CSA GCR cloud security alliance
GREATER CHINA REGION ®

云原生可观测性技术研究与应用

@2023 云安全联盟大中华区－保留所有权利。你可以在你的电脑上下载.储存.展示.查看及打印，或者访问云安全联盟大中华区官网（https://www.c-csa.cn）。须遵守以下：（a）本文只可作个人.信息获取.非商业用途；（b） 本文内容不得篡改；（c）本文不得转发；（d）该商标.版权或其他声明不得删除。在遵循 中华人民共和国著作权法相关条款情况下合理使用本文内容，使用时请注明引用于云安全联盟大中华区。

联盟简介

云安全联盟（Cloud Security Alliance,CSA）是中立、权威的全球性非营利产业组织，于2009年正式成立，致力于定义和提高业界对云计算和下一代数字技术安全最佳实践的认识，推动数字安全产业全面发展。

云安全联盟大中华区（Cloud Security Alliance Greater China Region, CSA GCR）作为CSA全球四大区之一，2016年在香港独立注册，于2021年在中国登记注册，是网络安全领域首家在中国境内注册备案的国际NGO，旨在立足中国，连接全球，推动大中华区数字安全技术标准与产业的发展及国际合作。

我们的工作

联盟会刊下载地址
了解联盟更多信息

加入我们

CSA大中华区官网
(https://c-csa.cn)

点击会员

加入联盟

填写相关申请信息

成为CSA会员

JOIN US

致谢

《云原生可观测性技术研究与应用》由CSA大中华区云原生安全工作组内企业云原生可观测性白皮书项目组专家撰写，感谢以下专家的贡献：

项目组组长：王亮

主要贡献者：

高巍　　陈　磊　　浦　明　　郑　伟　　申屠鹏会

杨文宏　　刘　刚　　李卓嘉　　谢奕智

研究协调员：

罗智杰　　闭俊林

贡献单位：

北京沃东天骏信息技术有限公司　　安易科技(北京)有限公司

北京神州绿盟科技有限公司　　中国工商银行股份有限公司

天翼云科技有限公司

(以上排名不分先后)

关于研究工作组的更多介绍，请在 CSA 大中华区官网（https://c- csa.cn/research/）上查看。

在此感谢以上专家及单位。如此文有不妥当之处，敬请读者联系 CSA GCR 秘书处给与雅正！联系邮箱 research@c-csa.cn；国际云安全联盟 CSA 公众号

© 2023 云安全联盟大中华区版权所有

序言

2018 年，可观测性（Observability）被引入 IT 领域，CNCF-Landscape 率先出现了 Observability 的分组。自此以后，“可观测性”逐渐取代“监控”，成为云原生技术领域最热门的话题之一。Gartner 发布的《2023 年十大战略技术趋势》中，提到了可观测性的发展趋势解读。文中谈到：“可观测性应用使企业机构能够利用他们的数据特征来获得竞争优势。它能够在正确的时间提高正确数据的战略重要性，以便根据明确的数据分析结果采取快速行动。可观测性是一种强大的工具。如果能够在战略中予以规划并成功执行。可观测性应用将成为数据驱动型决策能力建设的最强支撑。”

本白皮书介绍了可观测性在云原生场景的架构设计，阐述在云原生场景使用 eBPF 技术完成可观测性数据采集的技术实现及优势。云原生可观测性的应用场景涵盖了事件根因分析、安全溯源分析等主要内容，旨在阐述云原生可观测性的生产实践。云原生场景业务弹性变化节奏加快、工作负载生命周期缩短、工作负载直接的访问关系更加复杂。在轻量、多变、短生命周期的云原生环境获得足够多的数据，得以对事件根因进行分析，对入侵行为进行溯源，对未知威胁进行预测，是将可观测性能力运用至云原生安全领域后带来的安全能力提升。这也是可观测性对云原生场景安全的价值体现。

希望通过本白皮书为读者深入浅出的介绍云原生可观测性及云原生可观测性对安全能力的赋能。使得广大云原生基础设施运维者、云原生业务使用者能够借鉴本文的评估自身系统云原生可观测性的成熟度及发展战略。

李雨航 Yale Li

CSA 大中华区主席兼研究院院长

© 2023 云安全联盟大中华区版权所有

目录

© 2023 云安全联盟大中华区版权所有

© 2023 云安全联盟大中华区版权所有

1.可观测性概述

1.1 云原生可观测性发展

CNCF 云原生定义对云生技术描述为：在现代动态环境（如公有云、私有云和混合云）中构建和运行可扩展的应用程序的能力。容器、服务网格、微服务、不可变基础设施和声明性 API 均是基于云原生技术的特征。采用云原生技术使松散耦合的系统具有弹性、可管理和可观察性。结合强大的自动化功能，能够以最少的工作量频繁且可预测地进行高影响力的更改。

Pivotal 公司 Matt Stine 在 2013 年首次提出云原生概念。2017 年 Matt Stine 将原生特征归纳为六大点，分别是：

- 模块化 Modularity
- 可观测性 Observability
- 可部署性 Deployability
- 可测试性 Testability
- 可处理性 Disposability
- 可替换性 Replaceability

© 2023 云安全联盟大中华区版权所有

作为六大特征之一的可观测性是保证云原生应用稳定性的基础。在云原生时代，应用规模不断扩大，复杂度愈来愈高，而其中潜藏的问题和风险也随之增多。这对支撑平台及业务自身的稳定性提出更高要求。能够支撑业务的快速迭代、故障快速响应能力、适应复杂的微服务拓扑、保证高效运维。

在数字化大趋势下，云计算成为企业数字化转型的关键。以云上开发为核心的云原生技术得到了广泛的使用。云原生在企业上云和基础实施架构上的大量应用，也对企业的运维监控安全提出了新的挑战。分布式、解耦合的新型系统架构，服务调用链长、系统行为复杂、软件系统稳定性保障困难，解决以上问题需要采用新的方式对系统进行观测。

1.2 可观测性定义

在控制理论中，“可观测性是从系统外部输出的数据衡量系统内部状态的程度”。可观测性是人类对机器可以观察、理解和处理所述系统状态的功能。可观测性是在没有考虑目标的情况下决定系统在实现时应该具有哪些输出。

在 IT 领域，可观测性是在日志与监控指标组成的传统监控基础上，依据由日志、指标、链路追踪三种核心数据来洞悉系统运行状态。通过统一的链路追踪洞察系统服务调用链，并与日志、指标数据联动分析，可实现对云原生系统的高效故障定位与故障解决，保障云原生系统稳定性。

© 2023 云安全联盟大中华区版权所有

可观测性具有三个方面的特征，首先是度量能力，可观测性的度量能力能够帮助使用者在系统处于非常极端复杂的场景时，也能理解和解释系统当前的状态。其次是探索分析，可观测性不应该预定调试/排查模式和路径，而应该能够自由地对所有采集到的各类状态数据在各种维度和组合之间进行关联分析，不断探索分析出新的关联性。最后是按需改变，可观测性最好是在不改变原有代码的情况下，按需进行埋点。

2.云原生可观测性成熟度

研究可观测性成熟度模型的目标是提供一种可衡量、可复制的理论基础用以评估、改进可观测性体系能力。遵循 PDCA 模型通过对可观测性能力持续改进，提高对云原生系统的感知能力，缩短运维过程中寻找根因、排除故障的时间。

衡量和评估云原生系统可观测性的成熟度模型，可定义为如下四个级别：

Level 1 —— 监控（Monitoring）

Level 2 —— 基础可观测性（Basic Observability）

Level 3 —— 因果可观测性（Causal Observability）

Level 4 —— 主动可观测性（Proactive Observability）

可观测性成熟度模型的每个级别，建立在前一级别已实现的基础上。

© 2023 云安全联盟大中华区版权所有

2.1 监控

2.1.1 目标：确定系统组件是否按预期正常工作

可观测性成熟度模型中，监控是第一个阶段。此阶段对资产、进程、资源使用等数据持续采样、度量和记录，获取实时或定期的信息和数据。跟踪单个系统组件的特定参数，度量系统组件状态。系统组件运行状态如超出预设范围，触发警报、状态更改、通知。监控级的目标之一是设置实时警报，在系统出现问题或达到预定阈值时能够及时报警。

2.1.2 能力

在 Level1 阶段，被监控的系统各组件之间无相关性，此级别主要目标是了解系统组件是否正常工作。监控级会开始对基本的性能数据进行采集，以确保系统在负载情况下不会受到显著影响。监控级的主要目标是建立起最基本的监控能力，以确保系统的基本稳定性和可用性。

关键功能：

系统输入：事件和组件级指标

系统输出：报警、日志

价值：获得基本信息，系统组件的健康状态出现问题时的警报和通知

2.2 基础可观测性

© 2023 云安全联盟大中华区版权所有

2.2.1 目标：确定系统故障

可观测性通常指基于对复杂系统外部输出的了解，能够了解其内部状态或状况的程度。系统越可观测，定位问题根本原因的过程就越快速越准确，而无需进行额外的测试或编码。

为保证复杂动态的系统可靠运行，需要知道系统组件是否正常运行，还需要了解它为什么不运行。当出现问题时，希望遵循“5W1H”的原则了解问题详情：who、when、where、what、why、how。

Level1 监控措施基于预置规则实现告警、通知。预置规则依赖于一个关键性的假设，即能够在问题发生之前预测将会遇到的问题类型。这种方法不能覆盖足够多的场景，无法回答 5W1H 的问题。云原生环境是动态的、复杂的、多变的，无法事先预知可能会出现什么样的问题。因此云原生应用下的可观测性方案，应可以根据更完整、更深入的可观测性数据，实时感知事件，并提供定位可能无法预料的问题根因的能力。

可观测性成熟度 Level 2 相较于 Level 1 的数据具有更大的广度和深度。可观测性系统主要关注三种关键类型的数据来提供系统洞察力：指标、日志和跟踪。可观测性的这三大支柱是从微服务、应用程序、数据库、云原生基础设施中收集的，旨在提供对系统行为的更为完整的视图。

每种数据提供不同类型的信息：

© 2023 云安全联盟大中华区版权所有

- **指标：**了解服务性能和状态的数值测量--四个黄金信号：延迟、流量、错误率和饱和度。

- **日志：**了解系统在给定时间点的行为--统中发生的相关事件（例如事务、警告、错误）的时间戳记记录

- **追踪：**解决性能问题--显示数据如何从端到端流经应用程序的详细快照（例如，用户请求）

可观测性强调数据的统一性，通过构建统一的平台来实现指标、日志和跟踪数据的汇聚与处理，突破单点工具的能力限制。建设统一数据平台可将各种可观测性工具整合在一个集中的界面，提高管理和维护系统的效率。

通过手工关联数据来推断事件的可疑原因，需要跨系统手动查询。在 Level 2 中，尚未涉及自动化方法来统一和关联来自各种工具汇聚的孤立数据，准确定位问题的根本原需要大量的人力投入。

2.2.2 能力

Level2 阶段，理解可观测性数据之间的关系，将上下文数据结合。

关键功能：

系统输入：Level1+链路、指标、日志

系统输出：Level1+图标、日志可视化综合仪表盘

© 2023 云安全联盟大中华区版权所有

价值：

通过从更多来源收集数据，更深入、广泛、全面地了解整个系统健康状况，更好地支持问题诊断

除已知故障类型外，还能够发现未知故障模式

从各种类型的数据中获得有益的洞察力--例如，跟踪有助于识别性能瓶颈，指标可提供出色的 KPI，日志可用于查找软件缺陷。

2.3 因果可观测性

2.3.1 目标：确定问题根本原因影响面及规避方法

可观测性的核心价值在于提高排查问题的效率。可观测性工具通过分析数据，定位问题，进一步确认问题的根本性原因（Root Cause）。可观测性体系用于因果判断，可以更深入全面地理解系统的运行和行为，得出系统中事件之间的因果关系。通过对因果关系分析理解，找出引发问题的根本性原因。具备因果分析能力的可观测性体系可定义为“因果可观测性（Causal Observability）”。具备因果分析能力的可观测性体系，通过收集、分析和解析足够多维度的数据，达到理解系统内事件、状态变化之间的关系，从而更深入地洞察系统运行状态。因果可观测性强调在数据分析中寻找因果关系，并将这些关系转化为对系统事件之间关系的可视化呈现。

© 2023 云安全联盟大中华区版权所有

因果可观测性与基础观测性有所不同，Level2 强调数据，Level3 强调关系。Level2 基础可观测性关注收集、分析数据以理解系统的状态和行为，Level3 因果可观测性强调数据与数据、实体与实体、事件与事件或更多维度数据之间的联系。构建因果可观测性，涉及数据收集、关系收集、数据处理、关系处理、因果推断等步骤，以揭示事件发生的因果过程。面对复杂性、不确定性和变化性的云原生应用场景，对事件因果的理解有助于更好地预测、解释、优化和管理系统。

调查故障根因时，需收集事件发生的时间、空间、参数变化等数据，从而了解导致问题出现、传播，以及随着时间的推移而变化的事件状态。解决这些问题，需要引入新的能力：网络数据、拓扑数据、时间、空间地图、自动化关联技术。这些能力可以更全面地理解系统运行状态，定位问题的根本原因。

为了建立因果可观测性，需要补充两种类型的数据要素：拓扑、时间。

拓扑	系统中各实体对象相互之间的连接关系（例如根据链路相关数据绘制的服务拓扑）
时间	持续抓取观测数据的时间维度

表 1 两种类型数据要素

拓扑是因果可观测性的第一个必要维度。 拓扑是 IT 环境中所有组件的映射，它跨越所有层，从网络到应用程序再到存储，显示一切是如何相关的。 拓扑结合了组件之间的逻辑依赖性、物理接近性和其他关系，以提供人类可读的可视化和可操作的关系数据。

© 2023 云安全联盟大中华区版权所有

拓扑信息（Topology）指的是系统中各主机、进程、容器、API、Service 之间的关系和连接方式。拓扑的价值在于提供系统的高级视图，帮助运维者理解不同实体之间的依赖关系、通信路径和层次结构。通过拓扑信息，能够更全面清晰地了解系统结构。

拓扑信息在可观测性数据中扮演着一种定位和上下文的角色，辅助理解数据所涉及的组件、服务、资源以及它们之间的关系。拓扑信息是一个系统的结构图，展示了系统内部各个元素之间的连接和依赖关系。这种结构图可以是物理上的（如网络拓扑、主机之间的连接），也可以是逻辑上的（如服务之间的依赖关系、数据流动路径）。将信息点连接至系统元素，使得每个维度的数据不是孤立的点。

时间是因果可观测性的第二个必要维度。在充满微服务、云资源和容器不断变化的动态环境中，拓扑信息的变化是非常迅速的。系统状态可能在问题多次出现的过程中发生变化。为了确立因果可观测性，需要引入一个至关重要的维度：时间。

为了深入了解现代 IT 环境的动态行为并获取实现因果可观测性所需的上下文。随着时间的推移，捕获拓扑信息的变化，并将其与可观测性数据进行关联，以跟踪整个堆栈的变化。当出现问题时，可以回溯到问题开始的确切时间点，并查看是什么变化导致了这个问题。通过这种时间维度的关联，能够更准确地定位问题的根本原因，实现对问题的全面分析和解决。

© 2023 云安全联盟大中华区版权所有

空间地图是拓扑的升维，提供 IT 环境中所有元素的关系映射，空间地图是一张三维的元素连接拓扑，涵盖水平的实体关系拓扑，垂直的依赖关系拓扑。空间地图结合了组件之间的逻辑依赖性、物理邻近性和其他关系，以提供可读的可视化和可操作的关系数据。

- 水平拓扑：在相同类型的元素之间建立的连接关系地图，如进程到进程、服务到服务、主机到主机
- 垂直拓扑：在不同类型的元素之间建立的连接关系地图，如数据中心到主机、主机到进程、进程到服务、服务到应用程序

通过技术实现水平层、垂直层的空间地图，自动化、实时性的绘制，将可观测性数据与空间地图的实体关联，拉动时间轴，展示随时间变化的空间拓扑、关联数据，能够比较变化前后的系统状态，是可观测性能力的集中式成果体现。

2.3.2 能力

拓扑可以极大地提高因果判定的准确性和有效性，Level 3 对空间地图的引入是重大的进步。

关键功能：

通过拓扑为指标、链路、流量、日志提供上下文，随事件推移关联相关数据，追踪变化；

系统输入：Level1+Level2+网络+拓扑+时间

© 2023 云安全联盟大中华区版权所有

系统输出：Level1+Level2+空间拓扑+数据关联+时序变化

价值：

通过统一时间序列拓扑中的孤立数据，获得统一、清晰、相关的环境状态上下文视图

通过拓扑可视化和分析了解因果关系，显着加快根本原因识别和解决时间

基本自动化调查的基础，例如根本原因分析、业务影响分析和警报关联

自动将与同一根本原因相关的警报集中在一起，从而减少噪音和干扰所需的上下文

2.4 主动可观测性

2.4.1 目标：自动输出问题根因、自动化响应，智能预测、主动预防

Level 4 主动可观测性，典型特征是引入“AIOps”技术，通过自动化和智能化的方法实现更深入更准确的洞察力，将传统的被动式运维转变为主动式运维。将人工智能（AI） 和机器学习（ML） 技术融入到可观测性体系中。在这一阶段中，更加强调分析结果和答案，而不仅仅关注原始数据和分析过程。主动可观测性将分析结果呈现给用户，并辅助决策和采取行动。

© 2023 云安全联盟大中华区版权所有

Level4 主动可观测性建立在该成熟度模型中因果可观测级别的核心功能之上。Level4 阶段添加了模式识别、异常检测和更准确的问题修复建议。 对于主动可观测性而言，因果可观测性是一个必要的基础：时间序列拓扑提供了一个必要的框架。

数据是原材料，原材料经过分析之后得出的答案。快速的把高质量结果和答案传达出来，快速做出决策、采取行动，是主动可观测性需要重点考虑的问题。任何一套可观测性解决方案或建设可观测性体系之前，都应解决三个最本质的问题：

- **通知：**在问题出现并造成影响前后，能够在多快的时间内得到通知？
- **诊断：**能迅速地对问题进行分类，了解影响？
- **理解：**如何找到潜在的原因以快速解决问题？

AIOps 利用因果关系，使用基于拓扑驱动的渐进式故障树分析算法。AIOps 具备强大的拓扑绘制能力，实时获得基础架构、流程和服务依赖关系的可视化关系，更好地理解系统的运行情况和交互关系。AIOps 依托拓扑地图、云原生系统产生的数据进行分析，能显示一个事件在垂直和水平方向的拓扑依赖关系，能够自动确定异常问题的根源。

© 2023 云安全联盟大中华区版权所有

在 Level 4 阶段，目标是关注更高效且无事故的 IT 运维。可观测系统基于 AI/ ML 算法模型，感知服务或组件偏离正常行为的趋势，并在出现故障之前采取措施规避问题。对未发生事件的预测能力是 Level4 阶段可观测系统的典型特征。

2.4.2 能力

Level 4，目前是 IT 技术领域中最高的可观测性成熟度级别。需要拓扑和时间提供的额外上下文，准确评估根本原因、确定业务影响、检测异常并主动确定何时提醒 SRE 和 DevOps 团队。

关键功能：

系统输入：Level1+Level2+Level3 + 大数据分析/ML 模型

系统输出：Level1+Level2+Level3+空间拓扑+自动化 RCA+预测

价值：自动根因定位、自动化 RCA、告警降噪、预测异常；

借用 Gartner® 2022 年 3 月“Innovation Insight for Observability”文章中的一段话作为本章总结：“发现异常很容易，因为它们一直都在发生。 当每天收集十亿个事件时，每两分钟就会发生一百万分之一的事件。 可观测性工具的关键是发现与手头问题相关的异常，然后从可能相关的日志文件/指标中链接其他信息。 通过在上下文中显示相关信息，操作员可以更快地找出问题的潜在根本原因。”– Gartner®“Innovation Insight for Observability”，2022 Mar，Padraig Byrne & Josh Chessman.

© 2023 云安全联盟大中华区版权所有

3.云原生观测体系能力构建

3.1 云原生可观测性信号

云原生可观测性的实现依赖于监控指标 / 监控度量（Metrics）、事件日志（Logs）和链路追踪（Traces）三种数据类型。这三种数据各有侧重，不是完全独立，三种数据之间有重合或者可以结合之处。

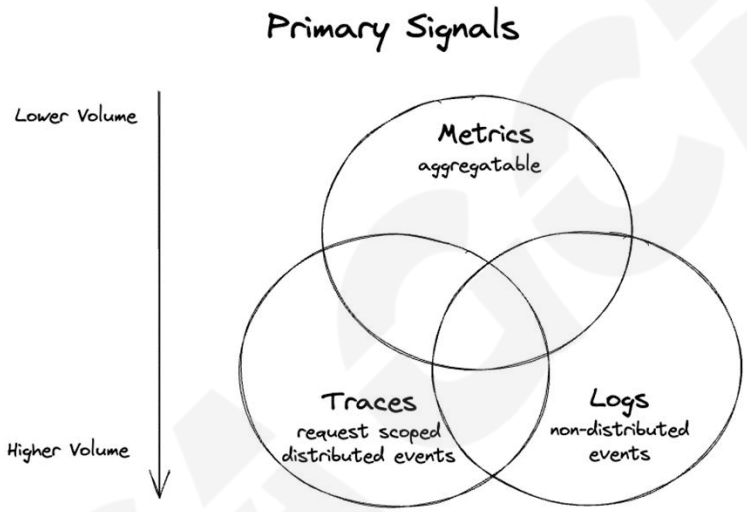


图 1 三种数据类型

3.1.1 指标

© 2023 云安全联盟大中华区版权所有

指标是数据的数字表示形式。它们分为两大类：已经是数字数据和提炼（聚合）成数字的数据。前者的一个例子是温度，后者例如在 Web 服务器上观察到的 HTTP 请求计数器。数字是存储数据的最有效方式，随着时间的推移，所有成熟的行业都趋向于指标优先。指标的典型示例用例 - 衡量主机（例如虚拟机）堆内存使用情况的仪表。我们称之为“堆内存字节”。指标由一个名称、一组标签（有时称为属性或标签）和每个时间点的数值（例如每秒一个值）组成。每个具有特定名称和标签的指标实例通常称为“时间序列”或“流”。

3.1.2 日志

事件日志（Logging）：日志的职责是记录离散事件，应用运行过程会持续输出日志数据，这些日志数据是业务系统运行状态的各种事件及业务处理逻辑时输出的，通过这些记录事后分析出程序的行为，基本上可以还原业务流程处理的全过程。事件日志可以详细解释系统的运行状态，但是存储和查询需要消耗大量的资源。

日志是描述操作系统、应用程序、服务器或其他设备中的使用模式、活动和操作的一个或多个文本条目。

日志可以分为不同的类别，例如：

- 应用程序日志 -应用程序内部事件创建的记录。日志可帮助开发人员了解和评估应用程序在运行过程中的行为模式。

© 2023 云安全联盟大中华区版权所有

- 安全日志 – 事件与系统中检测规则匹配后创建对该事件的记录。包括登录失败、密码更改、资源访问、资源更改、策略更改、发生时间。安全管理员通过安全日志可回溯与检测规则匹配的事件。

- 系统日志 - 系统日志记录操作系统中的事件，包括处理物理和逻辑设备的内核级消息、引导顺序、用户或应用程序身份验证以及其他活动、故障、资源状态消息。

- 审核日志 -事件和更改的记录。记录执行活动人员、执行活动内容以及系统如何响应。系统管理员根据业务需求确定审核日志收集内容。安全管理员可根据审核日志回溯人员对系统的使用。

- 基础架构日志 -涉及管理影响组织 IT 基础的物理和逻辑设备。在本地或云中通过 API、Syslog、主机代理收集获取。

日志记录应用程序或系统在发生故障时的状态,在执行根本原因分析时，这些记录特别有价值。存储在日志中的信息是自由格式的文本,因此很难得出含义。在过去的 30 年中，已经进行了许多尝试将架构应用于日志，但它们尚未特别成功。使用统一架构的原因使提取相关信息更易于访问。通常是通过解析、分段和分析日志文件中的文本来完成的。日志数据还可以转换为其他可观测性信号，指标和跟踪。一旦数据成为指标，它就可以用于了解随时间的变化。日志数据还可以通过日志分析技术进行可视化和分析。

基于对数据获取不同颗粒度，日志可设置不同级别:“错误”、“警告”、“信息”和“调试”。错误是最不详细的日志级别，调试是最详细的日志级别。

© 2023 云安全联盟大中华区版权所有

- **错误：**传达发生情况并详细说明发生故障的原因
- **警告：**需要注意的高级消息，不一定是失败信息
- **消息：**系统的工作状态
- **调试：**存储每个操作的详细信息。通常，仅在故障排除期间或具备充足得存储或性能而短期使用。

3.1.3 追踪

链路追踪（Tracing）：链路追踪大都是依据谷歌在 2010 年发表的论文《Dapper : a Large-Scale Distributed Systems Tracing Infrastructure》 Dapper 理论来实现的，调用链记录的是串联单个事务内全过程的日志数据，通过对请求打标、透传、串联，最终可以还原出一次完整的请求，可以帮助工程师轻松分析出请求中异常点。但是链路追踪和事件日志一样有着相同的问题就是资源消耗较大，通常也需要通过采样的方式减少数据量。

为了有效地进行分布式追踪，Dapper 提出了追踪（Trace）与跨度（Span）两个概念：

- 追踪（Trace）：从客户端发起请求抵达系统的边界开始，记录请求流经的每一个服务，直到到向客户端返回响应为止，这整个过程就称为一次追踪。

© 2023 云安全联盟大中华区版权所有

- 跨度（Span）：由于每次 Trace 都可能会调用数量不定、坐标不定的多个服务，为了能够记录具体调用了哪些服务，以及调用的顺序、开始时点、执行时长等信息，每次开始调用服务前都要先埋入一个调用记录，这个记录称为一个跨度。

Dapper 使用以跨度为基本树节点的跟踪树构建跟踪模型，并为每个跨度记录了一个可读的 span name, span id, parent id，这样就能重建出一次分布式跟踪过程中不同跨度之间的关系。没有 parent id 的跨度被称为根跨度。一次特定跟踪的所有相关跨度会共享同一个通用的 trace id 。换句话说每一个追踪实际上都是由若干个有顺序、有层级关系的跨度所组成一颗追踪树（Trace Tree），如下图 2 所示：

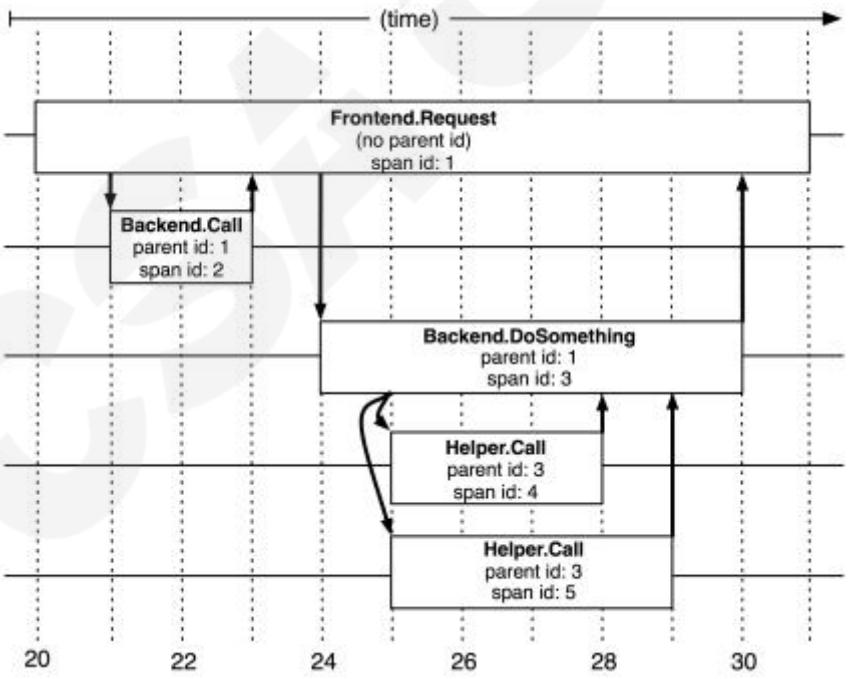


图 2 追踪树[1]

[1] 图片引用自 Google Dapper : a Large-Scale Distributed Systems Tracing Infrastructure

© 2023 云安全联盟大中华区版权所有

追踪（Tracing）通常表示一个具体的事务实例，即计算机通过特定程序的路径，使它们成为可观察性中的详细且昂贵的信号。跨度（Spans）高度情境化，需要进行上下文关联。除此之外，跨度（Spans） 还记录有关启动它的“父”跨度（Spans）的信息。这使得在分布式系统的不同参与者（如服务、队列、数据库等）之间建立因果关系成为可能。

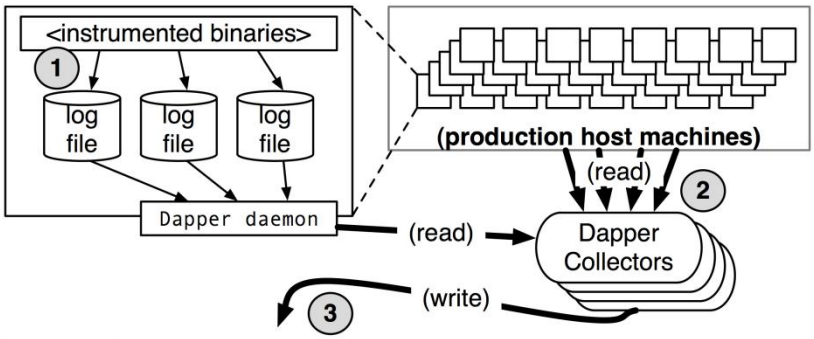
3.1.3.1 基于日志的追踪数据收集

基于日志的追踪的思路是将 Trace、Span 等信息直接输出到应用日志中，然后根据收集到的日志数据，从全局日志信息中反推出完整的调用链拓扑关系。基于日志的追踪数据收集的优点在于其对网络消息完全没有侵入性，对应用程序只有很少量的侵入性，对性能影响也非常低。但其缺点是直接依赖于日志归集过程，日志本身不追求绝对的连续与一致，这也使得基于日志的追踪往往不甚精准。另外，业务服务的调用与日志的归集并不是同时完成的，也通常不由同一个进程完成，有可能发生业务调用已经顺利结束了，但由于日志归集不及时或者精度丢失，导致日志出现延迟或缺失记录，进而产生追踪失真。

Dapper 的跟踪记录和收集管道实现如下图所示，共分为三个阶段：

1. 把 Span 数据写入到本地日志文件
2. Dapper 守护进程和采集器从主机中将日志他们拉取出来
3. 将拉取出的日志写入 Dapper 的 Bigtable 仓库中，其中 Bigtable 中的行表示一次跟踪，列表示一个 span。

© 2023 云安全联盟大中华区版权所有



trace id	span 12	span 23	span 34	span 45	span 56	...
123456	*nil*	*nil*	**<data>**	**<data>**	*nil*	...
246802	**<data>**	*nil*	*nil*	*nil*	**<data>**	...
357913	*nil*	**<data>**	*nil*	*nil*	*nil*	...
...	...	...	...	...	...	...

图 3 Dapper 工作流程[2]

3.1.3.2 基于服务的追踪数据收集

基于服务追踪的实现思路是通过某些手段给目标应用注入追踪探针（Probe），通过探针获取服务调用信息并发送给链路追踪系统。

探针在结构上可视为一个寄生在目标服务身上的小型微服务系统，它一般会有自己专用的服务注册、心跳检测等功能，有专门的数据收集协议，把从目标系统中监控得到的服务调用信息，通过另一次独立的 HTTP 或者 RPC 请求发送给追踪系统。

基于服务的链路追踪劣势在于比基于日志的追踪消耗更多的资源，也有更强的侵入性。基于服务的链路追踪优势为这些资源消耗换来收益的直接结果是精确性与稳定性均有所保证，从而不必再依赖日志归集作为追踪数据的来源。

基于服务的追踪被 Zipkin、SkyWalking、Pinpoint 等追踪系统广泛采用。

[2] 图片引用自 Google Dapper : a Large-Scale Distributed Systems Tracing Infrastructure

© 2023 云安全联盟大中华区版权所有

3.2 云原生可观测能力构建

3.2.1 信号关联

有两种基本方法可以关联数据：运维人员建立或者利用现有数据建立相关性。应该对所有的信号使用相同的元数据结构。尽可能对日志使用相同的标签。如有必要，运维人员可自建标签，附加到所有类型信号的数据。

连续收集所有可观测性信号，每条数据的范围都标记为某个时间戳。在不同的维度上，每个可观测性信号都需要绑定到某个“目标”， 能够查看目标的指标、跟踪和日志三个维度可观测性数据。

通过一致的元数据来切换来自同一目标（例如，同一应用程序）的可观测性信号。利用基于拉取的系统，如 Prometheus 或 OpenTelemetry Prometheus 接收指标，利用日志收集器（OpenTelemetry、Fluentd、Fluentbit ）收集日志。确保收集器附加一组一致的目标标签或属性，例如“集群”、“namespace”、”label“、“Pod”。在处理 OTLP（指标、日志和跟踪）等多维度集合数据时，应用附加目标标签信息，确保数据一致性。

运维人员可以基于相同的标签，在不同的可观测性信号之间切换。允许从每个信号中选择与特定过程、组件、时间相关的信息，从而快速浏览具有相同标签信号的细节内容。

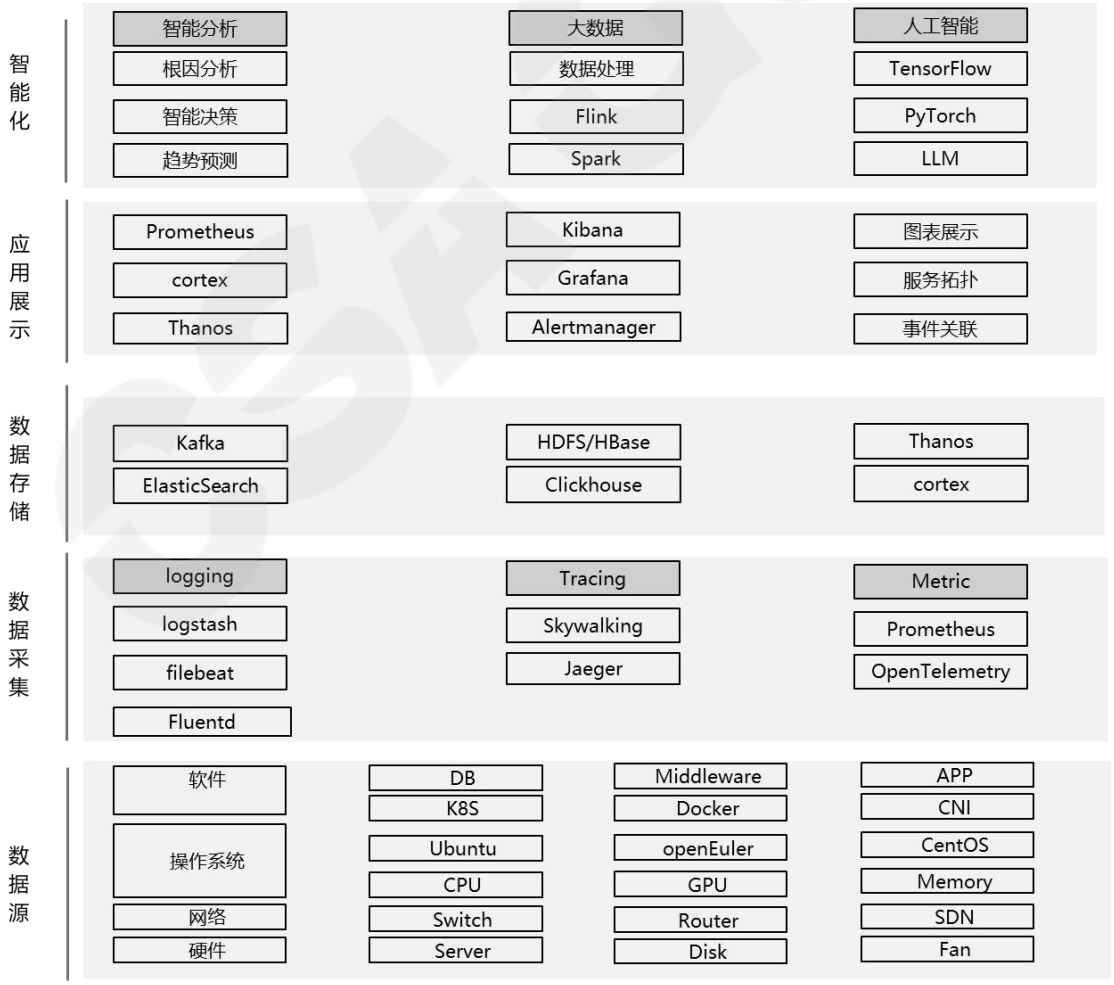
© 2023 云安全联盟大中华区版权所有

日志中共享与请求相关的相同信息（请求 ID、操作 ID）。确保日志记录和跟踪的这些 ID 是相关联的，从而确保数据可基于事件请求 ID 紧密地相互关联。运维人员能够通过跟踪绑定到单条日志的请求 ID 标签，快速定位日志。

3.2.2 典型业务架构

可观测性业务架构可分为以下五层（数据源、数据采集层、数据存储层、应用展示层、智能化层）。五层架构可作为可观测性建设的内容，也是当前大多数企业正在实践的部分。

典型可观测性业务架构如下图 4 所示：



© 2023 云安全联盟大中华区版权所有

图 4 典型可观测性业务架构

3.2.2.1 数据源

可观测的数据源大致可分为几大类，硬件、操作系统与网络以及软件。硬件如服务器的硬盘、电源、风扇和主板等；网络设备如交换机、路由器和网关等设备；安全设备如防火墙；机房设施如空调、电力等；这些硬件在运行时都会产生状态数据。操作系统其实也是一种软件，由于其特殊性这里将其单独归类，操作系统内会有 CPU、内存、存储相关的使用和状态数据，操作系统间会有通信的网络数据。软件包含集群编排系统、Docker 运行时、DB、中间件、业务软件。

3.2.2.2 数据采集

对数据进行分析总结出三种独立的数据类型，分别从三个不同的维度来展示应用的状态，这三种数据类型分别是日志数据（Logging）、链路数据（Tracing）和指标数据（Metric）

日志是记录了发生在运行中的操作系统或其他软件中的事件。常见于事件日志、事物日志、消息日志等，而与可观测性相关主要就是事件日志。事件日志（Event logs）记录了在系统运行期间发生的事件，以便于了解系统活动和诊断问题。它对于了解复杂系统的活动轨迹至关重要，尤其是只有很少用户交互的应用程序（例如服务器应用程序）。

目前，许多企业使用 CNCF 推荐的 Fluentd 作为主要的日志数据采集工具。此外还有一些企业采用 Filebeat 和 Logstash。

© 2023 云安全联盟大中华区版权所有

链路追踪即调用链监控，特点是通过记录多个在请求间跨服务完成的逻辑请求信息，帮助开发人员优化性能和进行问题追踪。链路追踪可以捕获每个请求遇到的异常和错误，即时信息和有价值的数据。链路追踪的技术选型目前在 CNCF 中主要有的 Jaeger，SkyWalking 或 Zipkin 几种工具供选择。

指标数据是应用程序运行时产生的内部指标，以 API 接口的方式提供查询。指标数据具有时间点的特性，不同的时间点对应的指标是不同的，因此用以存储指标数据的数据库一般称为时序列数据库。

3.2.2.3 数据存储

采集到的数据需要进行处理并进行存储，为下一步的数据应用和展示提供数据基础。一般场景日志数据和链路追踪数据存储可使用 ElasticSearch、ClickHouse 等非关系数据库。在大规模日志采集场景下可以添加 Kafka 作为缓冲。对需要进行大数据分析等场景时，也可以选择 HDFS/HBase 存储。

对于指标数据推荐使用 Prometheus 存储（Prometheus 本身也实现了 TSDB 数据库），但是原生的 TSDB 对于大数据量的保存及查询支持不太友好，该数据库不能保证可靠性，且无法支持 Prometheus 集群架构。而 Thanos 和 Cortex 都是在数据可靠性和集群高可用方面进行了优化和增强，目前都是 CNCF 孵化中的项目，也是不错的选择。在大规模场景下还可以选择 openTSDB 或 Clickhouse 来进行指标数据存储。

3.2.2.4 数据展示

© 2023 云安全联盟大中华区版权所有

展示层是对采集数据的基础应用，也是当前企业主要的应用场景。图表展示是可观测性面向用户最为直观的呈现，将复杂的数据以图或表形式展示出来，便于运维人员快速了解应用状态，基于经验做出判断或预测。对于日志数据和链路追踪数据的查看可以通过 Kibana 查看，对于指标数据可使用 Grafana 进行展示，也可以使用原生的 Prometheus、Thanos 或 Cortex 查看。

服务拓扑是通过数据流向和调用关系，以 UI 的方式将服务依赖关系拓扑呈现出来。实际业务中，应用之间的关联与依赖非常复杂，需要通过全局视角检查具体的局部异常。可以在服务拓扑查看应用在指定时间内的调用及其性能状况。

监控告警是最常用的场景，也是目前建设可观测系统的核心目标。监控告警通过事前配置好阈值，数据采集上来后通过计算与阈值比对，对于不符合规则要求的数据生成告警事件，通过告警渠道发送到目标设备。

对于可观测性数据的应用除以上几项外，还可尝试将三者的数据进行关联，使同一个应用不同维度的事件立体化的展示出来。如请求发生异常时，应用一般会将请求以日志的方式输出，调用链路也会上报调用异常，这两类数据可以通过 RequestID 或 TraceID 进行关联。

3.2.2.5 智能化

将人工智能和大数据应用于可观测性，辅助运维实现自动化、智能化，以达到业务服务高效稳定运行的目的。

© 2023 云安全联盟大中华区版权所有

智能分析依托大数据和人工智能技术对采集的日志、指标和链路数据进行分析，按需产生有价值的结果。智能分析的主要应用有趋势预测、根因分析和智能决策三类不同场景。

- 趋势预测：预测可能会发生的事件；
- 根因分析：确定事件发生的根本原因；
- 智能决策：基于根本原因后提供智能决策的解决方案。

3.3 核心能力-基于 eBPF 构建云原生数据采集技术

随着大量应用基于云原生技术进行开发、适配和迁移，应用出现微服务激增、多语言开发、多通信协议特征，系统架构复杂度越来越高，传统可观测方法存在采集数据粒度不够细致、资源消耗量大、外部代码侵入等问题。近年来，业界开始关注 linux 内核 eBPF 技术，独有特性能更有效地解决发展中问题。

3.3.1 Linux 内核 eBPF 技术原理

eBPF 起源于 Linux 内核，可以运行沙箱程序，安全有效地扩展内核功能，无需更改内核源代码或加载内核模块。eBPF 全称“扩展的伯克利数据包过滤器(Extended Berkeley Packet Filter)”，是一种数据包过滤技术，从 BPF(Berkeley Packet Filter)技术扩展而来。BPF 提供了一种在内核事件和用户程序事件发生时安全注入代码的机制，这就让非内核开发人员也可以对内核进行控制。随着 Linux 内核的发展，BPF 逐步从最初的数据包过滤扩展到网络、内核、安全、跟踪等，而且它的功能特性还在快速发展之中，这种扩展后的 BPF 被简称为 eBPF。

© 2023 云安全联盟大中华区版权所有

eBPF 是基于寄存器的虚拟机，使用自定义的 64 位 RISC 指令集，在 Linux 内核运行即时本地编译的 eBPF 程序。eBPF 程序并不像常规的线程那样，启动后就一直保持运行，它需要由事件触发后才会执行。这些事件包括系统调用、内核跟踪点、内核函数和用户态函数的调用退出、网络事件，等等。借助于强大的内核态插桩（kprobe）和用户态插桩（uprobe），eBPF 程序几乎可以在内核和应用的任意位置进行插桩。eBPF 程序通常包括 4 部分：

- 后端代码：在内核中加载和运行的 eBPF 字节码，它将数据写入内核 map 和环形缓冲区的数据结构中；
- 加载器：它将字节码后端加载到内核中，当加载器进程中止时，字节码会被内核自动卸载;
- 前端代码：从数据结构中读取数据，并将其显示给用户
- 数据结构：后端和前端的通信手段。它们是由内核管理的 map 和环形缓冲区，可以通过文件描述符访问。需要在后端被加载之前创建，数据会持续存在，直到没有更多的后端或前端进行读写操作

eBPF 程序的运行流程，如下图 5 所示：

© 2023 云安全联盟大中华区版权所有

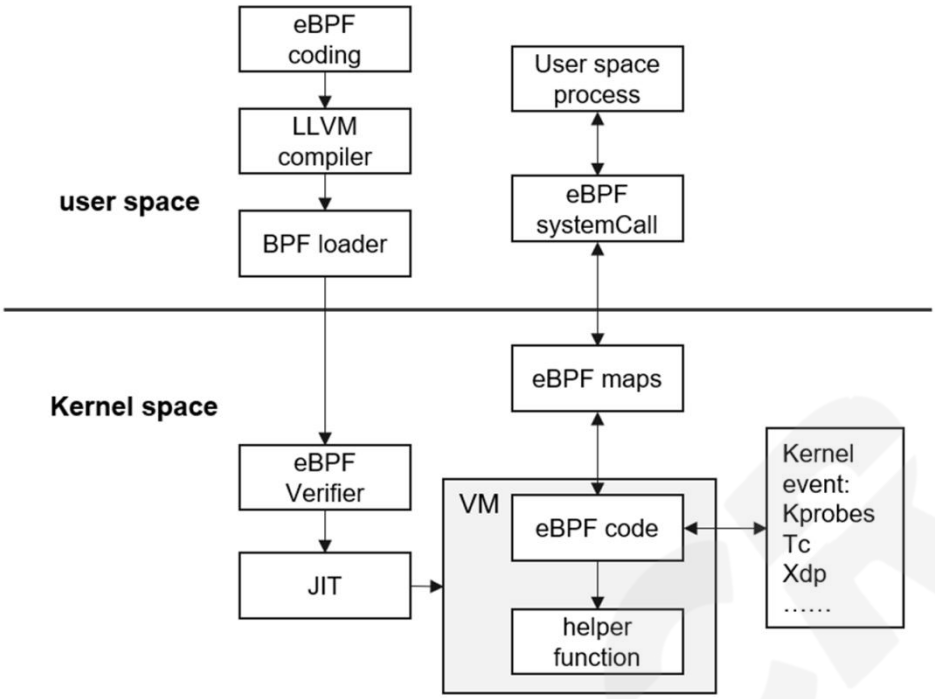


图 5 eBPF 程序的运行流程

1.用户态编写 eBPF 程序

2.使用 LLVM 编译成 bytecode 的 ELF 文件

3.使用 bpf 系统调用，把程序加载进入内核

4.内核的 verifier 会验证 eBPF 程序的合法性，确保其能够安全、合规地在内核中运行

5.内核会使用 JIT compiler 把 eBPF 字节码编译成本地机器码

6.eBPF 程序在内核中以 VM 方式安全运行

3.3.2 基于 eBPF 云原生可观测性技术架构

© 2023 云安全联盟大中华区版权所有

云原生环境中每台机器(或虚拟机)只有一个内核，所有运行在该机器上的容器都共享同一个内核，内核了解主机上运行的所有应用代码。通过对内核的检测，基于 eBPF 可以同时检测在该机器上运行的所有应用程序代码。当将 eBPF 程序加载到内核并将其附加到事件上时，它就会被触发，而不考虑哪个进程与该事件有关。eBPF 能够对基于广泛的可能来源的可视化事件的定制指标进行收集和核内聚合。

基于 eBPF 在操作系统内核特性优势，与 OpenTelemetry 结合实现云原生观测数据收集处理的架构，极大增强云原生环境可观测性能力。基于 eBPF 的可观测性架构如下图 6：

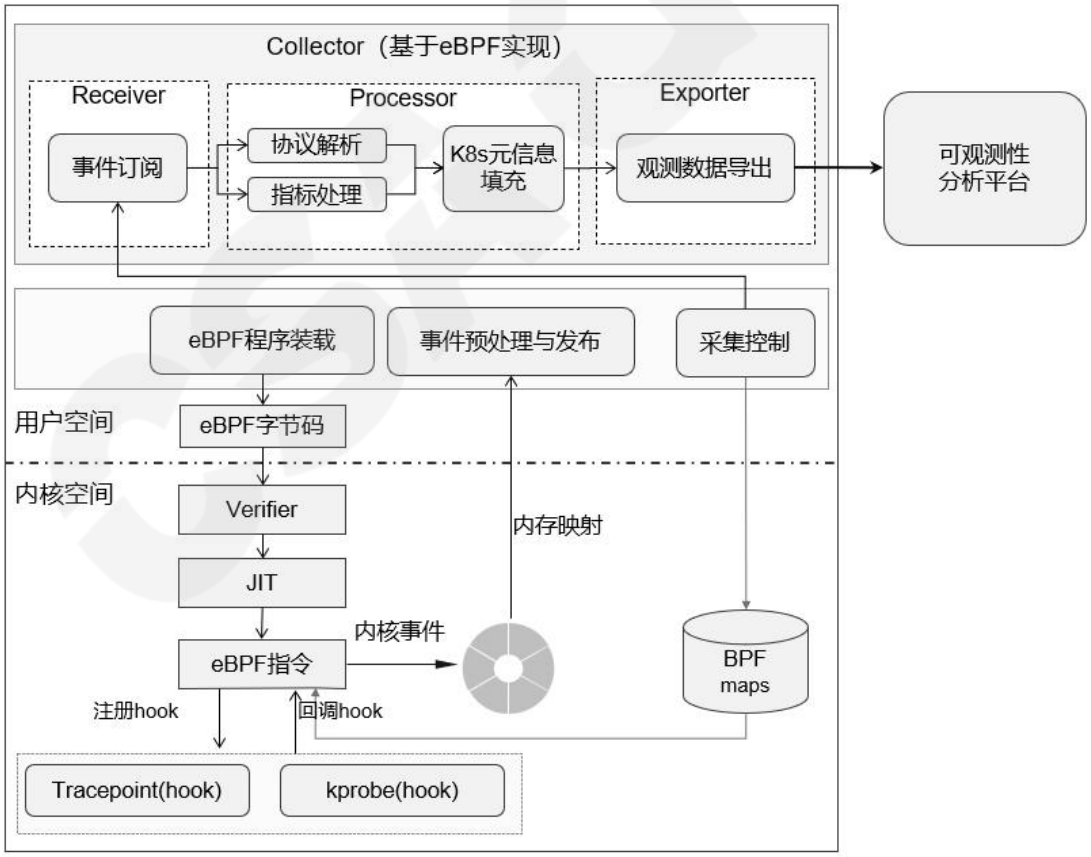


图 6 基于 eBPF 的可观测性架构

© 2023 云安全联盟大中华区版权所有

关键点在 eBPF 采集数据导入 OpenTelemetry Collector，步骤如下：

1. 数据接收：在 kubernetes 集群节点上基于 eBPF 的 tracepoint 和 kprobe/kretprobe，将内核采集到的应用请求、系统调用、网络传输性能等数据放到内存中，在用户态 eBPF 程序读取数据，进行预处理；基于 OpenTelemetry 规范实现 Receiver，以事件订阅方式接收采集数据；

2. 数据处理：基于 OpenTelemetry 规范实现 Processor，对采集数据进行协议解析和指标处理评估，然后填充 kubernetes metadata（元信息），实现 eBPF 采集的内核数据与 kubernetes 调度请求、上下文信息关联；

3. 数据导出：基于 OpenTelemetry 规范实现 Exporter 数据导出到可观测平台进行分析。

基于 eBPF 实现云原生可观测性数据采集具有以下优势：

1. 能够采集更加全面的数据，数据指标覆盖从程序调用、网络传输性能、网络协议栈性能、服务黄金指标等各个层面；

2. 资源消耗占比小，eBPF 程序以本机机器指令运行，性能很高，基于内核进行数据采集，对应用零侵入、零改造；

3. 更加灵活具备可伸缩性，eBPF 程序可以在运行时动态附加到系统中，无需重新启动目标系统；

© 2023 云安全联盟大中华区版权所有

4. 能够轻松应对容器启动、停止等动态特点，不需要任何的边车（ Sidecar）侵入，直接通过内核来观测任意的容器行为。

3.3.3 基于的 eBPF 云原生可观测性增强示例

3.3.3.1 动态网络性能监控

通过在每个 kubernetes 节点上部署一个探针 eBPF 程序。这个探针通过 hook 内核的 accept, connect, send, recv 等 L4(TCP、UDP)相关的系统调用，可以获取进程与绑定地址的关系、通信双方的地址、各连接收发的流量统计。探针会去获取当前 kubernetes 集群的 metadata 数据(pid, container, pod, service, node 等)并把它们保存在内存中，丰富原始 eBPF 数据。服务端在收集到通信双方的 eBPF 数据后做进一步数据填充，并存入数据库。查询时，根据指定的时间范围、主机/服务/pod 等筛选条件查询数据库，从而构造出该时段的各级别的动态拓扑图。并且基于 eBPF 能进一步采集内核级底层网络可观测性黄金信号数据,如 TCP 网络流量、网络延迟、堵塞等数据，并建立与 kubernetes 对象关联。

3.3.3.2 HTTP 黄金指标监控

云原生环境中有运行状况的三个关键 HTTP 黄金指标：请求速率、请求延迟、请求响应代码/错误。基于 eBPF 能够在不对应用程序进行任何更改的情况下提取这些数据，并且聚合相应的指标不是基于 IP，探针在获得网络报文之后，进一步解析 L7 内容，包括 HTTP、HTTPS、gRPC 等将微服务的各个会话（一次请求和响应）的 URL、latency、错误码关联到服务标识。探针定期将会话聚合信息推送到服务端进一步丰富数据，构建微服务链路、监控仪表盘等。

© 2023 云安全联盟大中华区版权所有

3.3.3.3 性能剖析（Profiling）

传统工具对系统中 CPU 进行定时采样，以特定的时间间隔或速率采集正在运行的函数或堆栈跟踪的计数，估计 CPU 的时间消耗分布，这种方式需要与应用强绑定，受开发语言局限，并且在跟踪多线程、多请求应用时存在困难。基于 eBPF 可以轻松地对堆栈跟踪和时间进行内核内摘要,获得系统级别特定进程的 On-CPU 和 Off-CPU 事件。基于 On-CPU 事件可以绘制火焰图，直观地展示各个调用栈所占时间比例。通过分析线程堆栈能够定位分析服务 CPU 使用率高的问题，如果堆栈被转储的次数更多，则意味着线程堆栈占用了更多的 CPU 资源，如下图 7：

图 7 CPU 资源

© 2023 云安全联盟大中华区版权所有

4.云原生可观测性应用场景

4.1 故障分析

故障分析是云原生可观测性的最基本的应用场景，也是可观测性技术的持续发展的重要驱动力。谷歌给出可观测性的核心价值快速排障（troubleshooting）。可观测性犹如整个 IT 系统的眼睛，它是运维人员发现问题、定位问题、解决问题的第一步，同时，也是运维监控的实现“先知、先觉、先行”的重要条件。

随着云原生持续发展，业务对可靠性、稳定性的要求越来越高。提前发现故障、快速发现故障、快速定位故障原因是做好稳定性的核心要素，良好的可观测性能够帮助用户在故障分析上事半功倍，有效提高业务上线运行地稳定性。

在故障分析领域，云原生可观测技术协助运维人员发现故障事件，分析故障发生原因，快速定位故障根因，可极大提升云原生领域运维效率。

4.2 事件预测

在安全运维领域，自动化成为技术发展的趋势。系统能够完成人类根本无法完成的任务，例如通过机器学习方法，在错误或事件发生之前进行预测。实现预测能力，需要通过在可观测性系统添加人工智能层。将人工智能的大数据分析能力及结果赋能于可观测性系统，使可观测性系统在问题发生之前提供预测能力。人工智能赋能的可观测性系统，除了提供预测能力，并且可对预测发生的问题提供解决方案。在安全运维领域提升对未发生事件、未知威胁的感知能力。

© 2023 云安全联盟大中华区版权所有

为使得可观测性系统具备更强的预测能力，需要更多的数据和更强大的算力支持，建立精确的模型和预测系统，提供更深刻的事件洞察能力。

4.3 日志审计

日志与审计对系统和应用行为的记录，对云原生可观测性的实现起到了重要的作用。通过日志系统获取系统以及应用的详细操作数据，是云原生可观测性重要的数据来源。

针对 Docker 和 Kubernetes，分别具有对应的日志采集机制，从安全审计的角度出发，可对日志数据进行存储、归类、查询等操作处理。

4.3.1 Docker 日志审计

Docker 支持多种日志记录机制，用以帮助用户从正在运行的容器和服务中获取信息，这种机制被称为日志驱动程序。Docker 从 1.6 版本开始支持日志驱动，用户可以将日志直接从容器输出到如 syslogd 这样的日志系统中。

每个 Docker 守护进程都有一个默认的日志驱动程序，通常这个默认的日志驱动是 json-file，也就是以 JSON 文件的形式保存日志信息。同时 Docker 还支持其他的日志驱动，比如 none、json-file、syslog 和 fluentd 等。下表 2 展示了当前 Docker 支持的日志驱动格式。

驱动	描述
none	不启用 log 功能，该容器 docker logs 没有可用的日志，并且不返

© 2023 云安全联盟大中华区版权所有

	回任何输出。
local	日志以自定义格式存储，旨在最大程度地减少开销。
json-file	日志格式为 JSON。这是 Docker 的默认日志驱动程序。
syslog	Linux 的系统 log 服务，将日志消息写入 syslog，确保 syslog 守护程序必须在 Docker 主机上已经运行。
journald	systemd 的 log 服务，可以代替 syslog 服务，将日志消息写入 journald，journald 守护进程必须在主机上运行。
gelf	将 log 写入 graylog 或 Logstash 等端点。
fluentd	将 log 写入 fluentd，确保 fluentd 在主机上已运行。
awslogs	将 log 写入 Amazon CloudWatch Logs。
splunk	使用 HTTP Event Collector 将 log 写入 splunk
etwlogs	将 log 写为 Event Tracing for Windows (ETW)事件，仅适用于 Windows 平台
gcplogs	将 log 写入 Google Cloud Platform (GCP)
logentries	将 log 写入 Rapid7 Logentries

表 2 Docker 支持的日志驱动格式

© 2023 云安全联盟大中华区版权所有

CIS Benchmark 对 Docker 的日志审计也提出了安全建议和要求，例如，在 CIS Docker Benchmark v1.6.0 版本中，2.13 小节要求需要配置集中和远程的日志记录，以确保所有的日志记录都是安全的，进而满足容灾的需要，而具体的日志驱动程序，则可以根据自身情况进行选择。 除了对容器的日志审计外，CIS Benchmark 还针对 Docker 主机，提出了相关的安全审计建议。对 Docker 主机的安全审计，一方面包括了常规的对 Linux 文件系统以及系统调用等进行审计。另一方面，也包括针对 Docker 守护进程等相关内容的安全审计。例如 CIS Docker Benchmark v1.12.0 版本中，1.1.3 小节要求，需要审计所有活动的 Docker 守护进程，可以通过 auditctl -l | grep /usr/bin/docker 命令，列出当前的审计规则，默认情况下，没有针对 Docker 守护进程的审计规则。

CIS Benchmark 中除了对 Docker 守护进程提出了安全审计的建议外，还对 Docker 相关的文件和目录提出了安全审计建议，例如：需要审计 Docker 文件和 /var/lib/docker 目录、需要审计 Docker 文件和/etc/docker 目录、需要审计 Docker 文件和 docker.socket 目录等。更多针对 Docker 详细的安全审计建议，可参考 CIS Docker Benchmark 标准。

4.3.2 Kubernetes 日志审计

4.3.2.1 应用程序日志

应用程序的日志记录可以更好的了解应用内部的运行状况，同时对调试问题、监控集群活动以及对应用程序运行过程的安全性分析有着非常大的作用。

© 2023 云安全联盟大中华区版权所有

当前，大部分应用程序都有某种日志记录机制，前文已经介绍过，以 Docker 为代表的容器引擎也被设计成支持日志记录的方式。针对容器化应用，最简单且最广泛采用的日志记录方式就是写入标准输出（stdout）和标准错误流（stderr）。

但是，由容器引擎或运行时提供的原生日志功能，通常不足以构成完整的日志审计方案。例如，当发生容器崩溃或者节点宕机等情况时，我们通常会想访问应用的日志，这时可能就会出现问题。在集群中，日志应该具有独立的存储和生命周期，与节点、Pod 或容器的生命周期相独立。这里通常会称为集群级的日志。

集群级的日志架构需要一个独立的后端用来存储、分析和查询日志。Kubernetes 当前并不为日志数据提供原生的存储解决方案，有很多现成的日志方案可以集成到 Kubernetes 中，具体可参考下 4.3.2.3 日志工具小节。

4.3.2.2 系统组件日志

在 Kubernetes 中，除了 Pod 中应用程序的日志外，Kubernetes 系统组件的日志同样需要有一定的方案来记录和存储。系统组件的日志主要记录了集群中发生的事件，这对于调试以及安全审计有着重要的作用。

系统组件日志可以根据需要配置日志的粒度，灵活调整日志记录的细节程度。日志可以是只显示组件内错误这种粗粒度的，也可以是更加细粒度的，例如记录事件的每一个追踪步骤（HTTP 访问日志、Pod 状态更新、控制器操作、调度器决策等）。

© 2023 云安全联盟大中华区版权所有

在 Kubernetes 中，系统组件根据部署运行方式的不同，可以分为两种类型：其中一种是运行在容器中的，比如，kube-scheduler、kube-proxy 等；另一种是不在容器中运行的，比如 kubelet、以及容器运行时等。在使用 systemd 机制的服务器上，kubelet 和容器运行时将日志写入到 journald 中。如果没有 systemd，它们会将日志写入到/var/log 目录下的.log 文件中。容器中的系统组件通常将日志写到/var/log 目录，绕过默认的日志机制。

Kubernetes 默认使用的日志库是 klog，专门用来做日志初始化的相关操作，klog 是 glog 的 fork 版本，由于 glog 不再开发、在容器中运行有不易测试等一系列问题，所以 Kubenetes 自己维护了一个 klog， 由于 Kubernetes 近期版本一直持续不断在精简系统日志组件的，因此在 Kubernetes v1.23.0 开始，klog 的一些命令行参数已经被废弃。

CIS Benchmark 对 Kubernetes 的日志审计也提出了安全建议和要求，例如，在 CIS Kubernetes Benchmark v1.6.0 版本中，1.2.22 小结要求需要配置—audit-log-path 路径，启动 Kubernetes API Server 的审计功能，设置合适的日志路径，进而可以获取 API Server 一系列按时间排序的与安全相关的记录。更多针对 Kubernetes 详细的安全审计建议，可参考 CIS kubernetes Benchmark 标准。

虽然 Kubernetes 没有为集群级日志记录提供原生的解决方案，但是 Kubernetes 官方给出了几种常见的参考设计方法（Logging Architecture）

4.3.2.3 日志工具

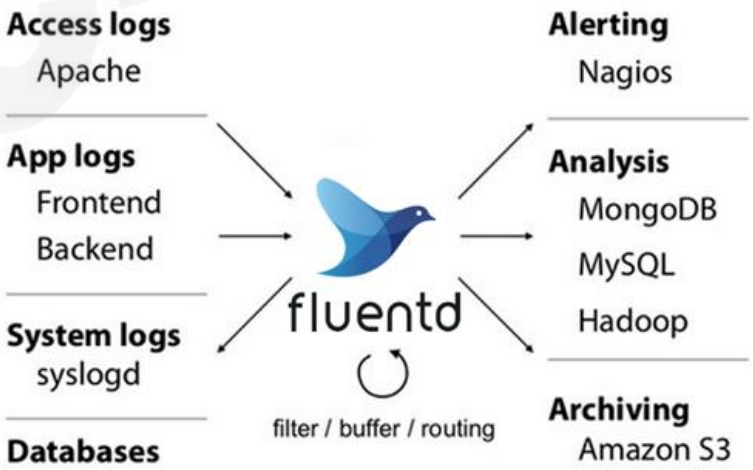
© 2023 云安全联盟大中华区版权所有

目前支持 Kubernetes 的日志管理工具种类也比较多，例如 Zebrium、Elastic Stack、CloudWatch、Fluentd 等，这些日志管理工具都有着一个共同的目标，那就是可以尽可能高效、快速的进行日志监控、记录以及分析处理。这里我们以 CNCF 项目 Fluentd 为例简单进行介绍。

Fluentd 由 Sadayuki “Sada” Furuhashi 于 2011 年提出，是一个跨平台的开源数据收集器，提供了统一的日志记录层，以便更好的使用和理解数据，不过它并不是一个独立的日志管理器。

这是一个非常流行的工具，有着数十个贡献者、数百个社区贡献的插件、有超过 5000 多名用户以及数以万计的事件被收集、过滤和存储，其用户包括 Atlassian、Microsoft 和 Amazon 等。

从架构上来看，Fluentd 创建了一个统一的日志记录层，可以更有效的使用数据，并在软件上对数据进行快速的迭代，可以每秒处理 12 万条记录。可扩展性方面，当前最大的用户集群可以从 5 万多个服务器中收集日志。因此，有着高可靠性、高可扩展性和良好的性能。



© 2023 云安全联盟大中华区版权所有

图 8 Fluentd 日志记录层

4.3 监控

监控（Metrics）在生产系统中，是必不可少的一部分，是系统稳定运行的重要基础，尤其是在云原生环境下，良好的监控系统，对于云原生应用的高效平稳运行，起到了重要的作用。

监控与日志有所不同，日志是对应用程序行为操作的一种记录，提供的是显式的数据。而监控更多的是通过数据的聚合，来对一个程序在特定时间内的行为进行衡量。

监控数据是可累加的，他们具有原子性，每个都是一个逻辑计量单元，或者一个时间段内的综合数据。监控的结果可以很好的观察系统的状态和趋势，但相比较于日志和追踪，监控结果对于问题的定位缺乏细节展示。作为云原生架构的重要支撑技术平台，我们以 Kubernetes 的监控为例，来介绍云原生环境下的主要监控指标。Kubernetes 的监控一方面需要包括对整个基础架构平台的监控，另一方面包括对正在运行的工作负载的监控。具体的监控指标，根据集群的特性不同而有所差异。

4.3.1 集群状态指标

集群状态可以说是一个基本的，也是关键的监控指标，我们需要知道集群中所有的聚合资源当前的状态以及使用情况，比如节点的状态、可用的 Pod、不可用 Pod 等。

© 2023 云安全联盟大中华区版权所有

通过对集群状态进行监控，进而对监控数据进行评估，以及由此产生的监控指标，可以让我们看到集群总体运行状况的概要视图。还可以了解到节点、Pod、Service 等相关联的问题。根据状态指标，可以判断集群的运行是否正常，是否存在相应的风险。

通过监控集群状态指标，我们还可以对节点正在使用的资源数量进行评估，包括共有多少节点、有多少节点可用等信息，从而可以根据需要调整所使用节点的数量和大小。

4.3.2 资源状态指标

CPU 利用率。清晰准确地知道节点 CPU 资源的使用情况，对于保障系统以及应用的平稳、安全运行有着至关重要的作用。如果微服务应用或主机节点已分配的 CPU 资源被耗尽，那么就需要考虑增加 CPU 配额或者增加处理节点，以满足业务的正常运行。同时，针对 CPU 资源使用情况的监控，还可以通过分析资源使用行为，发现挖矿、拒绝服务攻击等针对计算资源的恶意攻击行为。

内存压力。这个监控指标展示了一个节点正在使用的内存量，通过监控数据，我们可以实时的了解整个节点内存的使用状态，防止节点因内存耗尽而对应用运行产生影响。同时，还可以通过对每个组件、应用的内存分配情况进行分析，发现内存分配异常问题，比如，哪些应用的内存分配过度、不必要地增加了节点的开销，同时高内存压力还可以初步的判断应用程序是否存在内存泄露等问题

© 2023 云安全联盟大中华区版权所有

磁盘压力。磁盘在使用过程中，通常会设置相应的使用阈值，通过对磁盘使用情况的监控，结合既定的使用阈值，来判断节点磁盘空间的使用情况，进而确定是否需要增加额外的磁盘空间、当前应用程序的磁盘使用是否正常、是否需要对应用程序的磁盘使用进行调整等。

4.3.3 网络状态指标

对网络状态相关指标的监控，无论对于应用程序的通信还是安全，都有着重要的指示意义。

基于微服务架构的云原生应用，服务间的网络通信异常频繁，只有确保通信的正常，才能保证业务系统的顺畅运行。通过监控网络状态指标（比如，带宽、速率、连接状态等），及时的发现网络出现的问题，进而对问题进行定位、处置。

另外，对网络状态的监控，除了能够发现并解决网络故障问题，还可以通过对网络状态数据进行分析，判断是否存在网络层的攻击发生，比如拒绝服务攻击的检测，再比如异常网络行为的检测等。

4.3.4 作业运行指标

© 2023 云安全联盟大中华区版权所有

除了对基本的基础设施资源进行监控外，我们还需要对正在运行的作业任务进行监控，保证任务的准确运行。Kubernetes 中，使用了 Job 和 CronJob 两个资源，来提供一次性任务和定时任务的特性，这两种资源使用控制器模型来实现资源的管理。Kubernetes 的 Job 可以创建并且保证一定数量 Pod 的成功停止，当 Job 持有的一个 Pod 对象成功完成任务之后，Job 就会记录这一次 Pod 的成功运行。当一定数量的 Pod 任务执行结束之后，当前的 Job 就会将它自己的状态标记成结束。

基于这种机制，可以有效地对 Pod 进行管理和控制，同时对这些内容进行监控，也可以发现相关的问题，比如作业失败、崩溃循环、资源耗尽等。

作业失败并不一定意味着应用程序变得不可访问，但是忽略作业失败可能会导致后续部署出现更严重的问题。密切监控作业失败可以帮助及时恢复，并在未来避免这些问题。

崩溃循环通常指，应用程序在 Pod 启动时崩溃，并在不断的崩溃和重新启动中循环。出现崩溃循环可能会有很多原因，通常很难确定根本原因。因此，实时对其进行监控，在发生崩溃循环时，可以快速告警，快速缩小原因列表，并采取紧急措施使应用程序处于正常状态。

4.4 微服务追踪

© 2023 云安全联盟大中华区版权所有

在基于微服务的云原生架构中，客户端的一次服务调用，会产生大量的包括服务和中间件在内的众多调用关系。针对这些复杂的调用过程进行追踪，对于微服务的安全性分析、故障定位、以及性能提升等，有着重要的作用。因此，分布式追踪系统是微服务架构下不可或缺的重要组成部分。

分布式追踪是实现应用链路追踪的一种重要技术手段，同时也是实现云原生可观测性的重要组成部分，其主要用于应用程序的性能分析（APM，Application Performance Management）和故障定位等。Google 在 2010 年公开其生产环境下的分布式跟踪系统 Dapper，奠定了整个分布式追踪以及 APM 模型的基础。

自 Google Dapper 首先提出分布式链路追踪的设计理念以来，许多分布式追踪工具不断涌现。当前，常见的分布式追踪工具包括 Dapper，Zipkin，Jaeger，SkyWalking，Canopy，鹰眼，Hydra，Pinpoint 等，其中常用的开源分布式追踪工具为 Zipkin，Jaeger，SkyWalking 和 Pinpoint。这些分布式追踪工具大致可分为：基于 SDK、基于探针以及使用 Sidecar。

基于 SDK 的分布式追踪工具。以 Jaeger 为例，Jaeger 提供了大量可供追踪使用的 API，通过侵入微服务业务的软件系统，在系统源代码中添加追踪模块实现分布式追踪。此类工具可以最大限度地抓取业务系统中的有效数据，提供了足够的可参考指标；但其通用性较差，需要针对每个服务进行重新实现，部署成本较高，工作量较大。

© 2023 云安全联盟大中华区版权所有

基于探针的分布式追踪工具。以 SkyWalking Java 探针为例，在使用 SkyWalking Java 探针时，需将探针文件打包到容器镜像中，并在镜像启动程序中添加-javaagent agent.jar 命令实现探针的启动，以完成 SkyWalking 在微服务业务上的部署。SkyWalking 的 Java 探针实现原理为字节码注入，将需要注入的类文件转换成 byte 数组，通过设置好的拦截器注入到正在运行的程序中。这种探针通过控制 JVM 中类加载器的行为，侵入运行时环境实现分布式追踪。此类工具无需修改业务系统的源代码，相对 SDK 有更好的通用性，但其可获取的有效数据相对 SDK 类工具较少。

基于代理实现。Sidecar 作为服务代理，为其所管理的容器实现服务发现，流量管理，负载均衡和路由等功能。在流量管理过程中，Sidecar 可以抓取进出容器的网络请求与响应数据，这些数据可以记录该服务所完成的一次单个操作，可与追踪中的跨度信息对应，因此可将 sidecar 视为一种基于数据收集的分布式追踪工具。Sidecar 无需修改业务系统代码，也不会引入额外的系统的开销。但由于 sidecar 所抓取的跨度不包含追踪链路上下文，要将 sidecar 所抓取的跨度数据串联成追踪链路是很困难的。

基于 eBPF 实现。eBPF 通过一种软件定义的方式,提供并支持了丰富的内核探针，同时提供了强大的动态追踪能力。开发者通过编写 eBPF 程序即可实现相应的追踪脚本，我们可以使用 eBPF 自身的实现机制，以保证在内核执行动态追踪时的效率及安全性。eBPF 结合 Opentelemetry 规范可实现微服务链路追踪，详细内容可参考前文 3.3.2 小节。

© 2023 云安全联盟大中华区版权所有

4.5 安全检测分析

4.5.1 容器运行时检测

使用 eBPF 追踪技术可实时捕获云原生环境中活动容器内关键事件。包括：容器对文件的可疑访问，容器对系统的可疑调用，容器之间的可疑互访，检测容器的异常进程等，结合安全主控引擎和安全策略集可进一步对活动容器的异常行为进行取证。例如：

- 检测容器中用户操作及可疑的 shell 脚本的执行
- 检测容器运行时是否打开了新的监听端口或者建立意外连接的异常网络活动
- 检测容器运行时是否存在文件系统读取和写入的异常行为，例如在运行的容器中安装了新软件包或者更新配置
- 检测容器运行时是否创建其他进程
- 检测容器中是否运行了黑客工具，是否存在反弹 shell 行为
- 检测容器中是否执行了挂载系统目录操作

© 2023 云安全联盟大中华区版权所有

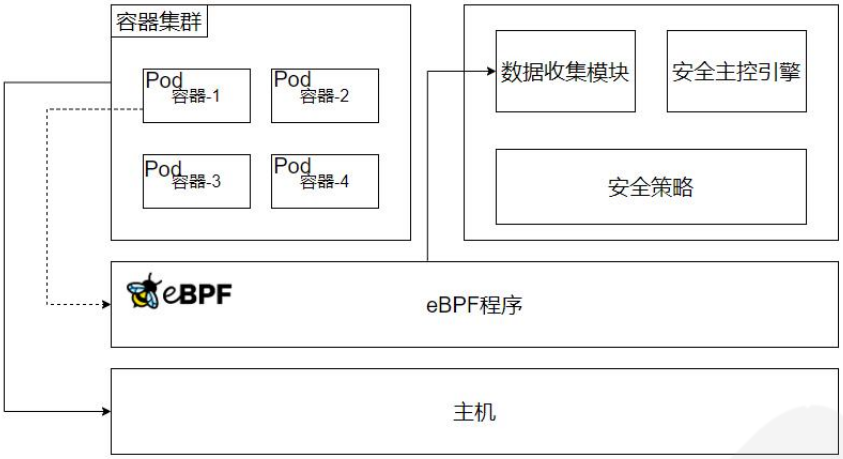


图 9 架构设计

eBPF 程序：捕获系统调用、网络数据包、文件系统等业务操作的 eBPF 程序，并加载至宿主机内核中执行。

数据收集模块：接收 eBPF 收集的容器内部系统调用和其他关键事件，例如创建文件、执行命令、网络通信等。这些事件可以包括正常的操作，也可能包括恶意行为或异常操作。

安全主控引擎：用于接收数据收集模块的数据，并根据预定义的安全策略集检测容器内部异常行为，若命中策略则实时触发警报并进行响应。具体可包括发送通知、记录审计日志、触发自动化的安全措施等。

安全策略集：定义一系列安全规则和策略，用于识别潜在的安全威胁和异常行为。这些规则可以基于特定的系统调用、文件访问、网络通信等事件，定义容器行为规范。

4.5.2 微服务业务异常检测

© 2023 云安全联盟大中华区版权所有

云原生环境中，可通过分布式追踪技术，实现获取 API 上下文信息及调用拓扑，这些信息可以做为检测引擎的输入源，通过业务异常检测引擎对正在运行的业务系统进行异常检测。

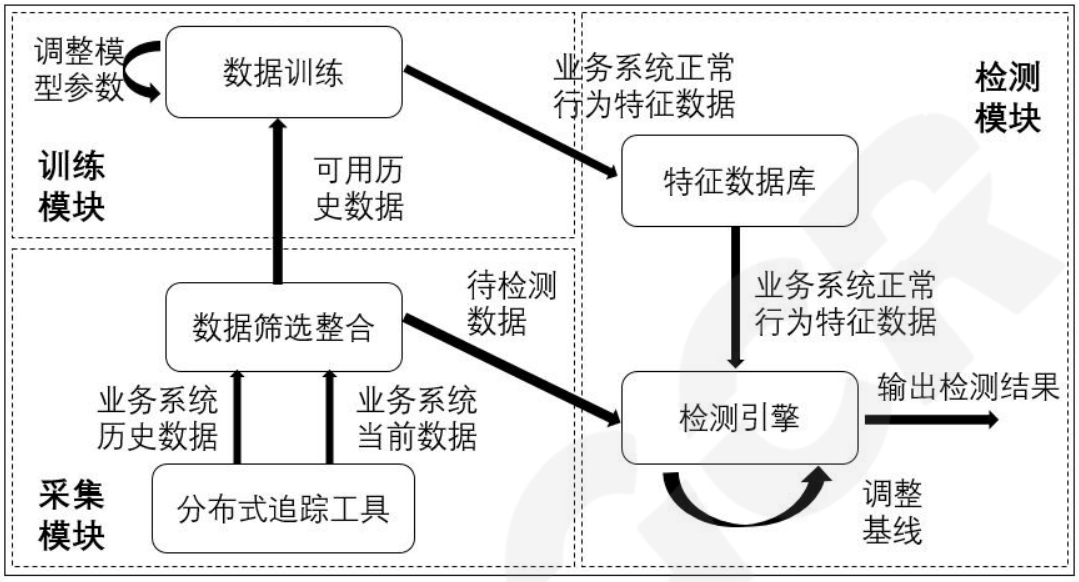


图 10 业务异常检测引擎

检测引擎具体功能介绍：

分布式追踪工具。相比 Skywalking、Sidecar，Jaeger 可获取的数据字段最多，能够检测的异常场景最丰富，然而，Jaeger 需要在业务系统的源代码中进行插桩，对开发团队而言有较强的侵入性。相反，Sidecar 模式没有代码和镜像的侵入性，但通过反向代理截取流量的模式也决定了它不能获得丰富的上下文，如云原生应用的 API 调用关系树（TraceID）是无法获得的。采用 eBPF 可解决上述的侵入性问题，且更轻量级，对系统性能的影响更小。从而实现更高效的性能跟踪和监控，同时提供更多细粒度的数据收集和分析能力。

© 2023 云安全联盟大中华区版权所有

数据筛选与整合模块。此模块主要功能为过滤掉数据集中的脏数据，以及提取出可以表示业务系统行为的数据。在云原生应用中，可以表示业务系统行为的数据为 API 调用关系树、服务名、操作名、HTTP POST 参数等。

数据训练模块。将预处理后的历史数据利用机器学习或统计学的方法，训练出业务系统中的正常行为，并生成与业务系统正常行为匹配的特征数据。这里进行训练的先验知识为，我们认为业务系统中大量存在的行为是正常行为，而数量很少的行为是异常行为。在训练过程中，需要根据专家知识对训练结果的检验不断调整训练模型的参数。

检测引擎。将业务系统当前数据与特征数据库中数据进行检索匹配，并利用序列相似性计算等方法找出特征数据库中与当前行为最为匹配的特征数据。检测引擎需要将特征数据与当前数据的相似性与基线进行比较，若比较结果显示当前行为与正常行为的差异在基线限制范围内，则为正常行为，若超出基线限制范围，则判定为异常行为。对于基线，首先需要根据专家知识设置合理的初始基线，并根据不同场景，或利用无监督模型自行调整基线，或由运维人员手动维护基线。

5.优秀云原生可观测项目介绍

5.1 Prometheus 项目

5.1.1 项目概述

© 2023 云安全联盟大中华区版权所有

Prometheus 是一个开源的云原生监控解决方案，用于收集、存储和处理各种指标和事件数据。它旨在解决现代应用程序架构中与传统监控系统相关的一系列挑战，如复杂性、可扩展性和平衡成本效益。Prometheus 成为了许多现代应用程序和基础设施中首选的监控解决方案，包括 Kubernetes、Docker 和 Cloud Native Computing Foundation（CNCF）中的许多项目。

项目于 2012 年开源，截止本文撰写日（23 年 11 月），Stars 数量高达 50K，参与的贡献者/组织达到了 838 个。Prometheus 于 2016 年加入云原生计算基金会 作为继 Kubernetes 之后的第二个托管项目。

5.1.2 整体架构

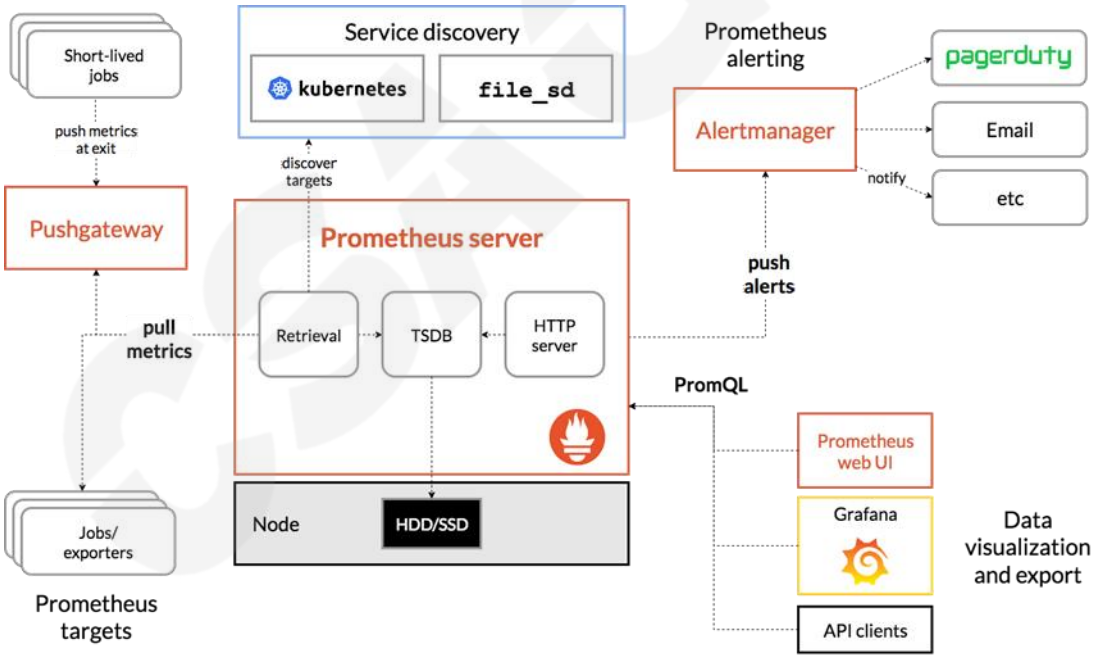


图 11 Prometheus 项目整体架构

© 2023 云安全联盟大中华区版权所有

Prometheus 的整体架构可以分为以下几个核心组件：

1. Prometheus Server：

Prometheus 服务端负责拉取和存储各种监控指标。它使用一种可扩展的 HTTP 协议来远程访问目标设备或服务，并支持多种数据导出格式，如 JSON、CSV、Graphite 等。

2. Client Libraries：

客户端库用于将应用程序内部指标转换为 Prometheus 可以理解的格式。这允许开发者轻松加入 Prometheus 的数据采集功能。几乎所有的编程语言都有对应的客户端库，如 Go、Python、Java 等。

3. Exporters：

导出器用于将第三方服务的指标转换为 Prometheus 可以理解的格式。这使得 Prometheus 可以轻松集成和监控各种第三方服务和设备，如数据库、消息队列、文件服务器等。

4. Remote Write

Agents 远程写入代理用于将采集到的指标数据发送到 Prometheus Server。它们在本地存储指标，并在接收到请求后将其发送到 Prometheus Server。

5. Service Discovery：

服务发现组件（如 Consul）负责在应用程序中注册和发现 Prometheus 实例。这允许将 Prometheus 部署在分布式环境中，以便收集各个服务实例的指标。

© 2023 云安全联盟大中华区版权所有

5.1.3 功能特点

Prometheus 具有以下功能特点：

- 采用多维数据模型，其时序数据由指标名称和键/值对标识；
- 采用 PromQL，一种灵活的查询语言，可利用此语音进行数据搜索；
- 不依赖分布式存储;单服务器节点是自治；
- 时序收集通过 HTTP 上的拉取模型进行；
- 支持通过中间网关推送时序数据；
- 通过服务发现或静态配置发现目标；
- 支持多种绘图和仪表板模式；

5.1.4 定位

Prometheus 是一个功能强大、灵活且易于扩展的云原生监控解决方案。Prometheus 作为云原生应用的监控基石，为复杂的云原生基础设施提供了稳定、高效的监控和报警能力，是云原生架构中不可或缺的组件之一。在未来几年的发展中，该项目也会是未来云原生监控发展的中坚力量。

5.2 OpenTelemetry 项目

5.2.1 项目概述

© 2023 云安全联盟大中华区版权所有

OpenTelemetry 项目最早于 2019 年由两个独立的项目 OpenCensus 和 OpenTracing 合并而成。OpenCensus 主要关注收集和传输观测数据，OpenTracing 主要关注定义和追踪分布式系统中的请求元数据信息，开源可观测性项目如 Jaeger、Zipkin 均使用了其标准，然而，由于两个项目的目标和方法略有不同，缺乏统一的标准和工具，不同的应用程序和服务使用不同的观测数据格式和传输方式，导致观测数据收集变得复杂和困难。为了解决这一问题，OpenCensus 和 OpenTracing 的维护者决定将两个项目整合在一起，形成一个统一的解决方案。在合并之后便形成了 OpenTelemetry 项目。

OpenTelemetry 项目是一个开源的可观测性框架，用于收集、检测、分析、导出可观测数据（追踪、度量、日志）。OpenTelemetry 与其他开源云原生可观测项目无关，因而可与其联合使用，如 Jaeger、Prometheus、Fluentd 等开源工具，此外 OpenTelemetry 也作为孵化类项目入选了了云原生计算基金会(Cloud Native Computing Foundation CNCF，通过使用 OpenTelemetry，用户可以使用统一的 API 和数据模型来收集和传输观测数据，而不需要关心底层的实现细节，从而可以降低观测数据收集的复杂性，并提高系统的可观测性。

5.2.2 整体架构

OpenTelemetry 的核心组件包括以下几部分：

© 2023 云安全联盟大中华区版权所有

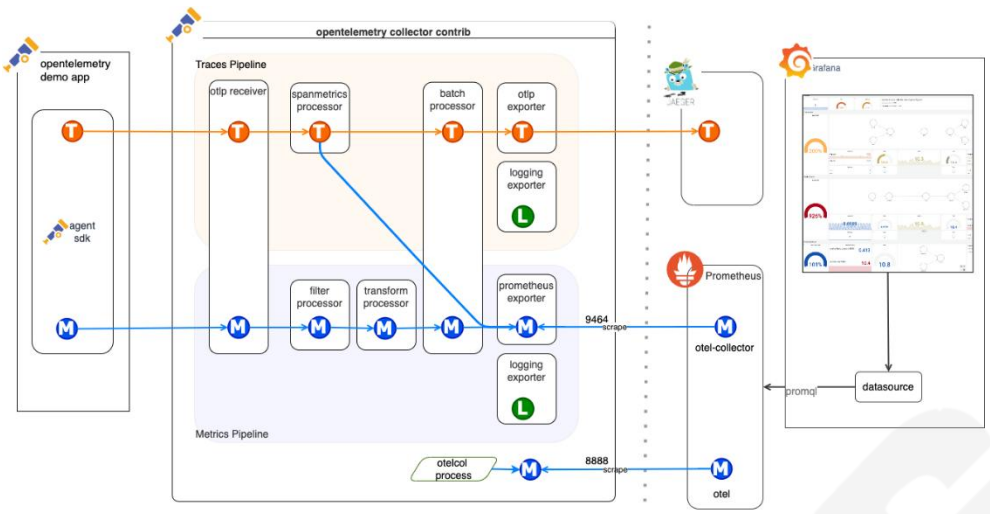


图 12 OpenTelemetry 核心组件

- 组件规范：定义了所有组件的规范和标准。
- 标准协议(OTLP)：定义了遥测数据的格式和结构的标准协议。
- 语义约定：定义了常见遥测数据类型的标准命名方案。
- API：定义了如何生成遥测数据的接口。
- 开发语言 SDK： 实现了定义各类开发语言的遥测数据的开发工具包。
- 自动化生成组件(Auto Instrumentation)：无需代码更改即可生成遥测数据的自动化组件。
- OpenTelemetry Collector：一个代理，用于接收、处理和导出遥测数据。
- 其他工具：如 OpenTelemetry Operator 用于 Kubernetes、OpenTelemetry Helm Charts 等场景。

© 2023 云安全联盟大中华区版权所有

5.2.3 功能特点

OpenTelemetry 的功能特点体现在其工作流程，以下为工作流程不同阶段所覆盖的功能特点：

生成（Instrumentation）：在微服务中使用 OpenTelemetry 提供的 SDK 或自动化组件进行埋点，以生成跟踪和指标数据。这类数据可包括微服务的函数调用、数据库查询、网络请求等。

处理（Processing）：生成的跟踪和指标数据可通过 OpenTelemetry 的 SDK 或自动化组件导出至 OpenTelemetry Collector 或其他支持的 Collector(第三方)。导出可以是实时的或批量的，具体取决于配置和需求，Collector 接收到 SDK 收集的指标数据后，将通过 Processor 组件进行一系列的处理操作，包括针对指标的过滤、采样、聚合、转换及添加自有的业务等，定制化能力较强。

导出（Exporting）：通过 Proccessor 处理后的指标数据将根据需求通过 Exporter 导出至可观测性平台，如 Jaeger、Promethues 等。

分析（Analysis）：导出的数据可通过可观测型平台进行进一步的分析和可视化。这些平台可提供追踪，指标查询、告警、仪表盘等功能，从而帮助开发人员及运维人员监控、优化应用程序的性能。

5.2.4 定位

© 2023 云安全联盟大中华区版权所有

OpenTelemetry 目前在市场上的应用情况正在逐步增加，尤其在容器化和微服务架构中。现今许多云服务提供商和容器平台，如 AWS、Google Cloud、Azure、Huawei Cloud 等均已经开始支持和推广 OpenTelemetry。值得注意的是 Opentelemetry SDK 现已支持多种开发语言，包括但不限于 C++、.NET、Go、Java、Javascript、Python、Rust、Swift 等。

OpenTelemetry 仍处于发展和演进过程中，在频繁更新版本给用户带来新的体验的同时，可能会导致不同版本间的兼容性问题。

OpenTelemetry 会持续聚焦于云原生生态系统支持，致力于将统一的数据格式和数据传输协议集成至更多主流的监控分析平台。

5.3 SkyWalking 项目

5.3.1 项目概述

Skywalking 是一个国产的开源框架，2015 年有吴晟个人开源，2017 年加入 Apache 孵化器，主要开发人员来自于华为，2019 年 4 月 17 日 Apache 董事会批准 SkyWalking 成为顶级项目，支持 Java、.Net、NodeJs 等探针，数据存储支持 Mysql、Elasticsearch 等，跟 Pinpoint 一样采用字节码注入的方式实现代码的无侵入，探针采集数据粒度粗，但性能表现优秀，且对云原生支持，目前增长势头强劲，社区活跃。

5.3.2 整体架构

© 2023 云安全联盟大中华区版权所有

SkyWalking 在逻辑上分为四个部分：探针，平台后端，存储和用户界面，其总体架构如下图 13 所示：

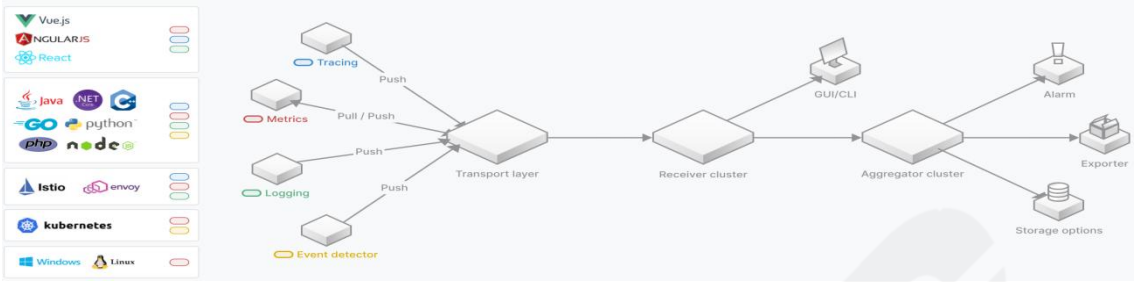


图 13 SkyWalking 总体架构

探针收集监测数据，包括各种格式（SkyWalking、Zipkin、OpenTelemetry、Prometheus、Zabbix 等）的指标、追踪、日志和事件。

平台后端支持数据聚合、分析和流处理，涵盖追踪、指标、日志和事件。可以作为聚合器角色、接收器角色或两者兼备。

存储通过开放/可插拔的接口存放 SkyWalking 数据。可以选择现有的实现，如 ElasticSearch、H2、MySQL、TiDB、BanyanDB，或自己实现。

UI 是一个高度可定制的基于 web 的界面，允许 SkyWalking 的最终用户可视化和管理 SkyWalking 数据。

5.3.3 功能特点

- 分布式追踪：端到端的分布式追踪能力。服务拓扑分析、以服务为中心的可观测性和 API 仪表盘。

© 2023 云安全联盟大中华区版权所有

- 多语言自动探针：Java、.Net

- Core、PHP、NodeJS、Golang、LUA、Rust、C++、客户端 JavaScript 和 Python 代理，处于积极开发和维护状态。

- 采用 eBPF 技术：Rover 代理作为由 eBPF 提供支持的指标收集器和分析器，用于诊断 CPU 和网络性能。

- 原生 APM 数据库：BanyanDB 是在 2022 年创建的一款可观测性数据库，旨在摄取、分析和存储遥测/可观测性数据。

- 一致的指标聚合：通过同一脚本管道进行处理 SkyWalking 原生的度量格式以及广为人知的指标格式（例如 OpenCensus、OTLP、Telegraf、Zabbix 等）。

- 日志管理管道：通过脚本管道支持日志格式化、提取指标、各种采样策略，性能高效。

- 警报和监测管道：支持以服务为中心、部署为中心、API 为中心的报警规则设置。支持将报警和所有监测数据转发到第三方。

- 扩展性：单个 SkyWalking 集群可以收集和分析超过 1000 亿条遥测数据。

- 兼容性：支持来自成熟生态系统的指标、追踪和日志，例如 Zipkin、OpenTelemetry、Prometheus、Zabbix、Fluentd。

5.3.4 定位

© 2023 云安全联盟大中华区版权所有

Skywalking 项目定位为分布式系统的应用程序性能监视工具，专为微服务，云原生架构和基于容器（Docker，K8S,Mesos）架构而设计，它是一款优秀的 APM（Application Performance Management）工具，包括了分布式追踪，性能指标分析和服务依赖分析等。

© 2023 云安全联盟大中华区版权所有

附件.引用文献

【1】B. Sigelman, L. Barroso, M. Burrows, Patrick Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, C. Shanbhag. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure 2010

【2】CNCF Observability Whitepaper 1.0 2023

【3】Dan Wendlandt. Grafana and Cilium: Deep eBPF-powered observability for Kubernetes and cloud native infrastructure, 2022.

【4】高巍.基于操作系统 eBPF 在云原生环境下的技术研究. 电子技术与软件工程，2022

【5】高巍.可观测性技术研究与探索.电子技术与软件工程，2022.

【6】Han Liu, Sheng Wu. Pinpoint Service Mesh Critical Performance Impact by using eBPF，2022

【7】刘文懋 江国龙 浦明 阮博男 叶晓虎《云原生安全：攻防实践与体系构建》，2021.

【8】Gartner® Top Strategic Technology《2023 年度十大重要战略技术趋势》

【9】Gartner® Market Guide for AIOps Platforms, May 2022

【10】The Forrester Wave™: Artificial Intelligence for IT Operation, Q4 2020

【11】中国信通院稳定性保障实验室《可观测性成熟度模型白皮书》2023

© 2023 云安全联盟大中华区版权所有

CSA GCR cloud security alliance GREATER CHINA REGION ®

Cloud Security Alliance Greater China Region

扫码获取更多报告

云安全联盟大中华区
官网：https://c-csa.cn 电话：0755-86548359
地址：上海市浦东新区东育路255弄5号3FB30（前滩国际经济组织集聚区）/
广东省深圳市南山区粤海街道北京航空航天大厦1号楼905