出色的Kibana教程（https://logz.io/ blog/kibana-tutorial/），或者如果您想投入更多時間，請潛入kibana docs（https://wwww

6.1.3 Other open source frontends

除了Grafana和Kibana/OpenSearch儀表板外，您還需要查看其他兩個開源前端，當您評估選項時：

Apache Superset（https://superset.apache.org/）在Apache許可證版本2.0下獲得，將自己定位為數據探索和可視化平台，該數據支持廣泛的數據庫（http://mng.bz/1qwj），包括Amazon Athena，Google bigquery，timequery，times，and eellasticsb，and yspeclics，andseack，包括但是，在寫作時，普羅米修斯的支持是不可用的。它於2021年初成為頂級Apache軟件基金會（ASF）項目，並擁有許多貢獻者，包括Airbnb（啟動的地方），Drop-Box和Lyft。

metabase（https://github.com/metabase/metabase）有望“公司中的每個人都提出問題並從數據中學習”，這非常適合可觀察性用例。它在AGPL和商業許可下發布，並具有強大的SQL重點。 您可以將（http://mng.bz/pzpn）連接到一系列數據庫（包括Google BigQuery，PostgreSQL和Snowflake），並通過合作夥伴和社區驅動程序獲得更多選項。除了開源DIY模型外，您還可以使用同名商業託管產品（https://www.metabase.com/pricing/），無論是在本地或云中。 現在，讓我們繼續前進云提供商和ISV產品。

6.1.4 Cloud providers and commercial frontends

在這個空間中，您經常會聽到或閱讀一詞*business intelligence*（bi） - 考慮業務流程的可觀察性。實際上，該空間中的許多產品都來自那裡，並已在觀察性上下文中適應或用於特有的用例區域。讓我們仔細看一下：

Amazon QuickSight (https://aws.amazon.com/quicksight/) - AWS的雲BI服務，您可以用於儀表板和報告。 它還支持許多受歡迎的後端，包括亞馬遜雅典娜，S3，OpenSearch和Salesforce。 *Looker Studio (https://lookerstudio.google.com/overview)* - Google的BI產品作為服務，支持一系列可視化和連接器與數百個後端的連接器。
Microsoft Power BI (https://powerbi.microsoft.com/en-us/) - Microsoft的BI雲提供，支持數據集市和應用程序深度集成到MIC-ROSHOFT OFFICE SUITE中。 另外，您可以使用Azure儀表板（https：// Learn .microsoft.com/en-us/azure/azure-portal/azure-portal-portal-dashboards），該板帶有幾種與Azure Monitoring的戶外集成。
Tableau (https://www.tableau.com/) - 一個可視分析平台，您可以在許多不同的設置中使用，從台式機到本地到雲，支持大量數據源和可視化。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

本節中提到的商業產品廣泛使用，主要是由房屋的商業方面而不是工程團隊。但是，在可觀察性的背景下，我們確實需要一種全面的方法，使盡可能多的工作家庭參與並從系統信號中受益。

6.2 All-in-ones

從技術上講，所有云提供商產品，包括CloudWatch，X射線，Azure Monitor和Google Cloud的操作套件，都屬於多一類別。我們在第5章中介紹了它們，因此，讓我們將其重點放在這個領域的其他產品上。

有一些開源產品，其中一些由Cloud Native Computing Foundation（CNCF）或Apache Software Foundation（ASF）託管，可為兩種或多種信號類型提供支持。它們具有攝入和查詢功能，通常允許將不同的後端插入或內存（非移動）存儲。

6.2.1 CNCF Jaeger

CNCF Jaeger（https://github.com/jaegertracing/jaeger）項目始於Uber，並受到Google的Dapper Project和Zipkin的啟發。如今，這是一個畢業的CNCF項目，將自己定位為一個分佈式跟踪平台，並為包括Apache Cassandra在內的各種後端提供了內置支持； ES/OS ;以及通過第三方存儲插件框架（例如Clickhouse）進行自定義的框架。

Jaeger的整體架構是一系列用GO編寫的微服務集合，它們正在共同努力。核心jaeger組件是

Agent (optional) - 守護程序將痕跡（通過UDP）發送到收集器，該收集器旨在部署到應用程序運行的所有主機。 代理被OpentElemetry代替。 *Collector* - 從OpentElemetry（SDK或Collector）或Jaeger代理中痕跡痕跡，並驗證，豐富並將其存儲在存儲後端中。 *Query service* - 從存儲中檢索軌跡並託管UI的API。 *UI* - 基於Web的GUI用戶與用戶進行交互以搜索和分析跟踪和/或查看系統體系結構（微服務的圖表表示）。 *Ingester* - 收集器的簡化版本，該版本讀取Apache Kafka（流行的開源消息代理）的痕跡，並將其寫入存儲後端。

對於實驗（DEV - 檢驗用例），Jaeger還支持純粹的內存存儲，很容易通過“一對一”分佈使用，將核心Jaeger components組合在一個包裝中。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

讓我們看看Jaeger的行動：我們再次使用Docker組成的設置，並使用Clickhouse作為跨度存儲（清單6.2）配置Jaeger。這次，我們將使用三個HO11Y實例的小規模微服務設置，以模擬呼叫兩個後端服務的前端服務。

列表6.2 Docker的相關位構成Jaeger的文件，並使用Clickhouse設置

```
frontend:
    image: public.ecr.aws/mhausenblas/ho11y:stable
    ports:
    - 8765:8765
    environment:
    - "DOWNSTREAM0=http://backend0:8766"
    - "DOWNSTREAM1=http://backend1:8767"
    - "OTEL_RESOURCE_ATTRIB=frontend"
    - "OTEL_EXPORTER_OTLP_ENDPOINT=otel-collector:4317"
    - "HO11Y_INJECT_FAILURE=enabled"

  jaeger:
    image: jaegertracing/all-in-one:1.35
    command: [
      "--grpc-storage-plugin.binary=/etc/jaeger-clickhouse",
      "--grpc-storage-plugin.configuration-file=/etc/clickhouse-
     plugin.config"
    ]
    volumes:
    - ./jaeger-clickhouse:/etc/jaeger-clickhouse
    - ./clickhouse-plugin.config:/etc/clickhouse-plugin.config
    ports:
    - "9090:9090"
    - "16686:16686"
    - "14250:14250"
    environment:
    - "SPAN_STORAGE_TYPE=grpc-plugin"

  clickhouse:
    image: clickhouse/clickhouse-server
    ports:
    - "8123:8123"
    - "9000:9000"
```

The frontend microservice, using ho11y. It's configured to call the two backends and send traces to the OpenTelemetry collector.

The Jaeger all-in-one distribution configured with ClickHouse as the backend, using the gRPC-based plug-in mechanism

ClickHouse as the backend to store traces (and make them searchable)

我們再次使用OpentElemetry收集器從基於HO11Y的微服務設置中收集痕跡，並將其攝入Clickhouse（通過Jaeger）。 以下列表中顯示了相應的opentelemetry收集器配置（另請參見第4章）。

清單6.3 opentelemetry收集器配置Jaeger帶有Clickhouse

接收器：OTLP：協議：GRPC：端點：0.
0.0.0：4317

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
exporters:
  jaeger:
    endpoint: jaeger:14250
    tls:
      insecure: true
service:
  pipelines:
    traces:
      receivers: [ otlp ]
      processors: [ batch ]
      exporters: [ logging, jaeger ]
```

The Jaeger exporter configuration (we don't do TLS, so it is configured insecure)

The traces pipeline, using the OTLP receiver and Jaeger exporter

Now, it's time for a docker-compose up, and if you now open up http://localhost:16686/ in your browser of choice, you should see something like what is shown in figure 6.6 (in the left-hand side form) if you've first set the Service dropdown to Frontend and entered http.status_code=404 into the input field labeled Tags.

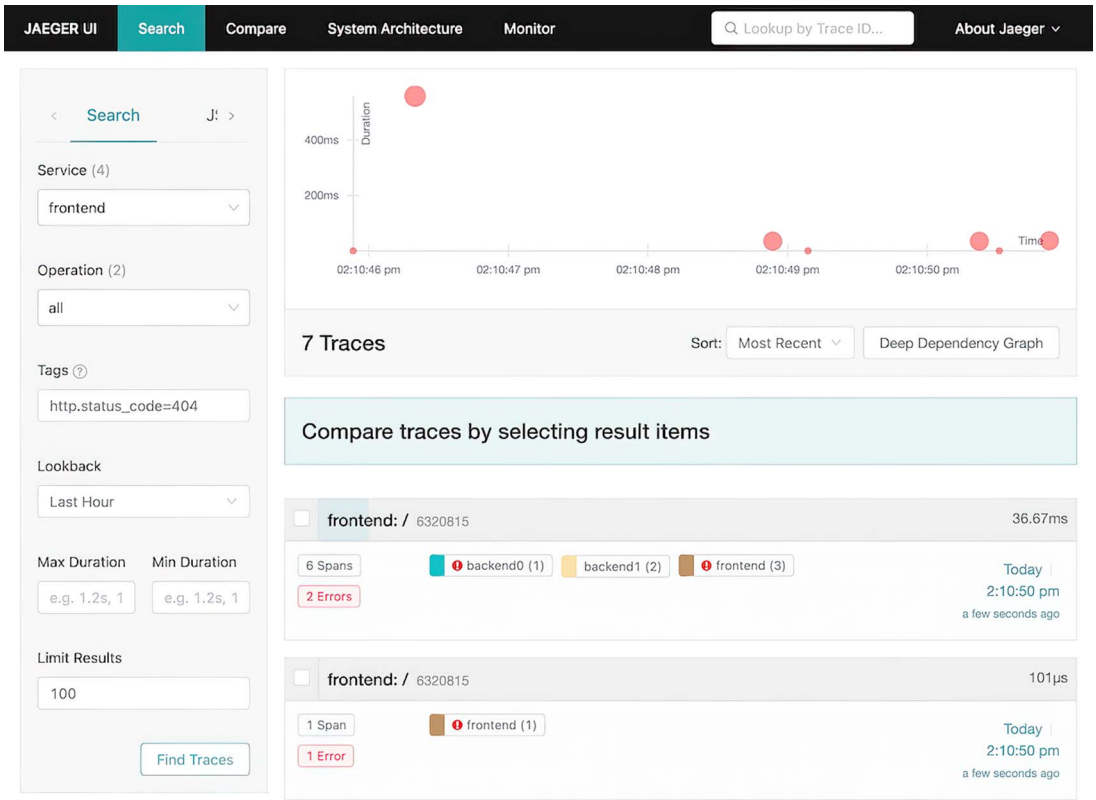


圖6.6 Jaeger UI顯示了某個標籤的所有跡線（http.status_code = 404）

概述所有軌跡都是有用的，但是通常，您想向下鑽並理解在請求端到頭的情況下發生的事情。為此，你

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

可以在圖6.7所示的視圖中單擊其中一個痕跡並降落。痕跡細節視圖使您可以跨越瀑布模型中的痕跡，包括持續時間和高級元數據，例如操作類型以及是否成功。讓我們轉到此類別中的另一個CNCF項目。

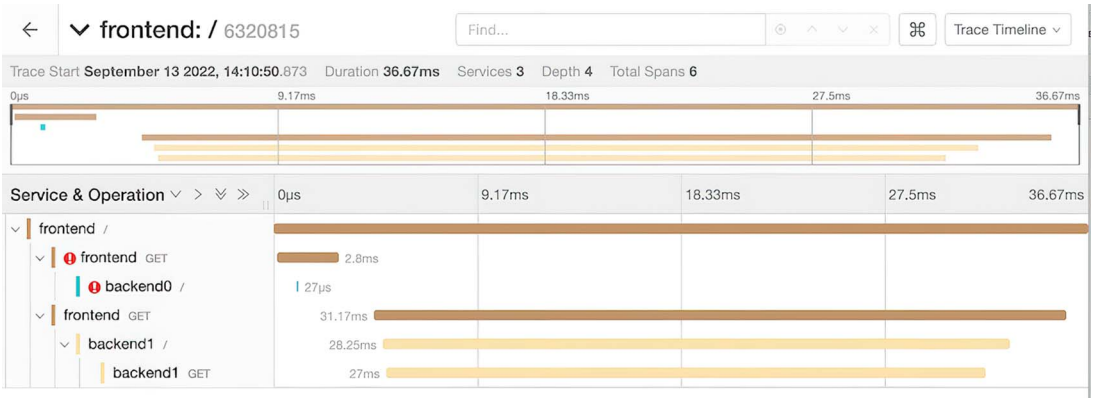


圖6.7 Jaeger UI顯示痕跡細節

6.2.2 CNCF Pixie

Pixie（https://px.dev/）是一個Kubernetes特定的實時調試平台，可以收集遙測數據，例如指標，痕跡和配置文件，而無需手動儀器或重新部署。 Pixie的前端可以幫助您查詢和可視化這些數據。它是由新遺物為CNCF貢獻的，並於2021年6月被接受為沙盒項目。

為了收集沒有任何代碼更改的數據，Pixie大量使用EBPF（https：//ebpf.io/）。 EBPF是Linux內核功能，可讓您直接在內核中執行沙盒程序。

由於EBPF允許您收集大量的細粒數據，例如完整的請求和響應主體，因此它將其所有數據存儲在內存中。但是，您可以將遙測數據發送到任何opentelemetry兼容端點（http://mng.bz/ jgxv）中，以進行長期數據保留。換句話說，通過設計，歷史數據是出於Pixie的範圍。

為了查詢遙測數據，Pixie使用一種名為PXL（https://docs.px.dev/tutorials/pxl-scripts/write-pxl-scripts）的相當聰明的域特異性語言，該語言受到了python Pandas的啟發。用戶可以使用PXL查詢和處理Pixie的基礎數據。 Pixie的所有儀表板也完全可拼寫，使用戶可以構建最適合他們的視圖和可視化。

Pixie擁有一個活潑的社區，並有一個有趣而有前途的路線圖。它具有許多集成，從Slack Alerts到在Grafana儀表板中使用它（通過OpentElemetry Export）。如果您正在運行kubernetes群集，我建議您服用

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

仔細觀察Pixie，因為它可能是工具帶中非常有用的工具。現在，讓我們繼續進行分佈式的追踪資深開源前端。

6.2.3 Zipkin

Zipkin（https://zipkin.io/）是一種開源分佈式跟踪系統，用Java編寫，由OpenZipkin志願者組織運行。 Zipkin具有關注的UI（見圖6.8），並集成了一系列後端，包括ES/OS，Apache Cas-Sandra和MySQL，其他後端作為第三方擴展。 Zipkin項目還受到Google Dapper的啟發，最初是由Twitter創建的，其根源可以追溯到2015年。

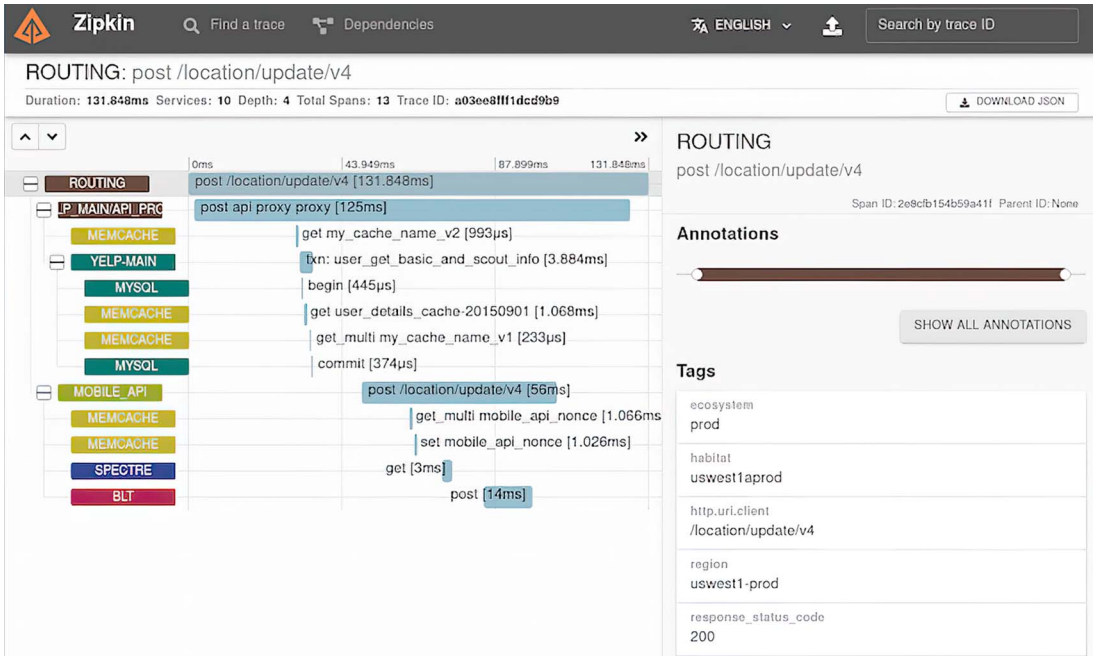


圖6.8 Zipkin在行動中的屏幕截圖（來自項目文檔）

我在這裡不會詳細介紹Zipkin的架構，因為它與我們在CNCF Jaeger的背景下看到的非常相似。如果您想嘗試Zipkin並在行動中查看它，請查看Zipkin Github repo，其中包含docker撰寫的示例（http://mng.bz/wv7p），該示例（http://mng.bz/wv7p）從使用Apache Cassandra作為後端，可與Apache Kafka追踪攝入量。 要查看Docker組成的示例，請在相應的目錄中進行Docker-Compose，然後訪問http：// localhost：9411，然後單擊“運行查詢”查看結果。接下來，我們將介紹一個令人興奮的Apache軟件基金會（ASF）項目。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

6.2.4 *Apache SkyWalking*

Apache Skywalking（https://skywalking.apache.org/）是一個ASF開源項目，如今，該項目為分佈式跟踪分佈式系統提供了支持，例如Kubernetes環境中的微服務。它最初是在2015年在中國Unicom的分佈式追踪系統，已被100多家公司（包括阿里巴巴，中國商人銀行和華為）採用。如果您想深入研究主題，請查看“ datahub中的Apache Skywalking和Apache Skywalking”（http://mng.bz/qr7r）的“應用程序性能監視器和分佈式跟踪”，liangjjun jiang看到它在作用中。這樣，我們轉到在Clickhouse上建立的兩個校園。

6.2.5 *SigNoz*

Signoz（https://github.com/signoz/signoz）將自己定位為Datadog和New Relic的開源替代品。目前，它使用Clickhouse作為後端；直到2021年中，它一直集中在Apache Kafka和Apache Druid組合（http：//mng .bz/7dbv）上。有關這些後端的更多詳細信息，請參見第5章。 Signoz易於設置，並配備了幾個教程，涵蓋了消費Kubernetes基礎架構指標（http://mng.bz/mv7a）的主題，以可視化Spring Boot應用程序中的JVM指標。

讓我們快速漫步以查看Signoz的行動。您所需要的就是安裝Docker，然後執行以下操作：

```
$ git克隆-b main https：//github.com/signoz/signoz.git && cd signoz/exploy/$ docker-compose -f docker/clickhouse-setup/docker-compose.yaml
```

如果您現在等待幾分鐘（要給捆綁的應用程序一些時間生成信號），請轉到http：// localhost：3301/，然後選擇一個跟踪，您應該看到類似於圖6.9中所示的東西（Signoz的跟踪視圖）。 接下來是另一個基於Clickhouse的開源多合一產品。

6.2.6 *Uptrace*

UpTrace（https://github.com/uptrace/uptrace）是一種全in-- in-in-in-in-in-in-in-int工具，它可以說明其軌跡，指標和日誌，並使用opentelemetry來收集數據，並在收集鏈球通信的信號中，具有具有可訪問的可靠數據的功能（歷史數據，您經常訪問您不經常訪問s3-compoptage）對象。為了提醒，Uptrace與多個通知渠道進行集成，包括電子郵件，Slack和Telegram。

Uptrace具有與您在Signoz所看到的相似的定位和架構，而在寫作時，它似乎不如Signoz成熟，但這里肯定有潛力。使用類似的設置，看到更多的開源越來越多，我不會感到驚訝。這樣，我們總結了開源產品，並將注意力轉移到該領域的商業產品上。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

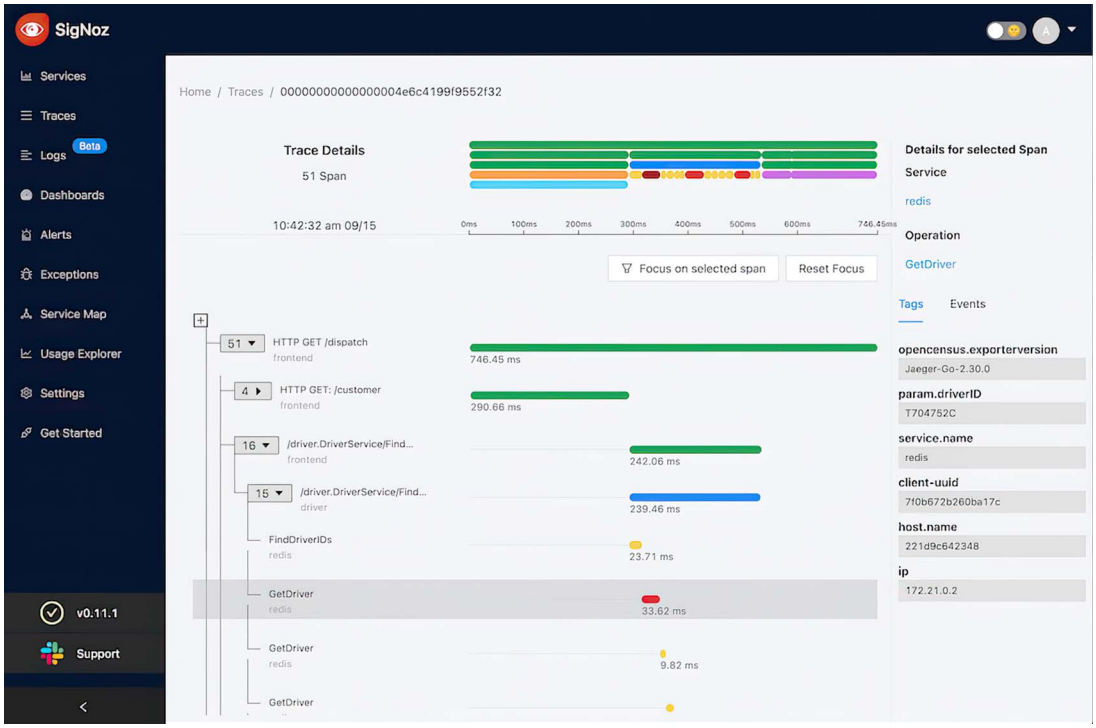


圖6.9 signoz的屏幕截圖，帶有痕蹟的視圖

6.2.7 Commercial offerings

多個類別中的許多商業產品始於數年，或者在某些情況下是十多年前。那時，通常用於其服務的標籤為*application performance monitoring*（apm）。 *APM*術語大多被*observability solution*取代，因為它更具包容性，並且大概比APM更具包容性和廣告搜索引擎優化（SEO）。

本節是有意的，因為我不想給人的印像是對所有供應商及其各自的產品提供全面的審查，甚至比較。確實，在寫作時，確實是一個繁忙而擁擠的空間，我們已經目睹了一些合併和收購，在未來幾年中更有可能跟隨。我想在以下列表中實現的目標是簡短的可用選項摘要，突出了它們的強項和整體定位。 這應該使您在選擇時擁有一些起點（在下一節中詳細介紹）。該空間中的供應商和供應商包括（但不限於）按字母順序排列：

Aspecto (https://www.aspecto.io/) - 一種SaaS提供的，重點是分佈式跟踪，您可以通過將軌跡，日誌和

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

指標。其背後的團隊在Opentelemetry項目中很活躍，因此，Stectiono大量利用它也就不足為奇了。 *AppDynamics (https://www.appdynamics.com/)* - 原始APM播放器之一，現在是思科的一部分。它將自己定位為具有專用代理的業務可觀察性平台（並具有OpentElemetry支持），並且您可以將其用於一系列用例，從最終用戶監視到異常檢測到儀表板。 *Datadog (https://www.datadoghq.com/)* - 提供許多功能，包括紅外結構，容器和數據庫監視； 日誌管理； 分佈式跟踪；以及連續分析。它可通過所有主要的雲提供者提供，具有各種基於自動啟動的集成和豐富的開箱即用功能。 2022年5月，Datadog還宣布了對OpentElemetry的支持。
Honeycomb (https://www.honeycomb.io/) - 一種具有驚人和獨特功能（例如泡泡）的雲本地可觀察性，用於調查異常。 它具有一個非常有趣的數據存儲（自定義構建），並在本地支持OpentElemetry。 Honeycomb非常關注手動儀器。 它的創始人在定義和確定過去幾年的可觀察性（O11Y）綽號方面至關重要。
Instana (https://www.instana.com/) - 由IBM收購的APM工具，具有豐富的監視和分析功能。它像這個空間中的所有主要ISV一樣，在opentelemetry上賭注而不是現在支持它。您可以在AWS，Azure，Google和IBM Cloud上使用Instana。
Lightstep (https://lightstep.com/) - 具有統一存儲層（https://lightstep.com/blog/lightstep-metrics-database）的SaaS產品，允許其存儲不同的信號； 一個有趣的定價結構； 以及上游的Opentelemetry的許多活動。 LightStep由Ben Sigelman（幫助在Google建立Dapper）創立，現在是ServiceNow的一部分。 *Lumigo (https://lumigo.io/)* - 一個較小的玩家，但在這個領域提供了一個有趣的SaaS，並具有競爭性定價。 它本地支持Opentelemetry，並專注於分佈式跟踪。 *New Relic (https://newrelic.com/)* - 具有APM根的SaaS提供，將自己定位為DevOps，Security和Business的可觀察性平台。它是opentelemetry的本地。 體育超過400個集成； 並支持跡線，指標和日誌。 *SolarWinds (https://www.solarwinds.com/solarwinds-observability)* - 在過去十年中，從收購中獲得了令人印象深刻的可觀察性解決方案，包括但不限於pingdom，plibato，papertrail，logicnow，loggly，loggly，loggly，vividcortex和sentryone。
Splunk (https://www.splunk.com/) - 觀察性ISV空間中的大玩家之一。如今，它可以作為SaaS和本地部署使用；它已經存在了一段時間（一直可以追溯到2003年）。 Splunk通常用於日誌，但也支持指標和痕跡，並具有廣泛的數組

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

功能，包括分析，儀表板和業務診斷。 Splunk大量投資並為OpentElementry做出了貢獻，因此，它在其產品中為其提供一流的支持並不奇怪。

Sumo Logic (https://www.sumologic.com/) - 與所有主要雲提供商集成的SaaS產品。它具有許多集成，支持一系列用例，並提供了與安全性和安全相關功能的功能，並支持開放遙測。

這樣，我們已經達到了多合一產品的末尾，現在選擇一個產品。

6.3 Selecting frontends and all-in-ones

鑑於您在本章中看到的前端和多大型選擇的數量，您可以想像選擇一個（或更多）並不是一件容易的事。如果可以的話，請從要交付給內部客戶，開發人員和業務利益相關者的價值後退。 考慮以下問題：您是否正在尋找解決方案來支持運輸功能的加速和修復錯誤？ 您是否對可以減少解決產品問題的時間的產品感興趣？您想提高開發人員的生產率嗎？

根據答案，您首先想對以下決定做出決定：構建或購買。 通常，除非您將其確定為競爭優勢，否則您希望盡可能地外包。換句話說，購買。但是，有幾家大型企業在最大程度地限制供應商和/或云提供商方面具有戰略興趣。在這種情況下，使用開源（components）並根據公開標准進行工作很有意義。 換句話說，構建。

您的里程可能會有所不同，但是在此空間中評估和選擇產品時，這裡有幾個注意事項：

您可以確定費用嗎？前端的成本通常是在託管產品和相對可預測的託管產品基礎上（觀看者和編輯器）的。總本身通常具有類似後端的成本計劃（並且可能具有基於座椅的定價）。 對於自託管設置，請確保您考慮修補，規模縮放，呼叫時間，與安全相關的要求和工作的運營成本，並與上游保持聯繫。正如我們在第5章中所討論的，可能還會有與數據存儲，攝入和查詢有關的成本。 多個玻璃通常是您可以獲得的最接近的玻璃窗格，如果您確保支持開放標準（例如OpentElemetry），則受支持，小型供應商鎖定，它們是一個絕佳的選擇。 對於所有開源產品，請確保您了解該項目背後的人（即通常由CNCF或ASF託管的單一供應商開源與社區），許可（可能會有可能更改）以及開源項目的健康狀況。關於最後一點，只是看著多少

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

github恆星還不夠（這純粹是一個受歡迎的指標）。 You need to invest the time to look at the release cadence (you should be able to accommodate the pace), how many contributors (ideally, multiple, from differ- ent places) and users (ideally, in different industry verticals) it has, the docu- mentation available, how many issues and pull or merge requests there are (and how quickly they get addressed), and a strategy for support (i.e., in-house or community-based), as support is key to enterprise 用法。

越來越有助於您更有效地使用所選工具的越來越相關和請求的功能之一就是相關性（https://www.youtube .com/watch？v = qviti34zfuk）。 確保您可以使用哪種相關性（基於時間的，示例等；另請參見第11章），以及假設您正在評估的工具確實支持多個信號類型，需要多少手動工作才能從一種信號類型轉到另一種信號類型，這是與工作相關的基礎。

鑑於前端和多個人的本質，作為可見零件最終用戶與之互動的提示，請確保將所有利益相關者帶入船上。純粹基於價格標籤的工具，或者是因為它的拋光方式可能會具有挑戰性。

現在，這是很多新信息！希望您可以在此空間中選擇工具時利用它來發揮自己的優勢。這是一個快速發展和令人興奮的空間，因此定期重新審視您的選擇是有意義的。

Summary

前端使您可以與存儲在後端中的可觀察性數據集進行交互。 您使用前端以各種圖形和文字方式找到（臨時）問題的答案。 有許多開源前端（Grafana，Kibana，Metabase等）可用，當您構建自己的內部可觀察性解決方案時，請考慮這些而不是從頭開始構建某些東西。 如果您設法避免過多的供應商鎖定，則是一個絕佳的選擇；它們使您可以外包大部分的運營負擔，並典型地為您提供一系列功能。 過去，現在是APM的，總的來說，以分佈式跟踪為單位。 在此空間中選擇工具可能很棘手，但是請確保您了解與之相關的成本，並清楚地了解要交付的價值（以及與誰）。 選擇開源工具是一種互操作和供應商不可知論的好方法；但是，請確保您了解開源項目的動態，並具有獲得支持的明確策略。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

工具支持的越開放標準越開放，您越有可能鎖定並且能夠越來越不滿意，並且能夠繼續使用另一個工具。 前端通常是發生相關性的地方； 也就是說，您希望UI支持操作和可視化，可以輕鬆從一種遙測信號類型切換到另一種遙測信號類型。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

7 Cloud operations

This chapter covers

如何管理事件健康監控和警報治理
和使用跟踪

到目前為止，在本書中，我們專注於雲本地設置的愉快道路，涵蓋了信號源，遙測和目的地，以存儲，查詢，可視化和與信號進行交互以了解和影響系統。在本章中，我們將討論我稱為*cloud operations*的雲本地解決方案的一個方面，該解決方案涵蓋了您可能會遇到的幾個主題，尤其是在操作角色中。

我們從事件開始：如何檢測何時無法按照其應有的方式工作，對異常行為做出反應以及從以前的錯誤中學習。然後，我們專注於警報或警報（我在此處互換使用這些術語，儘管警報可能是觸發的條件，並且警報可能會涵蓋Multiple Alerts） - 也就是說，是自動化的過程，以檢查條件並告知負責服務或有關其基礎架構的人。在本章的最後一部分中，我們討論了您的用法跟踪，即您的內部或外部用戶訪問權限或您用於提供雲本機應用程序的資源的成本。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

為什麼雲操作很棘手？ 好吧，在雲本地解決方案的背景下，是在Kubernetes上運行的工作負載或一堆Lambda函數，它們的共同點是許多具有許多不同所有者的活動部件。因此，當您的雲本地應用程序的最終用戶受到影響時，例如，謀殺之謎（謀殺之謎）開始：沿請求路徑沿線的哪些服務導致了問題？誰擁有它，因此應通知修復它？

請記住，對於最終用戶來說，您正在運行的事實，例如，數百個微服務是未知的，坦率地說，沒關係。 他們認為的是，某些事情不起作用，並且您的組織有責任確保您在從用戶那裡了解到這些問題之前（理想地發現這些問題），並擁有策略，不僅要盡快解決問題，還可以從問題中學習，以確保它不再發生。在這種情況下，我們可以考慮跟踪Customer的體驗（並非總是清楚誰擁有此）與基礎架構跟踪（通常明確定義所有權）。 讓我們從事件以及如何以結構化的方式檢測和響應它們開始。

7.1 Incident management

在本書的背景下，我們將考慮兩項事件：服務的意外行為或對其功能或性能產生負面影響的平台，以及重複出現的意外行為，而意外行為尚未應用這些學習。對於後一種情況，重複出乎意料的行為，有一本關於如何解決重複情況解決的前提事件的記錄可能會有所幫助。

通常，最好進行事件響應過程。 此事件響應過程通常是一組程序，可以定義您如何解析和從事件中學習。您可以從三個階段來想到它：

1首先，您需要能夠檢測出出現問題（健康和績效監控）。 2然後，您需要能夠以結構化的方式做出反應和處理事件。 3最後但並非最不重要的一點是，您需要從事件中學習。您想避免將來重複類似的事件（應該完成哪些行動項目以不重複出現相同的問題）。 我們將首先介紹檢測問題。

7.1.1 Health and performance monitoring

我們先前討論過，解決事件的第一步是能夠了解某事首先是破碎的。這聽起來很簡單，但是在具有許多組件的雲本地系統中，這是一個非平凡的問題。換句話說，手動方法，例如在瀏覽器中訪問您的網站，並檢查所有內容是否看起來不錯，這不是您想要考慮的事情。相反，以自動化為

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

許多針對您的服務和“外部視圖”的檢查，即盡可能的客戶體驗（CX）。 在這種情況下，要強調這種外部觀點至關重要。想像一下，您所有的服務都會按預期報告健康和工作的情況，但是由於DNS問題，您的最終用戶無法訪問該應用程序。

您需要使用的實際機制來執行健康和績效誤解確實是一個頻譜，包括以下幾個：

Reachability monitoring - 執行簡單調用（ICMP，TCP，HTTP），以檢查可及性或確保端點返回預定義的狀態代碼。 *Synthetic monitoring* - 執行腳本以模擬典型的最終用戶操作。這些模擬的最終用戶有時被稱為*canaries*（引用原始用例，其中這些小鳥被用於檢測地雷中的氣體洩漏）。幾種開源和商業產品允許您執行合成監控，例如Datadog（https：//docs.datadoghq.com/synthetics/，Elastic（http：//mng.bz/5wdo），new Relic（http://mng.bz/6dle）和Amazon CloudWatch）（http://mng.bz/o17z）。

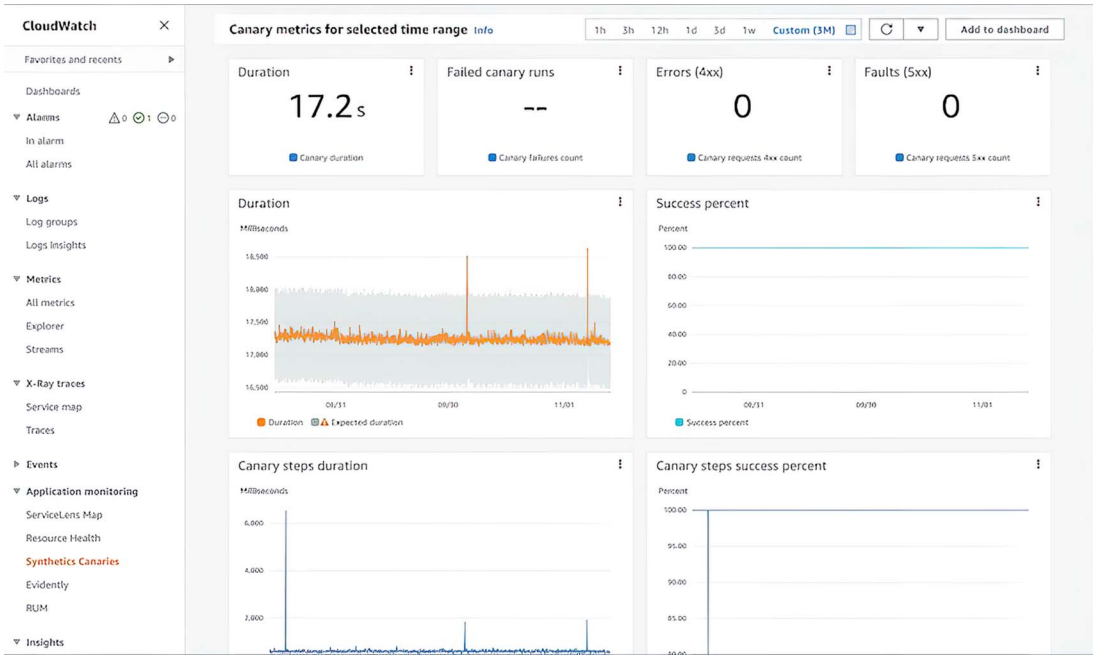


圖7.1 CloudWatch，綜合監視的一個示例，跟踪網站的可用性

您通常要自動化檢查條件的過程並告知負責人的人，這也稱為*alerting*或*alarming*。有一篇不錯的高級文章：“什麼時候提醒什麼？”（https://ali.sattari.me/posts/2023/何時to-to-alert-on-what/），這是一篇有用的文章，用於開始該主題，

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

從手冊到服務級目標（我們在第10章中詳細介紹）不等。

與之相關的是，您如何知道是否有什麼不正常的東西？您需要一個基線。您可以通過觀察和注意事物或使用更複雜的技術（通常稱為*anomaly detection*}（https://www.reddit.com/r/sre/r/sre/sre/sre/comment/comments/zxd2y/anomaly_detection/）來確定手動的基線（即“一切健康時的外觀？”）。隨著檢測基礎知識的發展，我們現在專注於在了解事件後如何處理該事件。

7.1.2 Handling the incident

現在，我們可以通過對雲本機應用程序進行健康和績效監控來檢測何時在我們的雲本機應用中側面進行側面，我們繼續解決如何處理事件的問題。讓我們首先討論“誰”，然後繼續進行“什麼”和“如何”。通常，人類需要進入循環，並分配給事件。這通常是通過通話機製完成的。在一定時間段內，從小時到一周甚至更長的時間，監視系統會注意到呼叫的人，並且需要盡快掌握問題。

> 提示有一些產品有時稱為*AIOps*，這些產品試圖優化人類的體驗（例如，Shoreline，https：// www .shoreline.io/）。

如今，該通知通常是通過應用程序（以及通過電子郵件，Slack，MS Teams Ping，電話等並行傳遞的），並且可以在白天或晚上的任何時間進行。您可以想像，如果這樣的通知（或頁面）在凌晨2點到達您，並且您有15分鐘的簽入，則要確保您有結構化的方法。

談論通話角色和模型時，有幾件事要考慮：

Traditional vs. on call - 在傳統的環境中，嚴格分開開發和操作，操作（OPS）團隊正在呼叫（技術響應者）。這可能會引起問題，因為不僅可以在呼叫設置需求，而且最熟悉某種服務的人當然不是待命的OPS人。另外，可以建立一個模型，其中開發和通話結合在一起。 例如，這就是AWS服務團隊的工作方式。這為開發人員提供了適當地啟動其代碼的激勵措施，從而使呼叫能夠盡快解決問題。

Engineering and communication - 毋庸置疑，要解決技術問題，您需要在呼叫上進行工程師。但是，還有另一個可以分開的角色：通信（事件經理或指揮官）。 一方面，這可能是內部溝通（使利益相關者保持最新），另一方面，它可以將問題的通信轉介給定制者。有人需要確定客戶是否受到影響（與

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

金絲雀）；有多少客戶受到影響；以及它們受到影響的地方（如果我們在談論全球系統）。根據結果的不同，您可能會看到更具個性化級別的更新，例如AWS個人健康破折號（http://mng.bz/nw75），或與所有用戶相關的全球視圖，例如github狀態（https://wwwwwwwwwwwwwwwwwwwwwwww.githubstatus.com/）。

關於*what*和*how*，要記住的最重要的事情是停止出血，也就是說，嘗試解決問題，理想情況下，沿途收集數據點。分析（為什麼出錯）稍後出現；不要讓分析減慢您阻止出血的努力。

為了支持呼叫的人，組織通常會開發劇本和運行簿（http://mng.bz/vn7p），這些劇本描述了在事件和文檔依賴項中檢查的內容，以了解下游效果或可能的上游麻煩來源。 此外，一些團隊在通常稱為*game day*（http：// mng .bz/4dnr）的東西中練習可用的過程或材料（包括運行簿的覆蓋範圍）。

7.1.3 Learning from the incident after the fact

假設呼叫團隊做得很好並解決了問題。那不是故事的結局。您想進行一個關注事件的過程，並允許您實現兩件事：了解基本問題是什麼，並確保您不會重複同樣的錯誤。

我建議遵循無罪郵政的路徑（http://mng.bz/ qpo6）; 在AWS中，我們稱這些*corrections of error*（coe; https：//wa.aws.amazon.com/wat.concept.coe.en.html）。這些課程可以幫助您確定是什麼原因引起了問題以及如何避免它，通常採用五種技術（https：//www.lucidchart.com/blog/5-whys-whys-analysis）。 您也可以遇到*root cause analysis*（rca; https：//www.techtarget.com/searchitoperations/definition/definition/root-cauce-cauce-analysis）一詞，這是指有關最初引起經驗數據問題的各種理論。

在這個繁重的理論部分之後，讓我們繼續前進如何處理接通角色的通知，換句話說，*alerting*。

7.2 Alerting

在本節中，我們首先考慮在Prometheus生態系統中提醒。然後，我們繼續使用Grafana提醒，最後，我們討論了雲提供商及其他地區的提示。

7.2.1 Prometheus alerting

我們談到了普羅米修斯，並在前幾章中看到了它的行動（例如，第2和第5章）。既然我們在儀器和後端方面有了更完整的情況，讓我們看看普羅米修斯如何處理警報。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

在普羅米修斯的背景下，警報包括兩個不同的部分：

Prometheus itself - 您定義了什麼是警報。這有效地意味著您使用Promql定義條件，並定期定期評估它們。一旦滿足條件，Prometheus就將其視為警報“啟動”，並將其發送給AlertManager API。

Alertmanager API - 提供了一種API，使客戶（通常是程序）可以發送警報並處理通知。例如，您可以定義一個策略，該策略通過電子郵件向某些類型的警報發送通知或實施*during work hours, send the notification to Slack; after work, send a text message*之類的策略。但這並不是Alertmanger所做的唯一事情：它可以重複使用警報，並允許您對通知進行分組和抑制以減少警報疲勞並減少警報風暴。

圖7.2顯示了Prometheus和Alertmanager如何一起處理警報。正如我們在第4章中討論的那樣，刮擦目標（例如，您的儀器應用程序或普羅米修斯出口商，例如節點出口商）在Pro-metheus exposition格式中揭示指標，該格式（換句話說，它調用了端點，否定端點，否則默認情況下會屬於 /計量）。如果您在Prometheus中定義了警報規則，請連續評估它們，並致電Alert-Manager，以防定義的閾值。 反過來，AlertManager的目的是發出盡可能少的通知。通過AlertManager中的配置接收器，它向一個或多個目的地提供了通知，其中操作員（人類）最終消耗了通知並相應地採取了通知。

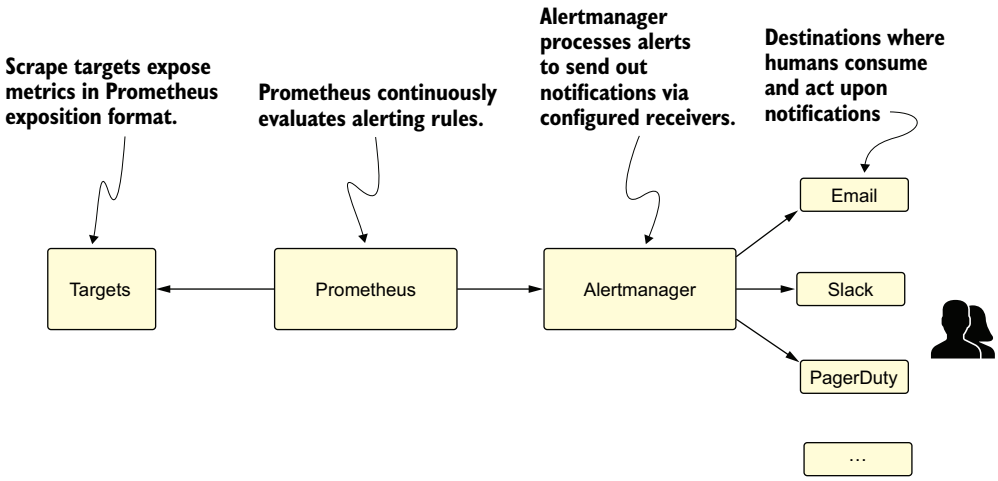


圖7.2 Prometheus和Alertmanager

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

如圖7.2所示，Prometheus和Alertmanager的分離允許靈活性和可擴展性。例如，您可以在不中斷其他工具的情況下擴展或重新啟動任何一種工具。

> 從操作的角度來看，在一天結束時，您想避免警報疲勞（https://en.wikipedia.org/wiki/alarm_fatigue）。 也就是說，您想最大程度地減少要採取行動的通知，同時確保您不會錯過任何重要的東西。 想像一下，您負責具有多個名稱空間的Kubernetes群集。 如果有數據平面可能無法達到或重新啟動節點，您只想收到一個通知，而不是為群集中每個POD的一個通知。

現在，讓我們更深入地了解警報規則和警報人員的工作方式。

普羅米修斯警告規則

從普羅米修斯開始，我們現在通過定義和處理警報來努力。在Prometheus配置中，您可以使用警報規則（http：//mng.bz/xnj9）定義警報。警報規則定義了您希望Prometheus評估的條件。一個具體的示例如下（改編自出色的參考網站Awesome Prometheus Alerts（https://awsome-prometheus-alerts.grep.to/），它提供了可以使用的大量策劃警報示例）：

The name of the alert. The convention is CamelCase.

The condition to evaluate—a PromQL expression Prometheus executes

Optionally, you can specify for how long the condition needs to be met before sending the alert to the Alertmanager.

Key–value metadata for filtering and routing in the Alertmanager

```
- alert: PrometheusAllTargetsMissing
  expr: sum by (job) (up) == 0
  for: 0m
  labels:
    severity: critical
  annotations:
    summary: Prometheus all targets missing {{ $labels.instance }}
    description: "A Prometheus job has no alive targets.\n
                  VALUE = {{ $value }}\n  LABELS = {{ $labels }}"
```

Key–value context for direct human consumption (content of the notification) using templating

雖然for字段是可選的，但您應該考慮設置它。原因是，在雲本地系統中，您正在處理許多短暫組件，網絡連接問題以及超時或回程。例如，您不希望觸發警報對您發送垃圾郵件，例如，這只是遙測的臨時問題。假設我們會選擇：3M在上一個示例中。在這種情況下，當第一次滿足條件時，Pro-Metheus最初會在待處理狀態下考慮警報。該警報被認為處於射擊狀態，如果條件持續了3分鐘，則將其發送給警報經理。 在前面的片段中，我們用於：0m，因此，警報直接射擊。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

關於註釋字段，您想為根據警報收到通知的人類操作員提供豐富的上下文。考慮到通知可能會在凌晨接聽電話，而在這種情況下，您想要的最後一件事是開始挖掘參考文獻，浪費寶貴的時間來停止出血，同時仍在試圖醒來。例如，除了我們在上一個示例中定義的摘要和描述字段外，您還可以添加以下內容：

指向儀表板的鏈接顯示了相關時間範圍的時間序列評估受影響的人（無論是真實的客戶還是金絲雀）鏈接到運行手冊的鏈接，已知問題和貢獻因素。如果您可以從手機上關注這些鏈接而無需登錄到八個系統以使它們渲染，那麼您做得很好。

現在，我們知道了什麼是警報規則以及如何定義它們，讓我們看看如何設置Prometheus以包括警報規則並建立與AlertManager的連接。 這是在Prometheus配置文件Prometheus.yml中完成的：

```
global:
  scrape_interval: 30s
  external_labels:
    environment: dev
    region: us-west-1
rule_files:
  - alert.rules
alerting:
  alertmanagers:
  - static_configs:
    - targets: ["alertmanager.local:9093"]
scrape_configs:
…
```

Adds external labels to provide context for the Prometheus instance

Points to the file where the alerting rules are defined

Configures the Alertmanager Prometheus should use

警報管理器

現在，讓我們轉到AlertManager（http://mng.bz/yq7p）。首先，我們看看AlertManager如何處理警報。 這是在所謂的通知管道中完成的，包括以下階段：

1 *Inhibition* - 如果其他（相關）警報已經觸發，則提示通知。 例如，想想上述情況，當您已經通知了群集數據平面問題時，您不需要POD級通知。 **2** *Silencing* - 以主動方式進行供應通知。例如，如果您在kubernetes群集中升級節點，那麼您已經事先知道節點會暫時下降，因此您不想為此獲得通知。 **3** *Routing* - 獲得通知的人。 **4** *Grouping and throttling* - 確保每個事件獲得一個通知。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

5 *Notification* - 使用所謂的接收器，定義應如何傳遞通知，例如通過電子郵件，Slack，Pagerduty或Amazon Simple Noticle Notification Service（SNS; https; https：//aws.amazon.com/sns/）。

AlertManager的示例配置（http://mng.bz/mbaq）應該讓您了解如何配置通知管道，但讓我們看看一個具體的端到端示例，以充分欣賞事物的工作方式。

在此示例中，我們實施了一個簡單的普羅米修斯自我監控案例。 目標是一旦超出了一定數量的Prometheus API呼叫，就要收到通知。

像往常一樣，我們需要使用docker撰寫文件7.1中的文件（從CH07/Prometheus/）中構成文件：

docker-compose -f docker-compose.yaml up

在Docker組成的文件中，我們傳遞了環境變量和配置文件。

列表7.1碼頭碼頭構成Prometheus警報設置的配置

```
version: "3.5"
services:

  prometheus:
    image: prom/prometheus:latest
    ports:
    - "9090:9090"
    volumes:
    - ./prom-config.yaml:/etc/prometheus/prometheus.yml
    - ./alerting.yaml:/etc/prometheus/alert.rules

  alertmanager:
    image: prom/alertmanager:latest
    volumes:
    - ./am-config.yaml:/etc/alertmanager/alertmanager.yml
    ports:
    - "9093:9093"

  webhook:
    image: kennethreitz/httpbin
    ports:
    - "9999:80"
```

Configures Prometheus with alerts and Alertmanager

Configures Alertmanager with grouping and notification

Sets up a generic webhook the Alertmanager can call

下一個列表顯示Prometheus.yl中的完整Prometheus配置，定義在哪裡找到警報並連接到AlertManager。

清單7.2 Prometheus配置

全球的：
Scrape_interval：5s

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
  external_labels:
   env: "docker"
rule_files:
- alert.rules
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      - "alertmanager:9093"
scrape_configs:
  - job_name: "prometheus"
    static_configs:
    - targets: ["localhost:9090"]
```

References where the alerting rule is defined

Configures the Alertmanager

Defines what to scrape (in this case, Prometheus self-scrape)

我們的警報定義為arter.rules，如下所示。一旦Prometheus認為超過一定數量的HTTP請求總數，我們想發射警報（在我們的示例中為40）。

清單7.3警報規則

```
groups:
- name: alert.rules
  rules:
  - alert: TooManyAPICalls
    expr: sum(prometheus_http_requests_total) > 40
    labels:
      severity: "critical"
    annotations:
      summary: "Too many API calls"
      description: "The Prometheus instance {{ $labels.instance }} is
     experiencing too
      many API calls"
```

The name of our alerting rule

The PromQL expression Prometheus evaluates

Using a Prometheus label to communicate how important the alert is

AlertManager配置AlertManager.yml被描述 XT。

清單7.4 AlertManager配置

```
route:
  group_by: ["env"]
  receiver: "mywebhook"
receivers:
- name: "mywebhook"
  webhook_configs:
  - url: "http://webhook:80/anything"
```

The grouping definition. We want all alerts with the env label grouped together.

Defining the receiver (how to send the notification) for this route

Configures the receiver. In this case, we want the notification via a webhook.

如果您訪問http：// localhost：9090，則應該看到Prometheus UI。您可以通過“頂級警報”菜單項找到警報規則及其狀態，如圖7.3所示。如果您想加快事情的速度，則需要執行一些查詢（例如，使用Promql查詢總和（Prometheus_http_requests_total））或否則請等待幾分鐘，直到您看到警報射擊。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

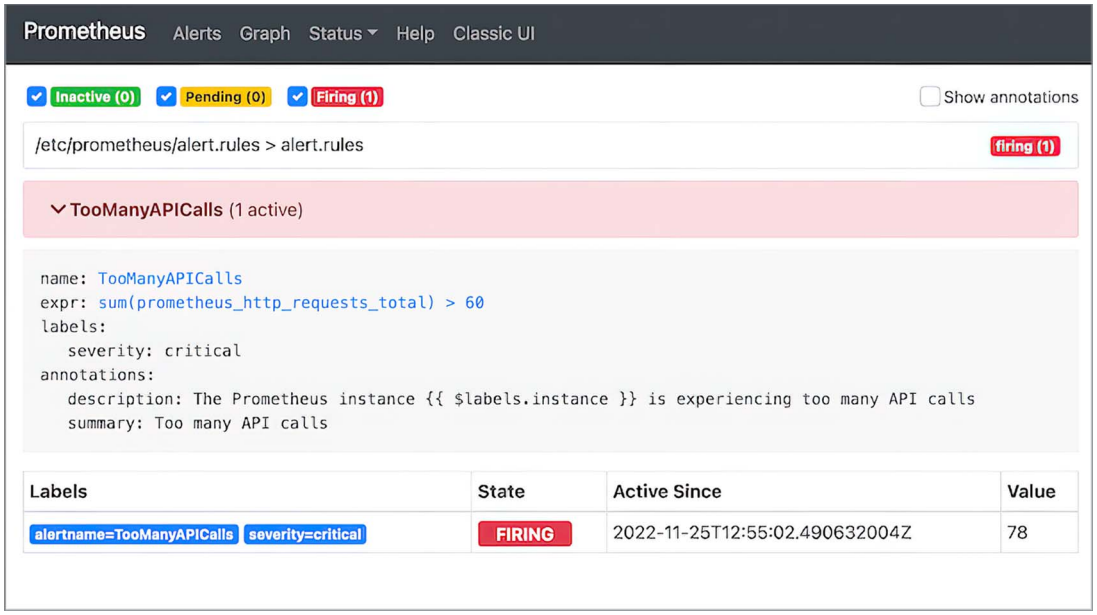


圖7.3 Prometheus中發射警報的屏幕截圖

現在，您應該在Prometheus中看到我們的警報正在發射。因此，調用了警報器。因此，一旦看到了在Prometheus向Prometheus開火的警報，請前往http：// localhost：9093，一旦通過Env = docker過濾，就應該在其中看到警報（圖7.4）。

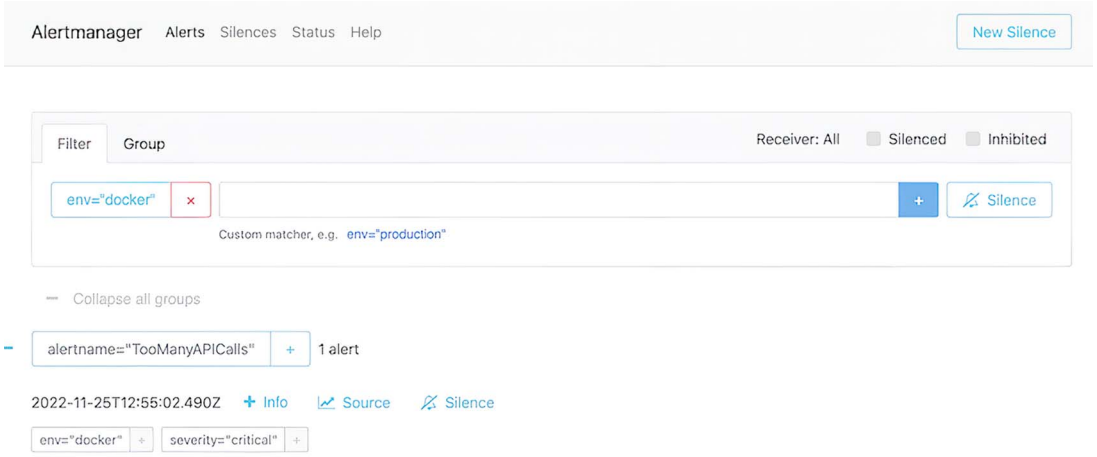


圖7.4在AlertManager中處理警報的屏幕截圖

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

圖7.5顯示瞭如果您配置了電子郵件接收器而不是通用的網絡鉤，您將收到的通知。它包含警報規則和外部標籤env中定義的標籤。

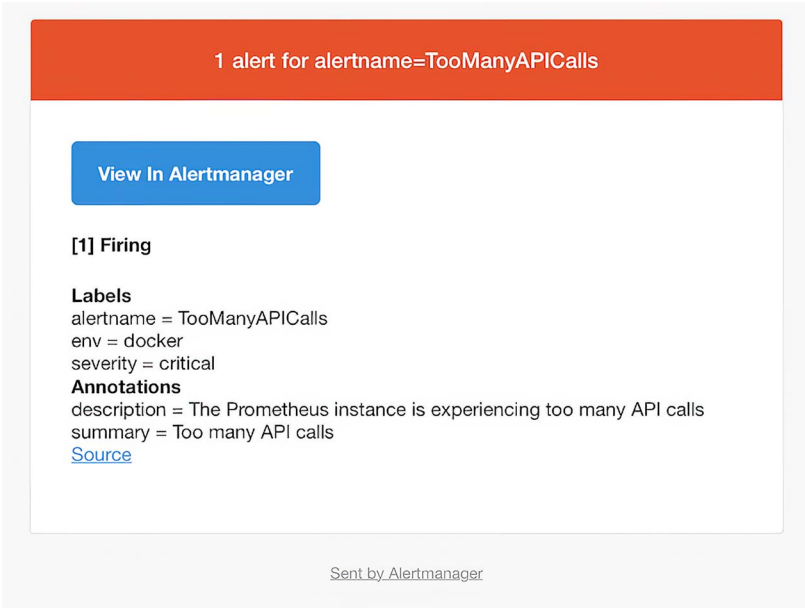


圖7.5通知電子郵件的屏幕截圖

提示AlertManager提供了許多出色的自定義，您可以使用這些自定義來使您的通知更加可行和有用（例如，Notifica-Themtion模板，http：//mng.bz/a1rj）。

這樣，我們已經完成了Prometheus警報端到端的示例。 這裡討論的問題也適用於Prometheus生態系統中的其他工具，例如CNCF Projects Cortex（http://mng.bz/gb2e）和Thanos（http://mng.bz/e1z9）;兩者都支持兼容兼容的警報規則。這使得易於遷移到聯合設置，有效地允許您重複使用所有配置。

如果您想深入研究Prometheus提醒的話題，我可以推荐一些資源：

Prometheus Up and Running by Brian Brazil (2018, O’Reilly, http://mng.bz/ OxeK) “Improved Alerting With Prometheus and Alertmanager” by Julien Pivotto (2019, PromCon talk, http://mng.bz/Y1Wo) “Life of an Alert” by Stuart Nelson (2018, PromCon talk, https://www.youtube .com/watch？v = pudjca23qa4）

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

接下來，我們將快速考慮使用Grafana提醒，使您能夠管理Prometheus以外的警報和通知。

7.2.2 Using Grafana for alerting

我們在第6章中討論了Grafana，現在我們將簡要介紹其警報功能。 直到版本8之前，Grafana支持以儀表板為中心的警報模型。這意味著警報僅限於儀表板，因此很難管理通知政策。 例如，回答問題：“是什麼通過Pagerduty向某個目的地發送警報？”將需要手動檢查所有儀表板。我們將專注於統一的警報（http://mng.bz/gyvm），可從Grafana Ver- Sion 9及以後獲得。

首先，您為什麼要使用Grafana進行警報？通常，您的用例涵蓋了多個後端（第5章）。也就是說，想像您正在使用Prometheus存儲指標和OpenSearch來存儲日誌和痕跡，並希望在中心位置定義這些後端的警報。

通過統一的警報，Grafana現在支持以下所有用例：

支持一系列數據源，包括Amazon CloudWatch，Azure Monitor，Google Cloud Monitoring，InfluxDB，Jaeger和Prometheus。 您可以按小組組織警報（類似於我們在Prometheus中看到的）。 支持不同的警報器，包括與Prometheus One的集成。 For notifications (http://mng.bz/zX7w), you use so-called contact points, sup- porting a wide array of types (https://grafana.com/docs/grafana/v9.5/alerting/ fundamentals/contact-points/), including Discord, Email, Kafka, PagerDuty, Slack, Telegram, and WebHook. 您可以使用沉默抑制通知。請注意，警報規則仍會由Grafana評估，導致呼叫後端的潛在成本，但不會創建通知。

現在，讓我們繼續警告雲提供商的解決方案。

7.2.3 Cloud providers

每個主要的雲提供商都提供一個警報解決方案，通常與其產品深入集成。 這些產品是您應該始終考慮的絕佳基準，因為它們涵蓋了雲提供商的API。這些包括

AmazonCloudWatch警報（http://mng.bz/0kjp），該警報使用Amazon SNS發送電子郵件或SMS通知AzureMonitor Alerts（http://mng.bz/keyp），支持電子郵件，SMS和PUSHIFICATION支持一系列通知類型，從電子郵件到pub/sub

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

如前所述，您也可以使用Grafana的這些警報解決方案（如果您想控制該窗格的警報，則可以使用CloudWatch數據源），從而提供單個玻璃窗格。現在，我們正在切換齒輪，以快速查看最終用戶跟踪以及資源使用率跟踪。

7.3 Usage tracking

警報總體上是一個反應性過程。這意味著正在發生某些事情，基於此，有人收到通知，對事件做出反應。在本節中，我們涵蓋了另一種雲操作技術，您可以用於各種情況，從trou-bleshooting到安全性，再到計劃。 我說的是跟踪用戶及其訪問權限。讓我們從雲本地應用的最終用戶開始。

7.3.1 Users

要了解最終用戶如何使用您的應用程序，您需要有一種方法來跟踪其應用程序中的活動。這使您可以識別流行的功能和潛在的UX問題。 多年來，*real user monitoring*（朗姆酒; https：//github.com/ open-telemetry/oteps/desise/169）術語已被廣泛發布。例如，對於Web應用程序，有廣泛使用的Google Analytics（https://analytics.google .com/adamletics/web/），並且您可以在AWS中使用CloudWatch Rum（http://mng.bz/9drx）。 所有這些產品都需要在網頁中將一些小JavaScript片段的前期嵌入，以在專用的位置查看使用信息（人口統計，區域，設備類型，用法路徑等）。 如果您想深入研究主題，我建議您仔細閱讀Chinmay Gaikwad（http：// mng .bz/jpjz）的出色文章“真實用戶監控（朗姆酒）：監視和可觀察性”。

> 請注意，朗姆酒要注意的一件事是單頁應用程序的興起（spa; https：//en.wikipedia.org/wiki/single-page_application）網站。在SPA的上下文中，傳統的指標，例如頁面加載時間，沒有longer物質（或有意義），因為該頁面僅在會話開始時加載一次，並且UI更新實際上是本地渲染。有關此主題的更多信息，請查看O11ycast的第60集，“以客戶為中心的觀察能力，具有Todd Gardner和Winston Hearn的能力”（http://mng.bz/wz94），以及文章，“可觀察的前端：browser opentemetry opentent the browser in the browser in the browser in the browsica by the browsica ker y jessica kerr”，

朗姆酒也是CNCF Opentelemetry項目的考慮。 Opentelem- Etry增強建議OTEP 169（http://mng.bz/eqoo）通過提議新的數據模型和語義來涵蓋這一問題，支持朗姆酒遙測的收集和導出，這是一種新的信號類型的收集和出口，以及痕跡，軌跡，儀表，logs和Profiles。雖然仍處於早期的早期，但您可以期望在這個領域中有更多的發展，尤其是因為日誌信號類型在2023年6月穩定（這意味著您可以使用OpentElemetry來處理產品中的日誌）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

朗姆酒專注於最終用戶，但還有另一個用戶跟踪類別：內部用戶，包括開發人員和運營商。 例如，您可能需要記錄用戶或服務帳戶採取的措施的記錄，或者出於合規性或風險審計原因。此類服務的一個示例是AWS CloudTrail（http://mng.bz/n2gn），支持一長串服務，包括Amazon API Gateway，AWS CloudFormaTion，Amazon Cloudwatch，Amazon CloudWatch，Amazon EKS，AWS Lambda和AWS Systems經理。

您還可以從上述內部用戶審核日誌中設置警報。話雖如此，現在讓我們繼續花費。

7.3.2 Costs

跟踪資源訪問的完全不同類別是成本跟踪。無論您是否想教育內部用戶有關成本和使用情況（梳理）或提供需要對某個宗旨收費（退款）的服務，您都想了解實際成本的盡可能良好的水平。我們可以區分計量，這是計算訪問權限的（例如，在過去的15分鐘內，API端點XYZ被稱為3,478次），以及可能取決於許多事情的實際費用，從預期承諾到您可能會消費的特殊交易的特殊交易。

雲提供商通常提供服務，使您（例如）探索您的用法（例如，AWS成本和用法報告[CUR]，http://mng.bz/d46n）。此外，您可能希望選擇使用提供成本分配和資源使用的解決方案，以調整為單個執行單位，例如OpenCost（https://www.opencost.io/）為Kubernetes環境所做的。

通過對用戶和用法跟踪和報告的高級討論，我們已經達到了雲操作章節的結尾。儘管我們討論的大多數主題主要是由運營商角色的人們理解和擁有云的系統，但至關重要的是，從開發人員到測試人員到釋放經理，都有所涉及的每個角色，對這些概念都有可靠的掌握。在完全無服務器的環境的情況下，這可能意味著實際上是開發人員正在呼叫（http://mng.bz/lw76）。

Summary

要成功地操作雲本地應用程序，您需要採用一種結構化方法來檢測，處理和從事件中學習。 健康和績效監控使您可以擁有系統 - 外部視圖，並了解某些東西無法正常工作。 如果您打電話進行服務並發生事件，請專注於停止出血； 對出現問題的分析是在後來的，理想情況下是無罪的屍體。 提醒的目的是使人類自動通知影響系統的事件。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

您想避免警報疲勞，因此需要製定政策，使您可以收到有意義的通知（但不是太多）。 Prometheus允許您定義警報規則，分組，過濾，付費和通知由單獨的組件：AlertManager處理。 如果您想集中管理各種後端的警報，則可以使用Grafana。 使用跟踪意味著捕獲應用程序中的最終用戶互動或捕獲對系統資源的訪問以及成本。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

8 Distributed tracing

This chapter covers

什麼是分佈式跟踪以及為什麼重要的是看到分佈式跟踪在進行故障排除考慮如何以及在何處使用分佈式跟踪如何選擇分佈式跟踪解決方案

在“過去的美好時光”中，從可觀察性的角度來看，運行整體應用時，事情很方便。您有一個Web服務器作為前端層，您的應用程序服務器（例如基於Java的應用程序）和一個數據庫，例如MySQL。在此設置中，弄清楚哪些組件引起了問題並不難。這些組件的日誌將涵蓋*where*和*what*。此外，您可以捕獲一些指標，表明整體健康和表現。

隨著雲本地系統和微服務的興起，與最終用戶相比，與整體相比，他們不應注意到用戶體驗的差異。從開發人員和操作的角度來看，我們希望從微服務中受益。這包括

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

feature velocity (different parts of the application can be written and deployed inde- pendentl y), partial availability (while certain parts of the application might not be available to the end user, the overall application might still be working), as well as enabling us to use the best-fitting programming languages and data stores for a cer- tain functionality (e.g., JavaScript for a frontend or an in-memory key−value store, like Redis, for a shopping 籃子 ）。

注意，我們先前介紹了特徵速度，這是基於微服務的架構的優勢。在這裡深入研究，增加特徵速度的原因是，較小的單元可以增加部署速度。 換句話說，如果您的部署單元較小，則更容易測試，升級和修復。 值得注意的是，即使將“小型單位”的想法應用於嚴格解耦模塊時，一些企業即使被認為是單個單體的發展速度也提高了。

另一方面，微服務確實是分佈式系統，因此，它們比單片應用程序更複雜。 可觀察性不再是可選的。問題的一部分是確定發生了什麼地方。您不必是eBay，Netflix或Uber，即處理數百個微服務，即可面對這一問題。即使您“只”擁有構成雲本地應用程序的幾十個服務，也可以準確弄清楚沿請求路徑的哪些服務導致錯誤或導致較慢的響應可能很棘手。

那麼，您如何概述您的所有服務如何一起播放以及發現無法正常工作的服務呢？也許，您可以深入研究特定的服務，以檢查其性能。

請注意，分佈式系統的一個挑戰是（潛在的）服務在不同的機器上運行。這些服務的日誌可能會帶有未100％對齊的時間戳（http://mng.bz/bmp0），這很難處理，因為它會破壞跨服務的日誌訂購，因此破壞了原因 - 效應分析。即使將日誌攝入中心位置時，通常需要根據標籤或標籤手動恢復執行的動態。因此，儘管可以在微服務的背景下單獨使用日誌，但它並不是最有效的時間使用，並且可能需要分析師的技能比我們在本章中討論的解決方案更多。

在本章中，我們仔細研究了上述挑戰的解決方案：分佈式跟踪。我們在第2章中的高級討論了這個主題，但是現在是時候深入研究了。我將首先在可觀察性空間中對這個相對較新的概念進行總體介紹，然後在具體用例的背景下，我們將看到分佈式跟踪。最後，我們將在選擇和使用分佈式跟踪解決方案的情況下結束本章。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

8.1 Intro and terminology

在本小節中，我將首先引導您介紹一個勵志例子，以顯示分佈式跟踪的力量。然後，我將定義整章中使用的術語。

8.1.1 Motivational example

為了設置場景，讓我們從一個基於微服務應用程序的具體示例開始，從而成為動機。圖8.1顯示了這樣的示例應用程序。這是電子商務應用程序的簡化版本，可允許最終用戶瀏覽和搜索項目，訂購併將其發貨。要變得更具體，請想像您正在構建一個電子商務應用程序，該應用程序可讓您購買望遠鏡，類似於我們將在附錄中介紹的演練示例。

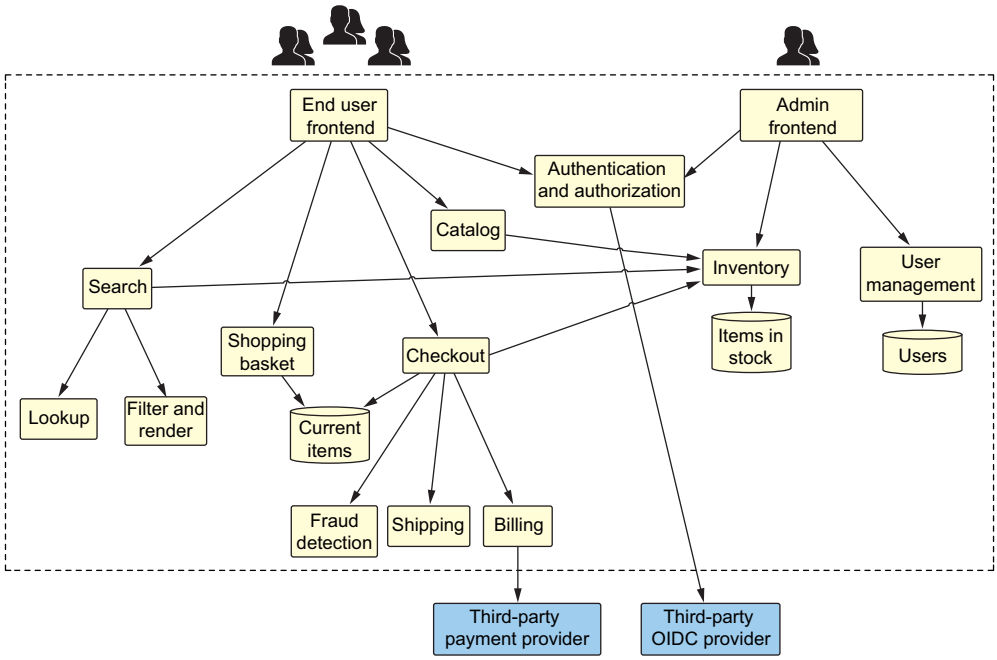


圖8.1基於微服務的示例電子商務應用程序（在虛線框中）

在我們的電子商務示例應用程序中，我們有兩個邏輯部件，其中一些功能（例如付款詳細信息）由外部（第三方）服務提供，以及在這些邏輯塊之間共享的一些（身份驗證）（虛線框是我們的Web應用程序）。

圖8.1的左側顯示了最終用戶 - 服務（客戶）部分，啟動功能，例如搜索項目並允許最終用戶將項目添加到一個

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

購物籃。 圖的右側顯示了管理部門的部分，使商店管理員可以維護庫存並管理客戶。

讓我們深入到最終用戶的請求路徑之一 - 我們電子商務應用程序的部分（請參見圖8.2）。這描述了消費者點擊購買按鈕後處理要求的各種組件；另請注意，在本節的其餘部分中，我們將重點關注最終用戶的零件，而不是管理部件。因此，在一個庫斯·托爾（Customer）將一對雙筒望遠鏡（例如雙筒望遠鏡）放入購物籃之後，他們想退房以支付商品並將其發貨。

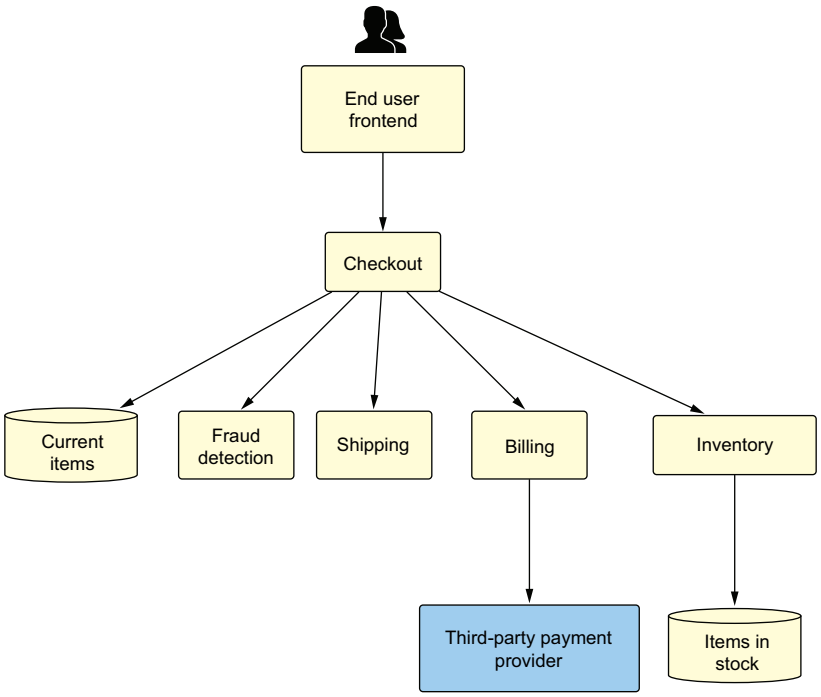


圖8.2應用程序中的單個請求路徑

那麼，最終用戶輸入所有必要的信息（例如船舶地址和付款詳細信息）後會發生什麼？如圖8.2所示，此情況的請求路徑如下：最終用戶前端呼叫結帳服務，而該服務又有幾個階段要完成，其中一些可能終止請求（例如，由於錯誤的用戶輸入或外部依賴性的超時）：

結帳服務查詢當前項目數據存儲。 調用欺詐檢測服務，檢查是否有任何理由不進行。 運輸服務的運輸信息用於確定船舶費用。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

計費服務收集了所有付款信息，並呼籲第三方提供商處理付款。 最後但並非最不重要的一點是，如果一切都在井井有條，結帳服務會調用庫存服務以更新庫存。

圖8.2中微服務的表示形式在觀測能力前端中具有不同的名稱，例如*service map*或*dependency graph*。該服務圖提供了架構的不錯概述，並顯示了信息流。但是，它不會捕獲時間方面。換句話說，我們需要一種方法來可視化沿時間軸的服務執行，通常稱為*waterfall visualization*。 圖8.3顯示了一種替代方法來表示基於時間的結帳服務所做的調用。

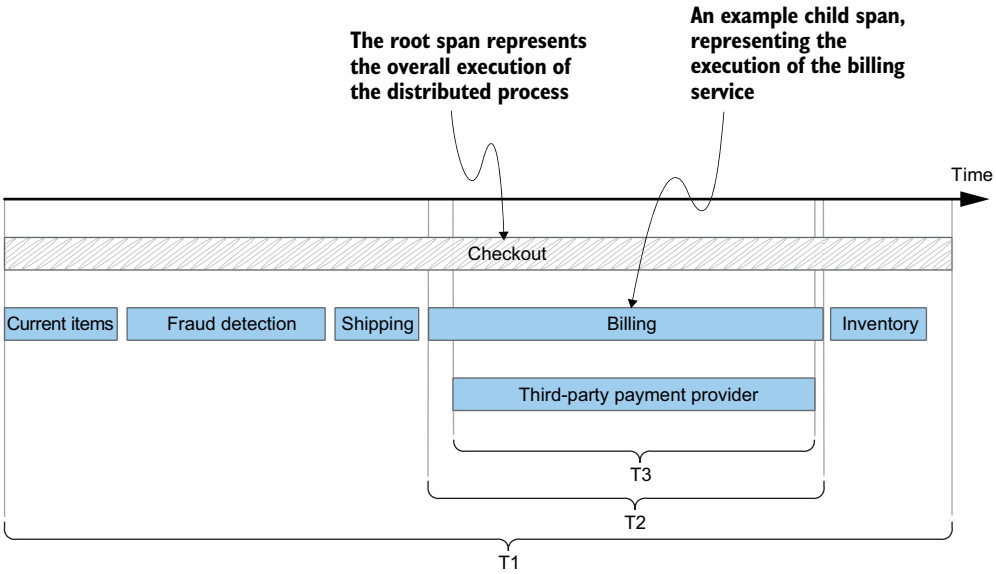


圖8.3應用程序中的單個請求路徑，作為時間（瀑布）可視化

圖8.3中描繪的整個單元稱為*trace*，表示瀑布可視化中目錄請求的端到端處理。每種服務調用，例如調用計費服務的結帳服務，都是一個跨度（比較我們在第2章中的討論）：

根跨度是結帳服務的整體執行，並花時間T1端到頭執行。 兒童跨度的示例是執行計費服務（需要T2才能完成）和執行第三方付款提供商呼叫（這需要T3完成）。除了我們的計費服務需要做的一些設置時間外，大多數情況下似乎是通過調用外部服務來佔用的。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

瀑布可視化的一個優點是，我們可以簡單地看出，通過視覺檢查痕跡中的所有跨度，大部分時間都花在那裡（在我們的情況下，在圖8.3中，結帳服務在計費階段的一半時間大約是一半的時間）。通過建立基於微服務的電子商務應用程序的這個具體的激勵示例，讓我們繼續對您在分佈式跟踪空間中遇到的術語進行更正式的定義。

8.1.2 Terminology

儘管各個供應商之間的術語存在差異，但以下術語通常在跨產品之間得到很好的定義和理解。我主要關注OpenTelemetry命名（http://mng.bz/d1qn），尤其是在分佈式跟踪解決方案之間沒有共識的情況下：

a *span*代表一個工作單位，並且捕獲了請求的一部分（與圖8.2進行比較）。這可以在服務範圍內發生，例如在服務中或跨服務中執行功能。跟踪中的一種特殊跨度是*root span*，它是在請求輸入應用程序時創建的。一個跨度具有一個ID，通常，它存儲它所屬的跟踪的ID以及它是從它觸發的根跨度以及包括有效負載的元數據（想想時間戳，標籤等）。 請注意，在AWS X射線中，該單元稱為段。

a*trace*是分佈式系統中請求的端到端處理，由一組暫時訂購的跨度組成。它具有ID和元數據。 在跨服務調用的背景下，*trace context*捕獲了各個痕跡和請求的所有唯一標識。這使得通過稱為*context propagation*的過程超越了single服務的痕跡，在該過程中，此跟踪上下文是在網絡調用中轉發的。調用服務（父）將跟踪上下文的所有必要數據附加到下游（兒童）服務的請求，例如，在計費服務和第三方付款提供商之間的示例電子商務應用程序（圖8.2）中。如今，隨著OpentElemetry將自己確立為標準，通常是基於跟踪上下文（https://www.org/tr/tr/tr/trace-context/）W3C推薦（使用HTTP標頭的標準方法來傳播上下文））。

要減少數據量（跡線和跨度），您可以使用一種稱為*sampling*的技術。這使您可以根據特徵有選擇性地刪除跡線，包括固定百分比或基於跟踪的性質 - 例如，如果其中一種服務中存在錯誤，並且我們看到在代表服務的跨度中陷入了錯誤。

現在，我們對分佈式跟踪術語及其含義有基本的了解，讓我們看一些用例。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

8.1.3 Use cases

分佈式跟踪不僅在分佈式系統中有用，在分佈式系統中，服務通過網絡呼籲其他人，而且對整體應用程序也有用。通常，在以下情況下，您可以從中受益：

Monitoring microservices-based applications - 這些允許您獲得應用程序實際體系結構的端到端概述，包括呼叫依賴。您不依靠該應用程序如何工作的靜態體系結構圖，而是可視化應用程序的實際工作方式。您可以確定服務映射的錯誤狀態（哪些服務錯誤）和服務調用的績效問題（服務調用的延遲圖）。 *Optimization* - 用於確定大部分時間都花在沿請求路徑的位置。您可以查明最慢的服務，並在那裡開始潛伏期的選擇，並有望在投資方面產生最大的收益。 *Release management* - 推出新版本的服務時，您可以評估服務對整體請求路徑的影響，並在新版本之前和之後進行比較。 *Troubleshooting* - 在事件的情況下，正如我們在第7章中討論的那樣，您可以使用分佈式的跟踪解決方案來找出哪些服務促成了該問題，並可以將其滾回已知的良好狀態以阻止出血。我們將在本章的下一個小節中看到這種常見的用例。

好的，那是足夠的理論！讓我們通過分佈式跟踪使我們的手弄髒，重點是故障排除用例。

8.2 Using distributed tracing in a microservices app

在本節中，您將在一個簡單的示例微服務應用程序的上下文中使用分佈式跟踪。 目的是了解服務如何共同運行，一旦我們開始模擬錯誤，請使用分佈式跟踪來確定哪些服務引起問題以及您遇到的錯誤。

8.2.1 Example app overview

我們的示例應用程序（稍後設置）由四個服務組成，如圖8.4所示。我們有一個前端服務作為入口點，該服務稱為兩個服務：服務A和服務B；反過來，服務B致電其他兩個服務（B0和B1）。一些服務還要求網絡上的靜態資產模擬對外部第三方API的呼叫。請注意，我們的示例應用程序無濟於事； 所有服務都使用HO11Y（http://mng.bz/rwxg）合成信號生成器來生成跟踪，我們可以配置它以模擬某些故障模式。

我們的示例應用程序堆棧包含構成微服務應用程序的五個服務以及OpenTelemetry Collector，以攝取從四個中生成的跨度

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

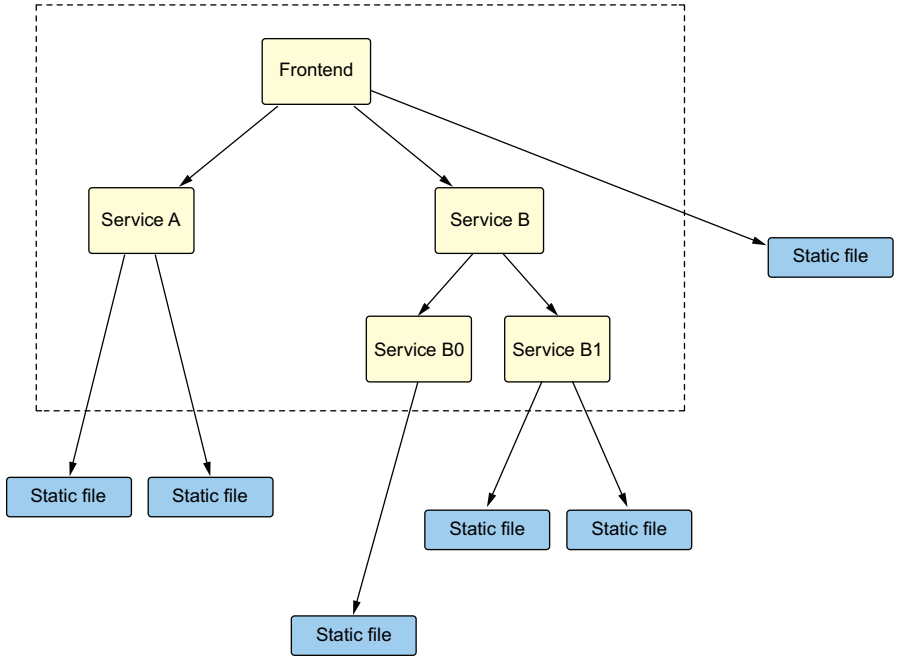


圖8.4示例微服務應用程序體系結構

將分佈式跟踪工具（CNCF Jaeger）提供服務。該設置基於您在第6章中已經看到的內容。

8.2.2 Implementing the example app

為了忍受我們的示例應用程序堆棧，使用清單8.1中顯示的docker撰寫文件，請從CH08 / jaeger/ Directory中執行以下命令：

```
docker-compose -f docker-compose.yaml up
```

請注意，HO11Y合成信號生成器具有通過環境變量定義的幾個配置選項。使用HO11Y_PORT，您可以在哪個端口HO11Y聆聽服務請求下定義，並且使用Downstream0到Downstream4，您可以定義HO11Y實例應呼叫的URL（並且，因此，允許您連接並模擬任意的複雜微服務）。為了模擬刪除請求，我們將使用HO11Y_INJEXT_故障環境變量，這會導致HO11Y實例在一半的情況下返回一些4xx或5xx HTTP狀態代碼，而不是返回200 HTTP狀態代碼。 最後，為了模擬另一種類似的故障模式（上游服務的節流），我們將使用HO11Y_CUTOFF_TPS環境變量定義一個截止點，除此之外，Ho11y將返回429 HTTP狀態碼（https:///httpstpstatuscodes.org/429/）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

清單8.1顯示了Docker組成文件的相關片段（專注於我們的微服務應用程序的接線和配置）。

清單8.1 Docker構成Jaeger設置的配置

```
frontend:
    environment:
    - "DOWNSTREAM0=http://service-a:8766"
    - "DOWNSTREAM1=http://service-b:8767"
    - "DOWNSTREAM2=https://opentelemetry.io/img/logos/opentelemetry.png"
  service-a:
    environment:
    - "HO11Y_PORT=8766"
    - "DOWNSTREAM0=https://mhausenblas.info/"
    - "DOWNSTREAM1=https://otel.help/"
  service-b:
    environment:
    - "HO11Y_PORT=8767"
    - "DOWNSTREAM0=http://service-b0:8768"
    - "DOWNSTREAM1=http://service-b1:8769"
  service-b0:
    environment:
    - "HO11Y_PORT=8768"
    - "DOWNSTREAM0=https://example.net/"
    # - "HO11Y_CUTOFF_TPS=1"
  service-b1:
    - "HO11Y_PORT=8769"
    - "DOWNSTREAM0=https://www.w3.org/TR/trace-context/"
    - "DOWNSTREAM1=https://example.org/"
    # - "HO11Y_INJECT_FAILURE=enabled"
```

Frontend service of our microservice app, which calls service-a and service-b

The service-a that calls out to static web content

The service-b that calls service-b0 and service-b1

The service-b0 that can be throttled and then returns HTTP status code 429

The serviceb1 that can simulate failure by returning HTTP 4xx/5xx status codes half of the time

下一個列表中顯示的OpenTelemetry集合器配置本質上與我們在第6章中使用的相同，單個跟踪管道以攝入跨度跨入Jaeger。

清單8.2收集器配置用於在OTLP中收集痕跡並攝入Jaeger

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
processors:
  batch:
    timeout: 1s
    send_batch_size: 10
exporters:
  jaeger:
    endpoint: jaeger:14250
    tls:
      insecure: true
service:
  pipelines:
```

The batch size is intentionally small for demonstration purposes.

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
traces:
  receivers: [ otlp ]
  processors: [ batch ]
  exporters: [ jaeger ]
```

Receives spans from the instrumented ho11y services in OTLP format

Sends spans in batches rather than all at once

Ingests spans into the Jaeger backend

請注意，與第6章中的設置相比，我在此處刪除了收集器的調試級記錄配置，因為我們不需要它，因此它更加分散注意力。

8.2.3 *The "happy path"*

現在，是時候前往http：// localhost：16686/打開Jaeger UI，我們首先要查看“ Happy Path”（見圖8.5），這意味著一切都按預期工作，沒有任何錯誤。這為我們提供了我們期望的基線，即服務應如何工作（例如執行時間）。

作為探索和理解我們的分佈式系統的初始步驟，讓我們創建服務映射（動態生成）整體架構。 為此，在標有搜索的左側框中選擇前端，然後單擊底部的“查找跟踪”按鈕。現在，您應該在右側看到一張隨著時間的推移執行持續時間的圖表，如果現在單擊其下方的深度依賴圖表按鈕，則應看到類似於圖8.5的內容。另外，請注意我們看不到的內容：外部服務或服務電話並未表示為跨度。

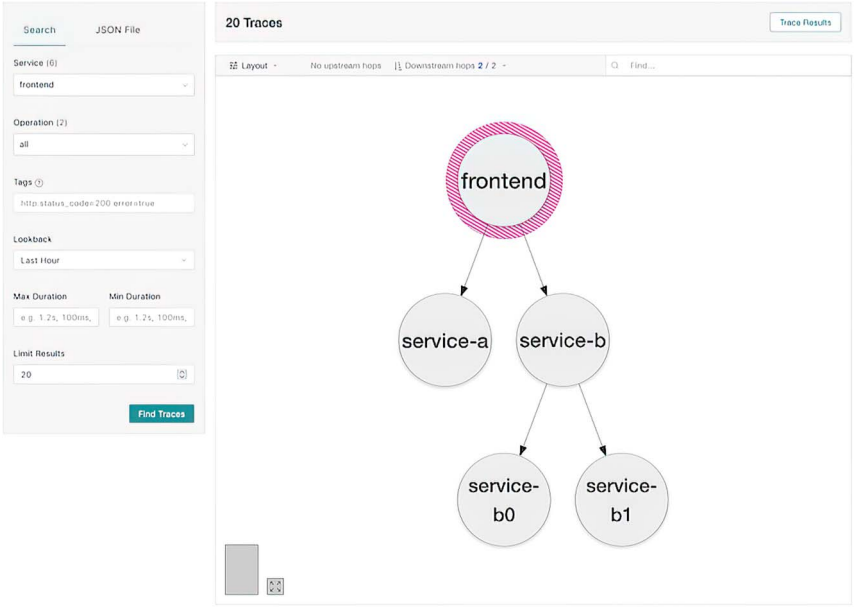


圖8.5故障排除演示微服務應用程序：快樂路徑依賴圖圖

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

接下來，我們希望看到瀑布模型。這意味著我們要可視化服務執行的節奏渲染，因為所有四個服務都可以共同努力服務。 為此，單擊“依賴關係圖”的上右上角的跟踪結果按鈕，然後選擇一個跟踪（哪一個都不重要；只需選擇列表中的頂部）即可。然後，您應該沿著圖8.6的線路看到一些東西。此屏幕截圖顯示了一個擴展的單個跨度，該跨度著眼於Service B0的調用。從中，我們可以說要執行大約135毫秒的時間，並且在呼叫外部資源的情況下，它花費了超過129毫秒。現在，我們已經看到了快樂的道路，讓我們模擬一些失敗，並使用Jaeger弄清楚發生了什麼。

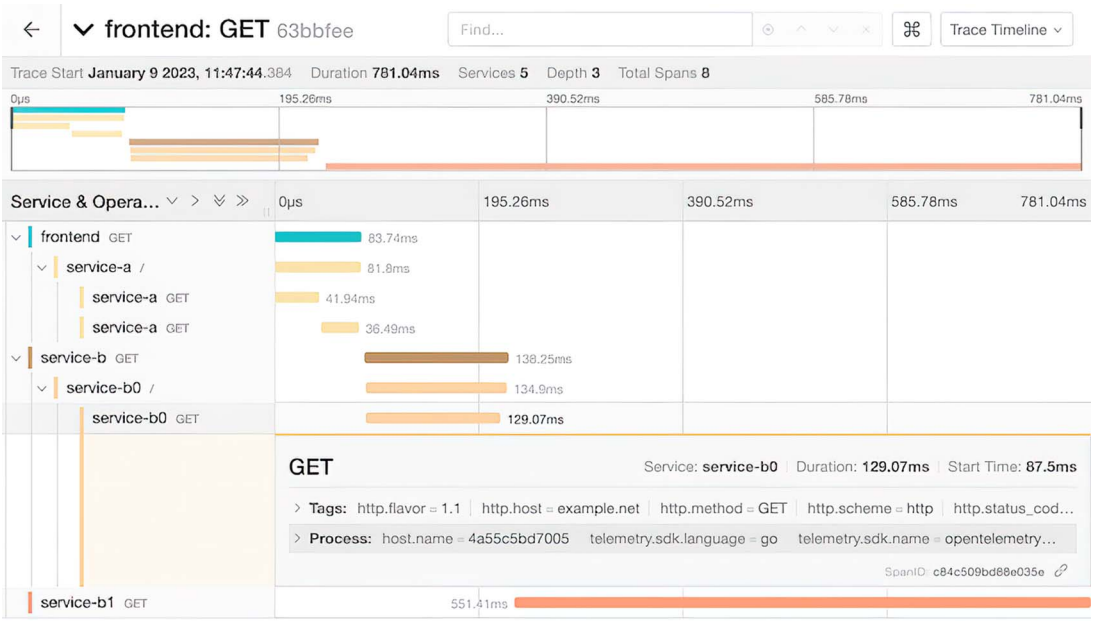


圖8.6故障排除演示微服務應用程序：一個快樂的路徑示例跟踪其跨度

8.2.4 Exploring a failure in the example app

為了開始我們的失敗示例，請停止使用CTRL-C組成堆棧，然後重置應用程序，還可以執行Docker-Compose -F Docker-compose.yaml向下執行docker-compose。現在，轉到CH08/ JAEGER/ DIRECTORY，然後在您選擇的編輯器中打開Docker- compose.yaml文件，在第62行中的HO11Y_CUTOFF_TPS前面刪除評論（#），以及HO1Y_INGENT_FAILURE的HOH1Y_CUTOFF_TPS以及第76行。這會導致服務B0和B1顯示錯誤。

將更改保存到編輯器中後，請執行docker-compose -f docker- compose.yaml以提高堆棧，然後回到您選擇的網絡瀏覽器中的Jaeger UI。 再次，在左側框中選擇標有搜索的前端，然後單擊底部的“查找跟踪”按鈕。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

如果您現在通過過濾HTTP狀態代碼（在標籤輸入字段中，添加501或通過錯誤= true）過濾或單擊顯示錯誤的跨度之一（您應該在該跨度上看到紅色的單詞標記），則應查看類似圖8.7所示的內容。這是什麼意思？ Jaeger為我們提供了視覺上的失敗渲染，使我們能夠深入研究顯示錯誤的服務。通過單擊跨度，您可以探索服務並比較諸如執行時間持續時間之類的內容與您從快樂路徑中擁有的基線以及查看錯誤代碼是什麼。這意味著您可以確定會導致問題的服務，然後專注於解決問題的服務。

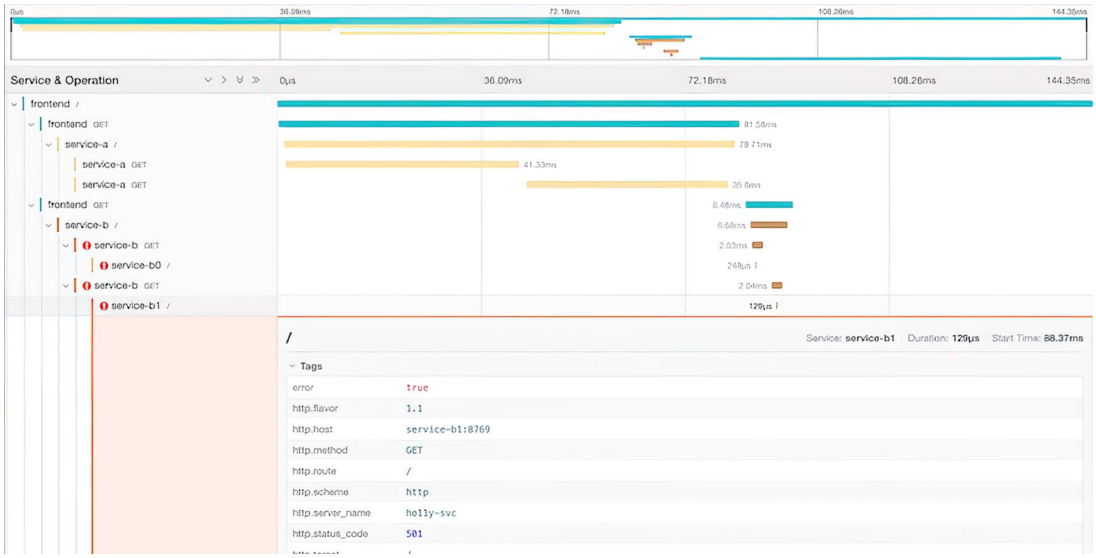


圖8.7故障排除演示微服務應用程序：示例故障跟踪和導致故障的跨度

現在使用分佈式跟踪來解決基於微服務的應用程序進行故障排除的動手練習現已完成。接下來，我們將討論如何在實用設置中從我們的新應用程序中受益。

8.3 Practical considerations

現在，您知道了分佈式跟踪的基礎知識，並且已經在具體用例的背景下在行動中看到了它，讓我們更多地談談在分佈式跟踪中全部考慮時要考慮的事情。在高水平上，重要的是要了解並非每個跨度的痕跡都具有相同的信息值。例如，如果在100次中有99次看到HTTP狀態代碼為200的跨度，那麼這會告訴您什麼？在很重要的情況下，帶有500 HTTP狀態代碼的一個跨度表明，發射跨度的服務可能會更深入地研究某種問題。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

此外，由於痕跡由一塊跨度組成，因此您希望在查看痕跡時，您會看到構成痕蹟的所有跨度。這聽起來很明顯，但是在實踐中可能會更加困難。以下幾個挑戰是最突出的挑戰，可能會導致破碎和部分痕跡：

Context propagation - 要查看所有跨度，每個服務都需要傳播跟踪信息（根跨度和元數據）。如果其中一項服務（您不擁有）不參與分佈式跟踪？ 如果您使用的是未使用儀器且不會傳播的雲提供商API怎麼辦？ API支持與您使用的內容不同（例如，專有與W3C跟踪標頭）的情況如何？為了解決這個問題，您可能需要升級API集成（支持新標頭類型）；使用中間件或翻譯層翻譯標題類型；而且，有可能與您的提供商遊說以支持追踪。

Dropped spans—This can be due to a faulty or starved agent (e.g., an Open- Telemetry collector’s in-memory queue overflows and may even be OOMed— that is, the Linux kernel kills the process, due to not enough memory being available) or some intermediary being configured to filter certain spans (e.g., a Gateway deployment in OpenTelemetry; https://opentelemetry.io/docs/ collector/deployment/），在邊緣使用的收集器將跨度發送給Central收集器，對您不知道。

您通過分佈式跟踪進行的投資包括跨度的產生（理想情況下，通過自動啟示）； 該集合（例如，請考慮opentelemetry collector）；攝入和存儲成本；最後但並非最不重要的一點是，前端的查詢或用法（例如，為了在瀑布圖中繪製痕跡或提供服務地圖和其他骨料）。

讓我們放大如何選擇值得收集並攝入後端的跨度的主題：換句話說，採樣。

8.3.1 Sampling

分佈式跟踪意味著在生產環境中，您定期看到大量數據正在生成。鑑於其與分佈式系統（微服務）的自然對齊，面對數十億跨度並不少見，每天代表大量數據。那麼問題如下：我們如何選擇相關跨度以收集和存儲？我們在此處用於選擇過程的術語是 *sampling*。請注意，採樣有一些先天的限制（https：// research .swtch.com/sampe），但忽略目前，可以使用以下高級策略：

No sampling - 您攝入所有跨度，這對於開發/測試環境可能很有用，可以評估工具，或者如果您的工具不收取數量的費用。 調查軌跡時，緩存可以為查詢路徑提供很多幫助，但是跨度攝入路徑仍然具有挑戰性。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Head-based sampling - 在這種情況下，您決定在請求開始時發出哪個跨度（根跨度）。實施很容易，但是鑑於您可能不知道是否會有任何問題（樹的節點和樹葉），因此您可能會錯過有趣或有價值的跨度。此外，鑑於一開始做出的採樣決定，必須在全球範圍內進行決定。

Tail-based sampling - 從某種意義上說，它與基於頭部的採樣相反，因為在請求完成後做出了決定。 太好了，因為您可以確定哪些跨度是相關的，但是在實踐中，很難知道何時真正完成請求的挑戰。 例如，沿請求路徑的一項或多項服務可能會超時。您將等待多長時間才能聲明請求完成？

如果您對抽樣的更多詳細信息感興趣，我強烈建議Alex Boten的書 *Cloud-Native Observability With OpenTelemetry*（Packt，2022; http：// mng：//mng .bz/rw9b）。這樣，讓我們繼續研究資源主題。

8.3.2 Observability tax

當我們談論該小節開始時，我們談論掉落的跨度時，我們稍微談到了資源使用情況。讓我們退後一步，看看我喜歡稱之為“觀察能力稅”（即，在分佈式跟踪的背景下您會遇到的成本（儘管它們不一定是它所唯一的 - 有些（即使不是大多數），您也會使用指標和日誌看到，但是控件可能會有所不同）：

Instrumentation costs—Cover everything that has to do with generating spans, including the manual work you have to put into emitting spans from your appli- cation, the costs of finding and fixing bugs related to instrumentation, the addi- tional resource usage your app has due to instrumentation (e.g., even in the best case, auto-instrumentation will consume CPU cycles and occupy RAM you have to pay for).

Transport and ingestion costs - 至少在雲提供商會計的情況下（通常需要為網絡使用付費），突破網絡入口； 並將跨度從您的環境（可能是本土）發送到後端，在這裡您可能必須處理每TB的兩位數美元或歐元的費用。此外，您的分佈式跟踪可能會收取每GB的費用或攝入的跨度。

Storage costs - 尤其是您在提供商賬單中找到的最高訂單項目，因為存儲經過優化的效率，而且通常是免費提供的。您通常的保留期限有限或固定（例如，AWS X射線有30天，Datadog有15天）。

Query costs - 表明所有查找和API調用以檢索一系列跡線，以查看它們或以匯總形式顯示它們，包括直方圖（http://mng.bz/v1ap）； 服務地圖（http://mng.bz/x4dw）；

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

和高級可視化，例如Honeycomb的Bubbleup（https：//www .honeycomb.io/bubbleup）。

實際上，這是對所涉及的潛在成本的一種非常簡化的看法，您應該與選擇的供應商合作，以更好地了解特定情況下的成本，包括適用的情況，包括適當的折扣。雖然*are*的成本，但不要忘記，歸根結底，從分佈式跟踪中獲得的回報是，您不再使用微服務應用程序盲目飛行，並且可以預期平均維修/恢復的時間會減少（MTTR）和開發人員生產率的提高。

8.3.3 Traces vs. metrics vs. logs

請記住，僅一種信號類型通常無法回答您可能必須得出的所有問題。我們將在第11章中深入研究該主題。

獨立於相關性，有時您可以彼此衍生信號。例如，您可以根據觀察到的跨度創建指標，例如使用OpenTemetry的SPAN指標連接器（http://mng.bz/aoez）。 分佈式的跟踪空間正在迅速發展，各種供應商和社區正在嘗試和迭代（https://keyval.dev/distributed-tracing-2025/），以提高其可用性和可用性，包括自動化的收集和電信信號的壓力。

Summary

分佈式跟踪為您提供了*where*和*what*，涉及分佈式系統的可見性。 執行是通過微服務分佈的，為了獲得整體圖，必須通過上下文傳播將各個運動部件縫合在一起。 跟踪是分佈式系統中請求的端到端處理，跨度代表一個工作單位，例如調用單個服務或對數據庫的調用。 從監視基於微服務的應用程序到延遲優化到故障排除，有很多可能的用例列表。 分佈式跟踪涉及幾項費用，包括用於查詢成本的運輸和攝入成本的工具成本。 您將必須從用例中向後工作，以確定投資回報率（請參閱第2章），確保您的利益相關者了解MTTR的降低，或者實際上是提高Devel-devel-teal-faster-loductitivity的價值。 一個信號類型，例如指標或日誌，通常不能產生可行的見解；因此，對信號相關的支持是選擇分佈式跟踪解決方案的關鍵因素。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

分佈式跟踪仍然是一個相對年輕的領域，並且正在發生許多創新，這意味著在諸如OpentElemetry之類的標准上建立構建應有助於降低投資風險，並在您想切換後端時使您的解決方案更加便攜。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

9 Developer observability

This chapter covers

開發人員可觀察性意味著什麼什麼是連續分析以及為什麼要關心的是cloud和開發人員可觀察性如何驅動開發人員生產力開發人員可觀察性空間中有哪些工具可用

在第1章中，我們討論了不同的可觀察性用例和目標受眾。在過去的幾章中，我們經常有相當偏見的觀點，即關註生產中運行的雲本地應用的主題。現在，我們正在改變齒輪，以專注於開發人員作為可觀察性的主要受眾。 我們將談論他們的自然棲息地（IDES和命令行）以及開發人員的可觀察性如何有用。因此，本章是關於開發人員的觀察能力，而不是關於開發人員或開發人員的觀察力（例如，創建的時間或提交或每天產生的LOC）。

在達到挑戰的核心之前，我們將首先定義一個與對話有關的概念：*shift left*。這個概念始於測試的背景

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

(http://mng.bz/zq1p)和ci/cd，然後擴展到安全性（http：// mng .bz/rxdk）。 如今，這個術語也是營銷部門喜歡在可觀察性中應用的東西。

左移動的概念（見圖9.1）是使開發人員能夠做傳統上測試人員或操作人員的任務和職責的事情。在“過去的美好時光”和相當傳統的環境中，開發人員仍然主要關註一件事：編寫代碼 - 也可能還有一些單元測試。歸根結底，開發人員“將工作扔在牆上”進行測試；打包（例如，進入戰爭文件）；最後，部署和操作。所有這些步驟都是由開發人員以外的其他角色（例如發行工程師，QA測試人員或運營商）執行的。如果您將軟件開發和操作過程視為連續性，而最左側的一側代表開發人員攪動源代碼，則最右側代表部署二進制的操作員，您可以想像*shift left*的意思。

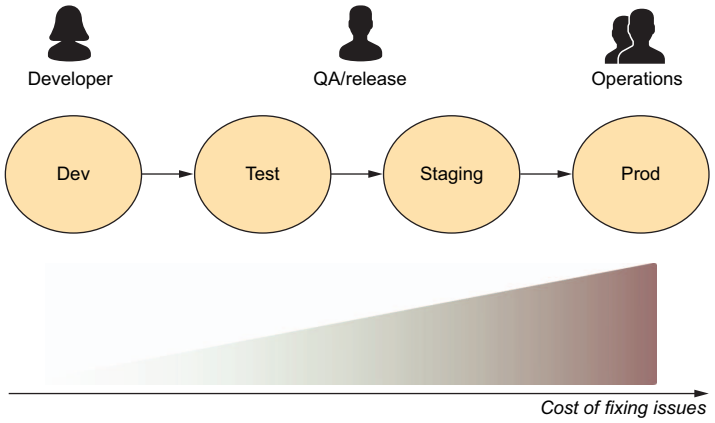


圖9.1向左移動

將左移動概念應用於可觀察性，*developer observability*意味著使用遙測信號為開發人員提供可行的見解，以加速開發，調試代碼或了解新功能在資源使用方面的影響。這導致縮短反饋循環，從而降低總體成本以識別和解決問題。 例如，與其需要等待四個星期的開發人員，以找出代碼是否通過測試套件或在產品環境中崩潰，而是可以，而且通常必須在“拉動或合併請求發送”過程中運行測試套件。

在本章中，我們將首先討論一種稱為*continuous profiling*的特定開發人員可觀察性方法，以及它如何幫助您了解您編寫的代碼的資源使用率（例如CPU和內存），即使是跨源代碼的不同版本。然後，我們將基於可觀察性工具的開發人員的生產力，這些工具深入融合到編輯和IDE中，使您作為開發人員洞悉

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

您引入的更改如何影響您的服務，遠遠超出了傳統的測試“全綠色”。在本章的最後一個小節中，我們將討論如何在開發人員的可觀察性空間中開始啟動以及要注意什麼。這樣，讓我們直接進入第一個主題：連續分析 - 它是什麼以及何時可以從中受益。

9.1 Continuous profiling

在本小節中，我們將查看連續分析，其用例以及可供您使用的工具。首先，我們將在操作系統的背景下查看其起源，然後我們將回顧可為您提供的通用技術和工具。最後，我們將在端到端的演練示例的背景下應用這些知識。

像往常一樣，我們將涵蓋開源工具以及商業產品。開源工具使您可以探索此空間，而無需投資於許可之類的錢。驗證了這項技術後，您可以確定投資回報率（ROI），其中有可能為其中一種產品付費。

那麼什麼是*continuous profiling*（CP）？這個術語有兩個詞。讓我們首先關注後者（分析），然後返回到本節後期（連續）的意思。在進入任何細節之前，讓我們在高級別上定義我們的含義：

> *Continuous profiling enables you to understand your operating system's and services' resource usage (such as CPU or main memory and function execution times), over time. You can benefit from it in troubleshooting and performance optimization scenarios.*

您可以使用持續分析不僅了解服務的當前性能和資源使用情況，還可以將其與以前的迭代進行比較。考慮以下用例：您在服務中添加新功能，並想評估現在的權衡。一方面，您的服務現在更加強大，更有用，但另一方面，它在CPU或內存方面可能會消耗更多的資源。

> 注意，從某種意義上說，由於您在服務的背景下運行，與分佈式跟踪的跨服務執行相比，連續分析與記錄有關。與日誌記錄一樣，通過分析，您對在服務的上下文中有興趣呼叫。但是，您對數值答案感興趣，例如執行函數或CPU利用率或某個代碼路徑中使用的主內存的數量，而不是日誌線發出的事件。從符號的角度（函數名稱，數據類型等），連續分析也與調試有關。但是，與調試相反，您正在整體上查看執行，而不是踏入單步。

歸根結底，新功能是否證明增加的資源使用是一個業務決策。但是，您現在可以通過連續分析，獲取原始數據來做出該決定。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

9.1.1 The humble beginnings

連續分析在哪里以及如何開始，術語中的*continuous*是什麼？讓我們退後一步。 Profiling has been around for decades, with GNU gprof (https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html) and Sun/Solaris DTrace (https://brendangregg.com/dtrace.html) as early examples, further developing in the context of Linux.通常在性能分析的背景下，通常在（操作）系統級別上考慮它。

現在，什麼是*profile*，個人資料在實踐中的外觀如何？我們將使用以下代碼中發出配置文件的應用。使用PPROF（https://go.dev/blog/pprof）來查看此GO代碼（通過CH09/CP/MAIN.GO，我們僅專注於相關位）：

導入（_“ net/http/pprof”）func QuickTask（）（結果字符串）{} func lure（）asttask（）（結果字符串）{} func main（）{go func（）{http.listenandserve（http.listenandserve（“ localhost：localhost 6060”，nil nil）（nil al）}（nil il al）}（fib for True for true for true true true）結果為%s \ n“，結果）結果= slottask（）fmt.printf（“慢任務結果為%s \ n”，結果）}}}
}

自動收集和公開配置文件的GO軟件包相對較快地完成（總CPU時間），一個相對較慢的函數（80％或更多的總CPU時間），一個簡單的Web服務器揭示了配置文件信息

請注意，函數slowtask（）和QuickTask（）除了使用CPU以外沒有其他任何有用的操作，以便我們可以在Profiler中看到一些東西。您現在可以在類似的終端會話中運行GO程序：

$去運行。快速任務結果是692530D563C476AEA00DD73067D7E146EDBECE4C17BFF4C慢任務結果是692530D563C476AEA00DD73067D7E14E146EDBECE4C17BFF4C9 ...

如果您導航到http：// localhost：6060/debug/pprof/，現在將看到NET/HTTP/PPROF GO軟件包的Web服務器輸出。查看此處的輸出顯示提供了各種類型的配置文件，包括有關Goroutines，Heap Cellecations等的詳細信息。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Next（並且您需要將GraphViz dot [https://graphviz.org/docs/ layouts/dot/]命令安裝），您可以使用image生成一個圖像，例如使用groutine call圖，使用

$ GO TOOL PPROF -PNG http：// localhost：6060/debug/pprof/goroutine > gr.png

http from http：// localhost：6060/debug/pprof/goroutine保存的配置文件{

好的，該網絡視圖或圖像可能會或可能不會有見地，但是還有另一種（更受歡迎的）使用PPROF：交互式模式（Think Shell）。 讓我們這樣做（收集CPU配置文件30秒，所以您需要等一點）：

$ go tool pprof“ http：//localhost：6060/debug/pprof/profof/profile？秒= 30”

從http：// localhost：6060/debug/pprof/profile？ seconsts = 30保存在~/ppprof/pprof.samples.cpu.002.pb.gz類型：cpu時間：cpu時間：cpu time：cpu time：cpu fime：cpu fime：cpu.002.3023，總計2023年3月17日， = 29.55S（98.09％）輸入交互式模式（鍵入命令“ help”，“ O” options）...

此時，您是在Pprof Interactive shell中，您可以列出前50個功能（在此處稱為*nodes*），並將累積焦點與Top 50 cum命令相似（編輯以顯示最重要的位）：

（PPROF）前50個暨活動過濾器：焦點=暨顯示節點，分別為15.54 s，29.55s的52.59％總數下降了47個節點（cum <= 0.15s）平flat％flat％sum％cum cum cum cum％3.72s 12.59％12.59％12.73％12.73％12.62％12.62％FMT。 7.17% 19.76% 4.15s 14.04% runtime.mallocgc 1.44s 4.87% 24.64% 1.44s 4.87% crypto/sha256.sha256block ... 0.12s 0.41% 50.19% 14.59s 49.37% main.slowTask ... 0 0% 52.59% 1.73s 5.85% main.quickTask

從PPROF中的TOP命令的輸出中，您可以說SlowTask（）函數（在30秒採樣期間）佔據了運行時的50％，並且同一時間的QuickTask（）功能佔6％。我不會在這裡詳細介紹，因為這項練習的目的不是教您Go的Pprof工具，而是要使您了解所涉及的努力。

因此，讓我們回顧一下我們必須做的事情才能看到一些東西：

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

1我們必須通過包括NET/HTTP/PPROF來儀器代碼。 2我們必須包括一個Web服務器來公開配置文件（或者，我們可以將數據寫入文件）。 3我們必須使用瀏覽器或curl（以及其他依賴項（例如DOT））獲得“時間點”結果，因此，對我們的資源使用問題的答案答案。 4我們必須等待30秒才能讓PPROF收集我們可以交互性查詢的配置文件。

請注意要說的真相，我一直在這里以GO為例，因為它帶有已經內置在其工具鍊和標準庫中的功能強大的參考器。 對於其他編程語言，同樣的練習可能意味著更多的努力。

例如，這裡有幾種特定於語言的方法來收集個人資料：

Java具有異步Profiler（https://github.com/jvm-profling-tools-tools/async-profiler）.net具有dottrace（https://wwww.jetbrains.com/profofiler/） CPROFILE（https://docs.python.org/3/library/profile.html）Ruby具有rbspy（https://rbspy.github.io/）rust rust pprof-rs（https：//github.com/tik.com/tikv/pprof-rs）

不要誤會我的意思；如您在Go示例中所見，分析是可能的，但需要專業知識，就像在2015年左右之前使用容器（當Docker可用時）是可行的，但需要對Linux內核效果（包括CGROUP和名稱空間）進行深入了解。如今，運行一個容器只是任何人逃跑的碼頭。

But then, in 2010, something happened: a bunch of Google engineers published the seminal paper “Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers” by Gang Ren, Eric Tune, Tipp Moseley, Yixin Shi, Silvius Rus, and Rob- ert Hundt (2010, IEEE Micro, https://research.google/pubs/pub36575/). 在本文中，作者描述了連續捕獲CPU和內存使用情況並將這些數據（配置文件）與源代碼相關聯的想法，以使工程師能夠定位慢速或錯誤的代碼行，以及使用其連續的配置文件進行大規模了解其連續的配置系統。本文影響了社區，在接下來的幾年中，我們目睹了上述空間中的許多商業產品。 總而言之，使用原始素可以進行分析，但是在該級別上創建和結合概況是複雜的，因此，它並沒有成為主流。

連續分析所需的權衡是在頻率相對較低的情況下進行剖面並隨著時間的推移而收集統計意義。這與傳統的分析相反，我們在很高的頻率上進行了介紹，因為我們希望在短時間內達到統計意義，例如10秒。現在，在我們進行連續分析工具之前，讓我們首先退後一步，討論在該空間中廣泛使用的一些常見技術和技術。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

9.1.2 Common technologies

在許多（如果不是全部）連續的工具中，您會發現三種技術。在開始進行工具本身之前，讓我們快速討論這些常見的技術。

pprof

我們已經在本節的引入中看到了PPROF，但讓我們放大一點。 PPROF（https://github.com/google/pprof）是一個可視化和分析工具，用於分析數據，由Google開發。 它已經很好地整合到了GO中，但對於我們的討論而言，它更重要，它定義了代表配置文件的數據格式。 This pprof for- mat is based on a protocol buffers (protobuf) schema (https://protobuf.dev/) defined in profile.proto (see the following site for the fully documented protobuf file and the following text for a description of the most important parts: https:// github.com/google/pprof/blob/main/proto/profile.proto):

```
message Profile {
  repeated ValueType sample_type = 1;
  repeated Sample sample = 2;
  repeated Mapping mapping = 3;
  repeated Location location = 4;
  repeated Function function = 5;
  repeated string string_table = 6;
  int64 time_nanos = 9;
  int64 duration_nanos = 10;
  repeated int64 comment = 13;
}
```

- Describes the meaning and measurement units of a value
- The set of samples recorded in the profile
- Mapping from address ranges to the binary/library mapped into the address range
- Program location reference
- Functions referenced by a location
- Common table for strings referenced by messages
- Time of collection (UTC), in nanoseconds, past the UNIX epoch
- Free-form text associated with the profile (description)
- Duration of the profile

配置文件消息使用其他幾條消息（再次編輯為最重要的部分；有關詳細信息，請參見Profile.proto），包括以下內容：

```
message ValueType {
}

message Sample {
}

message Location {
}

message Line {
}

message Function {
}
```

- A description of the samples associated with each Sample.value
- Each Sample records values encountered in a program context.
- Describes function and line table debug information
- The line number in the source code
- Captures the name of the source file as well as function and the start line number

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

儘管尚無關於如何表示電線或靜止狀態的輪廓的行業標準（例如，作為文件系統中的文件），但PPROF格式是我們最接近這樣的標準的格式，並且許多可用的工具可以讀取或生成它。

PPROF作為一種格式非常有效，因為它基於協議緩衝區，但這也意味著與您從HTTP API獲得JSON相比，它可能很難使用（並且可以使用Curl和JQ來過濾以獲取任何值）。有一些工具可用於處理Protobuf，例如

Protoc (https://grpc.io/docs/protoc-installation/)—Short for *protocol buffer compiler*, which you can use to compile .proto files but also to decode or deserialize them (useful for debugging) *Buf (https://github.com/bufbuild/buf)*—A powerful CLI tool for working with protobuf *Protoman (https://github.com/spluxx/Protoman)*—A Postman-like API client for protobuf-based messages
Postman (https://github.com/postmanlabs/postman-app-support/issues/2801#issuecomment-1002247567)—A popular tool that has added (indirect)通過新的GRPC API類型支持它

調試Protobuf文件是什麼樣的？

讓我們為此使用Protoc。首先，我們需要獲取PPROF Protobuf格式化文件。的確，我們已經在上一個示例中生產了幾個，我們使用了Go Tool Pprof進行分析。請注意，您的設置可能具有不同的路徑配置；查看GO Tool Pprof命令的輸出，其中說在…中保存的配置文件（我的~/pprof）。在CH09/ CP/ Directory中執行以下操作：

美元

現在，我們擁有Profof Protobuf文件，我們可以使用ProtoC進行挑選：

This is the pprof protobuf-encoded file we created in the previous step and that protoc is supposed to use as input.

We tell protoc to use pprof.proto as the schema and ask it to decode a specific message: perftools.profiles.Function.

```
$ protoc --decode=perftools.profiles.Function pprof.proto \
      < pprof | \
      grep --color=auto -e ".*Task"

6: "main.slowTask"
```

Once protoc is done, we search for all function symbols that contain "Task".

As expected, we found at least one function mentioned in the pprof file, and that happens to be slowTask() in my case.

這樣，我們現在就與PPROF一起完成了，但是我們肯定會在本章後面的行動中再次看到它。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

火焰和冰柱圖

Netflix名望的表演大師Brendan Gregg發明了火焰圖（https：//www.brendangregg.com/flamegraphs.html），當他在研究mysql perfor-mance問題時，他需要了解CPU使用。 布倫丹（Brendan）於2011年底發布了它們，這種可視化的核心思想（圖9.2）如下：

> 堆棧配置文件位於X軸上。請注意，x軸*does not*代表時間，但堆棧跟踪的集合。每個矩形代表一個堆棧跟踪（比較編程語言中的呼叫堆棧），或者更簡單地將呼叫對函數或操作系統syscall進行。 矩形的寬度代表了該功能的次數或運行時間的時間。例如，我們在圖9.2中看到的是，llowtask（）在CPU上花費的時間比QuickTask（）多。
>
> y軸顯示堆棧深度，從零計數（函數調用輸入點）。 頂部的矩形表示在堆棧跟踪集合點上的cpu（即實際運行）的函數調用。在我們的示例中，我們可以看到從main（）調用了slowtask（）。
>
> 最初，布倫丹使用隨機顏色來幫助區分堆棧框架。但是，它們可以用來傳達其他信息。換句話說，顏色本身沒有語義，而只是為了幫助您瀏覽火焰圖。

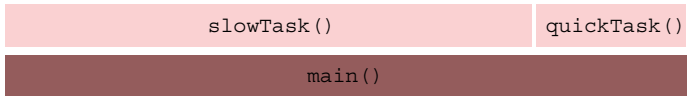


圖9.2使用我們的pprof go示例的火焰圖

儘管用於資源的火焰圖最初集中於CPU使用情況，但它們已經成長為涵蓋更廣泛的資源，包括主內存（RAM）或NET-WORK-WORK-WORK-WORK-WORK-WORK-WORK-WORK-WORM-WORK-WORK-WORM-WORK-WORM-WORM-WORK-WORM-WORK-WORM-WORK-WORM-WORM-WORM-WORM-WORK-WORKENTION和操作系統級別（例如，Java Virtual Machine或GO中的垃圾收集）。

話雖如此，不要將火焰圖視為僅限於系統級任務。實際上，能夠看到您的操作系統（或JVM）的作用，以及您的應用程序在執行時間或CPU和內存使用方面的表現是真正的最佳位置。您也可能會遇到*icicle charts*術語，它只是指倒置的火焰圖（因為冰柱將從頂部到底部生長）。如果您想深入研究主題，我建議閱讀Polar Signal的文章“ Ice and Fire：如何閱讀冰柱和火焰圖”（http：// mng.bz/2dp0）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

EBPF
許多（開源）連續分析工具使用的另一種技術是Linux內核功能EBPF。如今，它本身被用作術語，但最初，該術語源自*Berkeley Packet Filter*（bpf）。

Linux內核v3.18（2014年12月7日發布）引入了對EBPF的支持。這個內核版本的要求以及您需要對調試符號進行良好支持的事實，才能有意義地使用它，這意味著EBPF至少在2023年初寫作時仍然僅限於某些環境（包括Dev/Test）（包括您對使用Linux版本的Linux版本，相反，與生產環境相比，使用調試符號可以使用調試符號降低執行情況）。

請注意，您可能會想知道為什麼我會說EBPF在查看引入的內核版本時僅限於某些環境。在撰寫本文時，已經快十年了。但是，由於發行或其他方式，雲提供商和大型企業需要更長的時間將其默認Linux版本移至“較新”內核。詢問您的平台團隊是否可以在產品中使用EBPF，但如果答案是“我們正在努力”，請不要感到驚訝。

現在，我們為什麼需要EBPF，它如何工作？它使您可以通過添加合適的新功能來安全有效地擴展Linux內核。 它通過BPF（2）SYSCALL暴露於Linux用戶空間（另請參見MAN頁面[https：//man7.org/linux/man-pages/man-pages/man2/bpf.2.html]，有關更多詳細信息），並在內核中作為虛擬機（VM）實現，以定義定義自定義的64-BISC RISC RISC指令集。圖9.3顯示了Brendan Gregg（2019年，Addison-Wesley，http：//mng.bz/1q1j）的書籍*BPF Per- formance Tools*的高級概述。如果您想了解有關EBPF的更多信息，我強烈建議您閱讀Liz Rice（2022，O’Reilly，http://mng.bz/pzln）。

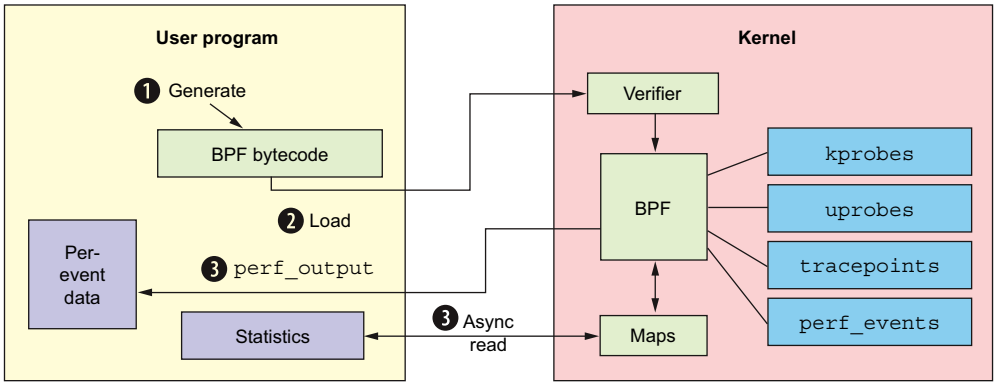


圖9.3概念級別的Linux內核中的EBPF呼叫流（來源：Brendan Gregg。根據CC的許可由4.0）

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

9.1.3 *Open source CP tooling*

讓我們首先查看連續分析空間中的開源工具。這是一個大約2020年左右爆炸的領域，我們看到對它的興趣越來越多，以至於Opentelemetry於2022年9月開始了一項活動（另請參見OTEP 139，http://mng.bz/jgjv），以探索如何將配置文件作為信號類型。按字母順序，我們現在將查看流行的OSS工具進行連續分析。

公園

PARCA（https://www.parca.dev/）是Prometheus-啟發的開源連續分析工具，專門針對使用EBPF進行零啟示，因此您無需更改有關應用程序或將任何庫將任何庫放入代碼中即可使其起作用。 它使用Prometheus Service Discovery查找個人資料來源。 PARCA支持一長串編程語言（https://www.parca.dev/docs/parca-agent-language-support），包括C，C ++，Rust，Haskell，Java，Java，Python，Python，.net，.net和eblesembly。

您可以安裝它並嘗試一下，或在demo.parca.dev（https://demo.parca.dev/）上使用零安裝的在線環境來嘗試一下。圖9.4顯示了PARCA中Kubernetes應用程序的CPU配置文件，在本節的末尾，我們將在端到端的步行中使用PARCA。

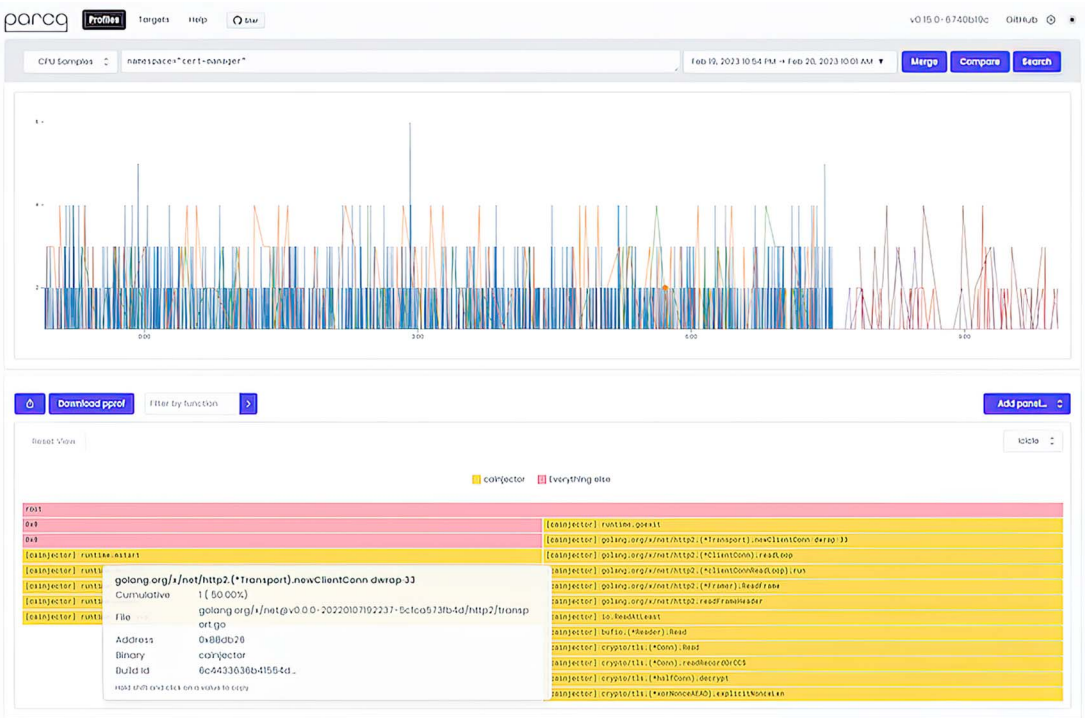


圖9.4 PARCA演示環境的屏幕截圖

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

小精靈

Pixie（https://px.dev/）是一個關注Kubernetes群集的CNCF項目，我們在第6章中介紹了該群集。這不僅僅是持續分析； Pixie還提供了完整的主體請求（例如，HTTP API的JSON BLOB返回值）以及資源和網絡指標以及其他有用的項目。

由於Pixie收集了很多豐富的數據，因此Pixie中的遙測數據保存在記憶中，庫伯尼特集群本地保存。 簡而言之，您可以將Pixie視為“我的Kubernetes群集正在發生什麼？”工具。在資源使用方面，Pixie聲稱最多使用少於5%的群集CPU。 Pixie可以幫助您在特定的POD中看到您的應用程序配置文件，跨節點等等，而無需您進行任何代碼更改或重新部署應用程序。在撰寫本文時，Pixie支持諸如Go，Rust和Java之類的編譯語言的連續分析。

載焦鏡

Pyrospope（https://pyroscope.io/）是一種開源連續分析工具，具有豐富的客戶端收集機制，用於多種編程語言，包括GO，Python，Python，Java，Ruby，Ruby，Ruby，Rust，Php，.net和Node.js，以及Node.js，以及EBPF支持。 2023年6月，Grafana Labs（http：// mng .bz/wvnp）收購了載速，這是在Grafana Cloud中連續分析的基礎。

Tho The Maker of Pyrospope建議採用混合方法集合收集方法：也就是說，使用EBPF來介紹該系統（例如，VM或Kubernetes Node）與每項服務具有特定語言儀器的組合。此外，Pyrospope sports sport flameql是一種查詢語言，使您能夠探索分析數據。您可以在https://demo.pyroscope.io/上使用零安裝的在線環境來嘗試pyro- scope或在本地（如果需要）本地安裝。

圖9.5顯示了Pyroscope UI的概述（以及帶有縮放的覆蓋層）。在圖的上部，顯示基於時間的配置文件的可視化，當您單擊任何配置文件時，您可以生成並放大特定的配置文件（在這種情況下為內存使用情況）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

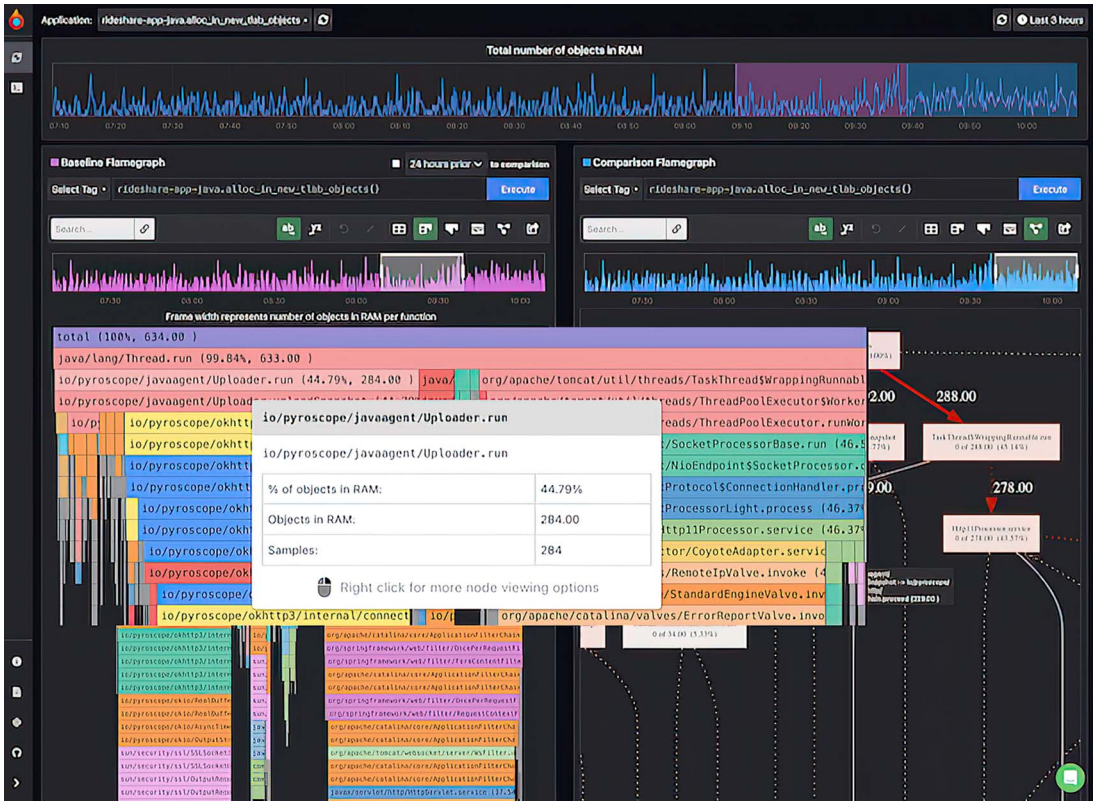


圖9.5載速度演示環境的屏幕截圖

9.1.4 *Commercial continuous profiling offerings*

從雲提供商開始，連續分析空間中有很多商業產品：

> AWS提供了Amazon Codeguru Profiler（http://mng.bz/qrkr），已優化以微調您的應用程序性能。它提供機器學習算法，以找到最昂貴的代碼線，超越該算法，提供了提高效率的方法。 Azure提供應用程序Insights Profiler（http://mng.bz/7dov），它使您可以查看應用程序的性能軌跡。它捕獲了每個請求的中值，最快和最慢的響應時間，並幫助您確定特定請求的熱代碼路徑支出，支持.NET和JAVA。 Google使用其云剖面（http://mng.bz/mvya）開創了該空間。它是一個低開銷的抽樣剖面，因此適合生產環境。 雲profiler支持通用語言和環境

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

(例如，GO，Java，Node.js和Python)以及一系列配置文件類型，包括CPU時間，堆和線程。

然後，有幾個ISV捆綁分析或具有獨立產品。有些以源可用的方式發布，包括但不限於以下內容：

Datadog Profiler（https://docs.datadoghq.com/profiler/），全部產品的一部分，使您可以找到CPU，內存和I/O瓶頸；該文件通過函數，類名和代碼中的行號分解。 Dynatrace的產品（http://mng.bz/5w5o）是其整體套件的一部分，可讓您進行CPU分析，包括鑽探到功能級別，內存轉儲分析和過程崩潰。 G：Profiler（https：//granute.io/continouul-profiling/）是支持AWS，Azure和Google Cloud Envimonments的SaaS產品（但也可以作為開源）。它支持一系列語言（Clojure，Scala，C/C ++，Ruby，Java，Python，Python，JavaScript，PHP和GO），使用EBPF，並索賠比1%的utization罰款。 新遺物具有Java（http://mng.bz/op6n）的“實時分析”功能，該功能支持火焰圖以及線程分析器工具https：// docs。 newrelic.com/docs/apm/apm-ui-pages/events/thread-profiler-tool/可以用於Java，.net，Python和Ruby。 流行器（https://prodfiler.com/）將自己吹捧為分佈式重量級連續的全系統剖面介質，使您能夠深入了解CPU使用情況，支持火焰，需要Linux內核，以4.15版或更高版本，並在諸如Kubernetes和Aws Ecs等平台中使用。 Splunk提供了EwswardOn（http://mng.bz/5w5o）。

馬克·漢森（Mark Hansen）的Pro-fileerpedia（https://profilerpedia.markhansen.co.nz/）是一個很棒的概述和連續分析空間的查找，這是工具和探測器數據格式的目錄。如果您想深入研究主題，請考慮從Datadog中查看*The Busy Developer's Guide to Go Profiling, Tracing and Observability*（https://github.com/ datadag/go-profiler-notes/tree/main/guide）。

9.1.5 Using continuous profiling to assess continuous profiling

在本小節中，我們將使用連續分析來介紹Opentelemetry收集器的資源使用情況。 我在此處使用OpenTelemetry收集器作為示例CP源，因為它很容易揭示Pprof-Formatted配置文件，但是您可以真正插入任何其他HTTP API提供配置文件（例如，從本章開始時我們的Go示例）。

為了生成配置文件，我們將設置一個信號源（像往常一樣，我們將使用帶負載生成器的HO11Y）將跟踪發送到OpentElemetry Collector，然後我們將使用PARCA從收集器接收和查詢配置文件。 所有來源都在

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

CH09/PARCA，因此更改為該目錄。首先，要查看來自Opentelem-etry Collector的配置文件，您需要在收集器中使用PPROF擴展名（http://mng.bz/6d6e）config- uneed（有關詳細信息，請參見Otel-config.yaml文件）。因此，讓我們以通常的形式提出Docker組成的堆棧：

```
docker-compose -f docker-compose.yaml up
```

以下列表顯示了與我們的對話有關的撰寫文件的片段。

清單9.1 Docker的相關片段構成PARCA設置的配置

```
otel-collector:
    image: otel/opentelemetry-collector-contrib:0.68.0
    command: [
      "--config=/conf/otel-collector-config.yaml"
    ]
    volumes:
    - ./otel-config.yaml:/conf/otel-collector-config.yaml
    ports:
    - "4317:4317"
    - "1777:1777"

  parca:
    image: ghcr.io/parca-dev/parca:v0.15.0
    command: /parca --config-path=/opt/parca.yaml
    ports:
    - "7070:7070"
    volumes:
    - "./parca.yaml:/opt/parca.yaml"
```

We expose the pprof extension on this port in the OpenTelemetry collector.

Parca, as the server, providing service discovery and UI.

The Parca configuration file

OTELEMETRY集合器在Otel-config.yaml中所示的OpenTelemetry Collector配置配置了一個簡單的跟踪管道，通過日誌記錄出口商將跟踪發送到收集器的Stdout。

列表9.2用於生成配置文件的OpenTelemetry收集器配置

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317

exporters:
  logging:
    loglevel: debug

extensions:
  pprof:
    endpoint: "otel-collector:1777"
```

The pprof extension for collector self-telemetry profile generation

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
service:
  pipelines:
    traces:
      receivers: [ otlp ]
      exporters: [ logging ]
```

A simple traces pipeline, to have some load

Effectively means log to stdout and no ingestion into a backend

請注意，列表9.2中管道的詳細信息對我們來說並不重要，因為我們專注於PPROF擴展程序，而總體收集器而不是特定的管道。

接下來，我們需要確保Parca知道從哪裡獲得資料。類似於Prometheus，我們定義了刮擦配置（在parca.yaml中），如以下列表所示。

清單9.3 PARCA配置，用於從OpenTelemetry收集器收集配置文件

```
object_storage:
  bucket:
    type: "FILESYSTEM"
    config:
      directory: "./tmp"

scrape_configs:
  - job_name: "default"
    scrape_interval: "2s"
    static_configs:
      - targets: ["127.0.0.1:7070"]
```

Tells Parca that we are only using local storage

The source of our profiles (the collector's pprof extension)

現在，您應該能夠在瀏覽器中打開Localhost：7070，並且經過幾分鐘後（給Parca足夠的時間刮擦一些配置文件），請參閱圖9.6。 確保您選擇一個配置文件（我選擇了內存中的字節），然後按極右側的搜索按鈕。要查看PARCA調用Icicle圖的火焰圖，請單擊“線”圖。

好的，這要消化很多，但是我希望您不僅看到了CP的力量，以及如何利用它來解決具體問題，在我們的情況下進行故障排除。 因此，在完成此次演出後，讓我們繼續談論開發人員生產力的更廣泛的話題，在該主題中，您可以將CP視為特殊情況，甚至有時甚至是功能。而現在，我們將左派轉移到開發人員的生產力和行動中的觀察力。

提示您也可以在生產環境中使用CP；但是，對資源的影響和工作負載的潛在破壞的理解尚不清楚。另請參見我的信息Q談話，“個人資料，缺失的支柱：practice中的連續分析”（http://mng.bz/o1zz），有關更多詳細信息。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

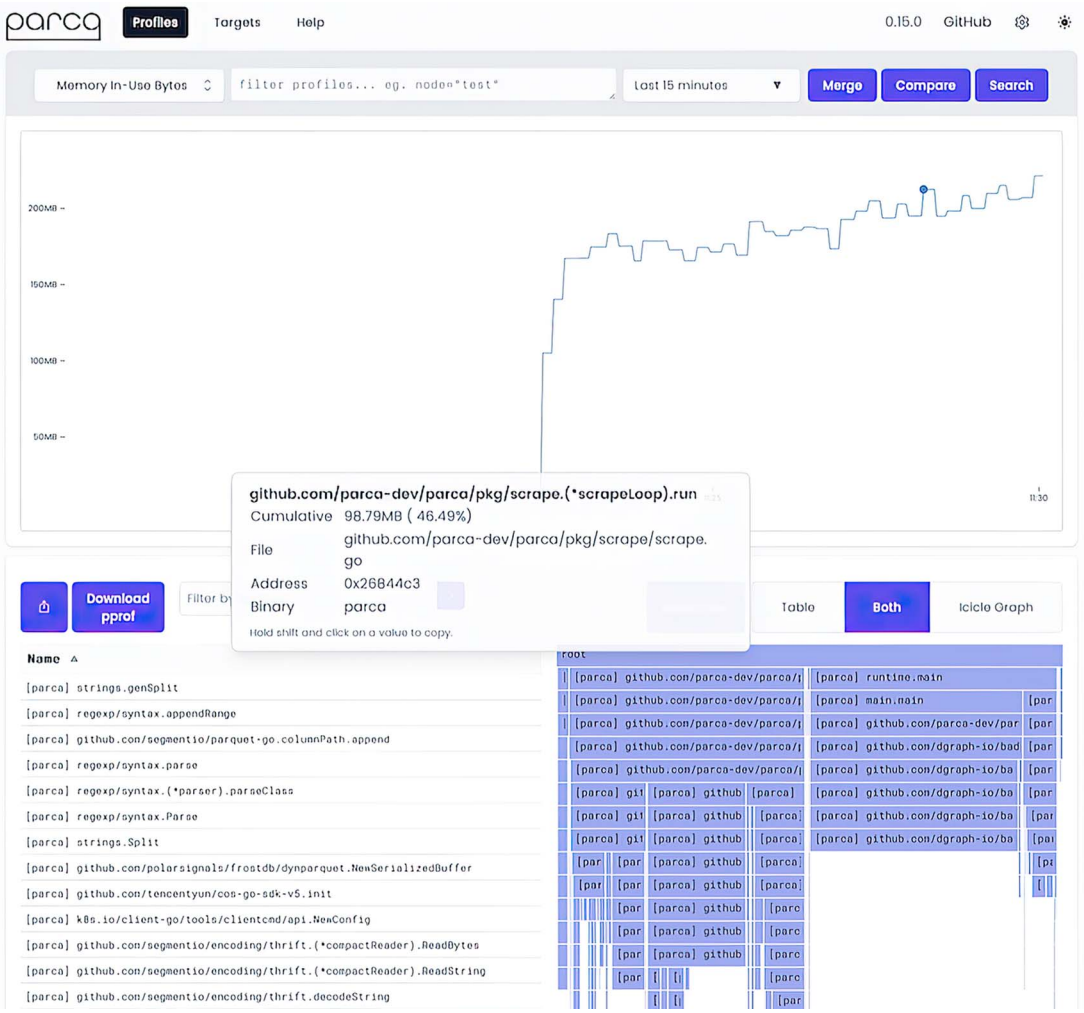


圖9.6 PARCA分析的屏幕截圖opentelemetry collector

9.2 Developer productivity

我們目睹了一組工具的興起，旨在通過向左翼趨勢為開發人員提供見解來提高開發人員的生產力。在本節中，我們將查看該領域的工具，並描述它們與連續分析的關係。

9.2.1 Challenges

開發人員正在尋求提高生產率的可行見解？也就是說，我們如何在安全的管理中更快地添加新功能或更快地修復錯誤？挑戰大致分為以下類別：

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Volume of data - 您寫為開發人員的源代碼以及所有庫中的庫，軟件包和模塊，在化妝中構建服務的很大一部分。 然後，有上下文數據集，包括問題跟踪器（GitHub，GitLab等）以及路線圖，用戶輸入和反饋，例如訪談或記錄。然後，有與源代碼和獨立項目或產品文檔以及API有關的文檔，以及基於模式的動作和數據類型。所有這些數據都是輸入，作為開發人員，我們傾向於嘗試將所有這些連接保持在我們腦海中的所有這些連接。這越來越具有挑戰性，我們使用的抽象層越多。

Testing - 隨著雲產品的引入，許多事情發生了變化，我不一定意味著基於雲的IDE；相反，我的意思是在哪里以及如何測試軟件。現在，我們將繼續進行單元測試（根據程序媒體語言，這是可靠和可維護的代碼的一個或多或少重要的方面），但是我們應該在哪裡進行集成和端到端測試？一個假設是通過金融店，A/B測試和功能標誌來測試生產並減輕影響（如aunchDarkly [https：// lunchdarkly .com/]）或Amazon CloudWatch（顯然是Amazon CloudWatch（https://aws.amazon.com/blogs.com/blogs/blogs/ways/abblogs/ways/abblogs/ways/abblogs/aws/aws/aws/clindwatch-vatch-vatch-evidly-evidelly/）。

Distributed systems - Microservices是分佈式系統，因此依賴網絡通信。 無論您是使用HTTP，GRPC還是Websockets，都需要了解通信行為，有效載荷（編碼，格式等），以及如何探索遠程API在體驗加載時所做的工作。 這意味著一方面被迫處理洩漏的抽象（通過抽象層閃耀的實現細節），另一方面，檢測和處理超時，回程和刺激。請記住，您的網絡可能不是可靠或安全的（https：// architecturenotes.co/fallacies-of-distributed-systems/）。

9.2.2 Tooling

開發人員生產力工具我們將在以下子規定中仔細研究通常基於或使用OpentElemetry，並提供深度IDE集成，通常將其用於VS代碼。公司越來越多地開始為包括eBay提供開源（通常基於OpentElemetry的）開發人員生產率的工具，該工具包括一個有趣的框架，稱為*flow telemetry*（https://github.com/ ebay/ebay/flow-telemetry）。請注意，本節中討論的工具正在迅速發展，您可以期望功能或與其他工具或平台的合併發生變化。

戰爭

Digma（https://digma.ai/）定位為*continuous feedback pipeline*（這似乎是一個營銷術語，因為我還沒有找到正式的定義），並帶有分析後端和IDE插件。它的目標是不斷分析可觀察性來源和

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

為您作為開發人員提供反饋。該插件在您編輯代碼時生成了與性能，錯誤和用法相關的代碼級見解。 Digma從您的OpenTelemetry Trace和Metrics中獲得了這些見解，這些見解是由Digma後端收集和分析的。 圖9.7（左側）顯示了VS代碼中的插件。

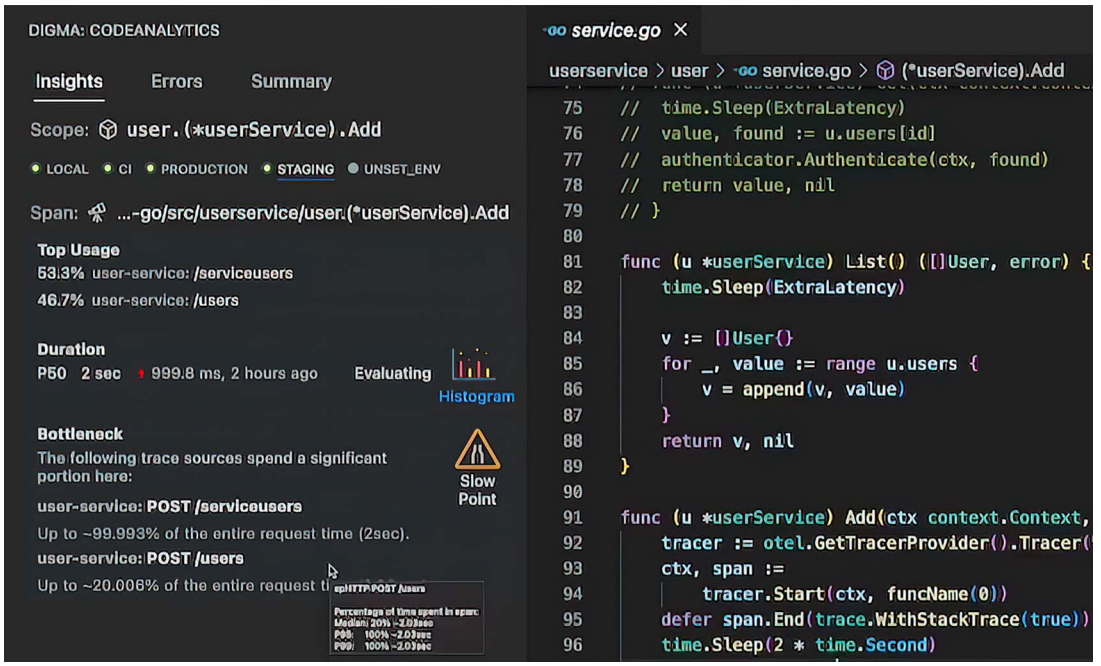


圖9.7 Digma的屏幕截圖

噴

sprkl（https://sprkl.dev/）將自己吹捧為“個人可觀察性平台”，在IDE中進行編碼時，對您的代碼更改進行了個性化反饋。 它使用opentelemetry來儀器代碼，並允許您在進行時探索代碼更改。您可以獲得代碼級的跟踪，了解您的代碼更改如何與系統中的其他實體進行播放以及代碼性能報告。

當前，SPRKL專注於Node.js代碼，並通過Kubernetes對Java，C＃和Python進行了部分支持。 SPRKL需要本地Docker，並專注於VS代碼（作為擴展名）作為IDE。在VS代碼中設置SPRKL並運行Docker後，探索其功能的一個很好的起點是Spample App Sprkl- dev/use-sprkl（https://github.com/sprkl-dev/sprkl-dev/use-sprkl）。

traceTest

Tracetest（https://tracetest.io/）與前兩個工具有點不同。它重點是使您能夠建立由Open Telemetry驅動的集成和端到端測試

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

跡線。這意味著您需要一個分佈式的跟踪後端，並將TraceTest指向它，或者，例如，使用OpentElemetry Collector直接將跨度發送到Tracetest。 您可以使用它來定義對沿請求路徑服務的測試和主張，並針對響應數據和跟踪數據定義斷言。此外，您可以手動或通過CI構建作業進行測試。圖9.8顯示了traceTest測試結果，以瀑布方式呈現請求。

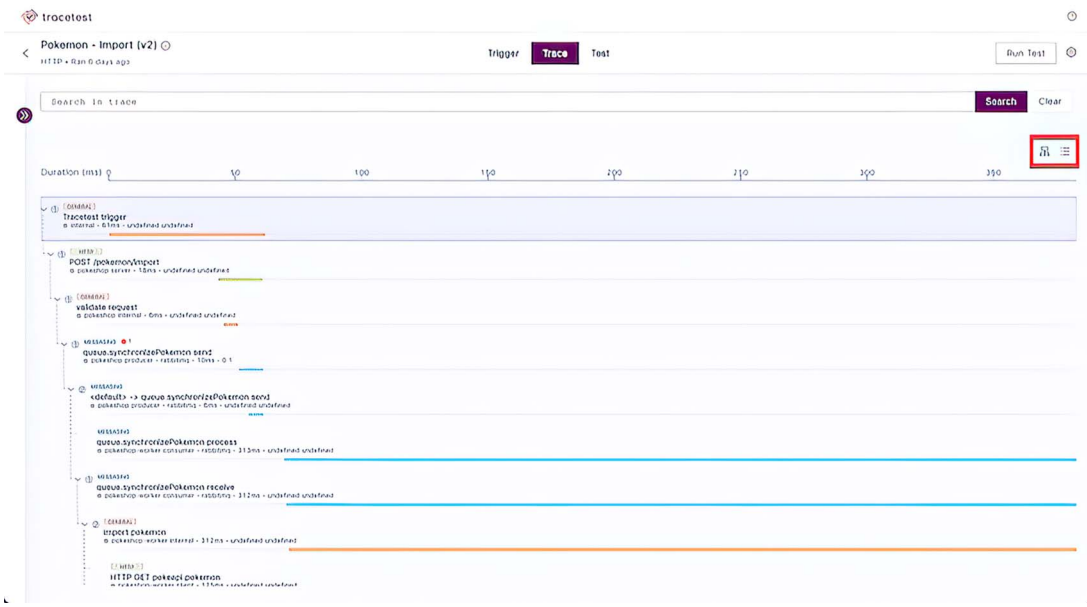


圖9.8 Tracetest UI的屏幕截圖（結果視圖）

您可以通過docker compose或helm圖表安裝tracetest（http://mng.bz/nwg5），以及許多示例集成，例如與AWS X-ray（http：// mng .bz/vnep）或grafana tempo（http：// mng . bz/vnep）或grafana tempo（http://mng.bz/mng.bz/4ddwr）。

Rookout

Rookout（https://www.rookout.com/）稱其為“開發人員 - first可觀察性工具”。 它捕獲了堆棧上的實時調試數據，支持Java，.net，Node.js，Python，Ruby和Go。 Rookout允許您在各種環境中進行調試以及可用斷點的視圖。

自動化

Autometrics（https://autometrics.dev/）建立在Prometheus和Open-Telemetry Client庫的頂部。它可以輕鬆地在代碼中儀器功能（通常用於Rust，Go，Python，TypeScript和C＃），其中包括通用指標，包括請求率，錯誤率和延遲，並為您生成ProMQL查詢。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

這樣，我們已經完成了工具評論。 現在，讓我們討論在空間中採用工具時該考慮的事情。

9.3 Tooling considerations

我們已經檢查了持續的分析，更普遍地是在本章中進行開發人員的生產力工具，現在您可能會很興奮地跳入深處。在您走得太遠之前，我想分享一些小節中的一些想法和謹慎的想法。雖然我個人超級看好和支持，但我們必須對尖銳的邊緣進行重新認可並正確設定期望。

開發人員的可觀察性是一個相對年輕的領域，許多競爭者在市場上已經不到一年了。以下是CP工具專門面臨的挑戰的討論。

9.3.1 Symbolization

由*symbolization*，我們的意思是將機器地址轉換為人讀取（符號）名稱的過程。 例如，如果您查看一個配置文件並且包含堆棧幀標籤，例如0x8C6FFF910，則您真的不知道該內存地址表示什麼。另一方面，如果您要看到/some/path/pare/file.go：42或calculatesummary（），那將是有意義的。

現在的挑戰是確定符號信息的位置。對於諸如c/c ++或go之類的編寫語言，符號表（包含符號）通常是二進制的一部分，對於解釋的語言，包括python，ruby或javaScript，符號通常是程序的一部分。

有關此主題的進一步閱讀，另請參見以下博客文章：

Kemal Akkoyun（http：//mng .bz/qpb6）撰寫的“奇妙符號和在哪裡找到它們” “BPF二進製文件：BTF，Co-Re和BPF Perf工具的未來和Brendan Gregg的未來”，Brendan Gregg（http://mng.bz/xnr9） Andrii Nakryiko（https：// nakryiko.com/posts/bpf-portability-and-co-re/）

一旦有了符號，我們還沒有完成。我們仍然需要存儲配置文件。但是，我們如何以有效且預算友好的方式做到這一點？

9.3.2 Storing profiles

當您檢查目前正在發生的事情時，在某些用例中，配置文件的內存存儲就足夠了。但是，如果您想比較更長的時間範圍或服務的不同版本，則需要能夠處理配置文件的量並以持久的方式存儲在某個地方。

TWo連續分析工具中發現的常見技術 S是

To efficiently store and query profiles, columnar storage is used (see also chapter 5, where we discussed the design and properties of columnar data stores).，例如在

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

PARCA的上下文，團隊決定開發FrostDB（https://github.com/ polarsignals/frostdb），這是一個專用的柱狀數據庫，用於Pogiles。 您可以嵌入frostDB，並且使用Apache Parquet進行存儲和Apache Arrow以查詢，它具有半結構化模式（其中某些或所有模式字段或類型是未知或未定義的）。

*To minimize the storage footprint for the timestamps, continuous profiling backends typically use XOR compression.*將其視為時間戳的Delta-of-Delta編碼。該技術首先是由Facebook工程師團隊在“大猩猩：快速，可擴展，內存時間序列數據庫”中的Tuomas Pelkonen等人描述的。（2015年，VLDB Endowment的論文集； http://www.vldb.org/pvldb/ vol8/p1816-teller.pdf）。 如果您想了解有關此域中壓縮的更多信息，請查看Joshua Lockerman和Ajay Kulkarni（https://wwwww.timescale.com/blog/blog/time-time-time-compers-compression-compression-algorith--algorithms-acleplation一下時間序列壓縮算法，解釋的時間序列壓縮算法”。

有效存儲後，您將需要查詢配置文件。那麼這裡的挑戰是什麼？

9.3.3 Querying profiles

理想情況下，我們將能夠支持有關概況的表達性查詢。當前，大多數簡單的基於標籤的查詢語言，例如在Parca（https://www.parca.dev/docs/ querying-parca）和Pyroscope的FlameQl（https://pyroscope.io/docs/ddocs/flameql/）中。本質上，這並不是一件壞事，因為Prometheus Propql的廣泛使用和流行的查詢語言也是一種基於標籤的查詢語言。但是，這裡的問題是（開放）標準是否以及何時出現，以及它是否將是與SQL相關的或與DSL相關的格式。

9.3.4 Correlation

在開發人員可觀察性空間中，關鍵挑戰之一以及承諾是相關性。自然地，人們希望從其他信號類型（例如分離的軌跡或度量標準）跳至概況（與Grafana插件一樣，是基於標籤的或通過前端支持）。這是一個積極研究的領域，我們可以期望在這裡發生更多的事情。我們將在第11章中更詳細地介紹該主題。

9.3.5 Standards

我們已經討論了幾次標準。 要點是，您要確保您使用的工具以實現開發人員的可觀察性是在頂部建立的，並使用包括OpentElemetry在內的開放標準，以獲取電線上的信號，以及在PPROF（例如PPROF）上的啟動標準，以代表電線上的配置文件。

9.3.6 Using tooling in production

您可以在開發環境以及生產環境中使用連續分析。在那裡，您將需要提出有關

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

對實時流量與有用性的影響，以了解正在發生的事情。這裡的挑戰是雙重的：

> 要獲得有意義的見解，您需要加載調試符號，該符號可以並且將更改執行時間並引起更高的資源使用情況。 然後，有跟踪，攝入和存儲配置文件的開銷（例如，以PPROF格式）。為此，您將需要另一個能夠有效存儲配置文件的後端，並允許您查詢它們或現有的後端之一，以引入個人資料處理的支持。

到目前為止，我們在現實生活中看到的是，CP在PROD中得到了驗證，但是最佳實踐可分享。 因此，您需要逐案決定CP的收益是否超過所需的投資（在存儲，攝入，潛伏期影響等方面）。

Summary

開發人員可觀察性意味著使用遙測信號為開發人員提供可行的見解以加速開發；調試代碼；或在運行時或資源使用方面了解新功能的影響。 您可以使用連續的分析來了解您服務的當前性能和資源使用情況，並將資源使用情況與以前的服務版本進行比較。 protobuf編碼的PPROF格式是我們最接近的標準，用於在電線上或靜止。 火焰圖以視覺方式呈現呼叫堆棧，EBPF是一種Linux內核擴展機制，用於在許多連續塗抹工具中進行配置集合。 要自動從應用程序代碼或操作系統中的配置文件集合，您可以使用EBPF，越來越多地獲得吸引力。 開源連續分析工具是相對較新的開發，包括Parca，Pixie和Pyroscope。 雲提供商和可觀察性供應商都在連續分析空間中提供商業產品。 開發人員生產力工具通常基於OpenTelemetry，為開發者提供了有關性能和資源使用視圖的代碼更改的見解。 有一些挑戰，包括缺乏互操作性以及被收購或停止報價的小玩家的風險； 但是，開發人員生產力工具的總體好處是明確的，並且可能成為主流。 如果您知道對系統的影響，則不僅可以在開發環境中而且在產品中的連續分析中受益。 對開放標準的支持至關重要。您將必須進行盡職調查，以確保所選的供應商（或開源項目）完全致力於支持他們的產品。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

10 Service level objectives

This chapter covers

我們按服務級別目標的含義如何實現服務水平目標SLO空間中可用的工具實施SLOS的考慮因素

在書中的這一點上，您應該對*ROI-driven observ- ability*的含義有一個很好的了解。但是，有一個我們尚未討論的操作主題，這是完成圖片所需的：服務的消費者如何滿足，我們如何根據數據知道消費者是否滿意？ Consumer不必是外部客戶，尤其是在大型組織中，消費者可能是不同的業務部門。 現在，不要誤會我的意思 - 沒有什麼比在Orange網站上發表刻薄的推文或周到的評論（又名Hacker News）更具動力的了。但是，如果我們可以自動確定整個過程，那不是很好嗎？

如果您退後一步，您會發現DevOps和網站可靠性工程（SRE）在過去十年中起飛，前者的自下而上，後者顯然是由Google驅動的。 本章中的核心概念和想法確實是Google發明，如果您想研究所有最後一個細節，包括最佳

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

練習，我鼓勵您前往Google SRE書籍網站（https：// sre .google/books/）並閱讀所有內容。 在本章中，我們將採用更實用的方法，迅速涵蓋基本面，然後展示如何使用它們。

主要目的是使可衡量的服務的可用性並比較從服務中得出的指標與目標。 一旦您達成了目標，並且可以自動衡量它們，那麼您就可以做更多的事情：

> 您可以繪製漂亮的圖表並查看它們，這非常適合在前端使用。 您可以提醒您是否有違反目標的風險（另請參閱第7章中有關雲操作的對話）。 您可以將這些指標用作與您的客戶合同的投入，以提交某些內容，如果您無法交付您的承諾，則可以承諾進行財務補償（可能是信用額度甚至罰款支付）。這意味著您以可量化和系統的方式從客戶那裡獲得信任。 您可以推動合規性（例如，在金融或醫療保健領域）。

> 筆記
> 與我們在現實生活中討論的想法相似：當我們的一個孩子和我同意，當一個乾淨的房間會帶來獎勵時，我們首先需要定義並同意*clean*的含義（這將是合同作品）。例如，我可能會說，一個月的時間，我希望在地板上只找到三件骯髒的衣服（這是孩子需要達到獎勵的目標）。然後，我會隨機檢查房間，並註意地板上發現了多少個髒衣服（測量值），並在本月底與目標進行比較。如果我要在地板上找到三件或更少的衣服，我會成功完成目標，孩子將獲得獎勵。
>
> 在任何時間點，孩子都可以查看我的清單，看看我發現了多少個衣服（在這裡的測量期，一個月）。這樣，他們可以確定他們要違反商定的目標或換句話說，他們剩下多少擺動空間（您可以稱為*budget*}，確實是官方術語：*error budget*）。

在本章中，我們將首先在理論層面上解決服務級別的目標。但是，恐怕我們首先需要解開一些術語。然後，我們將換檔並查看該領域的工具，您將完成實施可用性指標的動手會議，並與定義的目標進行比較。 在最後一節中，我們將討論一些重要的考慮因素並提供進一步的閱讀。

10.1 The fundamentals of SLOs

在本章中，我們區分了三個相似但完全不同的術語：服務水平協議（SLA），服務級別目標（SLO）和服務

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

水平指標（SLIS）。 在第一部分中，我們將深入研究這三個術語相互作用以及（角色扮演角色）如何相關的三個術語。

圖10.1顯示了SLA，SLOS和SLIS如何在高水平上相互作用。我們在這裡有四個角色在兩個水桶中工作：

*service consumer*或*customer*通常是團隊外部的，他們擁有服務（服務邊界線）

service provider，包括 - *sales team*，*sales team*，他們可以在事物的業務方面處於業務方面，解決方案，解決方案，客戶的成功工程師/服務的人，或者是服務的人，或者是該服務的人 - 團隊和工程 - *engineering team*，他從功能開發以及可能的操作角度擁有該服務（儘管組織中可能有專門的角色，例如SRE，例如SRE）

服務客戶在組織內（內部客戶）完全有可能；但是，這並不能改變對它們的期望或承諾，而只會改變金錢的潛在流動。如果服務的成本歸因於內部客戶，我們將其稱為*chargeback*。退款的一種極端形式（沒有實際費用，僅僅是成本和使用意識）是*showback*，其中成本顯示給內部客戶。 在我們的討論中，重要的是服務客戶超出了服務團隊的界限。

好的，但是所有這些真正的含義是什麼？ SLA，SLOS和SLIS的真正是什麼，您應該如何定義和實施它們？讓我們從圖10.1的底部開始，然後努力努力。首先，讓我們定義*service*的含義。

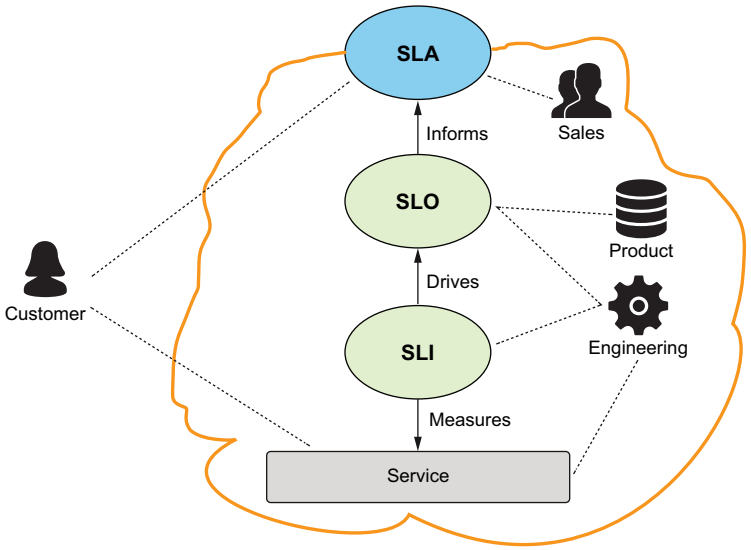


圖10.1 SLA，SLO和SLIS之間的相互作用和依賴關係

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

10.1.1 Types of services

在本章中，我將使用*service*和*API*的術語。無論哪種情況，我的意思是同步服務，異步服務或*sui generis*（unique）服務類型的API。讓我們仔細研究現在的服務類型：

> *Synchronous services*, generally request–response-based systems, such as long- running RESTful web services, WebSocket-based systems, and RPC-based (gRPC or SOAP) systems but also data pipelines (e.g., ETL) *Asynchronous services*, including event buses, queues, and Pub/Sub systems in which there is no obvious causal relation between a publishing action, such as *put event X on a queue with topic Z*, and a consuming action, like *read the most recent event from queue with topic Z Sui generis services*, including – Batch jobs triggered by a certain event, run for a certain time, and expected to terminate – Storage, such as POSIX-compliant local or remote file systems (including ext4 and NFS) or object storage, like Amazon S3 – Databases and datastores, including relational databases, such as Postgre- SQL, and NoSQL datastores, like AWS DynamoDB或MongoDB

根據服務的類型，有不同的方法來衡量服務客戶如何看待服務。輸入SLIS。

10.1.2 Service level indicator

服務水平指標（SLI）是確定服務方式的定量手段。簡而言之，所有SLI都是指標，但並非所有度量標準都是SLIS。 SLI的目的是允許您根據數值來衡量服務客戶如何體驗您的服務。您可以匯總此值，並在其中進行精美的數學（例如百分位數）。但是歸根結底，SLI確實只是服務指標，例如

> 服務服務的正常運行時間同步服務的錯誤率（例如，總請求錯誤）持續時間或延遲，例如同步服務需要多長時間來響應請求成功完成或失敗

如果您對要考慮哪種指標（用於同步服務）的詳細信息感興趣，我建議閱讀Joab Jackson（http://mng.bz/a1nj）的“紅色方法：一種新的監視微服務方法”。現在，您知道了SLI在高水平上是什麼，您作為工程師定義和跟踪的服務指標，讓我們繼續設定目標或SLO。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

10.1.3 Service level objective

SLO是系統 - 外部目標，您可以在告知SLA的背景下使用其中一個或多個。為服務提供多個SLO，您可以將它們組合起來進行全面的通過/失敗測試。與SLA相反，SLO的受眾不是您的服務客戶。 這是您的服務團隊。 作為產品所有者，您將SLO與您的工程師一起與銷售合作，以確定服務用戶可以接受哪些錯誤。例如，您可能同意，對於特定的服務，您希望在一個月的時間內達到90%的可用性。然後，在本月底，您可以將SLIS與SLO進行比較，以確定您是否達到了目標。

設定SLO閾值的平衡行為是在可用性和速度的競爭要求之間提出的。想想據稱可用性為0%的SLO的極端情況。這很容易實現；該服務可以隨時關閉，您可以隨意將新代碼運送到生產中。 從開發人員的角度來看，這很棒，因為開發人員可以隨時推動新功能或錯誤修復。從服務的角度來看，它不太偉大。這意味著他們根本不能依靠服務。頻譜的另一端將是100%的可用性，這意味著，有效地，不允許更改，也沒有錯誤或其他因素降低您的服務。 顯然，極端都不是理想的或現實的。

> 請注意，在某些方面，SLO有點像光速：您想越接近它，您必須投入的能量就越多。從目標的角度來看，將SLO設定為99%和99.9%的SLO可能看起來只有0.9%，但要確定必須實現這一目標的工作。另外，請記住，請求路徑中的弱鏈接可以決定您可以實現的目標。 例如，如果您有三個服務 - a，b和c，則a正在調用b； B正在打電話C；而且，假設A和B的SLO為99.9%，C的SLO為90%，您不能指望總體上承諾99.9%，因為C不會支持這一點。有關更多信息，請參見Steve McGhee的Sloconf 2021 Talk，“ Slo Math”（https://www.youtube .com/watch？v = -lhpdx90ppg）。

還有更多與SLO有關的術語，例如*error budget*，它定義了您在一個時期內留下的SLO的WIGGLE空間，而*burn rate*定義了您使用錯誤預算的速度。儘管術語可能有點密集，但簡而言之，錯誤預算使您可以顯著提高警報準確性，降低嘈雜的警報並改善練習領先的指標。 NOBL9代表告訴我，有一些案例研究顯示，實施SLO後，有90%+減少了誤報警報。

我們不會在這裡詳細介紹。 如果您有興趣了解有關高級術語（在Google以外未廣泛採用的），請查看免費的SRE書籍（https://sre.google/books/）。要了解有關SLO最初源於位置的更多信息，我建議Akshay Chugh和Piyush Verma（http://mng.bz/gbme）推薦“服務級別目標的起源”。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

從實際的角度來看，還有一些工具，例如Nobl9的Sli Analyzer（https://docs.nobl9.com/slocademy/sli_analyzer/），使您能夠基於（歷史）SLIS生成SLOS。現在我們已經知道了什麼是SLO（目標）以及如何衡量我們在多大程度上遇到的SLO（SLIS），讓我們將服務提供商與服務客戶之間的合同上堆疊。

10.1.4 Service level agreement

SLA是您作為服務提供商向您（服務）客戶承諾的，這主要是合同主題。臉頰方式的一種更舌頭說：“ SLO是工程師的寫作，而SLA是用律師寫的”（對Jamie！）。 這主要是在銷售人員和客戶之間，基於該產品和工程所承諾的SLO並告知。

例如，如果產品和工程公司承諾將API提供在本季度的99%，並且在季度後，您只能使API在98%的時間內可用，那麼您就違反了可用性SLA。這可能會導致後果，例如退款甚至支付罰款，具體取決於您經營的服務類型或與客戶的關係類型。

10.2 Implementing SLOs

現在，您了解了術語，讓我們討論如何實施假設的SLA，SLO和SLIS。我們將從銷售，產品和工程之間的示例對話開始，然後查看如何在Prometheus生態系統中實現SLO的具體示例。

10.2.1 High-level example

想像以下內容：您的服務團隊計劃啟動一個新的API，以進行對話聊天機器人，稱為Chat2Much。假設的對話如下所示：

> *Sales → Product: Yo, I hear Chat2Much is soon ready to launch. I'm working on the SLA and wanted to understand what we can promise to customers about an API availability rate over the month as well as the request failure rate.*
>
> *Product → Sales: Sure thing, let me get back to you on this.*
>
> *Product → Engineering: I'd like to establish an SLO that sales can use as a basis for the Chat2Much SLA.*
>
> *Engineering → Product: What SLO did you have in mind for Chat2Much?*
>
> *Product → Engineering: I was thinking of committing to 99% API availability over a period of a month—what do you think? Also, can we do request count and errors?*
>
> *Engineering → Product: That sounds reasonable, I believe we can achieve 99.9% with the current head count and technology stack. We will implement the service metrics* a *,* b *, and* c *, which you can use as SLIs to track the SLO.*

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Product → Engineering: OK, thanks—sounds good. I will work with Sales on the details.

Product → Sales: Alright, we can commit to 99% availability over the time period of a month for the Chat2Much API, and there is some wiggle room on our end already baked in with it. We are using SLIs a *,* b *, and* c *to track this SLO.*

Sales → Product: Perfect, thanks! I will use this and attach the financial implications—that is, the credits that a customer receives in the case we fail to meet the SLO.

提示要達到可用性百分比，您最多可以容忍多少個錯誤，您可以向後工作。假設您已經觀察到，您在負載測試中的服務已在一段時間內收到了1,000,000個請求，並且在同一時期內有6,942個失敗的請求。要實現一些淨空，您可以匯總，因此將其製作7,000。因此，實際上，您可以實現99%的可用性。

繼續示例，Chat2Much API然後進行生產，Engiering正在使用儀表板和警報來跟踪SLI S A，B和C。他們每月報告如何滿足SLO的方式，並通過多少。客戶知道SLA的可用性，並可以看到API對自己的儀表板上達成的目標的同意。每個人都很高興 - 故事結束。

如果您想深入研究此主題，我建議使用以下資源：

Alex Hidalgo（2020年，O’Reilly Addison- Wesley，http://mng.bz/5w5q）by *Implementing Service Level Objectives*“從關鍵用戶到slo/slis” Adam Roberts（http：// / mng：// mng.bz/6d65）的slo slo lifeclecle; .slodlc.com/）“有效的slos and you”（http://mng.bz/o1zj）PrathameshSonpatki和Sau-Rabh Hirani（http://mng.bz/nwg8）的“實施SLO的實用指南”

現在，讓我們轉動齒輪，看看一個混凝土域：普羅米修斯。我們還將研究如何在本生態系統中實現SLO。

10.2.2 Using Prometheus to implement SLOs

在我們對SLA，SLOS和SLIS的理論處理以及如何從概念上促進它們之後，我們現在將仔細研究如何使用Prometheus生態系統中的開源工具專門促進SLOS和SLIS。與本書中的所有其他章節一樣，您的想法是，您可以在不涉及的成本或（許可證）承諾的情況下嘗試和評估事物，並且該步驟應或多或少地應用於該領域的任何提供商。

pyrra

pyrra（https://github.com/pyrra-dev/pyrra）是一種開源工具，旨在使Prometheus易於使用。 Pyrra專注於Prometheus作為

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

事實證明，SLIS是一個很大的優勢，因為Prometheus在雲本地環境中被廣泛採用。從建築的角度來看，它包括下降的組件：

作為操作員，您使用（YAML）文檔定義SLO，並使用SLIS。 用於ServiceLevelObjective資源的後端手錶（來自本地文件系統或Kubernetes API），並根據此輸入創建Prometheus記錄規則。 pyrra api從後端（例如Kubernetes）收集了Slis。 pyrra UI使用pyrra API顯示SLO，SLIS，錯誤預算等。

提示，如果您不想投入在此處設置基於Docker的示例所需的時間，則可以在Demo.pyrra.dev（https://demo.pyrra.dev/）上與在線環境一起玩。

要使用清單10.1中顯示的Docker組合文件站立堆棧，請從CH10/ Pyrra/ Directory中執行以下命令：

```
docker-compose -f docker-compose.yaml up
```

以下列表顯示了我們用於此練習的Docker組成的文件的相關片段，重點是Prometheus和Pyrra的配置。

清單10 .1 Docker的摘要構成PYRRA設置

```
prometheus:
    image: prom/prometheus:v2.40.0
    restart: always
    ports:
      - "9090:9090"
    command:
      - --config.file=/etc/prometheus/prometheus.yml
      - --storage.tsdb.path=/prometheus
      - --storage.tsdb.retention.time=33d
    volumes:
      - ./prometheus/prometheus.yaml:/etc/prometheus/prometheus.yml:ro
      - ./prometheus_pyrra:/etc/prometheus/pyrra

  pyrra-api:
    image: ghcr.io/pyrra-dev/pyrra:v0.5.5
    restart: always
    command:
      - api
      - --prometheus-url=http://prometheus:9090
      - --prometheus-external-url=http://localhost:9090
      - --api-url=http://pyrra-filesystem:9444
    ports:
      - "9099:9099"

  pyrra-filesystem:
    image: ghcr.io/pyrra-dev/pyrra:v0.5.0
```

Tells Prometheus where the rule files are stored locally

Configures Pyrra with the Prometheus info

Sets up Pyrra's local file system provider (to store generated rules)

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
    重新啟動：總是
    命令：
    - 文件系統
    --- prometheus-url = http： / / prometheus：9090
    卷：
    - ./pyrra:/etc/pyrra
    - ./prometheus_pyrra:/etc/prometheus/pyrra
```

以下列表顯示Prometheus config（Prometheus/Prometheus.yaml）。

列表10.2 Prometheus配置

```
global:
  scrape_interval: 5s
  scrape_timeout: 3s
  evaluation_interval: 5s

rule_files:
  - 'pyrra/*.yaml'

scrape_configs:
- job_name: prometheus
  static_configs:
  - targets:
    - localhost:9090
- job_name: svc
  static_configs:
  - targets:
    - svc:8765
```

Configures where Prometheus reads the rules Pyrra generated

Defines scrape jobs, Prometheus self-scraping, and our ho11y service

下一個列表是yaml文件pyrra/ho11y-http.yaml的內容，其SLO定義了我們的HO11Y服務。我們將目標設置為總體上的75%。

列表10.3 pyrra slo定義

```
apiVersion: pyrra.dev/v1alpha1
kind: ServiceLevelObjective
metadata:
  name: ho11y-api-query
  namespace: monitoring
  labels:
    prometheus: k8s
    role: alert-rules
spec:
  target: '75.0'
  window: 7d
  indicator:
    ratio:
      errors:
        metric: ho11y_total{job="svc",http_status_code=~"4.."}
      total:
        metric: ho11y_total{job="svc"}
```

The name of the SLO

Our target value, which is 75% availability. Note that it's a string because kubebuilder doesn't support the float64 type

Defines error SLI—what we mean by errors (not available)

Overall SLI

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

我們前面提到的是，Pyrra有效地將ServiceLevelObjective資源轉換為Prometheus記錄規則。看起來如何？如果將列表10.3作為輸入，則在以下列表中顯示了相應的輸出（您還將在Prometheus_pyrra/ Directory的本地驅動器上找到該輸出）（請注意，請注意，該文件總體上約有94行，因此我們僅在此處顯示第一個規則）。

列表10.4普羅米修斯記錄規則從ServiceLevelObjective得出

groups: - interval: 1m name: ho11y-api-query-increase rules: - expr: sum by(http_status_code) (increase(ho11y_total{job="svc"}[1w])) labels: job: svc slo: ho11y-api-query record: ho11y:increase1w - alert: SLOMetricAbsent expr:缺失（ho11y_total {job {job =“ svc”}）== 1 for：1m labels：job：job：svc嚴重性：critical slo：ho11y-api-query

...

現在，讓我們來看看它。前往http：// localhost：9099/在瀏覽器中，您應該看到圖10.2，pyrra著陸頁面，提供所有SLO的視圖。

圖10.2 Pyrra著陸頁（SLO列表）

圖10.3描繪了Prometheus Slo。您可以看到我們處於合規性（滿足SLO）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

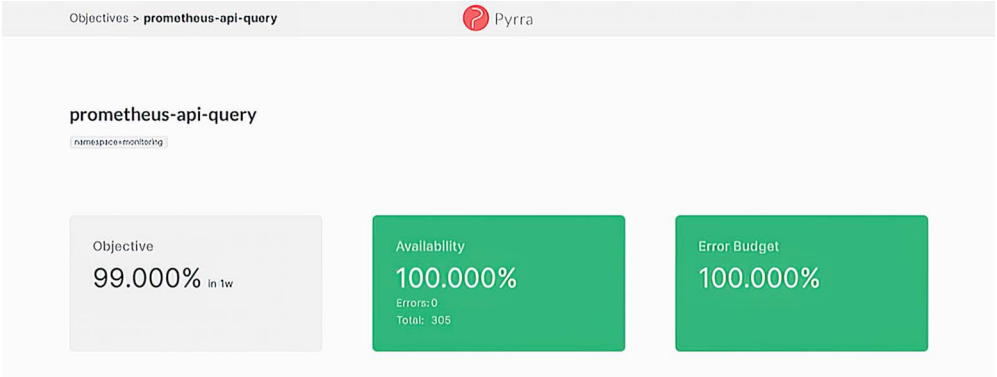


圖10.3 Prometheus slo in pyrra（SLO細節）

圖10.4和10.5顯示了pyrra中HO11Y的SLO，並且（如預期的，因此，我們看到75%的SLO被破壞。

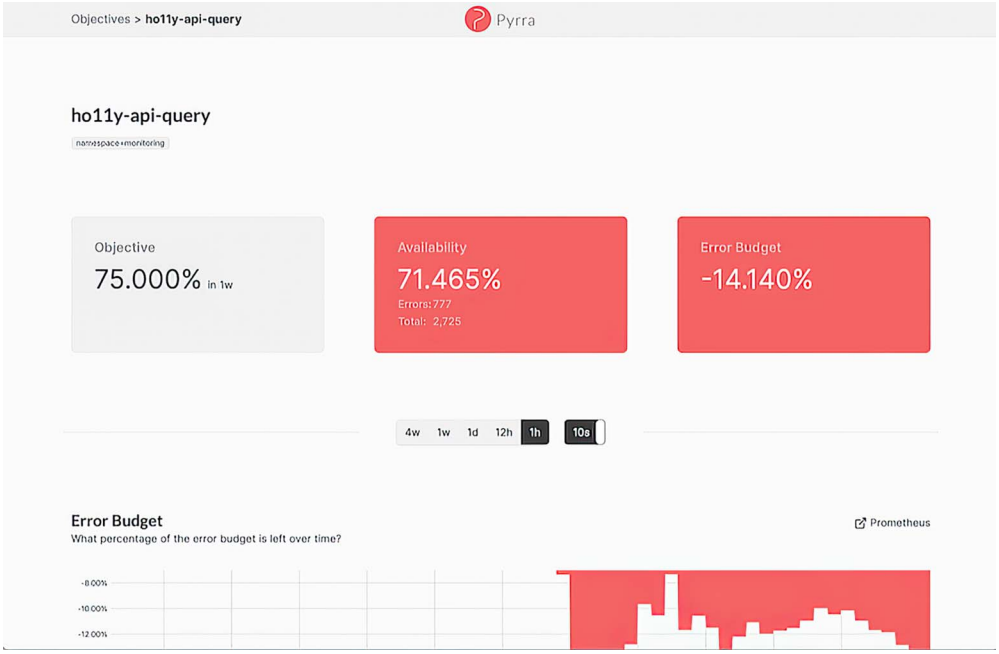


圖10.4 Pyrra中的HO11Y服務SLO（概述）

如果您想了解有關Pyrra的更多信息，請與Pyrra一起查看出色的Kubecon 2022 Talk Talk Talk“基於SLO的可觀察性”。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

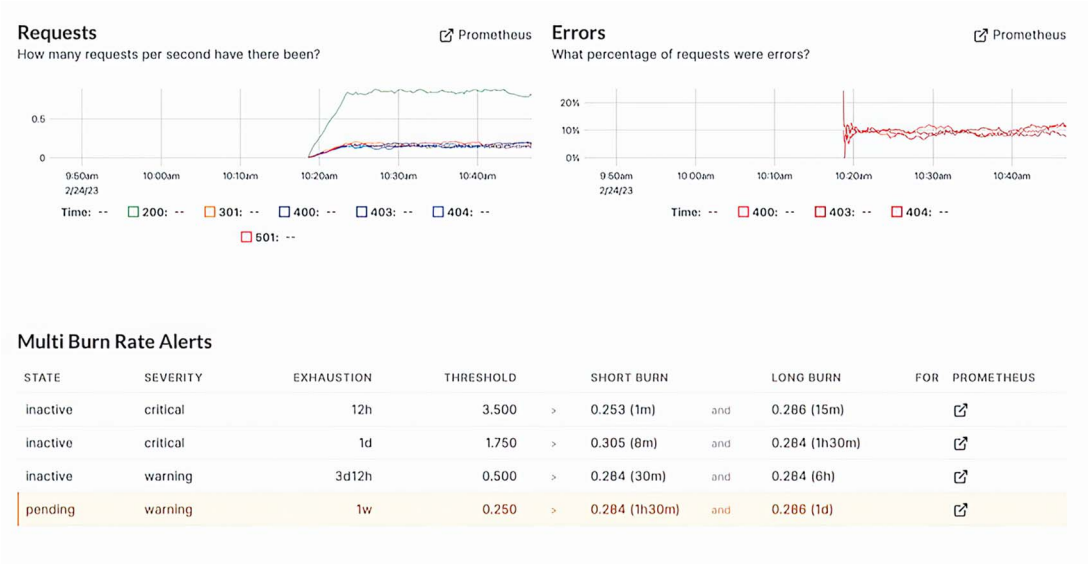


圖10.5 pyrra中的HO11Y服務SLO（詳細信息）

鉛維護者Matthias Loibl和Nadine Vehling（https://www.youtube.com/觀看？v = ac38jtxxe50）。

懶惰

懶惰（https://sloth.dev）還使用Prometheus生成Slos。簡而言之，它每個SLO都構成了三個規則組，其中包含有關SLIS，信息指標和警報的記錄和警報規則。 此外，它還具有所謂的SLI插件（https：//sloth.dev/sli-plugins/coredns-availasiety/），允許您抽象和擴展SLIS，包括用於Coredns，Istio和Traefik的插件。像Pyrra一樣，它也隨之支持的錯誤預算（SLO的擺動空間）和燃燒率（您使用預算的速度）。

鑑於與Pyrra的相似性和實質性重疊，我們不會潛入懶惰的工作原理。只需選擇在此空間中更適合您需求的工具。接下來，我們將介紹商業SLO和SLI工具。

10.2.3 Commercial SLO offerings

隨著SLO在撰寫本文時開始主流，您可以期望越來越多的商業產品。以下是當今可用的一些著名產品：

Nobl9（https://www.nobl9.com/）是一個完整的SLO解決方案。它具有深入的整合和覆蓋範圍，如果您對SLO的認真對待，您將需要評估它。 從雲提供者的角度來看，Google Cloud通過其操作套件（http://mng.bz/vneq）提供了支持SLO的支持。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

服務級別目標狀態頁面（http://mng.bz/4dwg）和SLO摘要小部件都是Datadog警報產品的一部分。 Honeycomb（https://www.honeycomb.io/slos）提供基於跟踪的SLO優惠作為其套房的一部分。 Dynatrace允許您在其產品中定義SLO（http://mng.bz/qpbe）。 新遺物提供服務級別管理功能（http://mng.bz/xnrm）。 Splunk提供SLO和SLI監控（http://mng.bz/yqme）。

SLO商業產品的空間正在迅速發展，您可以預期，越來越多的雲提供商和ISV來支持SLOS的前進。

10.3 Considerations

像往常一樣，公開標準規則。在這個空間中，OpenSlo（https://openslo.com）是相關（新興的）開放規範，用於定義SLOS以啟用供應商 - agnostic Tracking並與SLO進行交互。

例如，在Openslo中，您將代表本章前面的Pyrra漫步示例（與清單10.3進行比較），如下所示：

```
apiVersion: openslo/v1
kind: SLO
metadata:
  name: hello-api-query
  labels:
    prometheus: k8s
    role: alert-rules
spec:
  service: hello
  indicator:
    metadata:
      name: hello-error
      displayName: hello error
    spec:
      ratioMetric:
        counter: true
        good:
          metricSource:
            metricSourceRef: prometheus-datasource
            type: Prometheus
            spec:
              query: hello_total{job="svc",http_status_code~="2.."})
        total:
          metricSource:
            metricSourceRef: prometheus-datasource
            type: Prometheus
            spec:
              query: hello_total{job="svc"}
  objectives:
    - displayName: hello total errors
      target: 0.75
```

We are defining an OpenSLO SLO object.

The reference to the targeted service for this SLO

In contrast to Pyrra, where we defined errors, here, we define what the "happy path" looks like.

The goal that this SLO tracks: 75% availability (or non-error responses)

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

通常，系統定義SLO和SLIS，例如Kubernetes。有關一個示例，請參見“ Kubernetes量表和性能SLIS/SLOS/SLOS”（http://mng.bz/mbem）。 最後但並非最不重要的一點是，如果您想開始在空間中貢獻或只是了解更多信息，請考慮一下出色的Sloconf（https://www.nobl9.com/community/sloconf）活動，這是SRE社區的年度聚會，以交換與SLO相關的更新和學習。

Summary

可靠性法規即將到來，就像15年前的安全性一樣。 使用SLO衡量和報告合規性的可靠性正在成為金融，通信和航空技術的熱門話題，並可能傳播到其他行業。 要在可量化和自動化的SLA中做出合同承諾，請定義SLIS推動目標（SLO）。 SLA是服務提供商和服務企業之間的合同構造。 SLO是您組織定義的目標，可用於告知SLA。 SLIS是實際測量或度量標準（取決於服務類型），以驗證目標（SLO）在多大程度上實現的程度。 有開源工具（Pyrra和Sloth）可用於為Prometheus生態系統實施SLO/SLIS。 有一些SLO的商業產品，尤其是Nobl9的工具。 您不必全力參與SLO（例如，錯誤預算或燒傷率）；實際上，野外沒有多少人這樣做，而是定義服務指標（SLIS）並跟踪他們針對既定目標（SLOS）（SLOS）有助於您和您的客戶。 SLO可以用作提高警報準確性的一種手段，即減少假陽性的預警系統（非緊急性），以及用於抑制警報並改善基於趨勢的早期警告的一次性規則的數量，這些規則可與運行簿和自動補救相關。 確保以開放規格（Openslo）在此空間中建立。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

11 Signal correlation

This chapter covers

信號相關的含義為什麼您需要相關如何
實施相關實施相關性的挑戰

雲本地系統是*distributed systems*，它是指許多服務來處理請求的許多服務。當許多服務共同努力時，事實證明，尚不清楚哪種服務會導致故障或導致延遲問題。

四種基本的可觀察性信號類型（日誌，指標，痕跡和文件）各自提供見解；但是，他們通常無法回答您的所有問題。請記住，可觀察性的目的是從信號中獲得可行的見解。 那麼，如果單一信號類型不產生您希望的見解，該怎麼辦？使用一種以上的信號類型 - combine它們。

相關性可能會派上用場？很高興您問：

Incident response - 想像中，您正在致電基於微服務的應用。您將分類，並且警報包含指向Grafana儀表板的鏈接。來自

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

儀表板，您會看到應用程序中的錯誤率提高，並確定哪些服務有助於該錯誤，您使用指標 - 跟踪相關性來查找導致錯誤的服務痕跡。

Root cause analysis - 支持最近發生的事件，您想弄清楚是什麼原因造成的。指標遙測為您提供有關計算環境（例如，區域）的信息，您可以從日誌中收集版本信息（例如，有關所使用的庫，服務本身等）。為了弄清發生故障的原因，您可以使用指標與書記相關性來確定問題是否與某些機器類型，服務版本，區域相關。

Service performance improvements - LET假設您是開發人員，正在撥打機票，該機票召集一定數量的客戶報告端點的執行速度較慢。通過分佈式跟踪提供的服務圖允許您查明服務，並共同處理該端點的請求路徑。為了確定要改進的服務，您可以使用Trace -Profiles相關性並比較您可以優化的服務版本的配置文件。

雖然可以根據手動相關性一起使用多種信號類型，但這並不是您時間的有效利用。由*manual signal correlation*，我的意思是從信號類型開始，例如軌跡，並基於涵蓋的時間範圍，試圖查找有助於跟踪的服務的日誌線，並希望它們有些相關。希望不是策略。我們將首先在本章中的概念層面上討論信號相關性。 然後，我們將看到行動中的相關性，使用您已經熟悉的工具。最後，我們將總結一些用於實施信號相關的考慮因素。

11.1 Correlation fundamentals

讓我們首先定義*signal correlation*的含義，然後查看其工作原理。我將信號相關性定義基於BartekP otka的思想，即“有效地關聯現代可觀察性”（https://www.bwplotka.dev/2021/corralelations-exemplars/）。 我不僅引用了P otka的作品，因為我認識並相信他，還因為他在CNCF標籤可觀察性中擔任技術領導。

定義*Signal correlation*是連接不同信號類型的元數據驅動的過程，其目的是更快地獲得可行的見解和/或更準確。 元數據實現了多信號觀測能力的自動化。相關路徑是從一種信號類型到相關的相關路徑。

相關堆棧如圖11.1所示，顯示瞭如何實現信號相關的概念層。請記住，我們希望盡可能多地自動化，因此集成層越好，我們就會越接近完全自動化。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

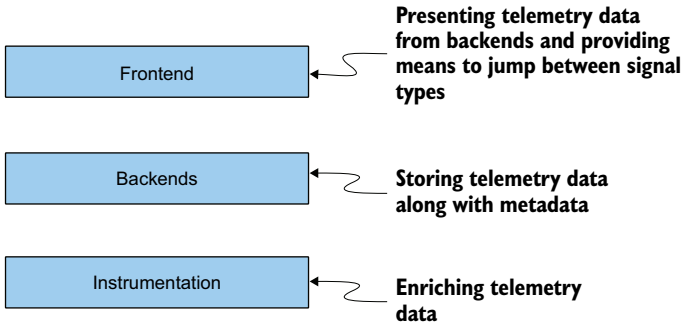


Figure 11.1 The correlation stack

從下到頂部的相關堆棧包括以下內容：

The instrumentation layer - 該圖層包括您的代碼和遙測代理，豐富了遙測數據並將其攝入後端。我們在第3章和第4章中詳細介紹了這一點。請注意，如果沒有該層的支持，則較高的層是具有挑戰性的。許多與源相關的元數據片（例如，lambda函數或pod的命名空間，其中kubernetes群集發出a（例如，跨度）僅可用，並且很難在沒有上述信息的情況下提供全面的相關性。

The backends layer - TeleMetry數據與元數據一起存儲在此層中。這可能包括每個信號類型的單獨後端或具有多種信號類型的後端，例如OpenSearch。 另請參見第5章，我們將更詳細地介紹此主題。 該層主要與驅動下一層有關，並提供原始數據，然後將其提供給用戶選擇具體的相關路徑。 *The frontend layer* - 有時也稱為*presentation layer*（請參閱第6章，我們介紹了主題，包括工具），此層是呈現信號的地方，可用於消費相關的遙測數據。各種可視化和渲染可以幫助您快速選擇，過濾和跳躍信號類型。

驅動相關性所需的元數據可以在事實之後進行標準化（例如，使用Prometheus Relabeling功能在第4章中討論的Prometheus Relabeling特徵時，也可以在第4章中討論。 後一種方法是建立定義的語義，是Opentelemetry在其語義慣例中所做的。現在讓我們仔細看看。

11.1.1 Correlation with OpenTelemetry

OpentElemetry通過語義慣例（https://opentelemetry.io.io/docs/concepts/semantic-cententions/），定義了元數據方案，包括常用名稱，有時是不同類型的遠程操作數據的共同值。更具體地說，OpenTelemetry支持以下元數據以在後端啟用信號相關：

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Resource attributes (http://mng.bz/a1nY) - 有關記錄遙測數據的實體的關鍵信息。他們可能會捕獲實體識別的整個層次結構。您可以使用它們來描述指定的雲區域中特定群集中的kubernetes節點，或者在運行您關心的應用程序的POD中的特定容器中描述。

Signal attributes are specific to a signal type, including – *Traces (http://mng.bz/gBMn)*— Semantic conventions, including for HTTP cli- ent and server spans, for remote procedur e calls, such as gRPC (https://grpc .io/) spans, messaging system (Apache Kafka, Rabbit MQ, etc.) spans, excep- tions associated with a span, and many others. - *Metrics (http://mng.bz/e1XZ)* - HTTP客戶端和服務器指標的儀式約定； RPC客戶端和服務器指標；標準系統指標，例如CPU使用或磁盤I/O；硬件;以及其他指標的列表。 - *Log (http://mng.bz/pP00)* - 在出版時，本質上仍然是實驗性的。僅定義了兩個：一個用於日誌媒體（描述日誌的來源，例如文件名或I/O流的源），以及一個可以例如描述業務上下文（例如瀏覽器或設備）的事件屬性。

資源和信號屬性的具體示例是什麼樣的？ 讓我們假設我們正在運行由某些服務組成的應用程序。假設其中一個是svc-wow，在歐盟-West-1區域的EKS群集中的名為boom的名稱空間中，該服務揭示了HTTP指標。

該方案將以POD作為信號源來表示，因此我們將使用以下資源屬性：

對於EKS群集，服務所在，我們將使用cloud.region（http://mng.bz/oxdo）屬性，具有EU-WEST-1的值和AWS-Specific Aws.eks.eks.cluster.arn（http://mng.bz/y12q）的屬性。 ARN：AWS：ECS：EU-WEST-1：123456789123：群集/prod-42。 對於kub ernetes命名空間該服務所在，我們將使用通用資源屬性（http://mng.bz/gyq8）servic e.namespace，具有boom and service.name的值。 我們在此處查看的信號類型是指標，這意味著我們正在使用signal屬性在發出的指標上提供元數據。假設SVC-WOW服務公開了HTTP API，我們想報告入站請求持續時間。我們將使用http.server.dura tion（http://mng.bz/zxyg）met- ric，具有http.method，http.status_code和net.host.nam e等屬性。偶然地，HTTP_STATUS_CODE屬性是屬性的一個示例，其中不僅名稱是標準化的，而且是其值，即HTTP響應狀態代碼（https：//wwwww.rfc-editor.or.or .or.g/rfc/rffc/rfc/ rfc7231＃extifter-ections-e節），以及固定的fefinited infinited。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

表11.1概述了可能的組合和從一種信號類型到另一種信號類型的過渡，其跡線代表交叉服務執行，代表數值聚合物的指標（例如，錯誤的百分比），將日誌作為沿請求路徑代表單個服務的signal類型。請注意，這些轉移不是特定於格式的特定格式，例如opentelemetry。

表11.1信號相關概述

From/To	Traces	Metrics	Logs
Traces	Related spans	Trace to representative metrics	Trace to log lines from services
Metrics	Metrics to representative spans	Related metrics	Metrics to logs from services
Logs	Logs to spans from services	Logs to relevant metrics	Related logs

術語*correlation path*在以下小節中是指從一種信號類型到另一種信號類型的過渡。這可能包括多次轉移。現在，讓我們剖析表11.1，按信號類型討論特定的相關性。

11.1.2 Correlating traces

在第一種情況下，相關路徑的起點是痕跡，代表交叉服務執行，沿請求路徑涵蓋同步服務，並且在電位上，異步服務（例如消息經紀人）。從這裡，我們將過渡到特定的相關性。

到跡線

*Traces–traces correlation*是指根據環境或服務找到相關的跨度。例如，您可以在Opentelem- etry中使用跨度鏈接功能（http://mng.bz/0knw），它使您可以將跨度與一個或多個跨度相關聯，從而定義了異步的fire fire and-forget fire and-firgent fire and-firgent fire and-firgent fire and-firget工作，父母的範圍在其中的孩子不等待它的孩子們在關閉之前就無法自行關閉。這對於諸如消息總線和消息隊列之類的異步服務特別有用，包括Apache Kafka（https://kafka.apache.org/）和Amazon Eventbridge（https://aws.amazon.com/ eventbridge/）。

指標

*Traces–metrics correlation*表示輸入是表示交叉服務執行的軌跡，輸出由相關指標組成，該指標來自對輸入跟踪跨度的服務。

到日誌

*Traces–logs correlation*表示輸入是表示交叉服務執行的跟踪，輸出由對日誌線組成，該日誌線從貢獻跨越跨度的服務線組成

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

輸入跟踪。 Java（http://mng.bz/keae）的AWS X射線SDK中的相關支持是一個示例。

11.1.3 Correlating metrics

在這種情況下，相關路徑的起點是指標的時間序列 - 數量匯總值。從這裡，我們可以過渡到特定的相關性。

到跡線

*Metrics–traces correlation*意味著您有一個指標作為輸入（例如，在儀表板上或警報中繪製），預期的產出由代表跨度的跨度組成，這些跨度來自服務時間序列的服務。在下一個小節中，我們將對這種相關案例進行具體查看，在該小節中，我們將使用Prometheus，Jaeger和Grafana來實施基於典範的度量與跨度之間的相關性。

指標

*Metrics–metrics correlation*是指相關的指標（按類型或服務）（例如，客戶端和服務器端錯誤）。

到日誌

*Metrics–logs correlation*表示輸入是度量標準（例如，來自儀表板），輸出由對匯總值的服務的日誌線組成。

11.1.4 Correlating logs

在這種情況下，相關路徑的起點是來自一個或多個服務的日誌。從這裡，我們可以過渡到特定的相關性。

到跡線

*Logs–traces correlation*意味著我們將服務線路線作為輸入，並尋求從編寫日誌線的服務中發出的跨度跨度。

指標

*Logs–metrics correlation*使用服務的日誌線作為輸入，並從上述和相關服務（例如，沿請求路徑）輸出相關的會議。

到日誌

*Logs–logs correlation*試圖通過服務，環境或嚴重性找到相關日誌。例如，您可以沿請求路徑詢問來自服務的所有錯誤日誌。

11.1.5 Correlating profiles

我們在第9章中討論了概況的性質和用例。 它們在某種程度上與日誌線相媲美，因為它們的範圍是一項服務。輪廓可以自動發射（通過EBPF或特定語言手段）或手動發射。與日誌相反，文件是數值的。因此，儘管相關觀察的前面許多之前的許多先前，但要考慮到概況是數值的價值，但在一天結束時（因此，從相關的角度來看，它們的行為就像指標一樣）。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

11.2 Using Prometheus, Jaeger, and Grafana for correlation

現在您知道了概念上的相關性是什麼，讓我們研究如何實施它。為此，我們將使用一個具體示例，該示例顯示瞭如何進行指標 - 基於示例的跟踪相關性。在指標 - 跟踪相關性的情況下，這意味著嵌入從指標表示到跡線的鏈接。這是通過插入跟踪ID來完成的，以便在時間序列圖中呈現度量時，您可以單擊示例並直接跳到代表性跟踪並導航其跨度。從概念上講，這種“指標中的跟踪ID嵌入”看起來像以下內容（使用Prometheus exposition Format或更精確，開放度量，因為此功能僅存在）：

a_counter 42.0＃{traceid =“ a6427ef2021cce03”}

現在，我們將繼續進行示例相關設置。

11.2.1 Metrics–traces correlation example setup

我們將使用Prometheus和Jaeger作為後端，Grafana作為前端。此示例中的示例服務基於我們在第2章中使用的迴聲服務。它發出了跨越opentelemetry收集器到Jaeger的路線，同時，收集器刮擦了帶有嵌入式跟踪ID的指標的服務 /指標端點，並使其可用於刮擦Prometheus後端。 您應該已經知道所有這些工具以及如何配置它們，因此我將在此處簡要介紹。

所有顯示的來源均在第11章中，因此更改為該目錄。首先，Echo服務用GO編寫，發出跨度並揭示指標。此外，它現在還支持OpenMetrics示例（http://mng.bz/9dz0）。源代碼在echo/main.go文件中；我們只專注於與如何發射範圍相關的添加，我們只會涵蓋快樂的道路（錯誤情況看起來或多或少相同，但它在40％的情況下返回500代碼，而不是200）：

```
invoke, _ := invokes.GetMetricWith(prometheus.Labels{
        "http_status_code": strconv.Itoa(http.StatusOK),
    })
    invoke.(prometheus.ExemplarAdder).AddWithExemplar(
        1,
        prometheus.Labels{"traceID": traceID},
    )
    invokes.WithLabelValues(
        strconv.Itoa(http.StatusOK),
    ).Inc()
    span.SetAttributes(
```

Embeds the trace ID as an exemplar into the invoke counter

Increases the invoke counter with the http_status_code=200 label

Emits the span with the same metadata as used in the metric (HTTP status code)

除了發出跨度和指標，帶有嵌入式示例外，只有另一件事要注意主機。 我們需要確保使用OpenMetrics格式，這是這樣做的：

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

http.handle（“/量表”，Promhttp.handlerfor（註冊表，Prom
http.Handleropts {enableOpenMetrics：true，}，））

現在，我們需要拉動所有依賴項，因此（假設您已安裝了1.20版或更高版本）執行以下操作：

Go Mod供應商

現在，您應該看到一個子目錄，即供應商/，該供應商/它在當地提供了所有必要的軟件包。

接下來，讓我們回到主要目錄，CH11/Metrics-traces，並提出Docker組合堆棧中定義的docker-compose.yaml：yaml：

docker-compose -f docker-compose.yaml up

以下列表顯示了我們用於指標跟踪示例的Docker組合文件的相關片段。

列表11.1 Docker的相關片段撰寫了相關設置的文件

```
prometheus:
    image: prom/prometheus:v2.33.4
    ports:
    - "9090:9090"
    volumes:
    - ./prometheus.yml:/etc/prometheus/prometheus.yml
    command:
    - "--config.file=/etc/prometheus/prometheus.yml"
    - "--storage.tsdb.path=/prometheus"
    - "--enable-feature=exemplar-storage"
  grafana:
    image: grafana/grafana:latest
    ports:
    - "3000:3000"
    volumes:
    - ./grafana-storage:/var/lib/grafana
    - ./grafana:/etc/grafana/provisioning
    - ./grafana/dashboards:/var/lib/grafana/dashboards
```

Need to configure Prometheus to support exemplars (in-memory)

Used for automatically configuring the Prometheus and Jaeger data sources as well as the dashboard

讓我們繼續進入以下列表中顯示的OpenTelemetry Collector配置-Otel-config.yaml文件。

列表11.2相關設置的OpenTelemetry收集器配置

出口商：Prometheus：端點：“ 0.0.0.0
：1234” Metric_expiration：180m

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
    enable_open_metrics: true
  jaeger:
    endpoint: jaeger:14250
    tls:
      insecure: true
service:
  pipelines:
    metrics:
      receivers: [ prometheus ]
      processors: [ batch ]
      exporters: [ logging, prometheus ]
    traces:
      receivers: [ otlp ]
      processors: [ batch ]
      exporters: [ logging, jaeger ]
```

Enables exemplar support in the Prometheus exporter

The metrics and traces pipelines

如果您想查看原始示例，則可以在計算機上執行以下操作（假設Docker正在運行； http://127.0.0.1:8888/是Echo-SVC服務的地方）：

Sending this HTTP header is key to get the exemplar (asking for OpenMetrics).

The echo service endpoint, serving the metrics in OpenMetrics format

```
curl -H "Accept: application/openmetrics-text" \
        "http://127.0.0.1:8888/metrics"
```

執行前面的curl命令時，您應該看到類似於以下輸出的內容：

A simple dimension, represented via the label for the HTTP status code

```
# HELP echo Total invocations of the echo service endpoint.
# TYPE echo counter
echo_total 930.0 # {http_status_code="200",
                   traceID="f2a9e19d5c34427cf2f50a430de9fa3c"}
                   1.0 1.6801173123788908e+09
# EOF
```

The exemplar represented via the Prometheus label traceID

這樣，w　E具有使用信號相關的設置。讓我們d　　哦，現在。

11.2.2 Using metrics–traces correlation

現在該關聯了！ 轉到您的瀏覽器並輸入Localhost：3000，該主機打開Grafana（無需憑證）。現在，用示例打開名為Echo Service儀表板的儀表板，然後等待幾分鐘來收集足夠的數據。然後，您應該看到類似於圖11.2的東西。第一行包含echo_total度量，其中包含200和500個狀態代碼，當您仔細觀察時，您會看到圖形底部的一些小點。 這些使Grafana時間序列可視化得以呈現典範。懸停在它們上，您將看到跨度信息；如果單擊它們，您將進入圖11.3中描述的Jaeger Trace View。

圖11.2的下部顯示了您也可以用來導航到Jaeger Trace View的誤差跟踪。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

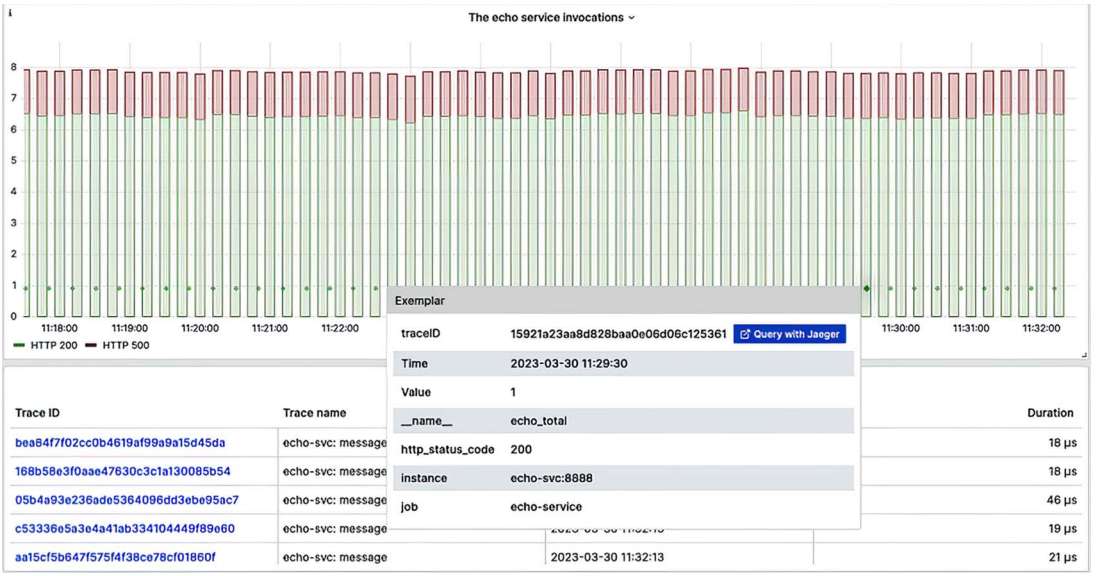


圖11.2 Grafana儀表板的屏幕快照，帶有ECHO服務的示例性（底部的小點）

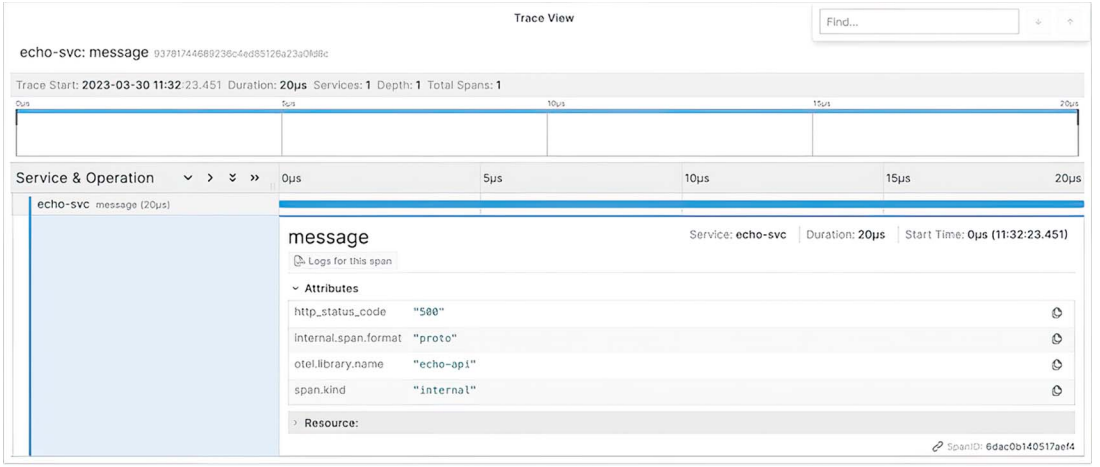


圖11.3迴聲服務的Jaeger View的屏幕截圖，重點是代表500響應的錯誤跨度

花點時間探索視圖以及如何使用示例，從示例到痕蹟的度量。這是很多東西，所以這可能需要一些時間。接下來，我們將在信號相關空間中快速審查商業產品。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

11.3 Signal correlation support in commercial offerings

雖然現在還很早，而且並非所有相關路徑都得到了每個提供商的支持，但已經對商業空間中的許多相關性都有核心支持。通常，這些與痕跡 - logs和指標 - 軌跡支撐有關。

首先，讓我們查看雲提供商及其本機監視和觀察能力解決方案：

AWS具有基於OpentElemetry（http://mng.bz/jpqv）的全面關聯策略，並具有通過AWS X-ray和Amazon CloudWatch（http://mng.bz.bz/wzwd）支持的Traces-Logs相關性。 像AWS一樣，Azure正在標準化OpenTelemetry。 例如，請參見應用程序見解 - 基於基於的相關支持（http://mng.bz/8raw）。 GoogleCloud提供了將日誌條目（https://cloud.google .com/logging/docs/view/corserletate-logs）相關聯的支持。

現在，我們將從可觀察性ISV轉移到商業產品。在觀察能力空間中，幾乎所有供應商都在其解決方案中提供一種或另一種形式的相關支持：

AppDynamics支持語言中的自定義相關性，例如Java（http://mng.bz/eqvd）和.NET（http://mng.bz/n2px），以實現對事件的指標。 在datadog中，您可以關聯度量標準（https://docs.datadoghq.com/dashboards/ correlations/），並且它們提供Traces-Logs相關性（https：//www.datadoghq .com .com/blog complast/blog unpess/replect unpess/replect unpess/replect-corness-log-correlation/）。 使用Dynatrace，您將獲得一組豐富的跟踪相關性（http://mng.bz/d4ng）。 Honeycomb提供了與其他APM相關的方法（https://www.honeycomb.io/compari-sons）的一個不錯的比較，指出了諸如其查詢速度和對高心電圖數據分析的支持之類的東西。 LightStep提供與跟踪相關的（http://mng.bz/lwmr）相關性支持。 logz.io提供基於OpenSearch的Trace -Logs（http://mng.bz/bm6l）相關性。 在New Relic的產品中，支持與決策（http://mng.bz/d1r1）配置相關邏輯，提供相關路徑的混合物。 Splunk具有一種有趣的方法，支持相關命令（http://mng.bz/rwxj），可讓您看到遙測數據中字段之間的共發生概述。 sumo邏輯允許您通過其云SIEM（https://www.sumologic.com/blog/siem-investigation-corlelation/）提供的雲調查中的信號。

現在，您對關聯空間中的商業產品和開源工具有一個想法，讓我們繼續對相關性的一些一般考慮以及在介紹它們時要注意什麼。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

11.4 Considerations

到現在為止，應該很明顯的是，在有效地獲得可行的見解的背景下，信號相關起著關鍵作用。我不會假裝很容易實現完整和全自動的相關性，即使在最好的情況下，您有多徵信號的後端和一個玻璃窗格。在出版時，仍然存在與實施相關性相關的許多挑戰，代表了可觀察到的最大風險類別。因此，一方面，您不應該將它們視為事後的想法，另一方面，您應該意識到限制並適當設定您的期望。 例如，如果您是一個平台團隊（內部開發人員平台簡化了基礎線和應用程序，以便避免售票處），則可能需要相應地將相關性優先考慮。

請注意，*TicketOps*不是指公司，而是略微貶低的術語，該術語用於描述開發人員在jira中創建門票的過程，例如，在平台團隊中（在平台團隊中）將它們逐漸解決。這可能需要一些時間（例如，幾天或幾週），與DevOps提出的自動化咒語相反。

而現在，事不宜遲，讓我們來看看組織信號相關性的具體挑戰。我已經將它們分為三個雄厚，並且對於某些挑戰，我建議您可以嘗試減輕風險。

11.4.1 Early days

首先，讓我們回顧一下與總體上，尤其是實施情況相關的主要挑戰。

缺乏標準

APM和監視工具（開源和商業都可以使用不同的格式）用於遙測信號以及各種術語。這使得將信號跨系統相關聯。

以下是適當的補救：如果您需要真正的互操作性並避免供應商鎖定，請使用OpenTelemetry。

缺乏元數據

為了驅動相關性，您需要以元數據為代表的結締組織，包括但不限於EC2元數據，Prometheus標籤，Kubernetes標籤以及OpentElemetry資源和信號屬性。迄今為止，很少有平台和系統是“首先可觀察性”（具有可觀察性支持），因此可能很難獲取上述元數據或涉及手動工作。

以下是適當的補救措施：

根據其元數據豐富度和自動化程度評估（計算）平台。 使用基於OpentEmetry的系統。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

11.4.2 Signals

接下來，我們將研究與信號本身有關的挑戰。

遙測信號量

從不同來源產生的數據的總體量使得很難有效地將信號效率相關聯。這不僅是一個成本問題，而且會對性能和流動產生嚴重影響。例如，如果您要索引日誌並且工具被過度使用的用戶查找請求，則操作員可能會放慢腳步，沮喪並有效阻止。

以下是適當的補救措施：

採用選擇性攝入，這意味著您可以限制預先限制音量。 這可能非常棘手，因為您不提前知道“有價值的”信號有助於獲得可行的見解。 對於記錄，請確保您的事件具有優先級或loglevel屬性。這使您可以做

*ingest all errors, 50% of all warnings, and 1% of informational log lines*之類的事情。

一種後驗方法是指採用機器學習（ML）來檢測模式。這可以幫助您處理該卷，因為它使您能夠等同於相關事件，並確定可能導致所需見解的相關事件。 不過，請注意，通過ML驅動的異常檢測，成本是在計算中，而不是存儲，因此攝入成本遠高於信號保留。

高基數

如第5章（尤其是基數爆炸挑戰）所討論的高心電圖數據可能使其很難關聯信號。這是因為值可能只發生幾次，或者在最極端的情況下一次。

以下是適當的補救措施：

限制基數。 這需要實施硬性限制，其缺點是，您不知道，在通常的情況下，正確的截止點是什麼，而您又可能會失去有價值的信號。 確保您選擇的可觀察性提供支持高速度性，這對於蜂窩和LightStep尤其重要。

採樣

在分佈式跟踪的情況下，採樣（請參閱第8章，我們對此進行詳細討論）可能是相關性的挑戰。缺失的跨度可能會導致孔中的孔，因此，總體用途降低。有一些供應商可以為您提供幫助。例如，HoneyComb有一個智能採樣器，該採樣器的錯誤使用率與快樂路徑不同。這是預製有價值的信號的變體。

數據隱私和安全性

許多行業和地理位置，例如歐洲的一般數據保護規範（GDPR; https：//commession.europa.eu/law/law/law/law/law/law/law/law/law-topic/data-protection_en en），都需要刪除數據，例如刪除個人身份的數據

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

信息（PII）。因此，可能是類似於採樣的情況，要驅動相關性的部分信息不存在。此外，隱私腿部或行業合規性標準可能要求嚴格內部處理信號，而不會發送給某個地區以外的SaaS提供商或云提供商。

11.4.3 User experience

最後但並非最不重要的一點是，我們將討論與用戶實驗（UX）相關的挑戰。

缺乏可行的見解

系統可能對相關性實施一些支持，但是我發現它使用可能很麻煩或不直覺。 在這種情況下，人們往往不會打擾，因此不使用該功能。

以下是一個適當的補救措施：除了（作為供應商）向UX支付ATCH，作為最終用戶，您還可以根據其產生的見解的價值評估產品。 Dynatrace和New Seric都提供（ML生成的）建議和見解，這可能是您評估的起點。

端到端的支持

有時，UX挑戰來自缺乏端到端的支持。 這種挑戰可能會影響儀器，代理，後端，前端等。

以下是適當的補救措施：

使用擁有整個堆棧的完全垂直集成產品，例如Datadog，New Relic和Splunk。 押注開放標準，特別是OpentElemetry。 在撰寫本文時，標準尚未涵蓋一個領域：查詢後端。該通信性已經確定了這個問題，並正在研究CNCF技術諮詢組（TAG）的上下文中的查詢標準化（http：//mng.bz/v1jx）。

11.5 Conclusion

考慮到實現信號相關性的這些考慮，我們現在不僅結合了本章，而且還要確定本書本身。堅持使用附錄，我們將使用OpenTelemetry在多輔助，多編程語言應用程序的上下文中獲得骯髒的實現性。

我很想听聽您的挑戰和成功案例，這些挑戰和成功的故事以及您想分享的其他內容。請在MH9@o11y。工程上伸出援手。願您的見解與您同在！

Summary

一種信號類型通常不足以獲得可行的見解。 信號相關性是元數據驅動的過程，該過程連接不同的信號類型，以更快，更準確地獲取可行的見解。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

相關路徑可以包含多個過渡，例如，從代表與特定服務的日誌線或配置文件相關的跨度指標開始的時間序列開始。 驅動相關性所需的元數據可以在事實（例如Prometheus Relabeling）或前面（例如Opentelemetry的半公約）之後進行標準化。 並非所有相關路徑都是相等的。有些，例如指標 - 軌跡或痕跡 - logs被廣泛使用和支持，而另一些則不是。 示例（在OpenMetrics和OpentElemetry中）允許您通過將指針嵌入指標中的指針來進行度量 - 跟踪相關性。 我們仍處於信號相關的早期；請注意局限性並相應地設定期望。 與實施相關性，尤其是關於UX和信號量，基數和採樣相關的挑戰。 使用OpenTelemetry可能是對沖相關空間中挑戰的最有效方法。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

appendix A Kubernetes end-to-end example

在本附錄中，我們將在Kubernetes中使用端到端可觀察性樣本應用。在整本書中，我們一直使用Docker撰寫作為我們選擇的工具。作為一種教育手段，它具有許多優勢；但是，實際上，您更有可能將Kubernetes，Amazon ECS或Hashicorp Nomad用作編排者。因此，本附錄將為您提供一種更本地和同意的方法，涵蓋我們討論的所有跟踪，指標和記錄主題，並允許您進行設置。

> 提示您*could*使用kompose（https://kompose.io/user-guide/）將任何docker撰寫我們介紹給kubernetes資源（例如部署和服務）中的任何docker撰寫示例文件，以獲得與書中討論的OSS工具一起工作的感覺。

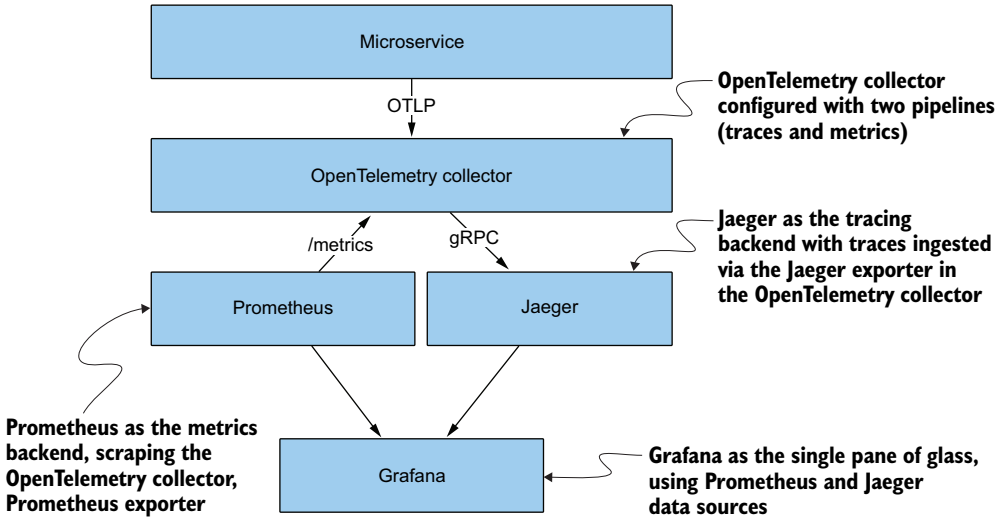
對我們來說幸運的是，已經有一些適合該法案的東西：開放式遙測社區演示（https://opentelemetry.io/docs/demo/）。該演示涵蓋了幾種使用不同編程語言編寫的服務，並功能我們討論過的工具。我們將首先討論演示及其工作原理，然後涵蓋運行它的先決條件，然後一起瀏覽演示。

A.1 Overview

OpentElemetry Demo是一個基於微服務的購物應用程序，使用12種不同的編程語言，包括Java，JavaScript，Python和.NET，用於實施服務（https://opentelemetry.io.io/docs/demo/services/）。 該演示具有所有服務，發射跨度，指標和日誌線。我們不會在這裡介紹演示服務及其互動的詳細信息，但是如果您是

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

有興趣深入研究，請查看討論演示體系結構（https://opentelemetry.io/docs/demo/architecture/）的OpenTelemetry文檔。 圖A.1顯示了OpenTelemetry演示中的遙測流。



圖A.1 OpenTelemetry演示遙測流量

遙測流中的相關位是

Microservices - 使用OTLP*The OpenTelemetry collector*將遠程測定數據發送到OpentElemetry集群，並配置了兩個管道（請參閱列表A.1）*Prometheus* - 尺度的後端后端the the the the the the the the the the the the the the the the the the the the the plaps and jee plaps ushth p under punder punder unious the plagn pean paine - 來源

OpentElemetry收集器配置（http://mng.bz/x4p6）在以下列表中更詳細地顯示。

列表A.1 OpenTelemetry Collector配置

接收者：
OTLP：協議：GRPC：HTTP：CORS：允許的_origins：- “http：//*”
- “https：//*”

我們通過OTLP接收信號。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

出口商：OTLP：端點：“jaeger：4317” TLS：in secure：true Logging：Prometheus：endpoint：“0.0.0.0：9464” resource_to_to_to_telemetry_conversion：enabled：enabled：enabled：enabled：true enable_open_metrics：true：true：true

A.2 Prerequisites

要在Kubernetes中安裝OpenTelemetry演示，您將需要一個Kubernetes群集：

如果您想在本地執行執行，那麼一些流行的選擇是 - Docker Desktop（https://docs.docker.com/desktop/kubernetes/） - ankn（https://kind.sigs.k8s.io/） （https://minikube.sigs.k8s.io/docs/） - microk8s（https://microk8s.io/）

雲提供商已管理Kubernetes產品，具體取決於您通常在哪里工作。請查看以下內容：- 亞馬遜彈性Kubernetes服務（EKS; https：//aws.amazon.com/eks/） - Azure Kubernetes服務（aks; https; https：//learn.microsoft.com/en-us/en-us/azure/aks/aks/aks/aks aks/aks/aks） - google kubernetes Engine（gke; 引擎）

您還需要以下CLI工具：

kubectl - 通過Kubernetes Docs中的指令（https：//kubernetes .io/docs/tasks/tools/）安裝。 EKSCTL - 如文檔（https://github.com/eksctl-io/eksctl #installation）所述的安裝。 *Helm* - 基於文檔（https://v3-1-0.helm.sh/docs/intro/install/）。 JQ - 通過文檔（https://jqlang.github.io/jq/download/）鍵入。 curl *and*觀看 - 可以在系統上可用； 否則，請諮詢您的Linux發行版或Windows文檔。

我建議您的數據平面中至少有兩個中型節點。例如，我使用的是具有三節點群集的EK，該群集使用t2.large（https：//stances.vantage.sh/aws/ec2/t2.large）EC2實例（每個節點有兩個VCPU和8 GB RAM）。

為了提供EKS群集，我將使用EKSCTL（https://eksctl.io/）與配置文件Otel-demo-cluster.yaml一起使用，如以下列表所示。

列表A.2 EKS群集配置

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: otel-demo
  region: eu-west-1
  version: "1.24"
iam:
  withOIDC: true
```

Make sure to set this to your region (here, in Dublin, Ireland, eu-west-1).

This will provision the OIDC provider so that you can attach IAM policies if you want to change the OpenTelemetry collector config, for example, to send metrics to Amazon Managed Service for Prometheus or traces to AWS X-Ray.

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

```
nodeGroups:
- name: default-mng
  instanceType: t2.large
  desiredCapacity: 3
cloudWatch:
  clusterLogging:
    enableTypes: ["*"]
```

The data plane—three t2.large EC2 instances

Enables control plane logging in CloudWatch

然後，提供並啟動EKS群集如下：

$ eksctl創建群集-f otel-demo-cluster.yaml

等待10到15分鐘，然後驗證（根據您的設置，您可能會看到較少的節點）：

```
$ kubectl get nodes
NAME                STATUS   ROLES    AGE     VERSION
ip-1..7.eu-west-1   Ready    <none>   2d23h   v1.24.10-eks-48e63af
ip-1..4.eu-west-1   Ready    <none>   3d14h   v1.24.10-eks-48e63af
ip-1..1.eu-west-1   Ready    <none>   2d23h   v1.24.10-eks-48e63af
```

現在，我們已經準備好進行演示。

A.3 Demo walk-through

首先，我們將安裝OpenTelemetry演示。安裝後，我們將準備好使用它！

A.3.1 Installing the demo

使用以下命令安裝演示。首先，我們需要添加演示的Helm存儲庫，就像這樣：

$ helm repo添加開放式\ https：///open-telemetry.github.io/opentelemetry-helm-charts

現在，我們可以安裝演示：

$ HELM安裝my-otel-demo \ open-telemetry/opentelemetry
-demo

驗證一切都出現了；這可能需要幾分鐘。您應該看到如下的某些內容（實際上，您要確保所有POD都處於運行狀態；請注意，我已經編輯了輸出以適合）：

$ kubectl -n默認獲取吊艙

```
NAME                          READY STATUS    RESTARTS        AGE
my-otel-demo-accountingse... 1/1   Running   4 (13d ago)     13d
my-otel-demo-adservice-86... 1/1   Running   0               13d
my-otel-demo-cartservice7... 1/1   Running   0               13d
my-otel-demo-checkoutserv... 1/1   Running   4 (13d ago)     13d
my-otel-demo-currencyserv... 1/1   Running   0               13d
my-otel-demo-emailservice... 1/1   Running   58 (83m ago)    13d
```

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

my-otel-demo-featureflags... 1/1 Running 1 (13d ago) 13d my-otel-demo-ffspostgres-... 1/1 Running 0 13d my-otel-demo-frauddetecti... 1/1 Running 0 13d my-otel-demo-frontend-65d... 1/1 Running 0 13d my-otel-demo-frontendprox... 1/1 Running 0 13d my-otel-demo-grafana-8445 ... 1/1運行0 13d my-otel-demo-jaeger-85484 ... 1/1運行2（11d ogo）13d my-otel-demo-kafka-77bfdc ... my-otel-demo-otelcol-b75c ... 1/1運行0 13d my-otel-demo-paymentservi ... 1/1運行0 13D my-otel-demo-productcatal ... 1/1運行0 13D my-otel-otel-demo-demo-demo-prometheus-s ... 1/1/1 my-otel-demo-recommendati ... 1/1運行0 13D my-otel-demo-redis-cf9dd8 ... 1/1運行8（7h27m前）13d my-otel-demo-shippingserv ... 1/1 1/1運行0 13d

現在，我們準備使用演示。恭喜！

A.3.2 Using the demo

要使用本地計算機中的演示（例如，訪問Grafana UI），您首先需要將網絡流量從Kubernetes群集轉發到計算機。

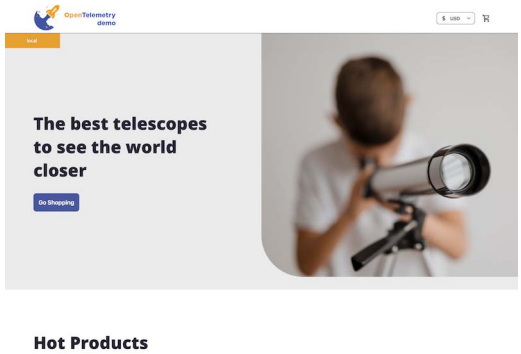
首先，我們將公開FrontendProxy服務。 使用以下命令實現這一目標：

```
Kubectl Port-Forward \ svc/my-otel-demo-frontendproxy 8080：8080
```

為了使跨度從本地瀏覽器正確收集，您還需要暴露OpenTelemetry收集器（4318是OTLP端口）：

```
Kubectl Port-Forward \ svc/my-otel-demo-otelcol 4318：4318
```

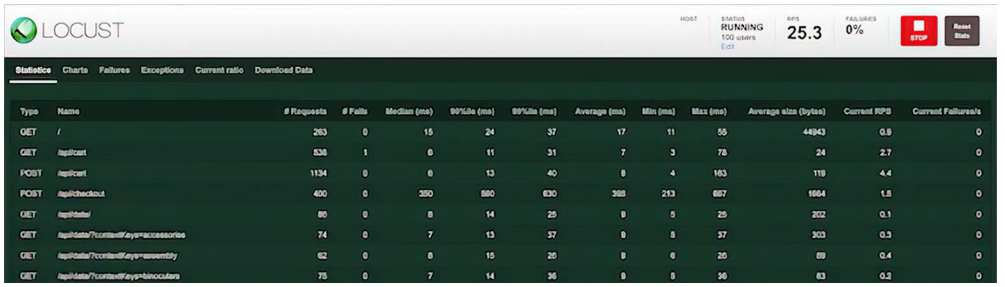
如果您現在轉到Localhost：8080（http：// localhost：8080/），則應該看到類似於圖A.2（OpentElemetry Demo Shop Shop首頁）的東西。



圖A.2 OpenTelemetry Demo Frontpage的屏幕截圖（商店）

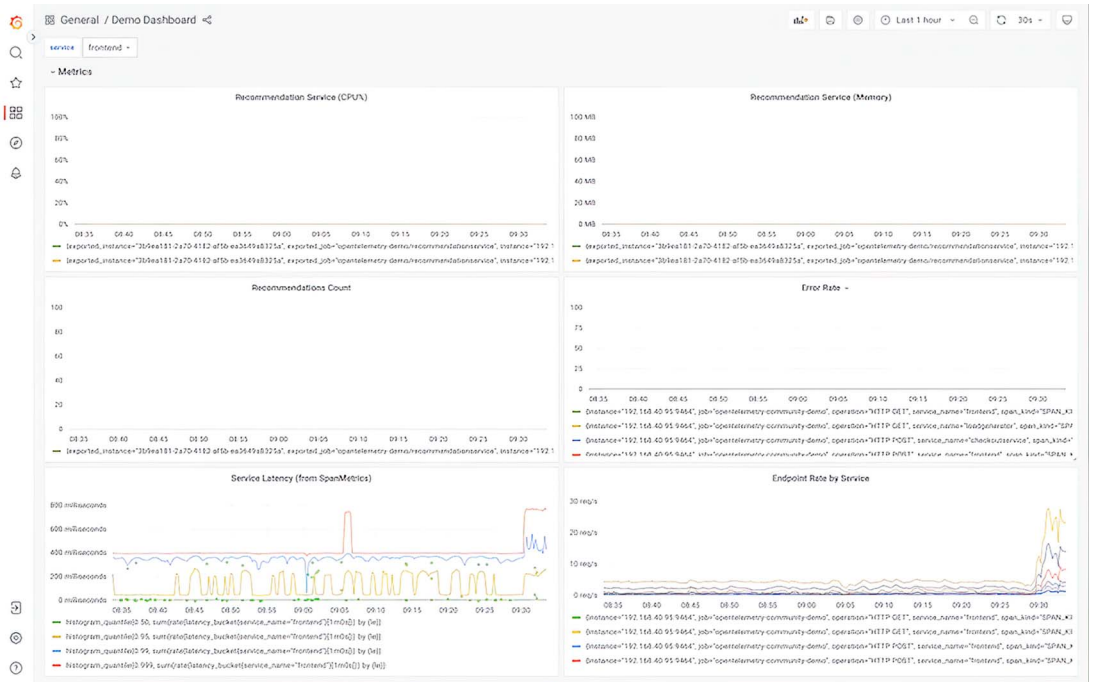
Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

單擊“去購物”，然後開始在購物籃中添加物品，然後查看這些物品（不用擔心 - 嘲笑，這樣您就不會使用真錢）。雖然點擊是一種開始的好方法，但我們不想等待太久就會產生遙測信號，對嗎？這就是為什麼我們使用負載生成器的原因（請參見圖A.3），您可以在http：// localhost：8080/loadgen/。



圖A.3 OpenTelemetry Demo Load Generator的屏幕截圖

圖A.4顯示Grafana是玻璃的單個窗格。準確地說，這就是演示應用儀表板向您展示前端服務指標。 訪問http：// localhost：8080/ grafana/到達那裡。



圖A.4 OpenTelemetry Demo Grafana的屏幕截圖（前端服務DA 滑板）

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

圖A.5顯示了Grafana的OpenTelemetry集合儀表板。

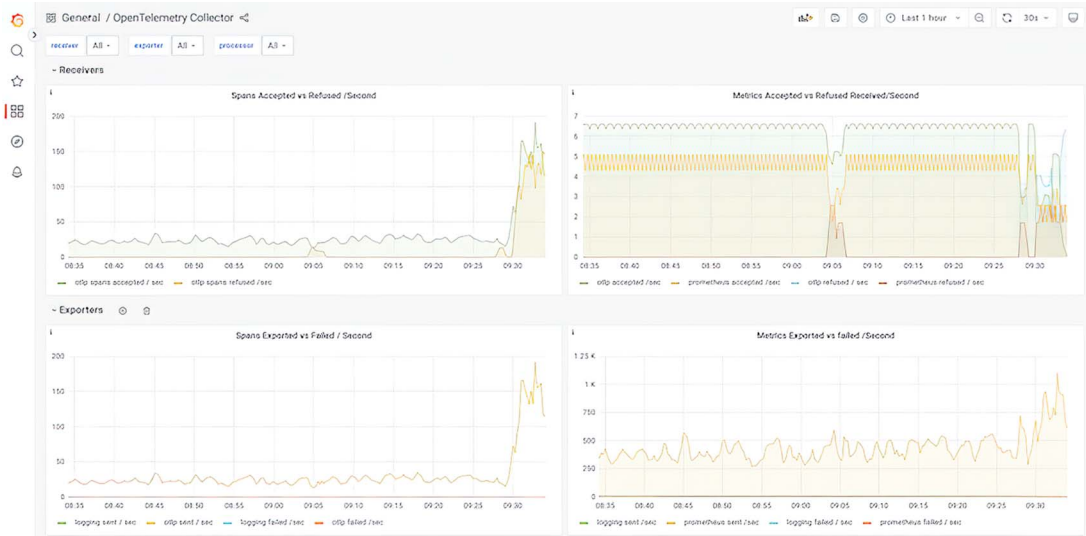
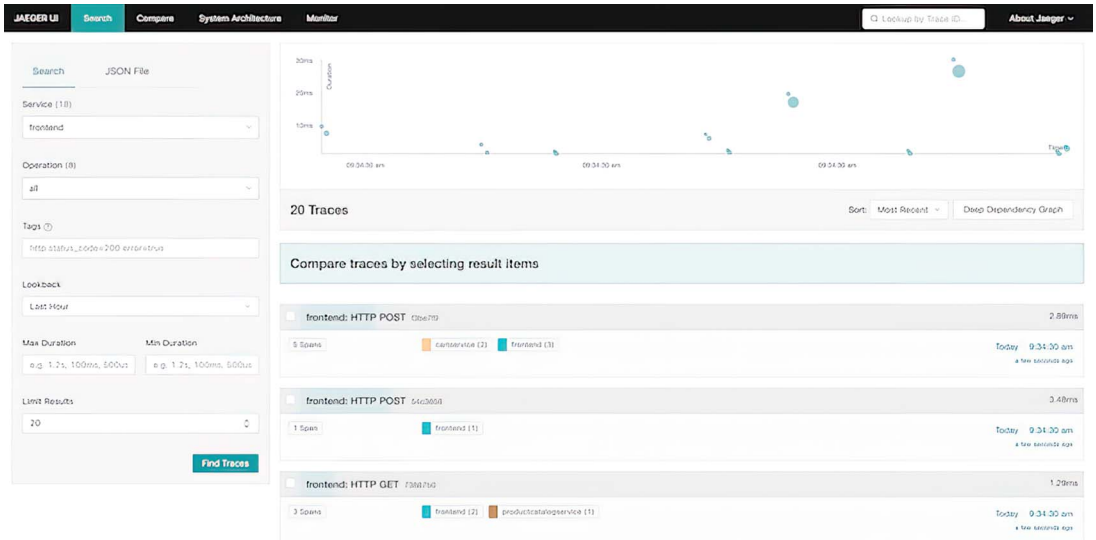


圖 ure a.5 opentelemetry demo grafana的屏幕截圖（收藏家da 滑板）

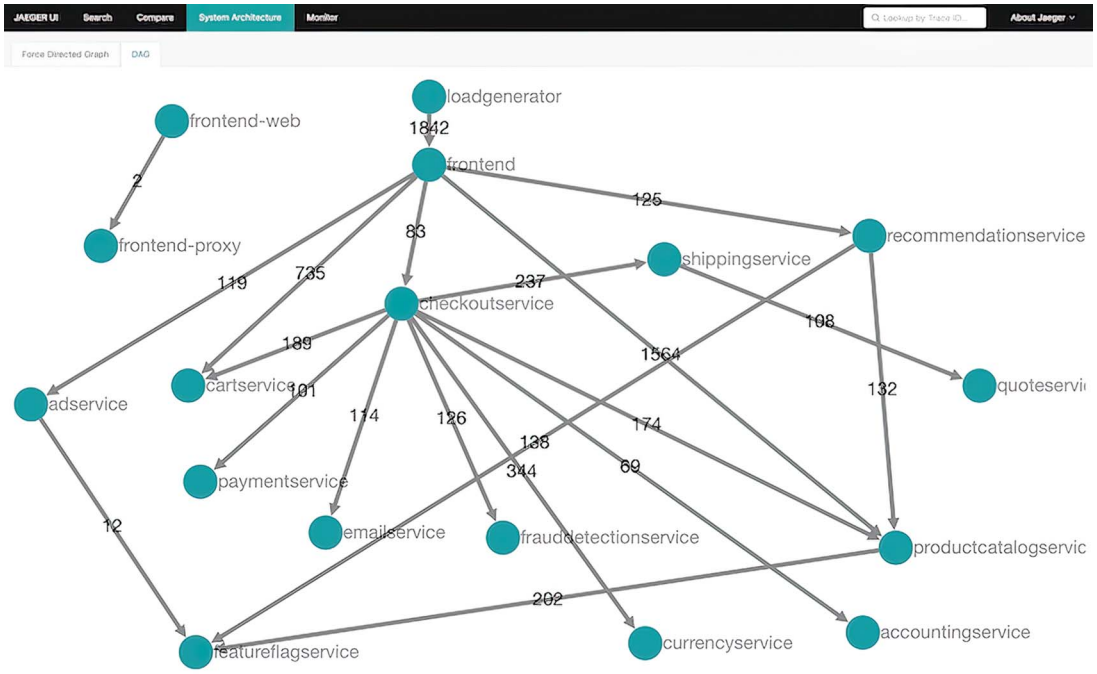
現在，讓我們將注意力轉向痕跡。圖A.6顯示了Jaeger UI，可通過http：// localhost訪問：8080/jaeger/ui/。您在這裡看到的是跨度或跟踪視圖。



圖A.6 OpenTelemetry Demo Jaeger的屏幕截圖（跟踪視圖）

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

圖A.7顯示了Jaeger UI，顯示了系統體系結構視圖中OpenTeleme-嘗試演示應用程序的動態渲染。



圖A.7 OpenTelemetry Demo Jaeger的屏幕截圖（服務地圖視圖）

如果您不想自己設置，則可以通過演示網站（https：// opentelemetry .io/doc/docs/demo/屏幕截圖/）提供更多屏幕截圖。

為了給您一些探索的想法，請考慮使用演示進行以下任何或全部：

在Jaeger中，搜索結帳服務的軌跡，並檢查跟踪。問你自己，大部分時間都需要什麼呼籲？ 現在，搜索付款服務並執行同樣的操作。您是否注意到與結帳服務類似的東西？ 搜索操作Oteldemo.shippingservice/getQuote。 這些痕蹟有什麼共同點？ 搜索FrontendProxy。為什麼您看到的痕跡相對較短？ 在“負載生成器”選項卡中，將用戶數量增加到較大的數字（10x，50x等）。現在，儀表板總體上如何變化？痕量持續時間如何變化？

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

在Opentelemetry項目的背景下，社區也在研究各種演示方案。例如，請查看“使用指標和跡線診斷內存洩漏”方案（http://mng.bz/zqn9），該方案已經在線可用。

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

index

一個

代理模式，收集器部署83代理模式，Prometheus 76代理定義3日誌路由器67–74指標集合74–77概述66–67選擇91-93列表的遙測費用24 AIOPS 145警報疲勞，避免了148個警報144，146-155 –155 –155 –155 –155 –155，MENTRICS 33，MENTRICS 33 33， all-in-one cloud provider offerings 121 frontends 131–139 selecting 139–140 AMA (Azure Monitor agent) 91 Amazon CloudWatch Agent 90 Amazon CloudWatch Logs 97 Amazon CloudWatch Metrics 103 Amazon CodeGuru Profiler 186 Amazon Managed Service for Prometheus 103 Amazon QuickSight 130 Amazon Timestream 103 anomaly detection 42, 145 Apache卡桑德拉110，135 Apache Druid 110 Apache Pinot 110 Apache SkyWalking 136 Apache Software Foundation (ASF) 135 Apache Superset 130 AppDynamics 138 Application Insights (Azure) 106 application-level metrics 27 application logs 67 application logs, Kubernetes 52–54 application performance moni- toring (APM) 108, 137 ASF (Apache Software Foundation) 135 Aspecto 137 asynchronous services 200 attributes, signal correlation 214 audit logs, Kubernetes 54 auto-instrumentation 64, 79 Autometrics 193 autoscaling, serverless compute 56 AWS Distro for OpenTelemetry (ADOT) 90 AWS X-Ray 106 Azure Application Insights Profiler 186 Azure Monitor agent (AMA) 91 Azure Monitor Logs 98 Azure Monitor Metrics 103

b

B2I (business logic to instrumen- tation) ratio 16 backend destinations columnar datastores 108–114 correlation layer 213 costs of 114–115 logs 97–101 metrics 101–106 overview 95–96 selecting 114–118 terminology 96 traces 106–108 vs. front end 95 back pressure, ingesting signals 116 baseline signal value 42 Berkeley Packet Filter （BPF）183 BI（商業智能）130個二進制大型物體（BLOB）60無罪後驗屍，事件管理146商業智能（BI）130商業邏輯與儀器（B2I）比率16商業指標27商業企業相關者，6個業務利益相關者，6個6個

c

Cadvisor 50基數爆炸33，102

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

cardinality issue backends 116 correlation 223 child span, microservice 162 Chronosphere 106 CI/CD pipelines as signal sources 63 ClickHouse 110–114, 136 Cloud Native Computing Foun- dation (CNCF) 5 cloud-native environments 2, 11 cloud operations alerting 146–155 incident management 143–146 overview 142–143 usage tracking 155–156 cloud providers警報解決方案154連續分析186相關221成本跟踪156前端130日誌記錄功能97-99指標功能103-104痕跡106雲提供商。 *See* all-in-one cloud provider offerings Cloud Trace 107 Clymene 105 cold data, logs 25 collectd 91 collectd, metrics agent 76 collectors metrics 74–77 OpenTelemetry 81–90 pipeline requirement 87 columnar datastores backend destinations 108–114 component identifier, collector config 86 compute-related sources basics 46–49 containers 49–51定義45 kubernetes 51–55無服務器計算55–57以容器為中心的無服務器計算56集裝箱49容器運行時接口（CRI）49 containers compute-related sources 49–51 context propagation, traces 163, 170 continuous profiling (CP) cloud provider offerings 186 correlation 216 definition 176 development of 177–179 ISVs 187 open source tooling 184–187 resource usage 187–189 technologies 180–183 corrections of error (COE) 146 correlation definition 212 developer observability 195 frontend destinations 140概述示例12相關性挑戰222-224基本原理212–216實施219–220概述211–212 Cortex 96，104成本跟踪156客戶體驗（CX）144

d

Datadog 138，187，209數據溫度25依賴關係管理容器49-51您的代碼為源，63個目的地3日誌存儲成本為25個指標33個跡線40 *See also* 後端目的地； 前端目的地開發人員定義5可觀察性挑戰194-196概述174–176生產率190 *See also* 連續分析DevOps角色5 Digma 192尺寸，控制33 distributed tracing costs of 171 definition 35, 163 microservices app example 160–163 overview 158–159 sampling 170 terminology 163 troubleshooting microser- vices app 168–169 use cases 164 vs. metrics vs. logs 172 distribution telemetry costs of logs 24 distributions OpenTelemetry 90 traces 41 Docker Compose 20–23 Docker daemon 49–50滴跨度，分佈圖170 dynatrace 187，209

e

EBPF Linux內核功能134，183 Echo Service GO程序15-16 Elasticsearch 99，107，128端到端支持需求，信號相關224最終用戶，跟踪使用155

f

FAAS (function as a service) 56–57 feature velocity 159 filesystems in cloud storage 60 flame graphs 182 Fluent and Fluent Bit 68–71 four golden signals of monitor- ing, metrics 27 frontend destinations all-in-ones 131–139 cloud providers and commer- cial offerings 130 correlation layer 213 costs 139 open source offerings 121–130 overview 120–121 selecting 139–140

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

前端目的地與後端95 Frostdb 110全文索引，日誌線26作為服務（FAAS）56-57

g

game day, incident management 146 gateway mode, collector deployment 83 Google Cloud for SLOs 208 Google Cloud Logging 98 Google Cloud Managed Service for Prometheus 104 Google Cloud Monitoring 104 Google Cloud Profiler 186 Go program 15–16 Go programming language, con- tinuous profiling 179 g:Profiler 187 Grafana 121–129 alerting with 154 correlation example 217–220 Grafana Agent 76 Grafana Labs Loki 100 Grafana Mimir 105 Grafana Tempo 107石墨代理77

h

“快樂路徑”微服務APP示例167–168基於頭部的採樣，分佈式跟踪171個基於頭的基於頭的痕跡抽樣40健康和性能監測，雲操作143-145 HO11Y合成信號生成器165 Honeycomb 138，209 Hot Data，Logs 25，日誌25，日誌25

我

冰柱圖182 icinga 105 incident management, cloud operations 143–146 InfluxDB 105 infrastructure operations, role of 6 ingestion distributed tracing costs 171 of signals into destination 97, 114 Instana 100, 138 instrumentation correlation layer 213 cost assessment 16 distributed tracing costs 171 logs 19–20, 24 metrics 28–29, 32 overview 66–67 traces 36–37, 39 your code as source 64 *See also* OpenTelemetry內部用戶，跟踪使用156

j

Jaeger 88，107組件和示例131相關示例217–220跟踪後端目的地85–86 Java虛擬機（JVMS）48 Journal Cttl 48 JVMS（Java Virtual Machines）48

k

Kibana Frontend 128–129 Kubernetes計算相關資源51–55集裝箱運行時界面49 Pixie 134 Kubernetes環境1日誌記錄21

l

基於標籤的索引，日誌行26基於標籤的方案，指標27 labels, Prometheus 74 Lightstep 138 LinDB 105 load balancers 62 local traces 34 Log4j challenges 68 log event 18 log levels 48 log routers 24 Fluentd and Fluent Bit 68–71 other routers 73 overview 67–68 logs signal type backend destinations 97–101 compute-related sources 47 containers 49 continuous profiling and logging 176 correlation 214–216成本和收益24–26儀器19–20，80 Kubernetes 52–54微服務應用程序8可觀察性26後端的開放標準116概述116概述17-19遙測20–23與TRACE vs. traces 172 logz.io 172

m

M3DB 105 manual instrumentation 80 Metabase 130 metadata, signal correlation 212–213, 222 metric name, Prometheus 74 metrics signal type backend destinations 101–106 collection by agents 74–77 compute-related sources 47 containers 50–51 correlation 214, 216–220 costs and benefits 31–34 instrumentation 28–29 Kubernetes 54微服務應用10

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

指標信號類型*(continued)*打開後端的標準116概述26–28遙測29–31與痕跡172指標信號類型。 *See* 服務級別目標微服務應用程序示例6-10，160–169 Microsoft Power BI 130監視與可觀察性5 MySQL 58

n

NAGIOS 77 NAGIOS CORE 105 NETDATA 105網絡摘要62雲存儲中的網絡文件系統60網絡接口61與網絡相關的源45與網絡相關的商店61-63 New Relic 138，187，287，209 Nightingale 105 Nobl9 Nobl9 Nobl9 Nobl9 Nobl9 Nobl9 Nobl9 208非功能性要求（NFRS），Agent 93 Nosqul DateStores 58 Nosql DataStores 58-58-58-59-59

o

對象存儲60可觀察性1-12挑戰10-12定義2–4對微服務的重要性

159 Microservices App 6-10個角色和目標5–6用例4–5 *See also* 開發人員可觀察性稅，分佈式跟踪171個呼叫角色和模型，Incperent Management 145在線分析處理（OLAP）108 online transaction processing（OLTP) 108 OpenMetrics 217–220 OpenSearch log backends 99 traces backends 107 OpenSearch Dashboard 128–134 Open SLO emergence and standards 209 open source tools considerations in selecting 139 continuous profiling 176, 184–187 frontends 121–130 log backends 99 metrics 104–106 traces 107 OpenTelemetry collector (agent) 81–90 correlation standards 222 correlation with 213–215 instrumentation 64, 79–80 open standards for backends 116 overview 67, 77–78 real user monitoring 155 OpenTelemetry Protocol（OTLP) 81 OpenTSDB 105 Ops Agent, Google 91 orchestrator, container. *See* kubernetes otlp（opentelemetry協議）81

p

PARCA 184，187–189付款方式，無服務器計算56績效代理92自動啟示影響79後端查詢117監視143–145 Pipelines AlertManager 149-154 Opentelemetry收集器81-83 Pixie 134，185平台操作，6個便攜性的作用12 PostgreSQL 58 PPROF 180–181隱私和安全性，信號相關性223 PODFILER 187生產環境，195個產品所有者連續分析，在6個配置文件中連續分析，追踪34個配置文件。 *See* 連續分析Prometheus 74–76警報146–154選擇後端與106相關示例217-220缺乏水平尺度103指標收集74-77 Scrape配置85 SLO實施85 SLO實施202-208 PROPSCALE 105 PROPS PROTER 224 PROPER 224 PROTER 224 PROTER SENMAL SENMAL SENMAL SENMEM 180 PROFUN PROFER SENMAME 180 PROX SENEMA SENMEMY 180 PROX SERMEL semy 180 PROX SENEMA指標方法27，74 Pyroscope 185 Pyrra 203–207

問

查詢後端97，117後端成本115分佈式跟踪成本171概況195 QuestDB 106

r

費率，錯誤，持續時間（紅色）方法，指標27 RCA（根本原因分析）146可及性監控144真實用戶監控（朗姆酒）155記錄規則功能，Prometheus 75

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

relabeling feature, Prometheus 74 relational database manage- ment s ystems (RDBMS) 58−59 release e ngineer, role of 5 resource attribute s, signal correlation 214 resource usage agent 92 continuous profilin g 187−189 retention period, signal s 42 retention period for signal sto rage 114 return on investment 11 R ookout 193 root cause analysis (R CA) 146 root span, microservice 1 62 row-oriented vs. columnar數據庫108–110朗姆酒（真實用戶監視）155

s

抽樣相關223分佈式跟踪163，170痕跡40安全劑91安全性和隱私，信號相關223安全補丁56服務器較少的計算環境55-57服務水平協議（SLAS）198–199，2019-201-201-201-201-201-202基本原理198–202實施202-209概述197-198與SLIS 201–202“左右”概念174信號屬性，信號相關214信號相關。 *See* 相關信號14–42定義2儀器成本評估16日誌8指標10範圍示例15選擇42 *See also* logs信號類型; MET-RICS信號類型Signoz 136單頁應用程序（SPA）155“玻璃的單窗格” 121 Site可靠性工程（SRE）197 SLAS（服務水平協議）198–199，20192 201-202 SLIS（服務水平指標）（服務水平指標）198–202 SLOS。 *See* 服務水平目標懶208雪花110襪店應用6 SolareWinds 100，138來源計算與46-57的46–57定義3個目的地33 Grafana Data 122與網絡相關的商店61-63 Proxy 64選擇45個信號和15個與存儲相關的信號和15個與儲存相關的58-61您的Coper 63 Spans，162 Spans 35-16 35，35-35-35，35 55，（單頁應用程序）155 Splunk 100，138，209 Splunk Enversodon 187 Sprkl 192 SRE（站點可靠性工程）197 Statsd 77 Stdout，記錄至18，49後端的存儲成本114 Signal of Signal的期望97

分佈式跟踪成本171個配置文件194與存儲相關的來源45，58-61 SUI GENEMIS服務200 Sumo Logic 139開發人員的象徵挑戰194同步服務200綜合監視144 System D 48 SystemD 48 System Log s 67 System Logs 67 System Metrics 27 System 27 System Persecting（SUO）44定義2 44定義2

t

Tableau 130 tail-based sampling, distributed tracing 171 tail-based sampling of traces 40 Telegraf 50 , 77 telemetry data volume challen ge for correlation 223 definition 3 in B2I calculation 17 logs 20−24 metrics 29−32 Pixie 134 regulator y requirements for storage 115 tra ces 37−38, 40 *See also* OpenTelem etry test engineer, role of 5 Thano s 105 time series數據庫（TSDB）101–103跟踪上下文163痕跡存儲和檢索40個跟踪信號類型作為記錄18個後端目的地的替換106-108 compute相關資源47相關性214-220

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

跡線信號類型*(continued)*成本和收益39-41儀器36–37 Kubernetes 55可觀察性41後端的開放標準116概述34遙測37-38 Zipkin 135 Zipkin 135 Trace Traces信號類型。*See* 分佈式跟踪Tracetest 193 TSDB（時間序列數據庫）101–103

你

統一警報154 UPTRACE 136用法跟踪，雲操作155-156用戶體驗，信號相關224利用率，飽和，錯誤（使用）方法，指標27

v

矢量開源項目91 VictoriAmetrics 106虛擬機（VMS）47-49虛擬私有云（VPC）63虛擬私人網絡（VPN）63 VPC（虛擬私有云）63 VPN（虛擬專用網絡）63

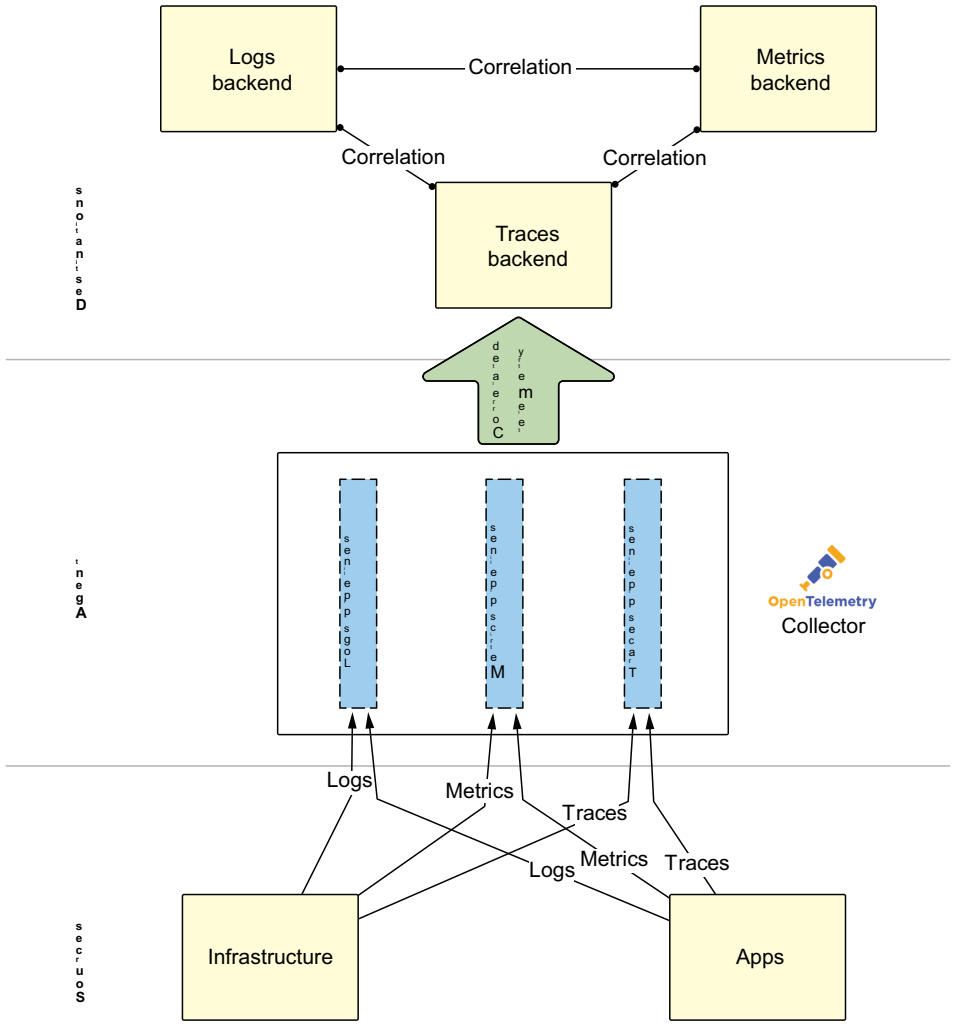
w

暖數據，日誌25瀑布可視化35，162編織工作6

z

Zipkin 107, 135

Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>



Licensed to Huan-Lin Tsai <huanlin.tsai@gmail.com>

CLOUD / DEVOPS

雲可觀察性IN ACTION

邁克爾·豪森布拉斯

C大聲的本地系統由數百個運動部件組成。當出現問題時，不足以知道存在問題 - 您需要知道它在哪裡，它是什麼以及如何fi x。這本書將帶您超越傳統監控，並解釋可觀察性系統，這些系統將應用程序遙測變成可行的見解。

行動中的雲可觀察性為您提供了成功將可觀察性引入基於雲的無服務器和Kubernetes環境中的背景和技術。 在其中，您將學習使用開放式標準和工具，例如開放式遙測，Prometheus和Grafana來構建自己的可觀察性系統，並最終依賴專有軟件。

您會發現來自Diff Erent遙測信號的見解，包括日誌，指標，痕跡和Profi les。另外，這本書的嚴格成本分析可確保您對觀察力投資獲得真正的回報。

裡面有什麼

- 雲本機系統中的可觀察性•儀表板，警報和slos/slis在{

適用於應用程序開發人員，平台所有者，DevOps和SRE。

Michael Hausenblas是AWS開源可觀察性團隊的產品主。

傑米·里德賽爾（Jamie Riedesel）是本書的技術編輯。

對於印刷書籍所有者，所有電子書格式都是免費的：https：//www.manning.com/freebook

“令人難以置信的書！這本書使您能夠達到速度，並使您為未來做好了準備。持續的深度和細節給我留下了深刻的印象，保護您的p ipeline的影響，提供建議，幫助您成為可觀察性的這是優秀者的重要指南。”

—Kit Merker, Google

“將雲原生環境的大量知識與最佳實踐和對可觀察性的見解相結合。必讀。”

—Jonathan Gomez, 為數的雲工程師的丹尼爾·戈麥斯布蘭科（Daniel Gomez Blanco）作證

“令人難以置信的書！這本書使您能夠達到速度，並使您為未來做好了準備。持續的深度和細節給我留下了深刻的印象，保護您的pipeline的影響，提供建議，幫助您成為可觀察性的這是優秀者的重要指南。”

—Ken Finnigan, Lumigo

ISBN-13: 978-1-633-43-969-7

MANNING