飞天大数据平台 APSARA BIG DATA PLATFORM | 阿里云开发者社区 ALIBABA CLOUD DEVELOPER COMMUNITY

Elasticsearch全观测
技术解析与应用

构建日志、指标、APM统一观测平台

Elasticsearch 技术应用系列

从原理到实践，一书全面了解Elasticsearch全观测

- 全观测技术原理与技术生态
- Elasticsearch全观测的行业应用
- 日志、指标、APM全观测应用实践

钉钉扫码加入
“Elasticsearch 技术交流群”

扫码了解
“更多阿里云 Elasticsearch”

扫码订阅
“Elasticsearch 技术博客”

阿里云开发者“藏经阁”
海量免费电子书下载

卷首语

Elasticsearch 全观测的核心是指把日志、指标、APM 甚至 Uptime 数据汇总到一个平台上，让运维人员、开发人员，甚至业务人员都可以在统一的大数据平台之上，对所有的数据从统一的视角进行观察，告警，以及可视化。

本书从介绍 Elasticsearch、全观测技术原理、行业应用到技术实践，全面系统地解读在大数据背景下，运维人员、开发人员等应用全观测技术的价值和实践上手指南。

目录

基础介绍篇

应用实践篇

走进阿里云 Elasticsearch

摘要：本文对 Elasticsearch 进行了整体介绍，包括 Elasticsearch 生态矩阵的构成，它所具备的低成本和强功能等特性，以及与搭建开源 ES 服务相比阿里云 Elasticsearch 所具备的优势。此外，还对 Elasticsearch 全观测产品能力、架构、技术难点和实践案例进行了分享。希望通过本文，大家能对 Elasticsearch 和全观测有更全面的认识。

观看视频：https://help.aliyun.com/document_detail/190544.html

分享人：沐泽

本文主要对 Elasticsearch 进行整体的介绍，以及在阿里云 Elasticsearch 上的全观测产品能力及最佳实践。

Elasticsearch 是业内比较热门和主流的信息检索分析引擎，在 DB-Engine 指数排行上是全球热度第 7 的数据库，也是全球热度第一的检索引擎。

Elasticsearch 在很多行业的场景下有非常多的应用，比如在企业搜索中的信息查询和在日志检索和分析中的应用。除此之外，它在金融、零售等行业中还能做数据分析和可视化，以及订单查询和 Elasticsearch 本身所包含的地理位置查询。

Elasticsearch 开源受到广大开发者的使用和接受并不只是基于 Elasticsearch 这一个检索引擎，除了 Elasticsearch 之外，它还包括 Beats，Logstash，Kibana 这一套生态矩阵。它是构建于 Apache Lucene 搜索引擎库之上的分布式全文搜索和分析引擎，提供搜集、分析、存储数据三方面能力。

通过 Beats 这个轻量级数据采集工具，数据能进入 Elasticsearch 系统。它集合了多种单一用途数据采集器，它们从成百上千或成千上万台机器和系统向下游发送数据。而在这套生态矩阵中，Beats 的下游就是 Logstash。Logstash 作为收集、过滤、传输数据的工具，能针对各种各样的日志数据做一些预处理和过滤。数据经过采集和处理，最后到 Elasticsearch 这样一个检索系统中进行存储，然后我们可利用 Kibana 去做业务上的可视化报表和大盘分析的搭建。

从 Beats，Logstash，Elasticsearch 到 Kibana 的这套开源生态矩阵，能帮用户解决各种各样的场景问题。目前阿里云上已提供了一个全托管的服务，用户不必再购买服务器和进行搭建，在阿里云上就可以直接一键开通整套服务。

一、那么阿里云在这套生态矩阵中做了一些什么事？它又有哪些特性和优势？

在开源生态下，Elasticsearch 有一套 X-Pack 商业插件，它包含数据权限、可视化、机器学习等能力，价值达到 6000 美元，而在阿里云上创建 Elasticsearch 服务，则可自动免费开启 X-pack 插件。

从下面这张图可以看到 Elasticsearch 端到端分析检索的架构，从数据采集、数据加工、搜索引擎到上层应用。在搜索引擎层，除了支持免费的商业插件外，我们在 Elasticsearch 的管控上还做了很多安全上的能力，如异地容灾和热重启能力，以便帮助用户更好地去运维这一套集群。

随着业务规模的扩大，整个 Elasticsearch 的运维成本非常高，而阿里云已经在 3 年间积累了丰富的大规模集群的管理和运维经验，并通过智能运维、高级监控报警等产品能力向我们的客户赋能。

我们很多用户使用的 Elasticsearch 内核是开源的，而阿里巴巴有专门的 Elasticsearch 内核团队，针对场景去做性能和成本上的优化。比如日志增强版内核，就专门针对大规模的数据量做了包括索引压缩和计算存储分离等在内的能力。除此之外，在一些搜索场景下面还会有达摩院分词，除了文本搜索，还提供一些向量检索的能力，以帮助客户更好地实现业务需求。

基于这样一套数据引擎，我们的用户能搭建自己各种业务运营的上层应用，比如监控告警、全要素搜索、APM 服务等。而在数据采集层，我们全方位托管了 Beats 和 Logstash，从而能帮助用户更好地同步数据，不论是来自阿里云的、开源的，或是其他云厂的。

通过这样一套端到端的分析检索架构，我们能为用户提供更丰富的分析检索能力，也让云上整体的 Elasticsearch 服务具备更高的可用性和安全特性。

目前，阿里云上有 30 多个行业上千位客户在使用我们的服务。在公共云的环境下，我们不仅能覆盖国内大部分地区和海外的一些数据中心，还能支持一些本地化的专有云的交付和提供混合云的方案，使不同行业的用户都能够很好地去使用我们这套服务。

二、与搭建开源 ES 服务相比，阿里云 Elasticsearch 的优势在哪里?

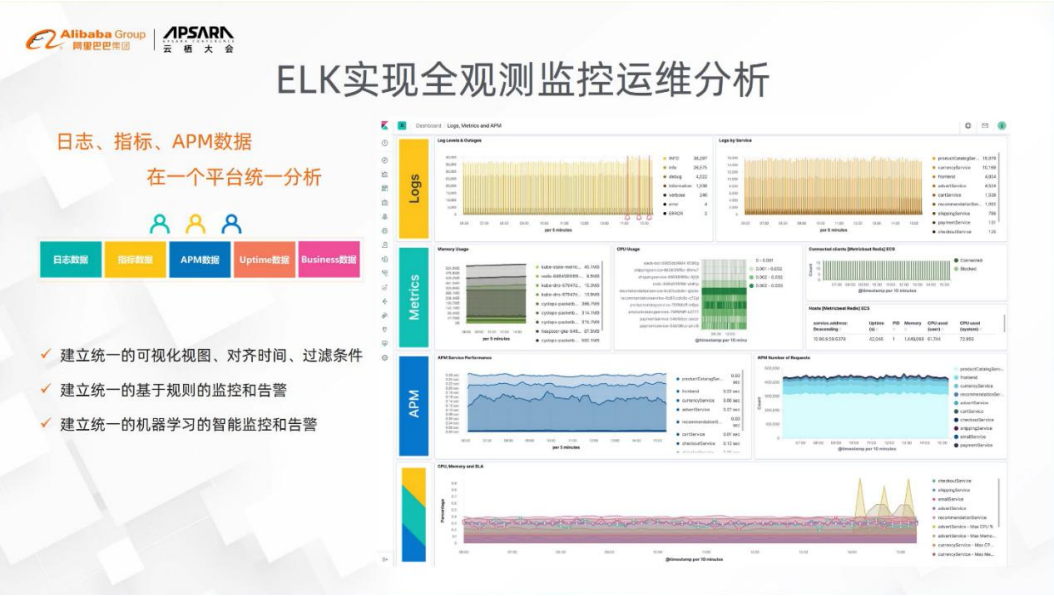
下面这张图我们整理了 Elasticsearch 与搭建开源 ES 服务的对比，在各个业务场景下，Elasticsearch 带来了全方位的能力提升与性能优化。包括云上的全套托管、超低的运维成本、降低大数据量的存储成本、一键搭建集群、集群平滑扩缩、向量检索、QoS 索引级别限流等等…

尤其在安全性和高可用方面，大家搭建开源 ES 服务的时候没有那么多精力去做安全特性的补充，所以我们做了一些 HTTP 的传输加密和内网环境管控等。同时，我们的数据可靠性和服务可靠性都达到几乎满分，能尽可能地保证客户在实现业务的时候不受到不稳定因素的影响。

三、什么是全观测? Elasticsearch 全观测能力如何 ？

我们对全观测概念的理解，是将日志、指标、APM 等数据在一个平台进行统一分析，而这样的能力正是 ELK，也就是 Elasticsearch 全观测解决方案所能提供的，它能帮助用户在 ELK 平台上建立统一的可视化视图。另外，通过全链路问题的追踪，还能设置统一的监控

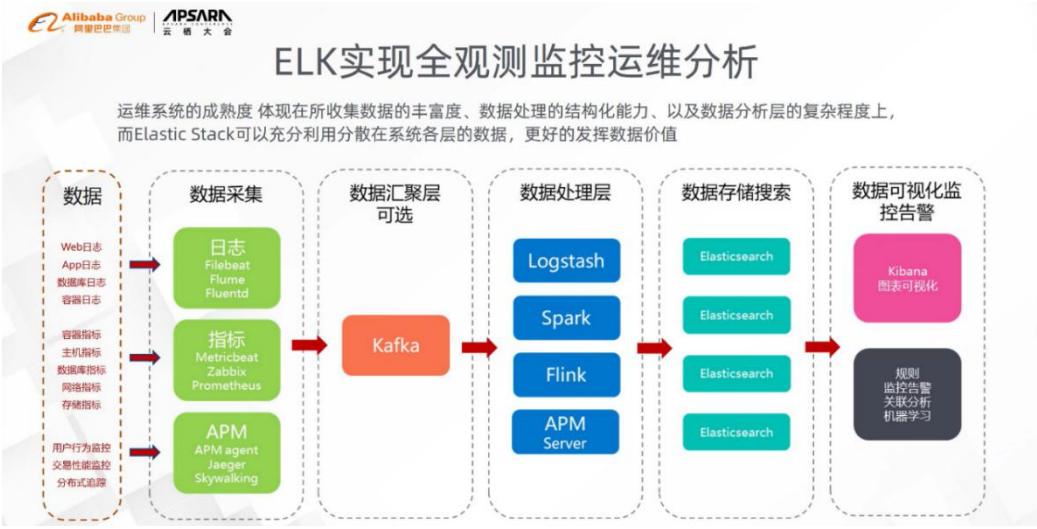
报警规则，甚至能利用机器学习的能力对未来情况进行预判。



从整体业务场景看，每个企业的系统中存储了各种日志和指标数据，而随着整个运维系统搭建得越来越成熟，我们收集的数据也越来越丰富。比如对一套日志的分析，日志分析能力已经不能满足全链路观测的需求了，我们需要收集更多的数据并做后续的处理和结构化，并通过可视化进行呈现。

另外，上层的数据可视化监控告警，最传统的也是基于一些规则进行监控告警。而随着这套运维系统的成熟，我们后续做的数据分析也不仅仅是简单的规则类分析和告警，更多的是可以做一些关联分析，以及利用机器学习的能力去利用分散在系统各处的数据，让其发挥价值。以上这些，正是全观测运维系统所需要具备的能力。

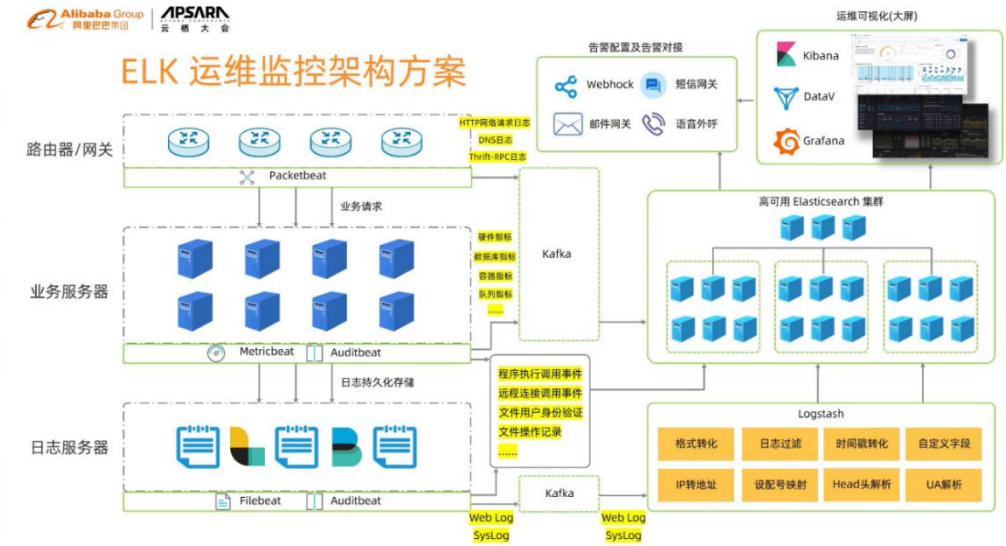
在直播、游戏等很多行业中，他们非常需要数据去做一些用户行为研究和整体链路的优化，从而为核心业务带来提升。而要让这套运维系统发挥业务价值，就取决于我们如何处理这些数据，如何更好地获取和分析数据所代表的真实含义。Elasticsearch 正具备这样的全观测能力，它区别于简单的日志处理和大数据分析，是能在同一个平台进行分析的全观测解决方案。



我们可以通过下面的架构图看到 ELK 在运维监控全链路上的能力。通过 Packetbeat 对网关的数据做收集，通过 Metricbeat 对业务服务器上的指标进行收集，通过 Filebeat 做日志相关的收集，以及利用 APM 的 agent 对用户实时行为做链路追踪。

通过对各种数据来源进行采集，我们会将其下发到 Kafka 组件，随后通过 Logstash 进行格式转化和结构处理，最后将数据传输到整个 Elasticsearch 集群里面，并基于上层的可视化组件搭建可视化的大屏。

除此之外，我们还能去接入非常完备的一套告警配置和告警对接，因为对数据进行实时链路追踪的同时，也需要我们对异常进行捕捉和判断，并通过短信网关等方式将这些判断及时返回给系统负责人，实现告警对接。



可以看到，这套日志分析和运维监控的链路非常复杂，而它们的难点也是共通的。

第一个是高并发写入。由于系统采集的数据的类型和量非常大，所以 Elasticsearch 集群很容易因为高并发写入和流量的波动导致其稳定性受到影响。

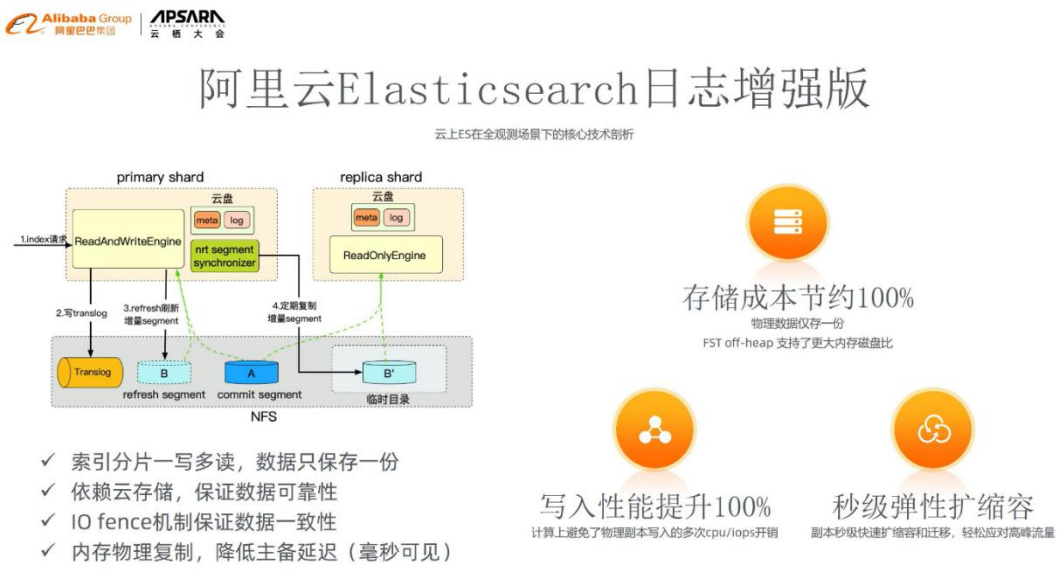
第二个是存储成本高。我们的系统通常涉及到 TB 级甚至 PB 级的数据，所以它们的存储成本也是需要考虑的一点。有些场景下数据可能只是做一些审计，所以需要我们对系统进行优化，比如通过冷热的方案降低同数据量下我们的存储成本。

第三个是时序分析性能差。由于 Elasticsearch 内核上的限制，当它遇到一些复杂的聚合、Range 等查询时有性能瓶颈，这是我们后续优化需要考虑的问题。

第四个是可伸缩性。比如游戏行业，可能白天和晚上出现数据的谷值和峰值，这就需要我们去弹性地解决流量波动问题，降低低流量场景下闲置资源的成本，同时快速扩容峰值所需要的集群配置。

针对这些问题，阿里云团队推出了 Elasticsearch 日志增强版，专门针对 Elasticsearch 内核做了一些计算存储分离相关的改造，从而能让整体写入性能得到提升。同时，通过底层映射同一块物理盘，我们在集群伸缩的时候不需要迁移里面的数据，实现秒级弹性扩缩容。

此外，我们还实现了索引分片一写多读，数据只保存一份，以及通过云存储保证了数据的可靠性，通过 IO fence 机制保证了数据的一致性。



四、什么时候用日志增强版?

当日志量达到 TB 级的时候，我们就建议使用日志增强版。此外，在增量日志并发高的时候，其峰值写入能达到 10W docs/s，并且会自动帮用户进行副本存储，保证数据不会丢失。

五、Elasticsearch 能给客户提供什么样的场景化解决方案?

有一些行业对全观测有特别的需求，比如游戏和教育行业。

现在线上直播课程很火，但是过程中学生可能会产生很多行为动作，比如进入和退出直播间、评论、对话等，而如果这些动作没有响应，我们就需要对整体链路进行测试、追踪和定位问题。而且，在线教育行业并不仅仅提供给内部，还会提供给外部第三方或平台方，这就更需要保证链路可以监控和观测。另外，在游戏行业中会产生了大量的日志数据和用户行为数据，我们需要更好地对其进行追踪、检索和做后续的分析，以推动游戏的进一步优化。以上这些，都是全观测所能提供的场景化解决方案。

六、有哪些场景痛点?

1. 流量波动，集群缺乏弹性。我们刚才说，集群需要能对业务进行适应性的收缩，如果是搭建开源 ES 服务方案可能会面临供应链成本冗余或不足的问题，而且因为没有专门的团队去运维集群，会导致运维成本高或降低业务开发者精力的问题。

2. 链路冗长，问题难定位。刚才我们说要追踪端到端数据链路的问题，如果因为链路冗长、监控不完备导致异常定位的话，成本也会非常高。

3. 高稳定性要求，成本高。就整体服务而言，这套系统有很高的稳定性需求，因为不止向内部提供，还要给第三方使用。

4. 搜索要求高。不仅是全文检索，还有非文本检索需求，这会导致搜索复杂。

七、产品有何能力?

1. 体系化产品能力。除了提供在日志场景下的内核能力之外，云上还有数据和服务高可用的能力，帮大家去实现了数据的存储加密和安全管控。

2. 数据时效性。毫秒级的数据时效性，全链路数据监控秒级监控。

3. 多云灾备解决方案。

4. TCO 成本优化。云上我们可以帮大家进行很多场景和性能上的调优，降低用户因配置不合理导致的资源浪费。

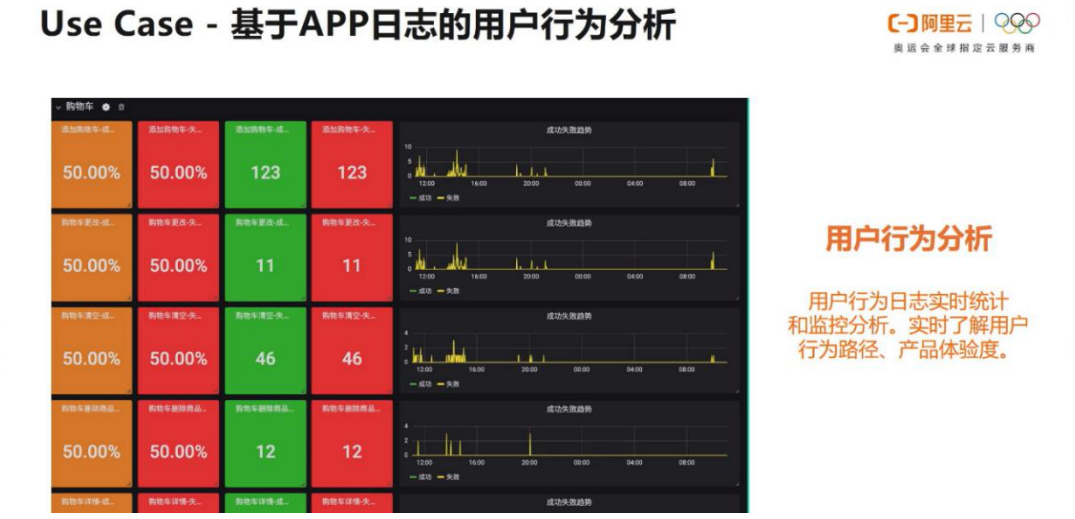
5. 专家级服务。云上我们有很多开发 Elasticsearch 和运维大规模集群的专家，能针对用户的实际使用场景进行解决方案和架构的优化，解决技术难点。

八、用户案例：基于日志的业务数据监控

不只是教育行业，在很多场景下，我们能搭建这样一套业务的监控看板，对各类业务数据进行监控，比如入侵监测、流量监测、交易额监测等。

九、用户案例：基于 APP 日志的用户行为分析

在游戏、电商和零售行业，他们会基于 APP 日志进行用户行为分析，我们可以在这里看到不同用户的行为路径和操作，并针对这些情况进一步优化，提升用户的产品体验。



当然，Elasticsearch 的场景不只是日志和全观测这一套，它还包括很多上层的检索能力，能应用到很多行业中，比如在电商行业中做商品和数据的检索，进行一些订单处理和业务数据分析等等。后面也会有专门的架构师和工程师对 Elasticsearch 进行分享，请大家继续关注我们的课程。

全观测技术原理与技术生态

摘要：本文从理论和技术层面介绍了全观测的技术，包括全观测与可观测的区别，如何实现可观测，如何构建可观测，可观测每一步所存在的问题，以及全观测如何解决这些问题，它又有哪些工具可以使用等进行了介绍。

观看视频：https://help.aliyun.com/document_detail/190545.html

分享人：朱杰

本文主要从理论和技术的层面介绍全观测的技术。

首先，监测和监控是有很大的区别的。

监控主要负责最上面两层的告警和系统概况，它的信号总量比较小。但是拿到监控后，我们并不知道系统发生了什么，所以需要结合日志系统、指标系统甚至 APM 系统进行排查，找到线索并进行剖析，甚至找出服务间的依赖关系。因此从可观测性的角度讲，我们要探查的内容要远大于监控范畴，且获得的信号总量也层层递增，数据量越来越大。

在谈全观测之前，我们先谈谈可观测。

构建可观测性有 4 个步骤。第 0 阶，我们会构建检查各个系统健康状况的检查机制。之后，我们会搭建采集系统各种性能的指标。然后，搭建集中化的日志平台，把所有系统的日志进行汇总并做一定程度的关联，帮助解决问题。最后，是涉及到应用的分布式性能的追踪，它要求更高，往往能从代码层面、API 层面直接度量性能的各方面指标。

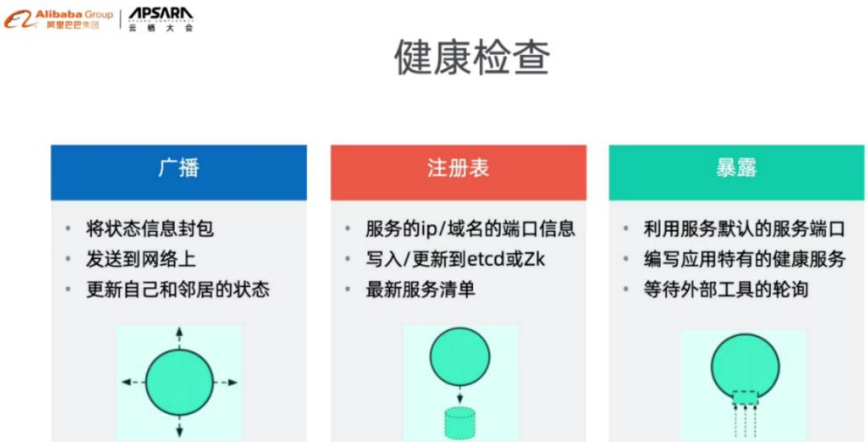
一、每一步具体是如何做的?

首先是健康检查。

健康检查有几种方法，第一是广播的形式，也就是把自己系统和邻居系统的状态信息发送到网络上，而接收端在收到广播包后就会获得这个系统的性能状态；

第二个模式是注册表，就是把服务的状态注册到中央的注册表系统里，比如 etcd 或者 Zk。当把服务状态写进去后，就相当于把自己的观测性暴露了，所以监测系统就可以从中心系统获得信息。这种模式在分布式系统中非常常用，可以通过查询中央注册表获得集群中每一个节点的状态；

第三种模式是暴露。我们在实现自己的应用服务器时，可能会设置一组查询健康度的 API 对外暴露，通过外部工具轮巡、调用 API 就可以获得这个系统的观测性。Elasticsearch 就有这样一组 API 去暴露它的健康度。

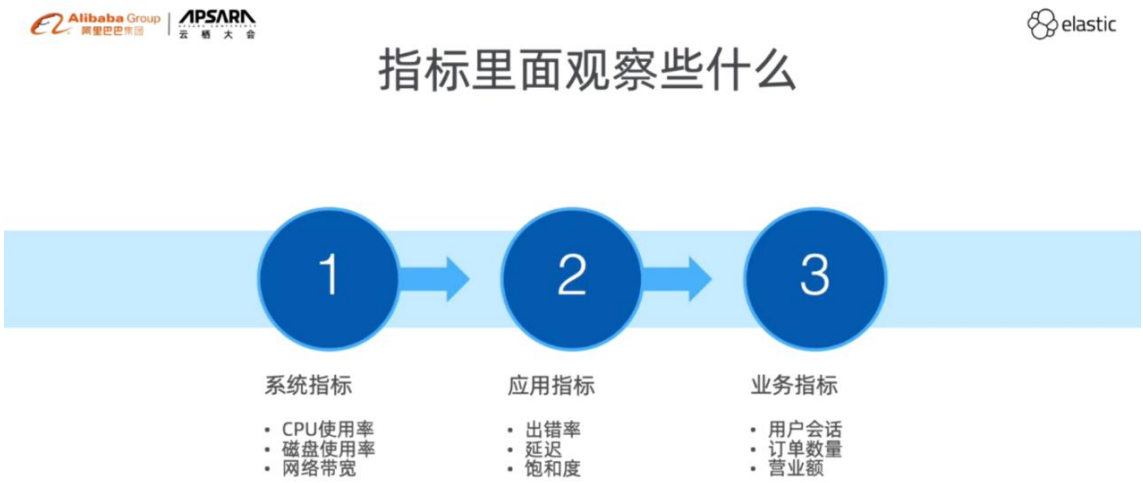


第二是指标，我们能在指标里观察些什么？

在建指标系统的时候我们会收集这几类指标。第一是最基础的系统指标，包含 CPU、网络、磁盘等，这些性能指标至关重要；

系统层之上是应用级别的指标，我们在做应用开发的时候要有意识地暴露很多指标，否则就不太好观测。这里面包含出错率、延迟、饱和度等应用的性能指标。它的暴露方式也可以用 API 的形式来调用，让外围系统轮巡，但更常用的是通过日志的方式去打指标。比如在交易系统中，我们会把与这笔交易相关的原数据打到日志中，然后能通过日志分析了解系统的健康度。另外，现在流行的做法是打成像 Json 这样的结构化日志，这对后续的日志处理有很大帮助；

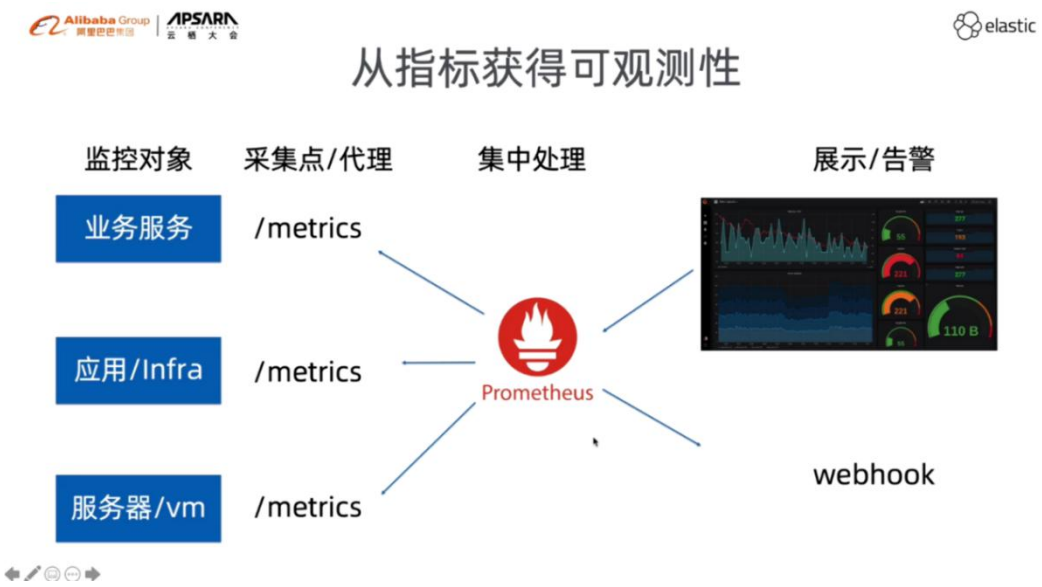
再上面，是业务性的指标，它会涉及到很多 BI 的分析，比如处理的订单的数量、营业额等。如果有业务指标的暴露，就可以反映系统支撑的各种业务量数据，这对运营人员比较重要。



如何从指标获得观测性？有哪些方式？

当前，像 Prometheus 这种系统应用得比较多，它的原理就是 Prometheus 提供的 agent 从各个采集点采集结构化数据，写入 Prometheus 数据库，然后基于一些开源的工具进行指标的可视化展现。从告警的角度，它也可以写一些告警的规则，通过 webhook 等对外告警。

这是做指标系统比较常用的一套堆栈。



第三是日志。如何从日志获得可观测性?

日志很多就是一串字符串，所以要从这里面获得可观测性，很重要的一步就是进行结构化的抽取工作。从下面的图中，我们可以看到各种各样的字段，比如 IP 地址、时间戳、进程号，以及具体请求的 URL。对于日志系统而言，就需要把文本的字符串变成结构化数据。

而要获得更好的观测还要注意，你的日志能够暴露多少状态，就意味着对你的监测程度能做得多好。比如下面这个例子，除了常规的发生时间、日志信息外，还有它提供的服务、它的开发团队信息、谁受到了影响等。这些对后续的分析、运维和提升服务质量都非常有帮助。所以从运维角度，要追踪哪些性能、哪些数据应该打在日志里是开发人员要进一步探索的。

第四，分布式追踪。如何从分布式的调用中获得可观测性?

APM 是从微观的代码层面捕获各种数据，所以它获得观测性的关键就在于安装 APM 探针。它能从每个代码块的层面帮我们度量每一步花费的时间、捕获报错等等。比如能把一笔事务处理的链路追踪到每个系统中，把它串联成一个瀑布图呈现给大家。这对开发人员去追踪问题，运维人员去定位问题都是非常重要。

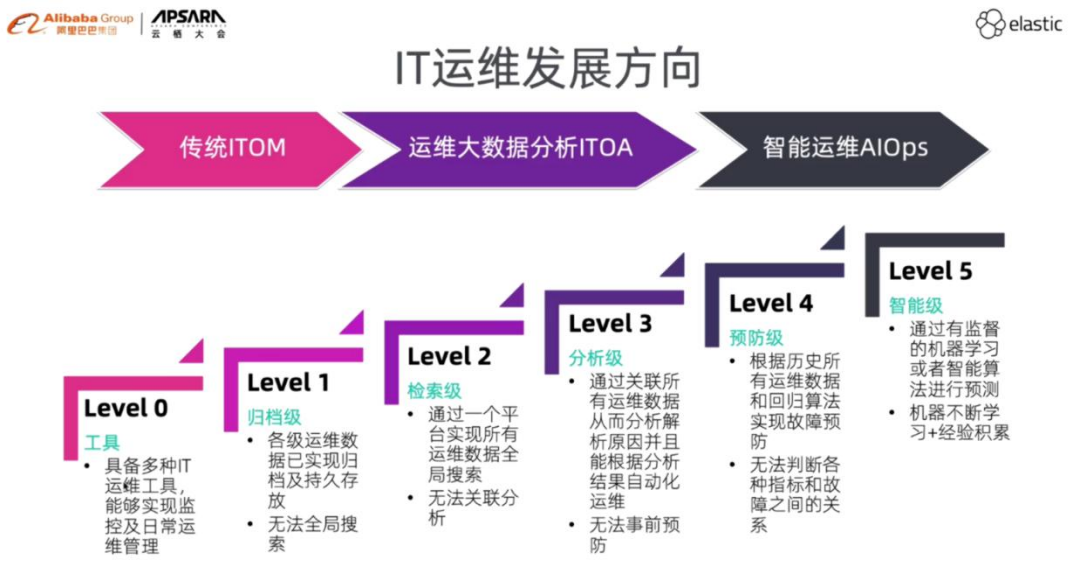
说完如何获得可观测性，下面讲讲如何去建设。

前两个 level 比较简单，比如做日志，把日志集中化归档到文件服务器上就算做完了。当我们上了 ELK，把日志集中化之后，我们就能很轻易地在日志中做检索，达到检索级。

不过检索级是事后的故障分析，从运维角度，我们要获得趋势的预测就要达到分析级的目标，分析级的特征是我们能看到各种分析图标。而从检索级到分析级的鸿沟在于，数据的结构化程度有多高，结构化程度越高，能呈现的图表和编写规则就越多。APM 系统因为采集的已经是结构化的数据，所以不需要复杂的加工，难点在于对日志的结构化提取。如果能够实现，后续就能进行聚合运算，产生丰富的图表，并编写很多规则。

有了分析图标和各种规则，就能建立一套预防的运维机制，这样系统初现端倪时就可以进行告警。而到了更高级，就可以通过加入机器学习等技术实现更智能化的预测。

构建可观测系统就是从初级开始，一步步走上高级。



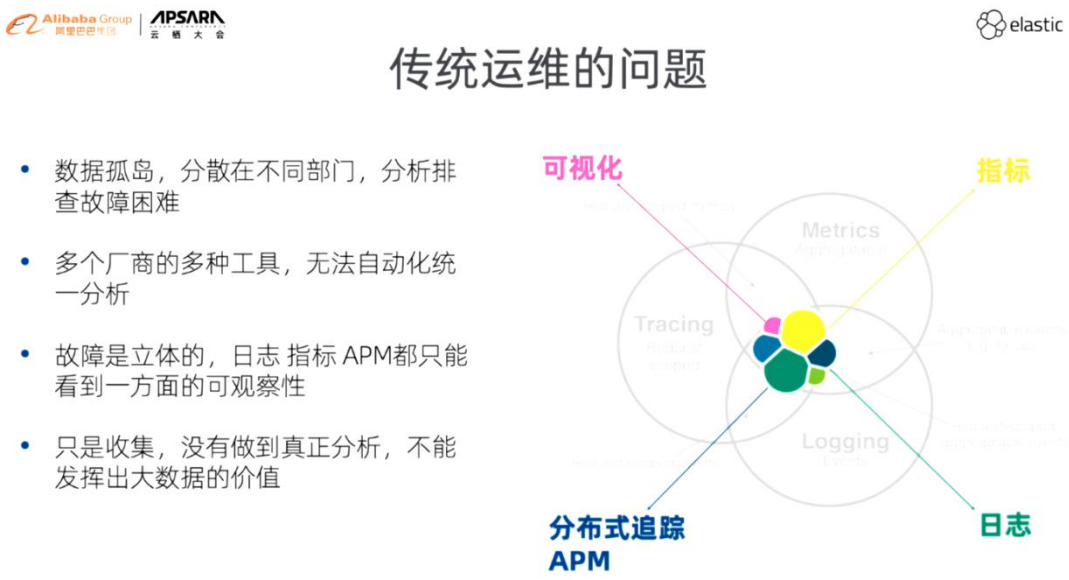
另外，从每个维度看，每一个级别里处理的内容会不太一样。

数据收集角度，从初级的日志集中化和分析，到中级的结合日志和指标综合判断系统状态，再到高级的融合日志、指标、APM 数据进行判断；数据准备角度，从初级的不抽取，到中级逐渐积累和解决故障、编写规则、实现少量结构化，再到高级积累更多数据、规则和告警，以及引入机器学习等帮助异常检测。

以上，我们谈了怎么可观测，以及如何去建设，下面谈谈全观测。

全观测其实是对传统运维的改进。

像上面所讲，传统运维是一步步进行搭建的，每一步都会出一个开源或商业产品，这会导致产品间出现数据孤岛的状况，非常割裂；第二是有各种厂商的工具，导致很难做自动化统一分析，甚至它们的 API 都不一样，严重制约了我们构建各方面观察的自动化平台；第三，每个方面只能提供一方面的观察，而故障往往是立体的，可能要多方面观察才能定位到具体的故障；最后，很多系统只是做了收集，没有真正进行分析，没有发挥出大数据的价值，也没有改进运维质量。



所以 Elastic 提倡的全观测，核心就是把包括日志、指标、APM 甚至 Uptime 这些数据汇总到一个平台上，让运维人员、开发人员，甚至是业务人员都可以在统一的大数据的平台之上，对所有的数据从统一的视角进行观察，进行统一的告警，以及进行统一的可视化。

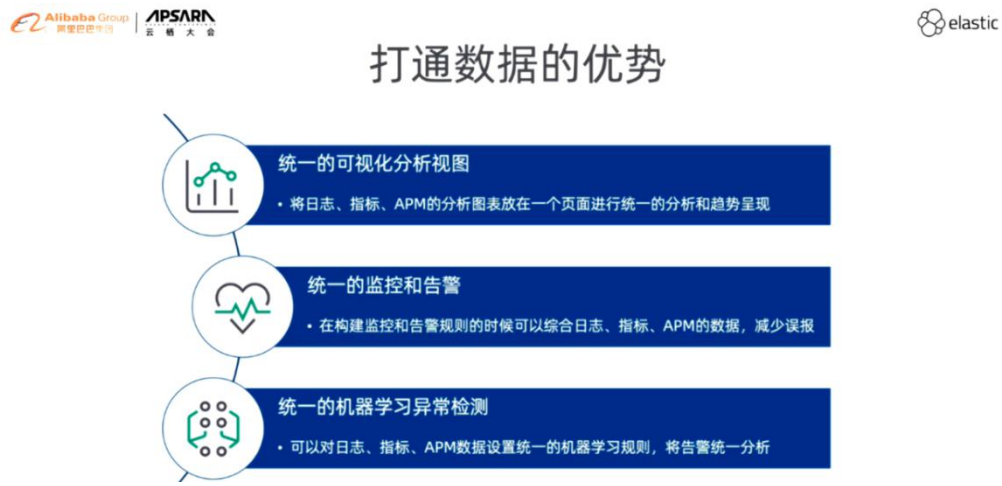
这套平台建立在 Elastic Stack 之上，核心是 Elasticsearch，而之上的 Kibana 也会提供像日志、指标、APM 各种应用。同时，Elasticsearch 也推出了像 Elastic Common Schema 这样的一个命名的规范，能够更好地去分析各个数据的来源。

二、打通数据有哪些优势?

第一，统一的可视化分析视图。可以把日志、指标、APM 的分析图表放在一个页面进行统一的分析和趋势呈现；

第二，统一的监控和告警。由于所有的数据都在这儿，所以在构建监控和告警规则的时候可以综合日志、指标、APM 的数据进行综合判断，减少误报；

第三，统一的机器学习监测。机器学习可以帮助监测大量指标，不管是分散在日志、指标还是 APM，都可以统一监测。

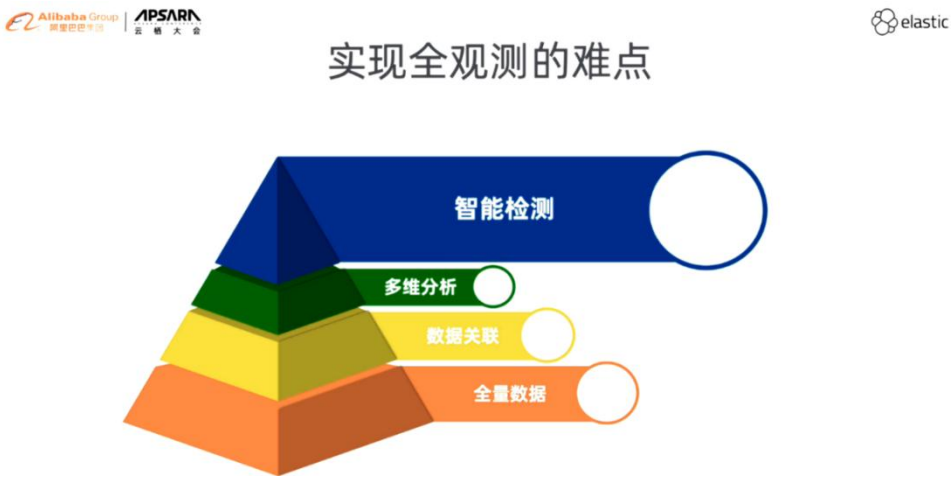


现在很多厂商也在想打通数据，比如日志厂商想融合指标和 APM，APM 厂商想把日志和指标融合进来。但这是否容易实现呢？实际上有几个难点。

三、实现全观测有哪些的难点?

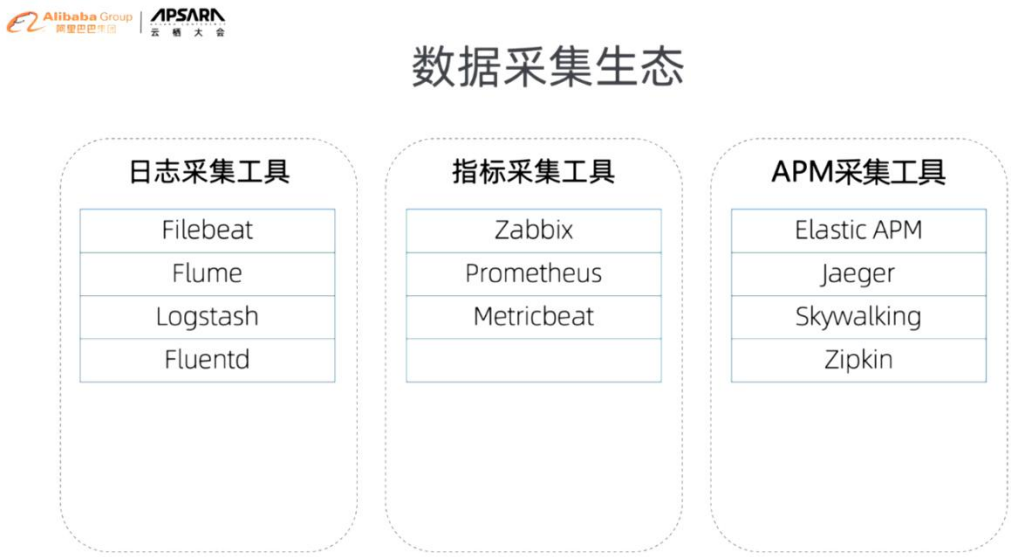
首先最大的难点在数据量。日志、指标这些数据量都很大，这就需要一个分布式的系统去做。另外因为因为要检测很多东西，所有要有数据的关联，进行多维度的分析。这个地方的要求更高，不仅要能够动态地生成字段，所有的列要能够索引，而且要能够灵活地写各种各样的查询。

所以，很多厂商会选择 Elasticsearch 作为底层数据引擎，因为 Elasticsearch 本身是分布式的，能够容纳海量的数据。第二个，它也有大量的多维分析的灵活度存在。同时，它还能进行智能化的检测。



要做这样一套全观测的系统，有哪些技术可以用到?

在数据采集生态中，我们可以用到以下这些工具，包括日志采集工具、指标采集工具和 APM 采集工具。



同时，数据存储搜索工具生态有如下工具，包括关系型数据库、时序数据库和 ES 搜索引擎，我们能从下面这张图看到它们各自的优势和劣势。

以下是分析展示工具。比如日志数据分析展示的 Kibana，配合 Prometheus 的 Grafana，还有专门做 APM 数据分析展示的 Skywalking。

现在我们讲究集成到一个平台上，这样日志、指标和 APM 就可以进行分析的联动和跳跃，而 Kibana 现在就能做到在一个平台纳入分析这三方面的数据，并且进行数据的跳跃和联动。

最后，是全观测的主要流程。它包含数据采集阶段、数据处理、数据搜索存储和可视化几个步骤。

在数据采集层，我们能用上面的各种工具对日志、指标、APM 进行采集，然后将其汇聚到 Kafka；在数据处理层，用相应的数据处理工具从 Kafka 进行消费；随后，数据经过各种各样的处理，流入到数据存储层，在 Elasticsearch 里对数据进行索引；最后，可以通过 Kibana 或第三方工具进行可视化展现。不过，可视化只是帮助我们进行人工监控，如果要做到自动化，就一定要安装各种各样的规则，能够进行基于规则的和基于机器学习的监控和告警。

以上就是全观测的基本原理和能够用到的一些工具，欢迎大家继续关注我们后续的课程。

全观测能力呈现与应用价值

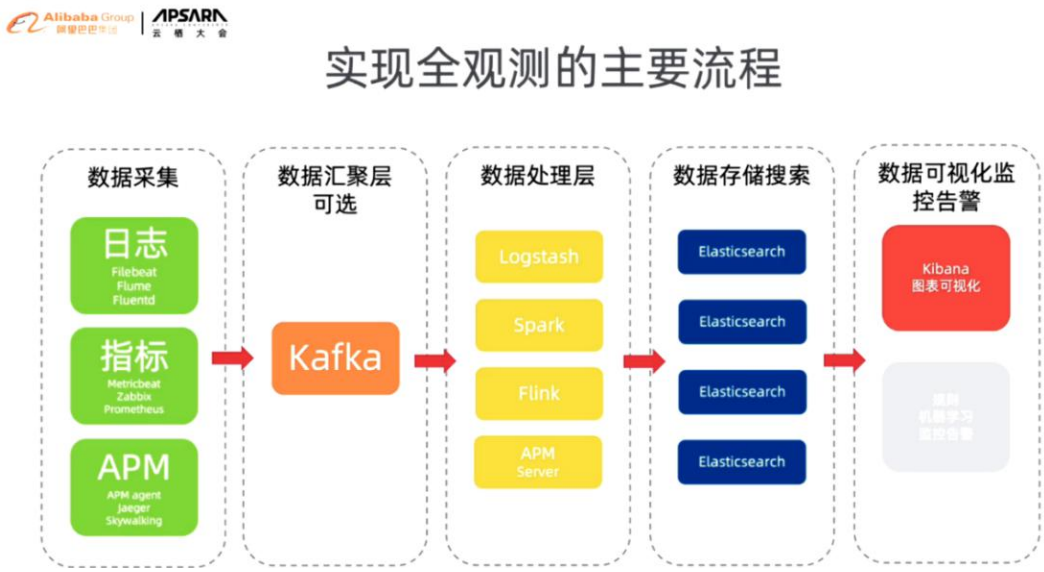
摘要：本文承接《全观测技术原理与技术生态》，介绍 Elastic 整套工具带来的能力，以及用 demo 展示怎么用这些能力构建全方位的观测性。

观看视频：https://help.aliyun.com/document_detail/190546.html
分享人：朱杰

实现全观测主要有以下步骤：

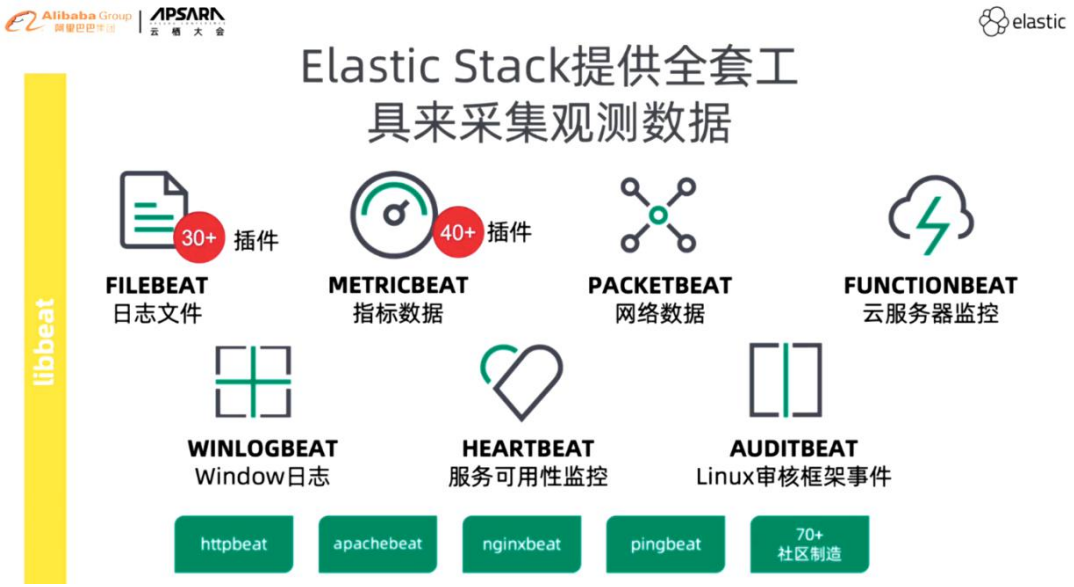
第一步是数据采集，这些数据可能来自日志、指标或 APM，每一种数据都有对应的采集工具；数据采集完成后，会进到 Kafka 这样的数据汇聚层，以经济高效的方式保证海量数据的涌入；随后在数据处理层，有 Logstash 等工具对数据进行加工、转换、清洗，变成结构化数据，然后交由 Elasticsearch 进行存储和提供搜索能力；最后，可以通过 Kibana 进行数据的可视化，以及用机器学习进行告警。

目前，除数据汇聚层的 Kafka 外，Elasticsearch 提供的工具已经贯穿了全观测的全链路，能在各环节提供相应的工具。



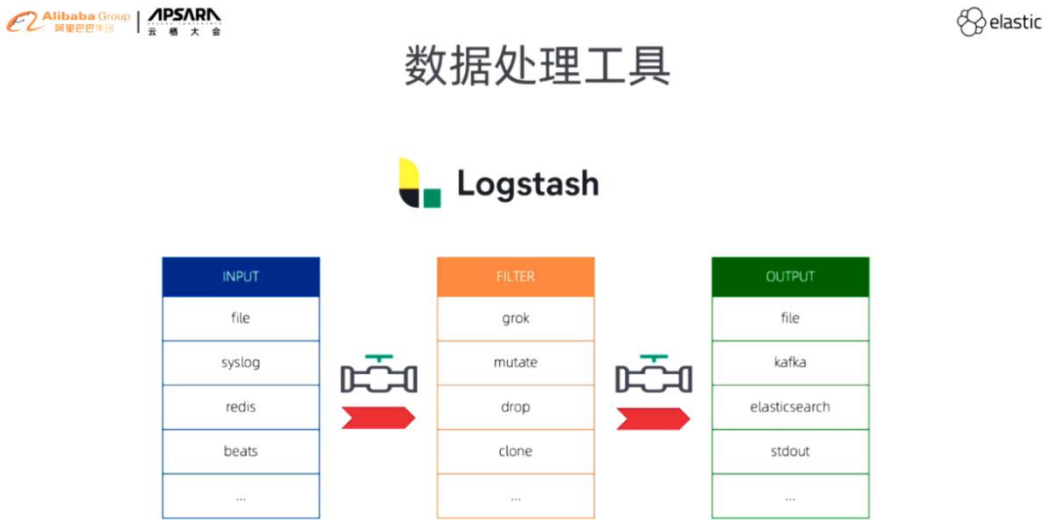
一、Elastic Stack 提供的数据采集全套工具

我们有丰富的 beat，比如采集日志文件的 Filebeat，有 30 多插件，能做到一定程度的开箱即用；Metricbeat 采集指标和底层的性能数据，有 40 多个插件；Packetbeat 从网络包层面采集数据；Functionbeat 主要对接在云端吐出来的指标日子；Winlogbeat 主要适配 Windows 上的日志系统；Heartbeat 主要检测服务的可用性，比如检测 API 是否在线等；最后，Auditbeat 可以连到 Linux 的 audit framework 来采集 Linux 各种各样的事件，汇总到 Elasticsearch。除此之外，我们的社区也制造了很多 beat，大家可以去看看。



二、数据处理工具

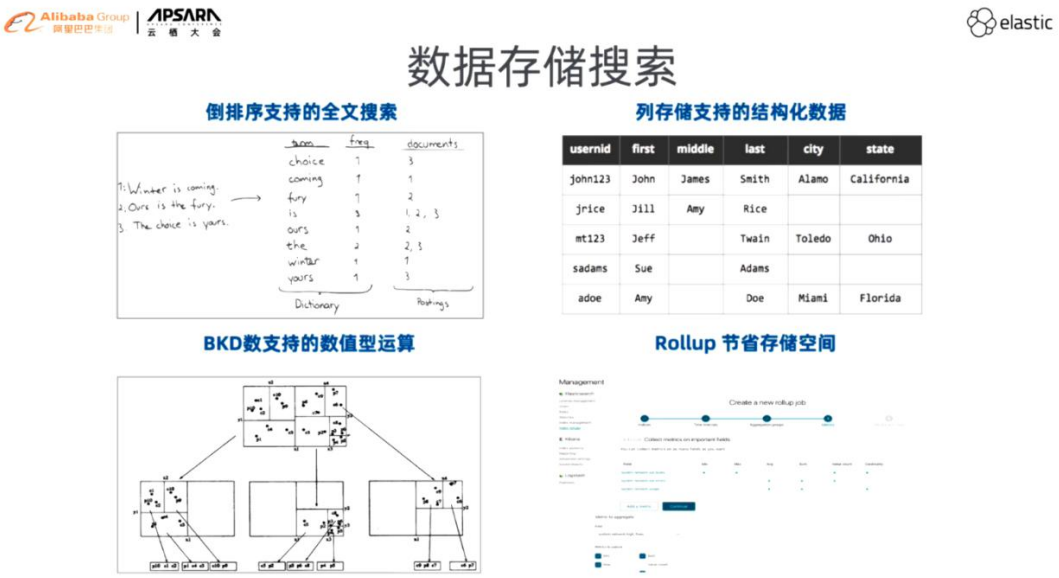
数据处理工具提供的是 Logstash，它分为输入、过滤和输出三个部分。它并不是独属于 Elasticsearch 数据输入和输出的工具，它有很多数据接入源，比如 syslog、redis 等，输出也可以到 Kafka、Elasticsearch 和其他数据库，而它的过滤部分主要体现在数据的加工和处理，比如用 grok 进行正则抽取。Logstash 能很容易地把像日志一样的流式文本抽取成 Json 的结构化数据，进而给后面的 Elasticsearch 进行存储和索引。



三、数据存储搜索工具

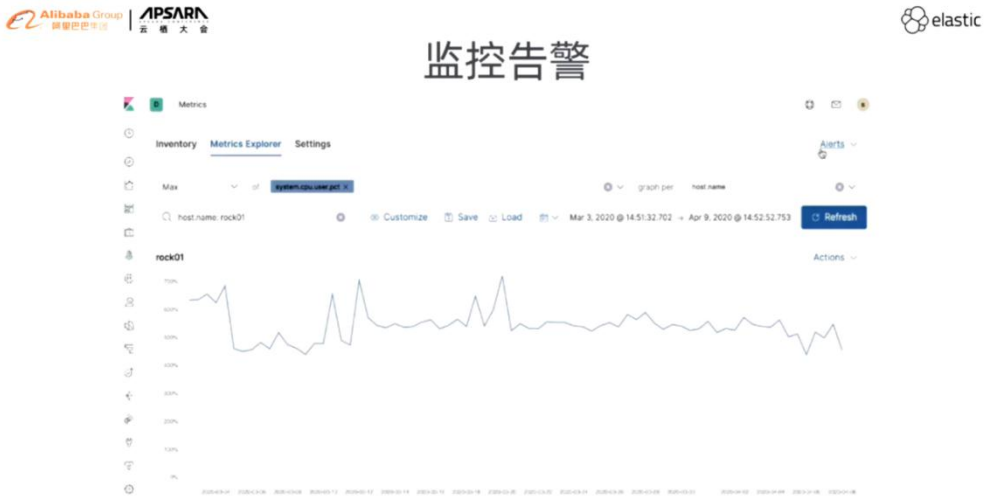
这部分主要由 Elasticsearch 提供核心的功能。Elasticsearch 经过了一系列演变，从倒排序仅支持全文搜索，到列存储支持结构化数据，加速排序聚合，再到 BKD 树支持的数值型运算，提升数值类型的范围搜索效率，以及数据上卷节省存储空间。

经过演变，Elasticsearch 能支持结构化的数据搜索和聚合，还有全文搜索、地理搜索等能力。这种丰富的处理能力可以应用到全观测的应用场景，产生很多有价值的图表分析。同时，它还有自动化的监测监控，并且执行告警。



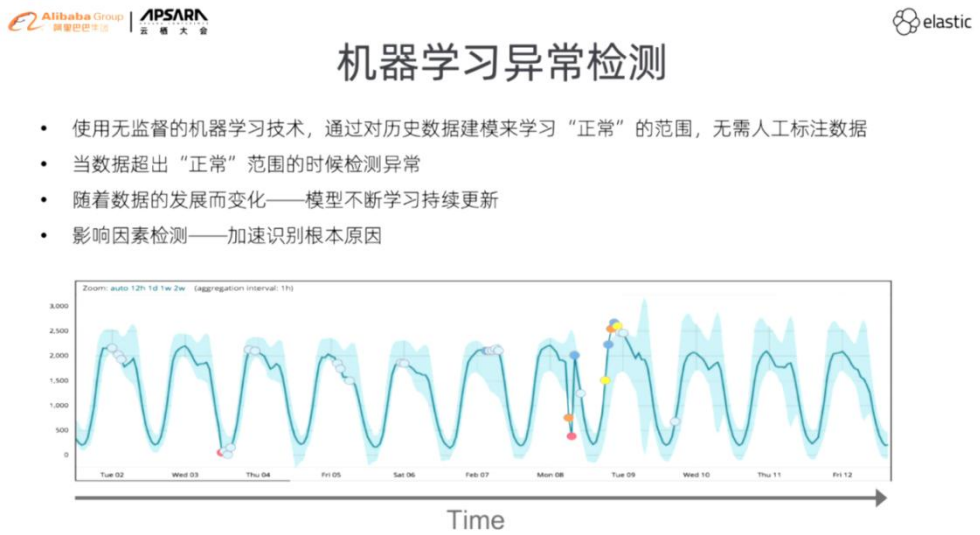
四、告警系统

全观测需要持续部署大量的监控规则，来自动化地进行监控和告警。我们在 Kibana 里植入了新的告警系统，它能跟上层的各种 APP 和解决方案进行无缝整合，大幅简化使用门槛。



除了基于规则的告警，Elastic Stack 还提供了机器学习的异常检测。

Elasticsearch 中存在大量的指标数据，它们随时间序列的波动是非常常见的，但当指标数量越来越多，就很难用传统的方式一个一个地设置规则。所以我们利用机器学习，通过对历史数据的建模去学习正常的波动范围，不再需要人工来标注数据。同时，模型也会根据数据的持续写入来不停地更新，以反映最新的指标状态。



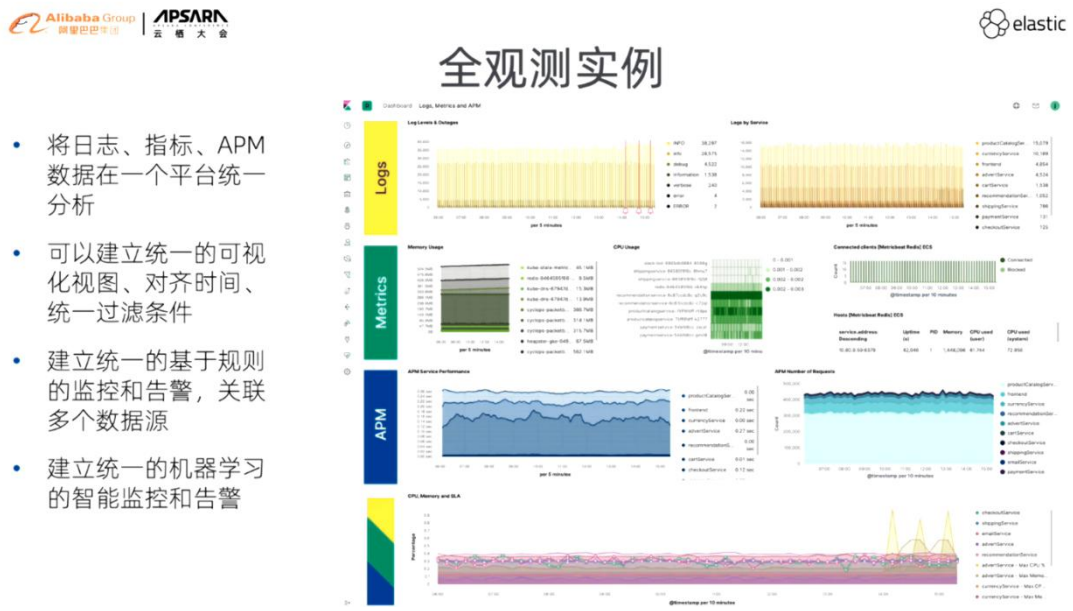
五、数据分析和展示工具

Kibana 经过演化现在已经拥有了丰富的可视化展示能力，并且我们引入了 Kibana Lens 这样更加便捷的制图工具，同时也不断添加各种图表类型，帮助展示 Elasticsearch 里的各种数据。

六、把工具组合起来使用——两个全观测实例

下面这张仪表板的图，融合了日志、指标和 APM 数据，能进行统一的过滤、搜索和展示等。我们能从多个维度进行统计和观测，同时，日志、指标和 APM 数据也能对齐时间，对照着进行分析。

另外因为所有的数据都汇聚到这儿，所以我们可以建立统一的基于规则的监控和告，并通过参考三份数据源，来判断是否要触发某一个告警，减少误判。此外，我们还能建立统一的基于机器学习的智能监控和告警。



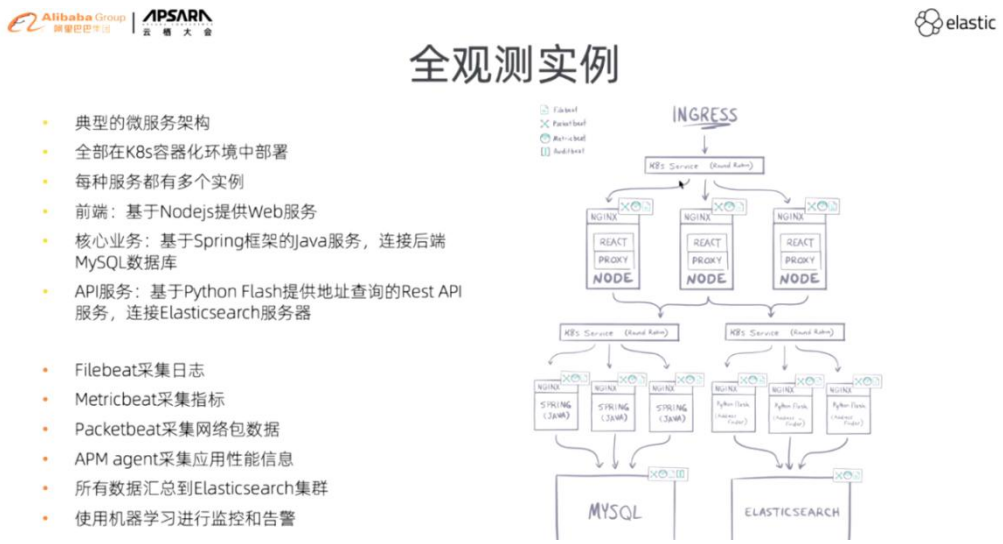
再看另外一个较复杂的实例。

下图呈现了当前常见的微服务架构，它所有的服务都部署在 K8s 容器化的环境中。在前端，它全部基于 Nodejs 提供 Web 服务，而核心业务是基于 Spring 框架的 Java 服务，并连接到后端 MySQL 数据库，同时它还有基于 Python Flash 提供地址查询的 Rest API 服务，通过连接 Elasticsearch 服务器实现全文搜索的功能。

那么我们如何用上面的工具对这一架构进行监测呢?

首先我们采用 Filebeat 去采集每一个 Pod 的日志，把它们汇总到 Elasticsearch 里，然后通过 Metricbeat 采集系统的性能数据，以及把 Packetbeat 安装在某些 Pod 中采集网络包数据，最后是用 APM 探针植入到 Nodejs、Java 代码和 Python 中监控代码层面的各种性能、响应、延迟等数据。

通过这些，我们就能把日志、指标、APM 数据汇总到一起，统一地进行观测。同时，由于数据量很大，所以也将使用机器学习来对里面的性能指标进行自动化的监控和告警。



七、从故障告警到故障定位的流程

全观测定位故障主要有以下几步。首先我们会收到来自机器学习的警告，告知我们可能遇到了问题，随后我们可以点击链接跳转到 Kibana 的分析平台。在机器学习的告警页面，会把告警全部对齐，方便我们去分析各个服务之间的状态和依赖关系等。

在看到告警之后我们可以进行排查，通过跳转到其他的 Kibana 应用来帮助我们进行侦测和定位各种故障。比如跳到 APM 应用程序中，从 APM 的角度观测发生故障时代码层面的一些异常，或者在综合仪表板中统一地观测各种数据，或者在指标的应用中看到发生故障时 K8s 基础架构的情况。

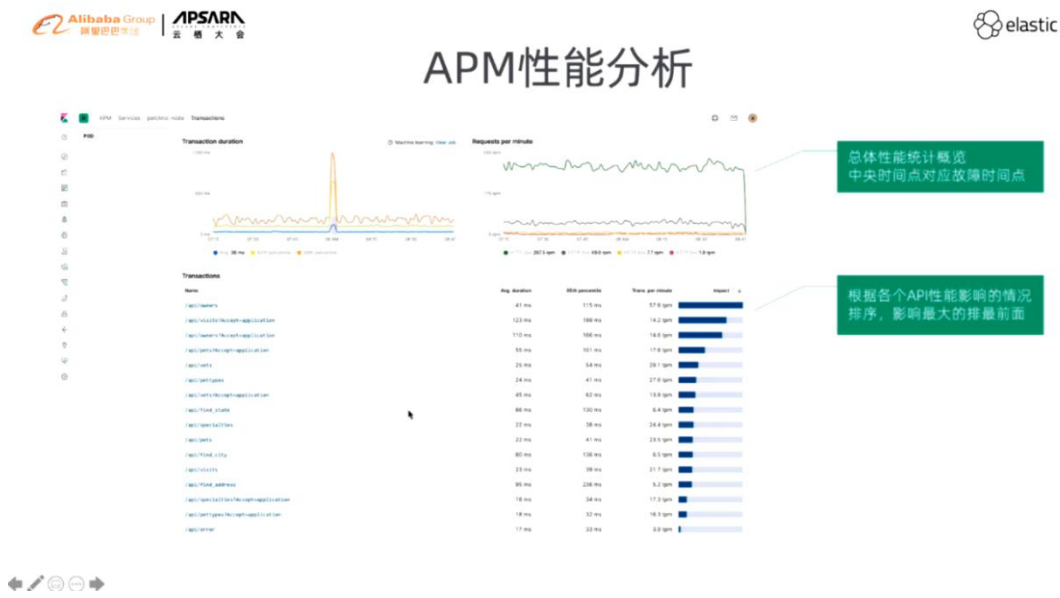
下面我们更具体地来看定位故障的每一步流程。

八、机器学习告警

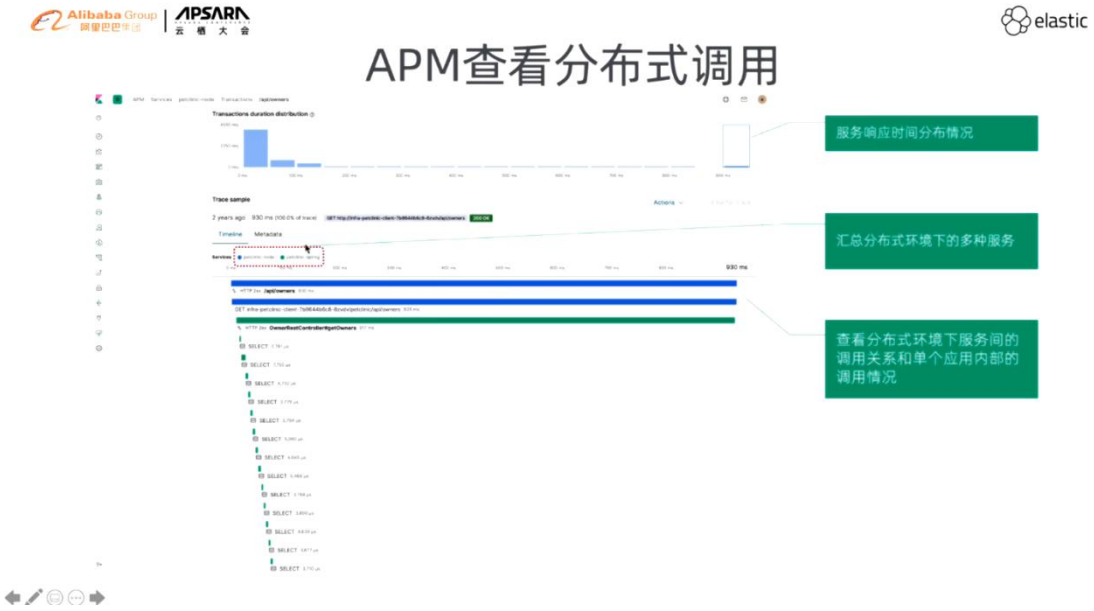
我们能在机器学习告警页面看到很多机器学习的任务，他们能够进行告警对齐。另外，机器学习根据 API 响应时间的历史情况自动建模，当监控值超过动态阈值就触发告警，并且可以指出是哪个 API 性能下降。这旁边还有 action，能引导我们到其他应用中做分析，比如跳转到 APM、仪表板、指标、Uptime 等来诊断这个故障。

九、APM 性能分析

在 APM 层面，我们不仅能看到总体性能统计概览，还能根据各个 API 性能影响的情况进行倒排。

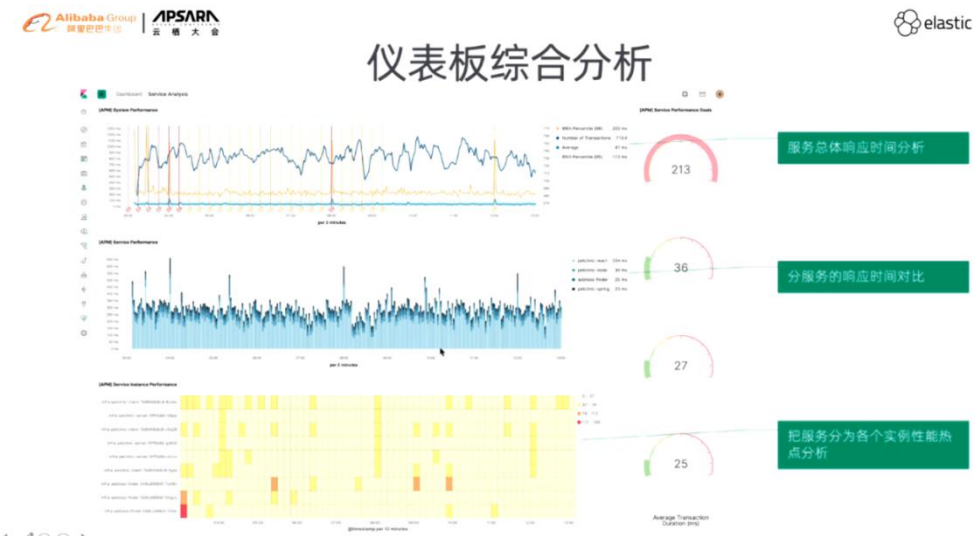


当我们点进 API 后，还能看到它在分布式环境下如何调用堆栈。比如下图中蓝色就代表 Nodejs，绿色代表 Java 代码的调用。可以看到，每一个服务不仅存在多个实例，而且是分布在不同的服务器上，因此这个工具可以把服务流经的所有处理环节串联到一起，实现分布式追踪。



十、仪表板综合分析

在仪表板中也是一样，会把故障的时间点定位到最当中，然后可以参照前后性能状况来综合判断故障的状况。这个仪表板是完全可定制化的，所以可以把各种分析图都放在里面，包括日志、指标、APM 等。



十一、指标关联日志 APM

在专门的性能指标应用中，我们能从主机、K8s、docker 等的角度观看所有性能的切片。这里面也体现了联动的精髓，比如当我们点击某一个 pod，就可以单独看到这个 pod 串联的日志、指标、APM、Uptime 数据，这样就方便我们灵活地进行跳转，更快地定位问题。

另外，当我们点进 Pod 的日志的时候，会进入流式日志分析器，这集合了所有服务器所有应用的庞大的日志流，按照时间戳进行排序。这里面有一个很强大功能就是搜索框，任何符合 ES 搜索语句的都可以写在这个地方，并且能用 and 这个条件继续进行过滤和定位。

ES 全观测性行业应用

摘要：本文主要解决 3 个问题：1.什么是全观测性？2.为什么是 Elastic Stack？3.全观测性行业应用场景。介绍了全观测性的概念，Elastic Stack 技术栈的 Kibana、Elasticsearch、Beats、Logstash 等产品，以及全观测在应用系统、中间件和操作系统的应用。

观看视频：https://help.aliyun.com/document_detail/190547.html

分享人：李猛

一、什么是全观测性

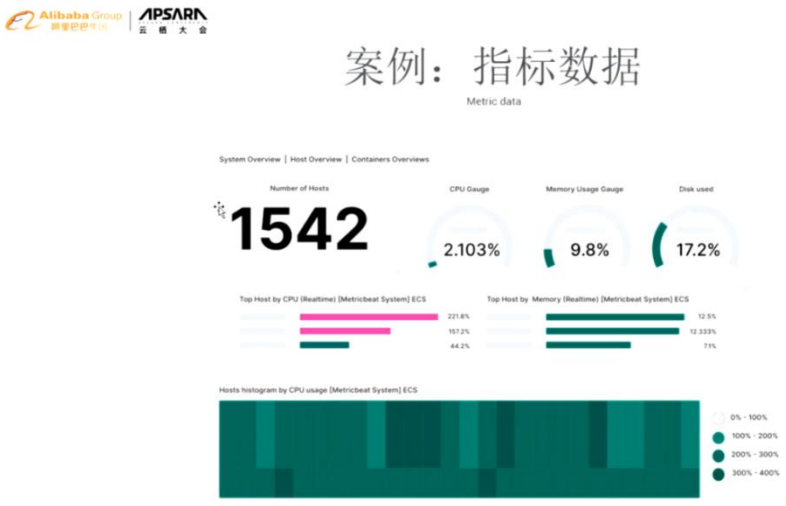
全观测性简单讲就是“监控”、“一体化的监控”。它包括几个方面：一方面叫日志数据，就是文本，第二方面包括一些指标数据，第三方面就是这套产品必须有告警通知。

日志数据

工作开发中日志是免不了的，它一般包含几个重要信息，比如发生时间、发生模块和详细信息等。

指标数据

指标可以理解为文本日志的高级抽象，主要用来记录数值类型的数据，如 CPU、内存图、磁盘等。



告警通知

在日志和指标采集后就要做一些告警规则，比如日志的错误达到某一数量，或 CPU 消耗占到某一比例的时候进行触发；另外，它还要能够发通知给我们，比如通过 Webhook 等；最后，在面对庞大日志和指标数据量的时候，还需要进行智能化，通过机器学习自动生成规则，并在规则基础上自动监测异常。

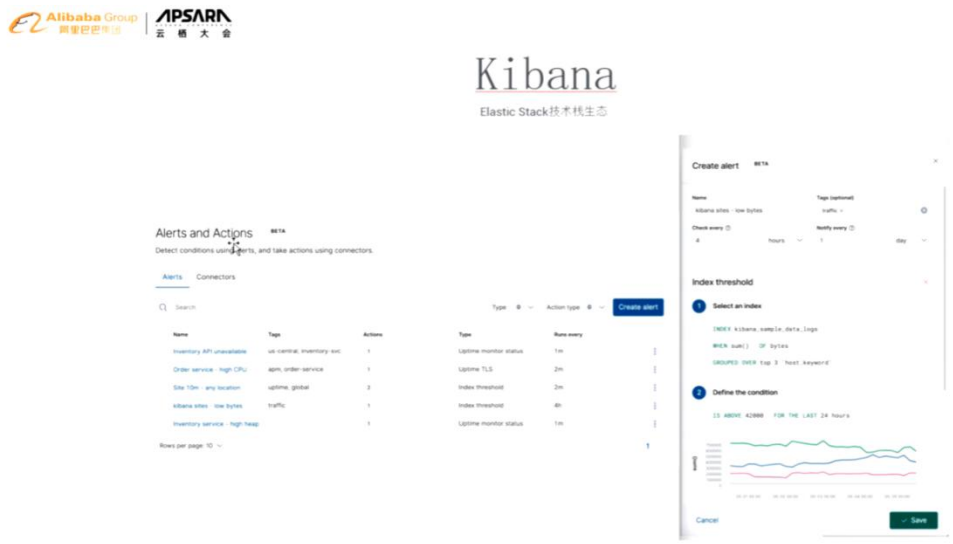
二、为什么是 Elastic Stack?

Elastic Stack 技术栈分由 4 个产品栈组成，包括做 UI 展示的 Kibana，负责数据存储、查询、计算、聚合的 Elasticsearch，负责数据采集的 Beats，以及 ETL 轻量级工具 Logstash。

Kibana

Kibana 能把所有的日志数据集中起来，在一个界面上就能搜索和查看。同时在这里我们可以看到，不管是日志还是指标，它们展示出来都一样。

另外，Kibana 还有个更高级的功能叫 Alerts and Actions，就是用来做安全告警通知。我们可以在里面写一些自定义的告警规则，然后创建一些 action。所以，Kibana 相当于把日志的展现和告警一体化了。

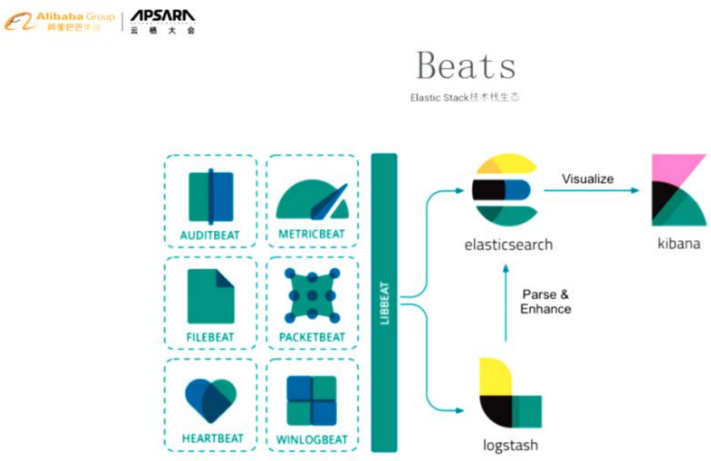


Elasticsearch

Elasticsearch 是我们的核心，它主要有 3 个职责，数据存储、数据查询和数据分析。首先，它能把 Pb 级海量的数据存储到 ES 上；随后，它能基于倒排索引支持海量的数据查询，在做多条件检索的时候它可以说是最快的索引算法；最后，它还能支持行式和列式的分析，比如用行式分析来做明细查询，用列式分析来做统计分析。

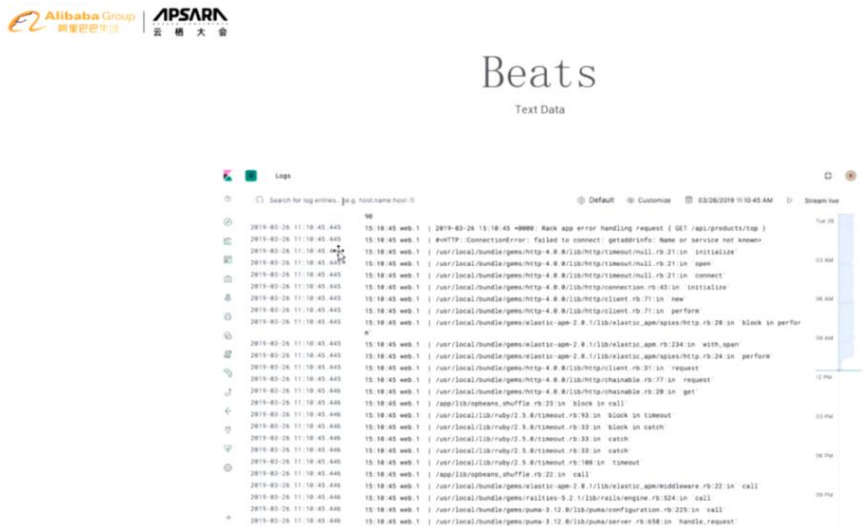
Beats

Beats 主要用于数据采集，非常轻量级，包括 Auditbeat、Metricbeat、Filebeat、Packetbeat、Heartbeat 和 Winlogbeat 几种。为了和 Logstash 做职责区分，Beats 就主要做日志数据采集，但如果遇到一些数据需要做自定义转换，就需要把采集完的数据先传到 Logstash 里进行一些处理再写到 ES 里。



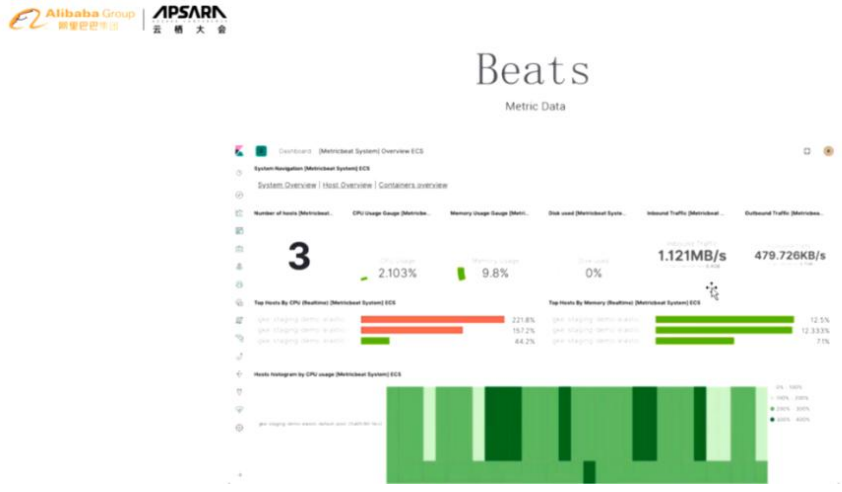
Filebeat

Filebeat 专门采集文本日志，比如服务器和应用上的文本 log。另外，Filebeat 已经写好了规则，非常方便。比如如果要抓取 ES 的日志，它里面直接有一个 ES 的模块，日志格式都帮忙解析完成了，只需要配一个日志路径就能完成采集。所以 Beats 家族更加“傻瓜化”，把 Logstash 的可编程性都替代掉了，直接做成最终落地。



Metricbeat

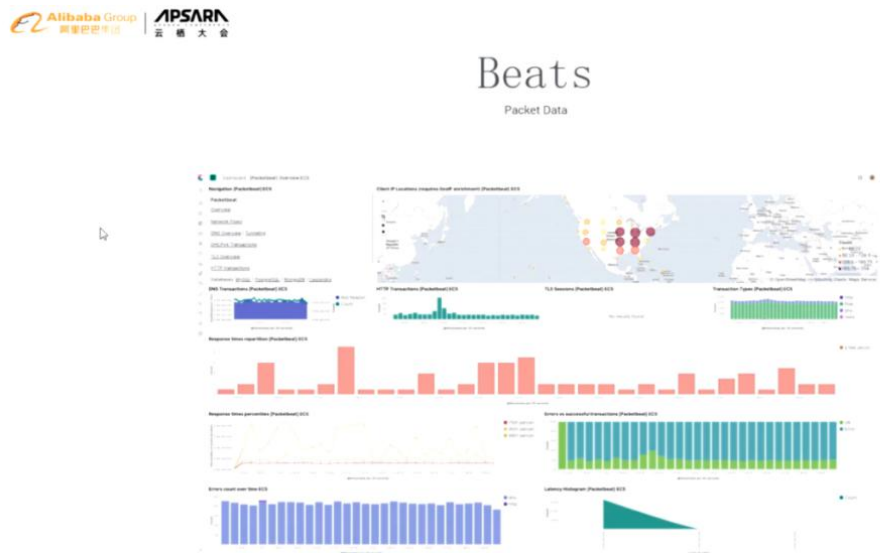
Metricbeat 专门采集指标数据，启动之后直接对接应用、中间件，它们都可以采集。比如这个图里面是采集操作系统的，我可以直接部署在这台操作系统上面，然后就可以监测到操作系统里面到底有多少资源在消耗内存、CPU、网络等等。我们也可以自己拓展里面的配置模块，自己编写规则，自己去采集。



Packetbeat

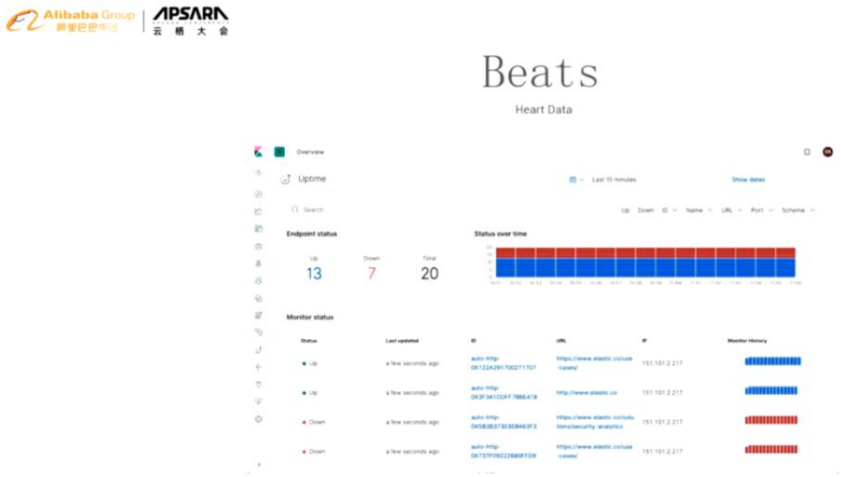
Packetbeat 主要采集网络数据，它比 Metricbeat 采集的网络数据更详细，Metricbeat 只能监测到我的网卡数据进了多少出了多少，而 Packetbeat 能知道网络的数据有多少，朝哪些 IP 去了，有多少是哪些 IP 访问进来。我们可以把它看作是日志数据和指标数据的结

合，因为网络地址是一个字符串，而流量又是指标，所以我们说全观测，不外乎就是这两种数据，而如果再抽象一点，最终只有文本数据。



Heartbeat

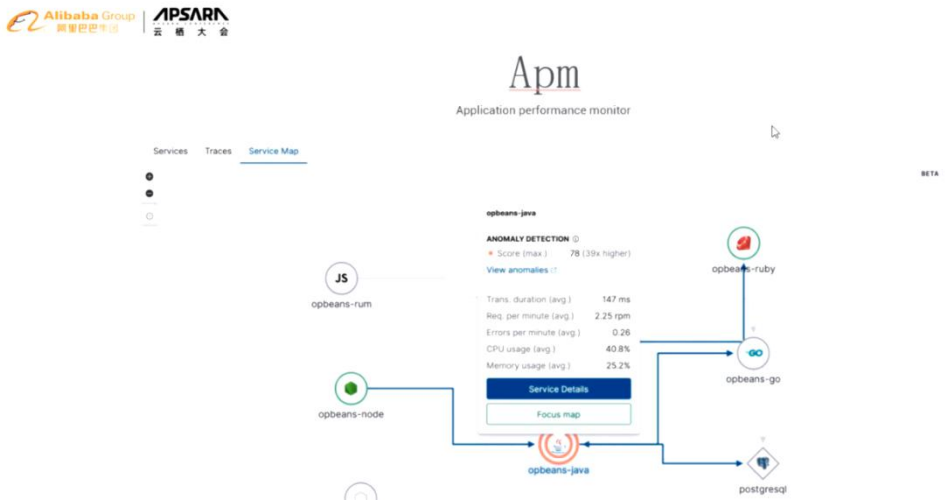
Heartbeat 负责监控应用的心跳。比如 Java 开发现在最流行的用微服务，虽然微服务里面可以监控应用的上线下线，但是它并不是很全面，仅仅只能监控自己的微服务。如果要监控别的，比如一套大型的分布式系统，就需要在 Heartbeat 里面配置。



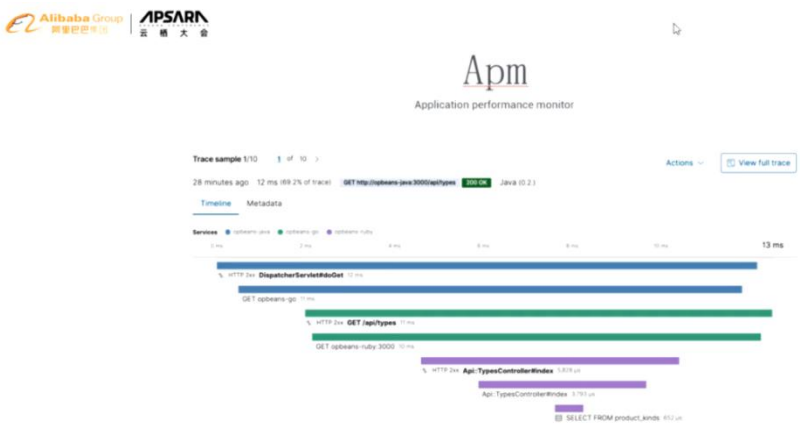
APM

最后再和大家推荐一个 APM，就是应用程序性能监控。这是我们开发人员最关心的，很多公司加入程序后什么监控都没有，请求数量、响应数量、请求时间、响应时间都不知道。但是如果了解过 Elastic Stack 就会发现，它能把你需要研发的东西一网打尽，这个就是 APM。比如这里面举了几个图，你可以看到程序每分钟的请求数量，也可以看到服务的调用链路……

我们一个一个来讲，下面这个是 APM 高层次的调用链路图。如果我的微服务有几十个或者上百个几百个不同的应用、服务，调用的时候就有严格的一个调用链路，那么我们能从这里面看到前段和后端整个链路，这是 ES 推出 APM 自动采集数据之后自动绘制的。

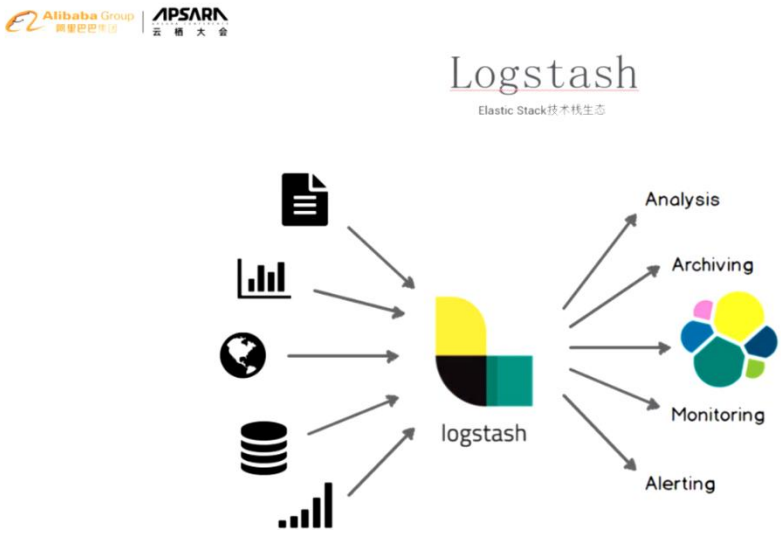


除了上面的应用程序，它服务之间也有绘制相应的图。比如下面你从 Java 调到 Go 再调到 Ruby，它都可以帮你自动来绘制出来，仅仅只需要把你的应用程序部署起来就可以了。



Logstash

Logstash 以前是日志采集的工具，基本上已知的所有数据它都支持，数据库、文本、日志、网络都可以采集进来，然后经过它的中间转换传输到 ES 中去。但是现在我们把它定位成数据的 ETL，比如前端 Beats 采集完数据后放到 Kafka，而数据到 ES 里面又需要 Logstash 来抽取，所以它相当于一个典型的 ETL。



以上就是我们对为什么选择 Elastic Stack 的解释，因为我们全观测性监控需要一体化，需要这么多数据的采集展示，而目前只有 ES 一家做的比较全。

接下来我们介绍一下全观测的应用场景，一些案例。

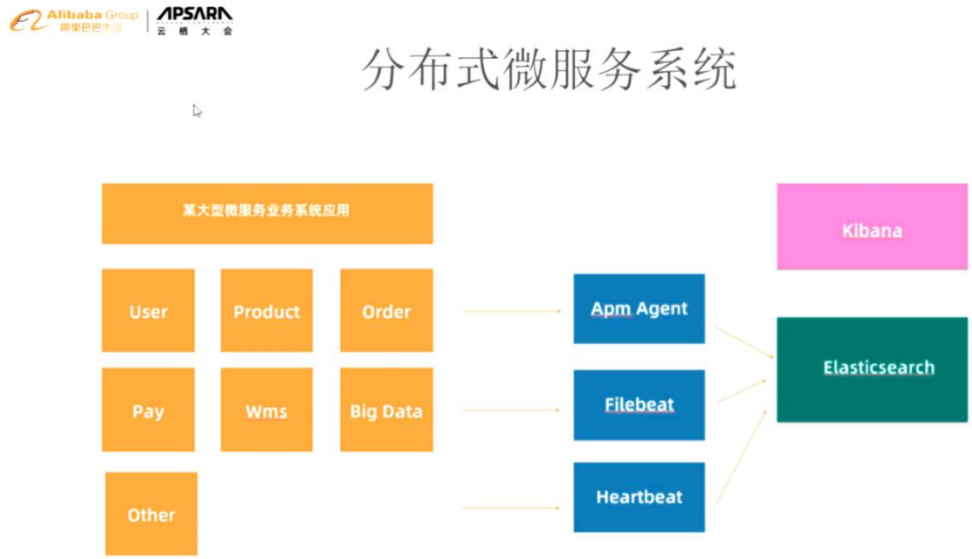
三、全观测性行业应用场景

例 1：微服务平台

现在做传统业务系统开发或者分布式软件都离不开微服务的理念。比如某大型微服务业务系统应用可能要部署几百个，然后在下面拆分为用户商品订单、支付、仓储、数据等等，这些可以按照领域拆分，也可以从垂直或水平方向拆分，而且它们部署的数量都不一样，比如如果用户的服务请求量特别大，它可能就要部署 20、30 个。

这些数据我们最关注的是 APM，就是综合的应用指标，来看它服务的请求数量，哪些服务比较慢，什么时间比较慢等等；第二个，微服务会写入大量的日志文本数据，有的是系统相关，有的是业务相关，我们都要通过 Filebeat 采集过来；第三个，我们会用到 Heartbeat。比如当用户的服务突然下线了，从 20 个下线到 12 个，我们就需要它进行告警。

还有最关键的这些微服务的链路调用，比如 User、Product 和 Order 之间的互相调用，可能调用过程中就把程序串联起来了，这对我们的分析非常重要。目前来说，没有哪个产品能像 ES 一样，把应用开发和微服务需要的这种监控做到一体化。



例 2：中间件平台

中间件是应用系统里面必不可少的，比如数据库、缓存等等。举个例子，比如我们现在做分

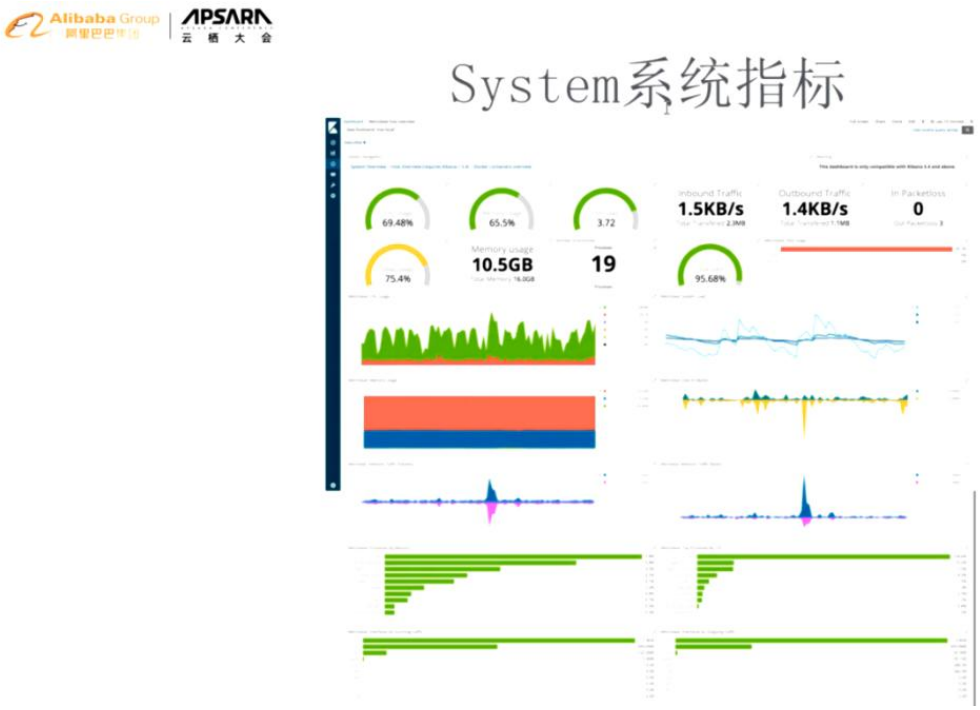
布式开发，要写一堆分布式任务调度，调度任务为了做到高可用就要部署多个高实例，这就要每一时刻只能有一个程序在运行。这里面会用到 Zookeeper 来做 Leader 选取协调，用它把中间件做一个监控，因为任何时候出了故障可能就意味着你的服务程序会失败，运营出错。

再一个就是 Kafka。我们前面讲到微服务调用，几百个微服务有那么多数据采集，数据量是非常大的，所以会引入 Kafka 承担信息缓冲，它非常重要，所以我们也会对它进行完整的链路监控。

例 3：基础平台

我们的服务程序一般是部署在操作系统上面，不管是用云，还是用自己的自主机房，还是用私有云，私有云或者是云平台的监控至关重要。比如网络突然出现瞬间的阻塞怎么办？磁盘空间满了不够了怎么办？这些都需要我们实时地去监控采集。

我们可以从这个系统指标的 demo 图看到，采集操作系统的指标，网络、磁盘、CPU、内存、交换区、负载、应用程序等。因为我们一台服务器实际部署的时候肯定不止一个，微服务可能部署了很多个，所以对这套操作系统的监控也很重要。



以上关于全观测性在具体行业的应用我们基本上讲完了，包括应用系统、中间件和操作系统。其实从个人的职业生涯开发周期来看，我们所有做的监控就是这三个场面。如果你是作为一个普通开发人员，你最关心的可能就是 Java 应用程序、服务接口、处理能力、数据量之类的；如果你稍微资深一点，可能关心 Java APP 的问题；如果你是一个架构师，可能要关心更加综合一点的东西，如果你是运维或者公司的总监，你需要通盘关注，而这就需要一个更加整体的技术平台。

使用 SkyWalking 和 Elasticsearch 实现全链路监控

摘要：SkyWalking 是分布式的应用性能管理 APM（Application Performance Monitoring）工具，也被称为分布式追踪系统。本文介绍使用阿里云 Elasticsearch 7.4版本的实例与 SkyWalking，实现对实例的全链路监控。

背景信息

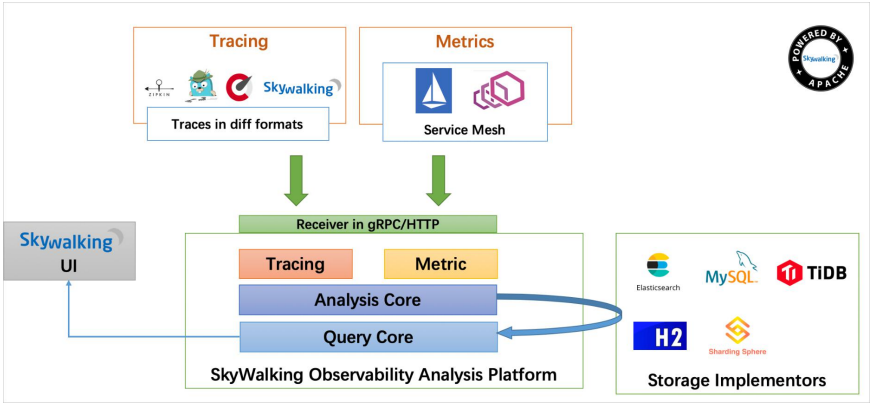
SkyWalking 具有以下特性：

- 全自动探针监控，不需要修改应用程序代码。
- 手动探针监控，提供了支持 OpenTracing 标准的 SDK。覆盖范围扩大到 OpenTracing-Java 支持的组件。

说明： OpenTracing 支持的组件请参见 OpenTracing Registry。

- 自动监控和手动监控可以同时使用，使用手动监控弥补自动监控不支持的组件，甚至私有化组件。
- 纯 Java 后端分析程序，提供 RESTful 服务，可为其他语言探针提供分析能力。
- 高性能纯流式分析。

SkyWalking 的架构图如下。



SkyWalking 的核心在于数据分析和度量结果的存储平台部分，通过 HTTP 或 gRPC 方式向 SkyWalking Collector 提交分析和度量数据。SkyWalking Collector 对数据进行分析和聚合，存储到 Elasticsearch、H2、MySQL、TiDB 等其一即可，最后通过 SkyWalking UI 的可视化界面查看分析结果。Skywalking 支持从多个来源和多种格式收集数据，支持多种语言的 Skywalking Agent 、Zipkin v1/v2 、Istio 勘测、Envoy 度量等数据格式。

说明：本文介绍 SkyWalking 与阿里云 Elasticsearch 7.4 版本的集成配置，您也可以通过 Skywalking 客户端上报 Java 应用数据，详细信息，请参见通过 SkyWalking 上报 Java 应用数据。SkyWalking 支持的中间件和组件，请参见 SkyWalking 官方文档。

前提条件

您已完成以下操作：

- 创建阿里云 Elasticsearch 实例，本文使用 7.4.0 版本。
  具体操作步骤，请参见创建阿里云 Elasticsearch 实例。

- 准备一台 Linux 服务器，并在服务器中安装 JDK，要求 JDK 版本为 1.8.0 及以上版本。
  建议您使用阿里云 ECS 服务器。购买 ECS 服务器的方法，请参见步骤一：创建 ECS 实例。

说明：安装 JDK 的方式，请参见步骤三：安装 JDK。如果未正确安装 JDK，启动 SkyWalking 后查看日志，可能会显示 Java not found 或者 java-xxx: No such file or directory 报错。

- 确保 Linux 服务器的 8080、10800、11800、12800 端口不被占用。
- 关闭 Linux 服务器的防火墙及 SELinux。

操作流程

1. 步骤一：下载并安装 SkyWalking
2. 步骤二：配置 SkyWalking 与 Elasticsearch 连通
3. 步骤三：验证结果

步骤一：下载并安装 SkyWalking

1. 在 Linux 服务器中，下载 SkyWalking。
   建议选择最新的 7.0.0 版本。由于本文使用的是 Elasticsearch 7.4.0 版本，因此选择 Binary Distribution for ElasticSearch 7 二进制包。下载命令如下。

```
wget https://archive.apache.org/dist/skywalking/7.0.0/apache-skywalking-apm-es7-7.0.0.tar.gz
```

2. 解压。

```
tar -zxvf apache-skywalking-apm-es7-7.0.0.tar.gz
```

3. 查看解压后的文件。

```
ll apache-skywalking-apm-bin-es7/
```

返回结果如下。

```
total 92
drwxrwxr-x 8 1001 1002   143 Mar 18 23:50 agent
drwxr-xr-x 2 root root   241 Apr 10 16:03 bin
drwxr-xr-x 2 root root   221 Apr 10 16:03 config
-rwxrwxr-x 1 1001 1002 29791 Mar 18 23:37 LICENSE
drwxrwxr-x 3 1001 1002  4096 Apr 10 16:03 licenses
-rwxrwxr-x 1 1001 1002 32838 Mar 18 23:37 NOTICE
```

```
drwxrwxr-x 2 1001 1002 12288 Mar 19 00:00 oap-libs
-rw-rw-r-- 1 1001 1002  1978 Mar 18 23:37 README.txt
drwxr-xr-x 3 root root    30 Apr 10 16:03 tools
drwxr-xr-x 2 root root    53 Apr 10 16:03 webapp
```

步骤二：配置 SkyWalking 与 Elasticsearch 连通

1. 在 config 目录下，打开 application.yml 文件。

```
cd apache-skywalking-apm-bin-es7/config/
vi application.yml
```

2. 定位到 storage 部分，将默认的 H2 存储库改为 elasticsearch7，并按照以下说明配置。

```
storage:
  selector: ${SW_STORAGE:elasticsearch7}
  elasticsearch7:
    nameSpace: ${SW_NAMESPACE:"skywalking-index"}
    clusterNodes: ${SW_STORAGE_ES_CLUSTER_NODES:es-cn-4591kzdzk000i****.public.elasticsearch.aliyuncs.com:9200}
    protocol: ${SW_STORAGE_ES_HTTP_PROTOCOL:"http"}
   # trustStorePath: ${SW_SW_STORAGE_ES_SSL_JKS_PATH:"../es_keystore.jks"}
   # trustStorePass: ${SW_SW_STORAGE_ES_SSL_JKS_PASS:""}
    enablePackedDownsampling: ${SW_STORAGE_ENABLE_PACKED_DOWNSAMPLING:true} # Hour and Day metrics will be merged into minute index.
    dayStep: ${SW_STORAGE_DAY_STEP:1} # Represent the number of days in the one minute/hour/day index.
    user: ${SW_ES_USER:"elastic"}
    password: ${SW_ES_PASSWORD:"es_password"}
```

说明： SkyWalking 服务默认使用 H2 存储，不具有持久存储的特性，所以需要将存储组件修改为 elasticsearch。

参数	说明
selector	存储选择器。本文设置为 elasticsearch7。
nameSpace	命名空间。Elasticsearch 实例中，所有索引的命名会使用此参数值作为前缀。
clusterNodes	指定 Elasticsearch 实例的访问地址。由于实例与 SkyWalking 不在同一专有网络 VPC（Virtual Private Cloud）下，因此要使用公网访问地址，获取方式请参见查看实例的基本信息。
user	Elasticsearch 实例的访问用户名，默认为 elastic。
password	对应用户的密码。elastic 用户的密码在创建实例时指定，如果忘记可重置。重置密码的注意事项和操作步骤，请参见重置实例访问密码。

注意： 配置中仅指定用户名和密码即可，请注释 trustStorePath 和 trustStorePass，否则会报错 NoSuchFileException:../es_keystore.jks。

3. **可选：** 修改监听的 IP 地址或端口号。
   SkyWalking 默认使用 12800 作为 Rest API 通信端口，11800 为 gRPC API 端口，可在 application.yml 文件的 core 中修改，本文使用默认配置。

```
core:
  selector: ${SW_CORE:default}
  default:
    # Mixed: Receive agent data, Level 1 aggregate, Level 2 aggregate
    # Receiver: Receive agent data, Level 1 aggregate
    # Aggregator: Level 2 aggregate
    role: ${SW_CORE_ROLE:Mixed} # Mixed/Receiver/Aggregator
    restHost: ${SW_CORE_REST_HOST:0.0.0.0}
```

```
    restPort: ${SW_CORE_REST_PORT:12800}
    restContextPath: ${SW_CORE_REST_CONTEXT_PATH:/}
    gRPCHost: ${SW_CORE_GRPC_HOST:0.0.0.0}
    gRPCPort: ${SW_CORE_GRPC_PORT:11800}
```

4. **可选：**在 webapp 目录下，修改 webapp.yml 配置。
   本文使用默认配置，您也可以根据具体需求修改。

```
server:
  port: 8080
collector:
  path: /graphql
  ribbon:
    ReadTimeout: 10000
    # Point to all backend's restHost:restPort, split by ,
    listOfServers: 127.0.0.1:12800
```

步骤三：验证结果

1. 在 Linux 服务器中，启动 SkyWalking。

```
cd ../bin
./startup.sh
```

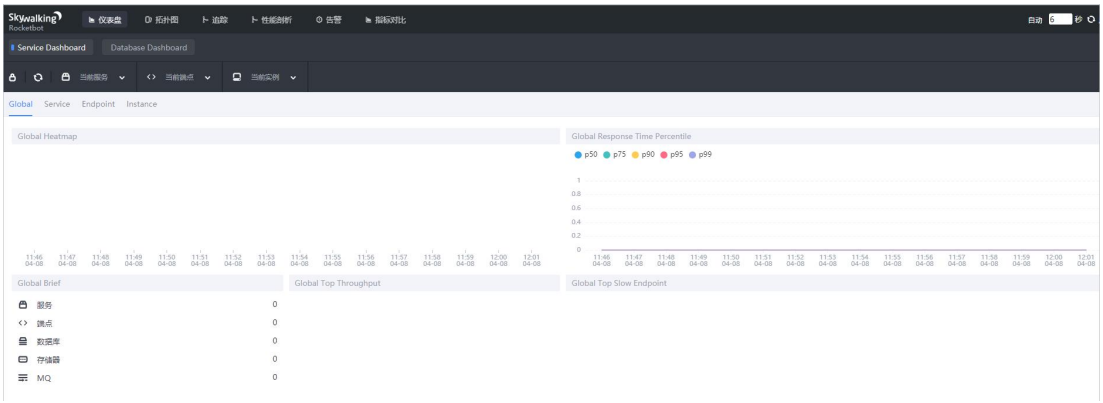
注意：

- 在启动 SkyWalking 前，请确保 Elasticsearch 实例为正常状态。
- 执行./startup.sh 命令，会同时启动 Collector 和 UI。

启动成功后，返回如下结果。

```
SkyWalking OAP started successfully!
SkyWalking Web Application started successfully!
```

2. 在浏览器中，访问 **http://<Linux 服务器的 IP 地址>:8080/**。



说明： 初次使用 SkyWalking 连接 Elasticsearch 服务，启动会比较慢。因为 SkyWalking 需要向 Elasticsearch 服务创建大量的 index，所以在未创建完成之前，访问这个页面会显示空白。此时您可以通过查看日志来判断启动是否完成，日志路径为<SkyWalking 的安装路径>logs/skywalking-oap-server.log。

3. 参见登录 Kibana 控制台，登录对应 Elasticsearch 实例的 Kibana 控制台，执行 `GET _cat/indices?v` 命令查看索引数据。
   根据返回结果，可以看到 Elasticsearch 实例中包含了大量以 `skywalking-index` 开头的索引。

```
green  open   skywalking-index_all_percentile-20200408
green  open   skywalking-index_endpoint_inventory
green  open   skywalking-index_service_apdex-20200408
green  open   skywalking-index_database_access_resp_time_month-202004
green  open   skywalking-index_service_relation_client_cpm-20200408
green  open   skywalking-index_database_access_sla_month-202004
green  open   skywalking-index_service_relation_server_call_sla-20200408
green  open   skywalking-index_endpoint_sla-20200408
green  open   skywalking-index_instance_jvm_memory_noheap-20200408
green  open   skywalking-index_service_relation_server_percentile-20200408
green  open   skywalking-index_service_instance_relation_server_resp_time-20200408
green  open   skywalking-index_profile_task_segment_snapshot-20200408
green  open   skywalking-index_endpoint_relation_cpm-20200408
green  open   skywalking-index_instance_jvm_old_gc_time-20200408
green  open   skywalking-index_service_sla-20200408
green  open   skywalking-index_top_n_database_statement-20200408
green  open   skywalking-index_service_relation_client_call_sla-20200408
green  open   skywalking-index_endpoint_percentile_month-202004
green  open   apm-agent-configuration
```

使用 Filebeat+Kafka+Logstash+Elasticsearch 构建日志分析系统

摘要：随着时间的积累，日志数据会越来越多，当您需要查看并分析庞杂的日志数据时，可通过 Filebeat+Kafka+Logstash+Elasticsearch 采集日志数据到阿里云 Elasticsearch（简称 ES）中，并通过 Kibana 进行可视化展示与分析。本文介绍具体的实现方法。

背景信息

Kafka 是一种分布式、高吞吐、可扩展的消息队列服务，广泛用于日志收集、监控数据聚合、流式数据处理、在线和离线分析等大数据领域，已成为大数据生态中不可或缺的部分。本文使用阿里云消息队列 Kafka 版，详情请参见什么是消息队列 Kafka 版。

在实际应用场景中，为了满足大数据实时检索的需求，您可以使用 Filebeat 采集日志数据，将 Kafka 作为 Filebeat 的输出端。Kafka 实时接收到 Filebeat 采集的数据后，以 Logstash 作为输出端输出。输出到 Logstash 中的数据在格式或内容上可能不能满足您的需求，此时可以通过 Logstash 的 filter 插件过滤数据。最后将满足需求的数据输出到 ES 中进行分布式检索，并通过 Kibana 进行数据分析与展示。简单流程如下。

操作流程

1. 准备工作

   完成环境准备，包括创建实例、安装 Filebeat 等。

注意：安装 Filebeat 的 ECS、阿里云 ES、阿里云 Logstash、阿里云消息队列 Kafka 必须在同一 VPC 下。

2. 步骤一：配置 Filebeat
   配置 Filebeat 的 input 为系统日志，output 为 Kafka，将日志数据采集到 Kafka 的指定 Topic 中。

3. 步骤二：配置 Logstash 管道
   配置 Logstash 管道的 input 为 Kafka，output 为阿里云 ES，使用 Logstash 消费 Topic 中的数据并传输到阿里云 ES 中。

4. 步骤三：查看日志消费状态
   在消息队列 Kafka 中查看日志数据的消费的状态，验证日志数据是否采集成功。

5. 步骤四：通过 Kibana 过滤日志数据
   在 Kibana 控制台的 Discover 页面，通过 Filter 过滤出 Kafka 相关的日志。

准备工作

1. 创建阿里云 ES 实例，并开启实例的自动创建索引功能。
   具体操作步骤请参见创建阿里云 Elasticsearch 实例和配置 YML 参数，本文以 6.7 版本为例。

2. 创建阿里云 Logstash 实例。要求该实例与阿里云 ES 实例的版本相同，并且在同一专有网络 VPC（Virtual Private Cloud）下。
   具体操作步骤请参见创建阿里云 Logstash 实例。

3. 购买并部署阿里云消息队列 Kafka 版实例、创建 Topic 和 Consumer Group。
   本文使用 VPC 实例，并要求该实例与阿里云 ES 实例在同一 VPC 下，具体操作步骤请参见 VPC 接入。
   创建 Topic 和 Consumer Group 的具体步骤请参见步骤三：创建资源。

4. 创建阿里云 ECS 实例，并且该 ECS 实例与阿里云 ES 实例和 Logstash 实例处于同一 VPC 下。
   具体操作步骤请参见使用向导创建实例。

注意：该 ECS 实例用来安装 Filebeat，由于 Filebeat 目前仅支持 Aliyun Linux、RedHat 和 CentOS 这三种操作系统，因此在创建时请选择其中一种操作系统。

5. 在 ECS 实例上安装 Filebeat。
   具体操作步骤请参见 Install Filebeat。本文使用 6.8.5 版本。

步骤一：配置 Filebeat

1. 连接安装了 Filebeat 的 ECS 服务器。
   具体操作步骤请参见连接实例。

2. 进入 Filebeat 安装目录，执行以下命令，创建并配置 filebeat.kafka.yml 文件。

```
cd filebeat-6.8.5-linux-x86_64
vi filebeat.kafka.yml
```

filebeat.kafka.yml 配置如下。

```
filebeat.prospectors:
  - type: log
    enabled: true
    paths:
        - /var/log/*.log

output.kafka:
    hosts: ["172.16.**.**:9092","172.16.**.**:9092","172.16.**.**:9092"]
    topic: estest
    version: 0.10.2
```

参数	说明
type	输入类型。设置为 log，表示输入源为日志。
enabled	设置配置是否生效。true 表示生效，false 表示不生效。
paths	需要监控的日志文件的路径。多个日志可在当前路径下另起一行写入日志文件路径。
hosts	消息队列 Kafka 实例的接入点，可在实例详情页面获取，详情请参见查看接入点。由于本文使用的是 VPC 实例，因此使用默认接入点。
topic	日志输出到消息队列 Kafka 的 Topic，请指定为您已创建的 Topic。
version	Kafka 的版本，可在消息队列 Kafka 的实例详情页面获取。 注意：不配置此参数会报错。

3. 启动 Filebeat。

```
./filebeat -e -c filebeat.kafka.yml
```

步骤二：配置 Logstash 管道

1. 登录阿里云 Elasticsearch 控制台。
2. 进入目标实例。
   i 在顶部菜单栏处，选择地域。
   ii 在左侧导航栏，单击 **Logstash 实例**，然后在 Logstash 实例中单击目标实例 ID。
3. 在左侧导航栏，单击**管道管理**。
4. 在**管道列表**区域，单击**创建管道**。

5. 在**创建管道任务**页面，输入**管道 ID** 并配置管道。

   本文使用的管道配置如下。

```
input {
  kafka {
    bootstrap_servers => ["172.**.**.92:9092,172.16.**.**:9092,172.16.**.**:909
2"]
    group_id => "es-test"
    topics => ["estest"]
    codec => json
 }
}
filter {

}
output {
  elasticsearch {
    hosts => "http://es-cn-n6w1o1x0w001c****.elasticsearch.aliyuncs.com:920
0"
    user =>"elastic"
    password =>"<your_password>"
    index => "kafka - %{+YYYY.MM.dd}"
 }
}
```

表 1. input 参数说明

参数	说明
bootstrap_servers	消息队列 Kafka 实例的接入点，可在实例详情页面获取，详情请参见查看接入点。由于本文使用的是 VPC 实例，因此使用默认接入点。
group_id	指定为您已创建的 Consumer Group 的名称。
topics	指定为您已创建的 Topic 的名称，需要与 Filebeat 中配置的 Topic 名称保持一致。
codec	设置为 json，表示解析 JSON 格式的字段，便于在 Kibana 中分析。

表 2. output 参数说明

参数	说明
hosts	阿里云 ES 的访问地址，取值为 http://<阿里云 ES 实例的内网地址>:9200。 说明：您可在阿里云 ES 实例的基本信息页面获取其内网地址，详情请参见查看实例的基本信息。
user	访问阿里云 ES 的用户名，默认为 elastic。您也可以使用自建用户，详情请参见 For more information, see 通过 Elasticsearch X-Pack 角色管理实现用户权限管控。
password	访问阿里云 ES 的密码，在创建实例时设置。如果忘记密码，可进行重置，重置密码的注意事项及操作步骤请参见重置实例访问密码。
index	索引名称。设置为 kafka - %{+YYYY.MM.dd}表示索引名称以 kafka 为前缀，以日期为后缀，例如 kafka-2020.05.27。

更多 Config 配置详情请参见 Logstash 配置文件说明。

6. 单击**下一步**，配置管道参数。

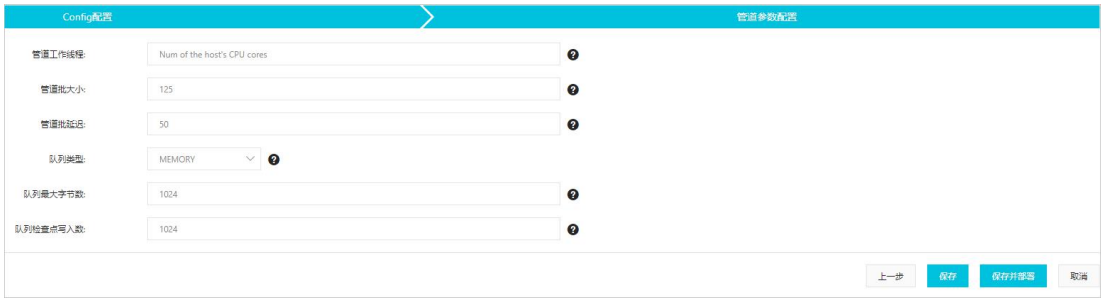


表 3. 管道配置参数说明

参数	说明
管道工作线程	并行执行管道的 Filter 和 Output 的工作线程数量。当事件出现积压或 CPU 未饱和时，请考虑增大线程数，更好地使用 CPU 处理能力。默认值：实例的 CPU 核数。
管道批大小	单个工作线程在尝试执行 Filter 和 Output 前，可以从 Input 收集的最大事件数目。较大的管道批大小可能会带来较大的内存开销。您可以设置 LS_HEAP_SIZE 变量，来增大 JVM 堆大小，从而有效使用该值。默认值：125。
管道批延迟	创建管道事件批时，将过小的批分派给管道工作线程之前，要等候每个事件的时长，单位为毫秒。默认值：50ms。
队列类型	用于事件缓冲的内部排队模型。可选值： MEMORY：默认值。基于内存的传统队列。 PERSISTED：基于磁盘的 ACKed 队列（持久队列）。
队列最大字节数	请确保该值小于您的磁盘总容量。默认值：1024 MB。
队列检查点写入数	启用持久性队列时，在强制执行检查点之前已写入事件的最大数目。设置为 0，表示无限制。默认值：1024。

警告：配置完成后，需要保存并部署才能生效。保存并部署操作会触发实例重启，请在不影响业务的前提下，继续执行以下步骤。

7. 单击**保存**或者**保存并部署**。
   - **保存**：将管道信息保存在 Logstash 里并触发实例变更，配置不会生效。保存后，系统会返回**管道管理**页面。可在**管道列表**区域，单击**操作**列下的**立即部署**，触发实例重启，使配置生效。
   - **保存并部署**：保存并且部署后，会触发实例重启，使配置生效。

步骤三：查看日志消费状态

1. 进入消息队列 Kafka 控制台。
2. 在左侧导航栏，单击 **Consumer Group 管理**。
3. 在 **Consumer Group 管理**页面，单击目标消息队列 Kafka 实例。
4. 单击对应 Consumer Group 右侧**操作**列下的**消费状态**。

5. 在**消费状态**对话框中，单击对应 Topic 右侧**操作**列下的**详情**，查看详细消费状态。
   正常情况下，返回结果如下。

步骤四：通过 Kibana 过滤日志数据

1. 登录目标阿里云 ES 实例的 Kibana 控制台。
   具体步骤请参见登录 Kibana 控制台。

2. 创建一个索引模式。

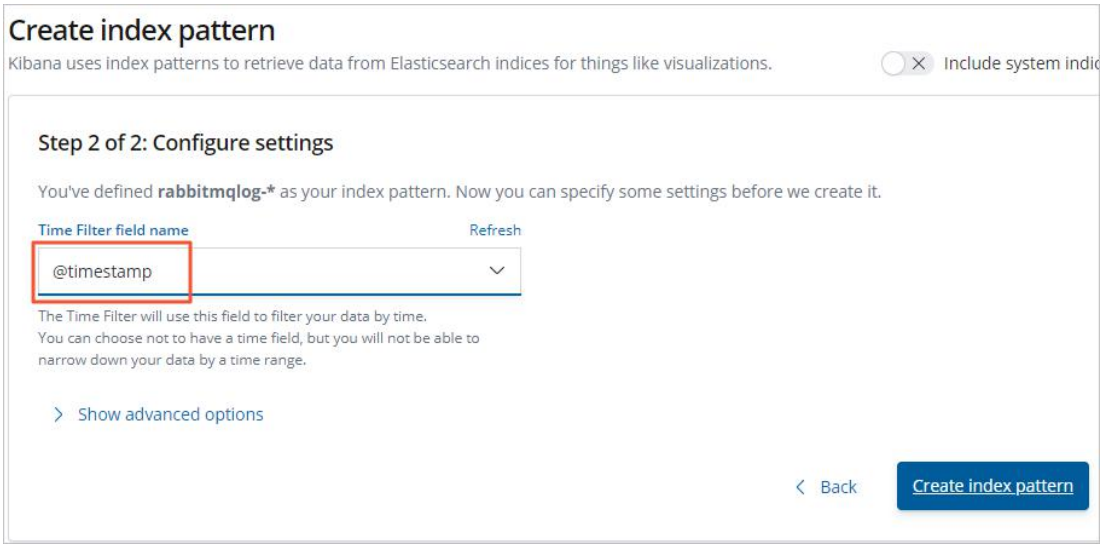
   i 在左侧导航栏，单击 **Management**。

   ii 在 Kibana 区域，单击 **Index Patterns**。

   iii 单击 **Create index pattern**。

   iv输入 **Index pattern**（本文使用 kafka-*），单击 **Next step**。

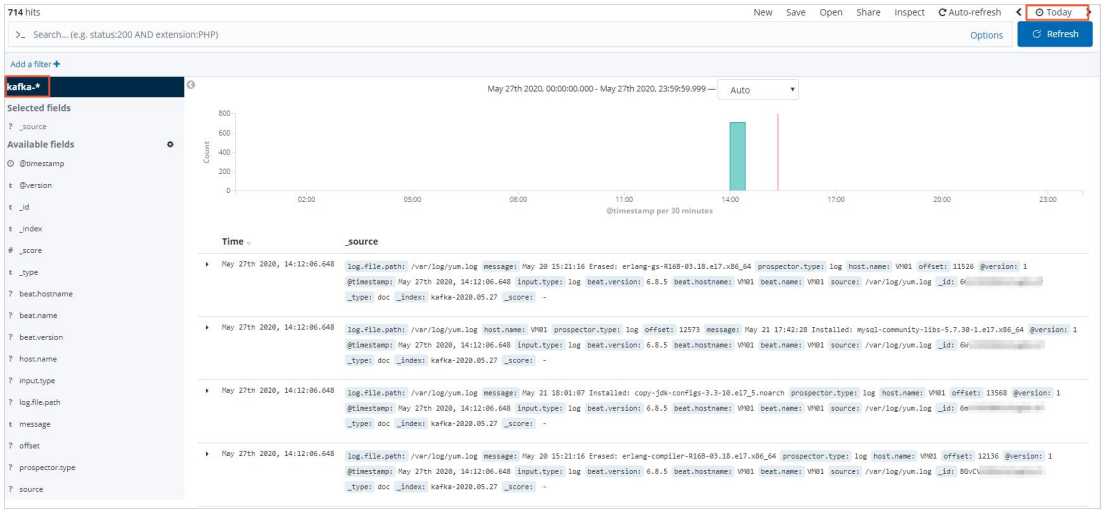
   v 选择 Time Filter field name（本文选择@timestamp），单击 Create index pattern。



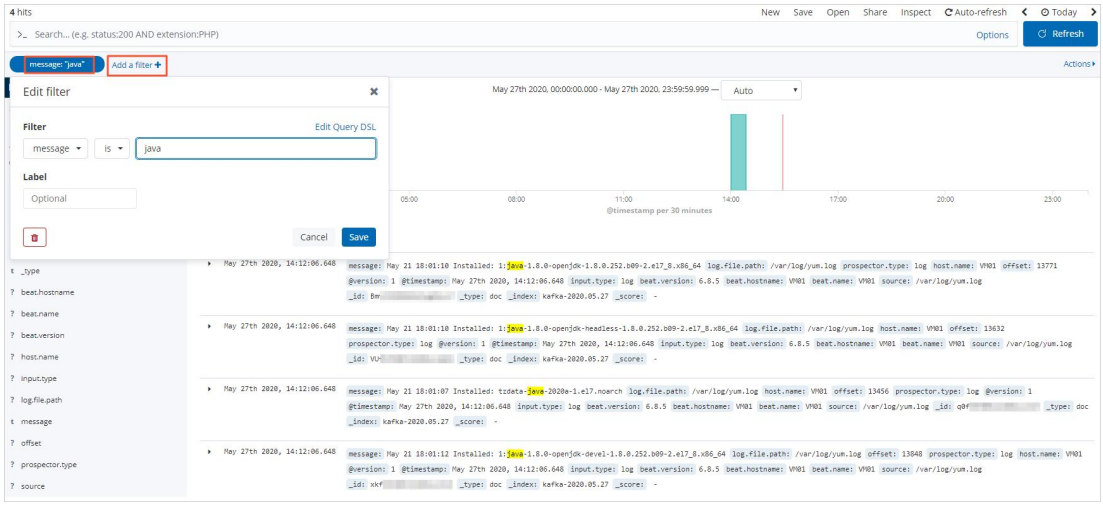
3. 在左侧导航栏，单击 **Discover**。

4. 从页面左侧的下拉列表中，选择您已创建的索引模式（kafka-*）。

5. 在页面右上角，选择一段时间，查看对应时间段内的 Filebeat 采集的日志数据。



6. 单击 Add a filter，在 Add filter 页面中设置过滤条件，查看符合对应过滤条件的日志数据。



常见问题

Q：同步日志数据出现问题，管道一直在生效中，无法将数据导入 Elasticsearch，如何解决?

A：查看 Logstash 实例的主日志是否有报错，根据报错判断原因，具体操作请参见查询日志。常见的原因及解决方法如下。

原因	解决方法
Kafka 的接入点不正确。	参见查看接入点获取正确的接入点。完成后，修改管道配置替换错误接入点。
Logstash 与 Kafka 不在同一 VPC 下。	重新购买同一 VPC 下的实例。购买后，修改现有管道配置。 说明：VPC 实例只能通过专有网络 VPC 访问消息队列 Kafka 版。
Kafka 或 Logstash 集群的配置太低，例如使用了测试版集群。	升级集群规格，完成后，刷新实例，观察变更进度。升级 Logstash 实例规格的具体操作，请参见升配集群；升级 Kafka 实例规格的具体操作，请参见升级实例配置。
管道配置中包含了 file_extend，但没有安装 logstash-output-file_extend 插件。	选择以下任意一种方式处理： 安装 logstash-output-file_extend 插件。具体操作，请参见 安装或卸载插件。 中断变更，等到实例处于变更中断状态后，在管道配置中，去掉 file_extend 配置，触发重启恢复。

更多问题原因及解决方法，请参见管道创建后，进程卡住了，实例变更进度不变，如何处理？ 。

基于 Elasticsearch+Flink 的日志全观测最佳实践

观看视频：https://developer.aliyun.com/live/248094

一、什么是全观测?

1、传统运维的问题

要了解全观测，我们先看看传统运维存在哪些问题：

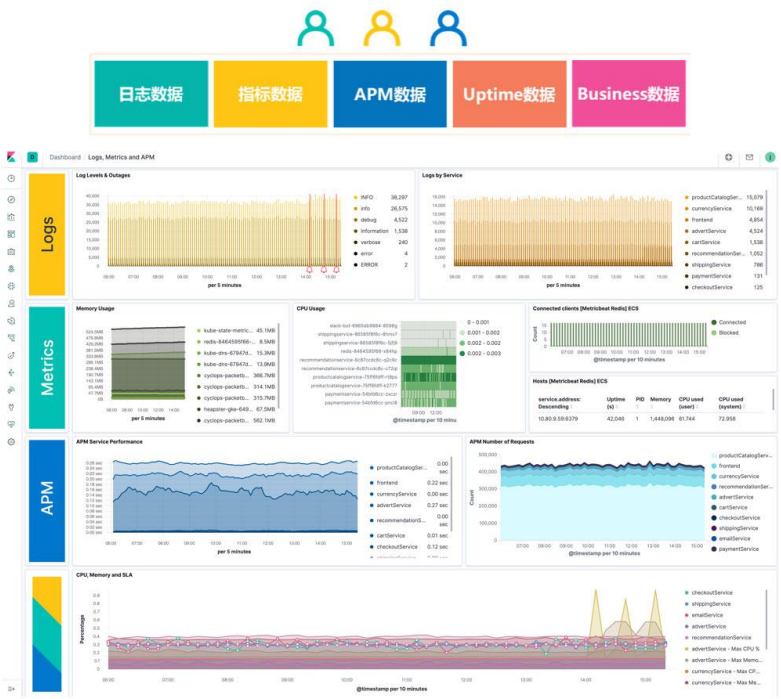
a）数据孤岛，分散在不同部门，分析排查故障困难；
b）多个厂商的多种工具，无法自动化统一分析；
c）故障是立体的，日志、指标等都只能看到一方面的可观察性；
d）只进行收集，没有真正深入分析，不能发挥大数据的价值；

2、全观测的定义

全观测是对传统运维的改进，它将日志、指标、APM 数据，汇总在一个平台，让运维、开发、业务人员对所有的数据从统一视角进行观察分析。

3、全观测可以做什么?

a）建立统一的可视化视图、对齐时间、过滤条件；
b）建立统一的基于规则的监控和告警；
c）建立统一的机器学习的智能监控和告警。



全观测架构图

二、全观测场景技术难点和解决方案

在全观测场景下会面临很多痛点，那么如何通过云上 ELK+Flink 能力，去解决全观测日志场景下的痛点呢?

1、痛点一：日志/指标获取难

机器、业务系统、网络链路、操作系统，诸多指标及日志获取手段不一，落地过程复杂；

- **解决方案：Beats/APM 获取日志/指标**

轻量化的提供各类 metic、logs、APM 数据采集能力；

2、痛点二：日志/指标规格化要求高

上下游链路配合衔接过程中，如何将有效信息从海量日志中获取；

- **解决方案：数据清洗 SQL 化更简易**

支持各类网络格式的日志/指标采模板，实时计算 Flink 提供完整流式 SQL 能力；

3、痛点三：高并发写入、系统稳定性差

业务/流量抖动，日志写入峰值往往会很高，旁路系统稳定性受到很大的挑战；

- **解决方案：云上 ES 写入托管及超强稳定性**

提供 Indexing service 自研 ES 写入托管服务，及跨机房部署、同城容灾、场景内核优化；

4、痛点四：海量数据存储成本高

日志场景涉及海量数据，TB 级别起步，甚至 PB 级；

- **解决方案：低成本数据存储**

阿里云 ES 提供冷热分离数据存储方式，及自研存储引擎 Openstore 优化存储压缩算法；

5、痛点五：日志分析和指标监控统一难

借助时序系统可以很好的完成监控，但异常分析困难相反，如何在统一平台完成；

- **解决方案：日志分析、指标监控、APM 能力齐全**

阿里云 ElastiStack 全托管，提供日志分析、监控、Tracing 一站式能力，针对时序场景，针对性优化引擎，保证时序日志监控和分析的性能；

6、痛点 6：系统可扩展性要求高

业务调整带来的技术演进一直在发生，技术组件更新快，运维框架需要有强大的兼容性；

- **解决方案：开源生态具备强大的可扩展性**

基于分布式架构，以及灵活开放的 RestAPI 和 Plugin 框架，支持各种扩展能力。

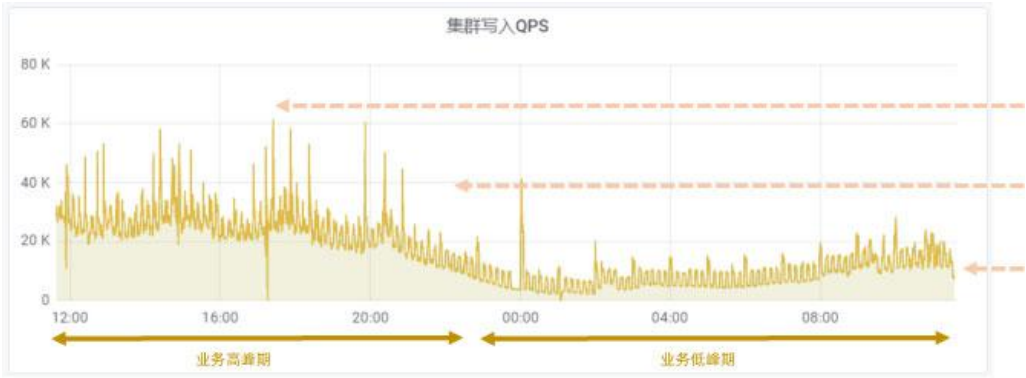
三、时序日志场景痛点分析

写多读少的日志场景下会遇到什么问题?

1、高峰期写入压力大弹性扩展难以有效实施；
2、海量计算+存储资源成本高低峰期资源闲置；
3、为保证系统稳定性集群运维管理复杂；

从下图我们可以看到在高峰和低峰期遇到的情况：

1、业务突发峰值写入 TPS 高达 60K，无法准确预测和预估，峰值写入瓶颈明显；
2、高峰期写入 TPS 平均 40K，需要准备至少 8 节点 ES 计算资源来满足高峰期写入能力；
3、低峰期写入 TPS 平均 20K，仅需要 4 节点 ES 计算资源即可满足低峰写入流量。



四、全链路日志分析与监控 ELK 技术难点

1、高并发写入

a）日志场景往往面临业务/流量抖动；
b）日志写入峰值往往会很高；
c）ES 集群容易被打爆；

2、存储成本高

a）日志场景涉及海量数据；
b）TB 级别起步，甚至 PB 级；
c）部分场景(如：审计)长周期存储；

3、时序分析性能差

a）ES 内核技术局限性；
b）日志场景中的时序查询性能差；
c）复杂聚合、Range 等查询性能瓶颈明显；

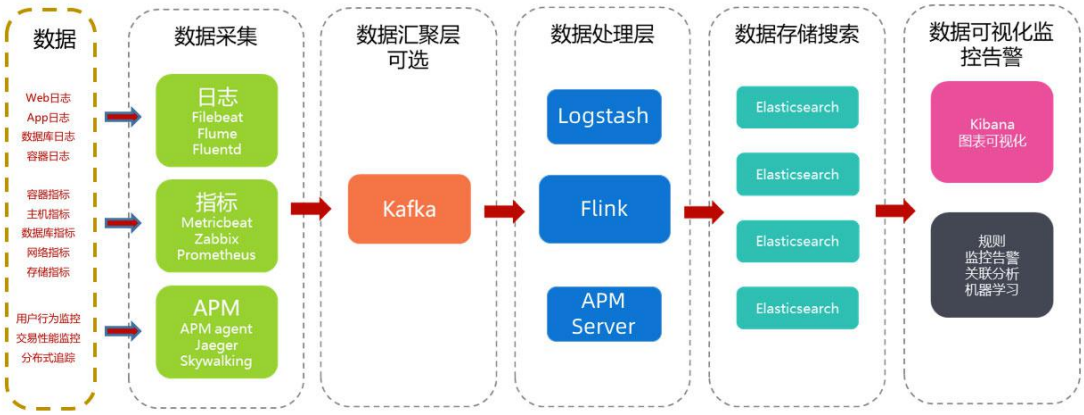
4、可伸缩性及运维要求高

a ）日志峰值/均值/谷值差异巨大；
b ）集群规模大管理运维复杂；

五、全观测解决方案实现日志监控/运维/分析

1、方案选型：100%兼容开源，与各类开源生态组件无缝衔接；支持多云/跨云的日志监控、运维分析场景。

2、方案优势：云上 Elasticsearch 及实时计算 Flink 产品，提供面向海量数据的高性能读写、及高弹性低成本解决方案。



全观测解决方案数据架构图

六、FLink 在方案应用中的优势

实时计算 Flink 版是新一代 Serverless 实时计算引擎，它在方案应用中具有以下优势：

1、流式 SQL

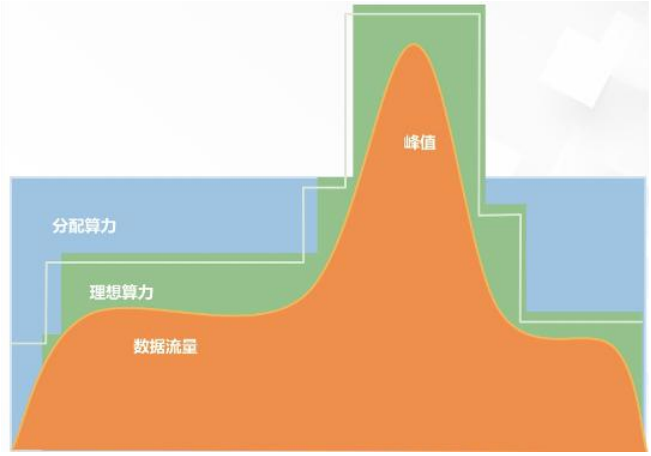
100%兼容开源，一站式开发平台，提供更适合日志场景下海量数据清洗的流式 SQL 能力，核心算子性能优化达开源 2 倍以上。

2、Serverless 服务

全托管免运维，用户无须关注集群运维和资源预留，100%投入业务开发。

3、Autopilot 能力

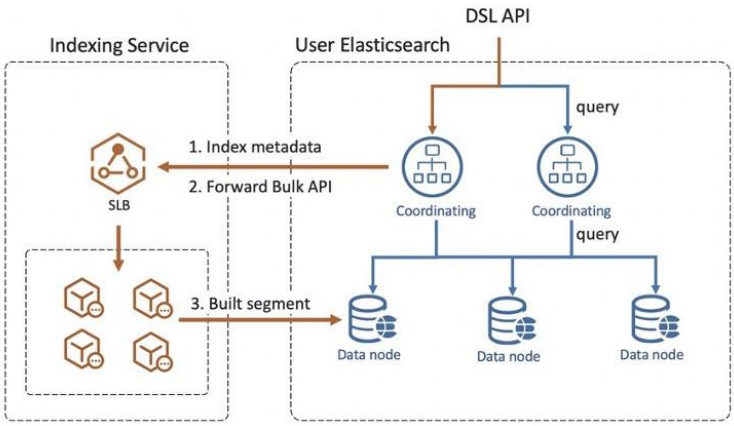
根据数据流量自动重新分配算力，智能削峰填谷，弹性资源分配，有效应对数据的高峰低谷，如下图：



Autopilot 自动分配算力

七、阿里云 Elasticsearch 日志增强特性

1、日志增强版 Indexing Service 写入托管



Indexing Service 架构图

写入托管的核心功能是 IndexingServerless 服务，在 ES 集群之外，依托云端海量计算能力，打破本地集群的物理资源限制，通过读写分离架构，将 ES 集群的数据写入在云端服务进行托管加速，以满足高并发数据写入要求，并且基于海量的资源，支撑快速弹性的扩展能力。

① 相同ES集群规模，提升近10倍写入能力

集群节点规模	日志增强版写入TPS	开源商业版写入TPS	写入能力对比
3个数据节点的2核8GB	226649	24883	910%
3个数据节点的4核16GB	419574	52372	801%
3个数据节点的8核32GB	804010	110277	729%

- 压测配置
选最低的配置 3节点 CPU(2核)/Memory(4G)/存储(高效云盘)
采用数据源为esrally的nyc_taxis数据集进行压测

- 吞吐测试结果
在未做任何配置优化的前提下，写入平均TPS达到 20万 docs/s 以上，吞吐流量超过百兆，比市面上所有极致优化过的ES集群性能高出10倍以上

② 相同业务流量，降低近70%计算成本

开源自建/商业版		日志增强版Indexing Service		
规格	小时价	规格	小时价	成本对比
32核 64G 20节点	296.8	8核32G*6节点	35.2	
		高峰平均 250MB/s	99	降低54.8%
		低峰平均 100MB/s	39.6	降低74.8%

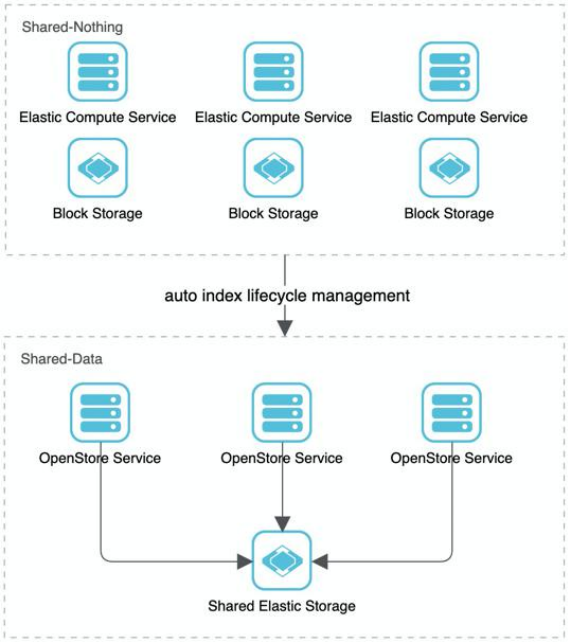
集群节点个数 减少70%，一天计算资源 成本降低68.1%

- 示例场景
以Elasticsearch内部监控业务ElasticMonitor持有集群为例，每天承载高流量的线上用户ES集群的指标和日志写入

- 集群数据
写入流量平均150MB/s，每天增量数据12.96T

写入能力和成本对比图

2、自研 ES 存储 Openstore



阿里云自研 ES 存储 Openstore 具有以下优点：

a ）低成本：

相较于本地 SATA 盘存储成本降低 60%，相较于高效云盘存储成本降低 70%；

b ）海量存储：

数据存储按实际用量用多少付多少，存储 Serverless 按量付费；

c ）高可用：

底层存储服务保证了集群的数据高可用，提供 99.9999999999%的数据持久性。

综上，基于 Elasticsearch+Flink 的日志全观测解决方案在开源的基础上做了优化，解决了传统运维日志场景下的诸多痛点，满足高并发的写入要求，可以达到比开源更高的写入性能，全托管免运维，存储按量付费，用户无需预先购置或预留过多节点来预防集群高并发的写入要求。

APM 应用性能监控分析最佳实践

摘要：通过阿里云托管的 Elasticsearch 应用性能监控 APM（Application Performance Monitoring）服务，结合阿里云 Elasticsearch，您可以搭建应用性能监控系统，实现系统的可观测性。本文介绍阿里云 Elasticseasrch APM 服务的概念、架构及功能，以及如何快速接入并使用 APM。

观看视频：https://developer.aliyun.com/live/247920

背景信息

可观测性的本质是度量您的基础设施、平台和应用程序，以了解它是如何运行的。与传统监控运维相比，目前主流的监控更加注重发现与预警问题，而可观测性的终极目标是为一个复杂分布式系统所发生的一切给出合理解释。监控更注重软件的交付过程中以及交付后的服务状态，而可观测性则要为全研发与运维的生命周期负责。

日志、基础架构指标以及 APM 应用程序性能监测构成了可观测性的三要素。其中，APM 弥补了指标和日志之间的差距。虽然日志和指标往往更具交叉性，涉及基础架构和组件，但 APM 更侧重于应用程序，允许 IT 和开发人员监测其堆栈的应用层，包括最终的用户体验。将 APM 添加到系统监测中，您可以：

- 了解服务的时间花在什么上，以及它崩溃的原因。
- 了解服务如何相互交互，以及它的可视化瓶颈。
- 主动发现并修复性能瓶颈和错误。
- 提高开发团队的生产力。
- 在浏览器中跟踪终端用户体验。

APM 通常被应用于以下场景：

- 用户体验监控：通过监控用户的行为提升用户体验。例如监控用户和 Web 界面或客户端的交互，并记录交互事件的时间。
- 运行时应用程序架构：理解服务间的依赖关系、架构中应用程序交互的网络拓扑。
- 业务事务（Business transaction）：产生有意义的 SLA 报告，并从业务角度提供有关应用程序性能的趋势信息。

- 组件监控（Deep dive component monitoring）：通常需要安装 Agent 并且主要针对中间层，包括 Web 服务器、应用和消息服务器等。健壮的监控应该能显示代码执行的清晰路径，因为这一维度和上述第二个维度紧密相关，APM 产品通常会将这两个维度合并作为一个功能。
- 分析或报告（Analytics/Reporting）：将从应用程序中收集的一系列指标数据，标准化地展现成应用性能数据的通用视图。

阿里云托管的 Elastic APM 和目前比较流行的开源 APM 系统 Apache SkyWalking 的能力对比情况请参见能力对比。

架构及数据模型

阿里云 Elasticsearch 应用性能监控分析服务，支持基于开源 Elastic APM 构建云上一键托管的 APM Server 服务，并灵活对接阿里云 Elasticsearch 服务，能够为您提供高效的应用程序性能优化与监控能力。整体功能模块架构如下。

Elastic APM 由四个模块组成，详细信息请参见应用性能监控分析服务介绍。其中 APM Agent 采集器从其监测的应用程序中收集不同类型的信息和数据，这些被称为事件。采集器收集数据后将这些事件流式传输到 APM Server，由 Server 验证并处理事件。事件支持的类型包括 Spans、Transaction、Errors 和 Metrics：

- Spans：范围事件类型。包含有关已执行的特定代码路径的信息。它们从活动的开始到结束进行度量，可以与其他 Span 具有父或子关系。
- Transaction：事务事件类型。是一种特殊的 Span（没有父 Span，只能从中派生出子 Span，可以理解为“树”这种数据结构的根节点），具有与之关联的额外元数据。你可以将 Transactions 视为您在服务中衡量的最高级别的工作，例如服务中的请求、提供的 HTTP 请求或运行的特定的后台作业等。

- Errors：错误事件类型。包含相关的原始异常信息，或发生异常时创建的日志信息。
- Metrics：指标事件类型。APM Agent 自动获取基本的主机级别指标，包含系统和进程级别的 CPU 和内存指标。除此之外还可获取特定于代理的指标，例如 Java Agent 中的 JVM 指标和 Go 代理中的 Go 运行时指标。

前提条件

创建阿里云 Elasticsearch 7.10 版本实例，具体操作请参见创建阿里云 Elasticsearch 实例。购买时，建议您选择日志增强版，以满足应用性能监控场景下低成本海量数据写入及存储的需求，详细信息请参见购买页面参数（增强版）。

使用限制

阿里云 APM Server 仅支持与阿里云 Elasticsearch 7.10 版本集成。

操作流程

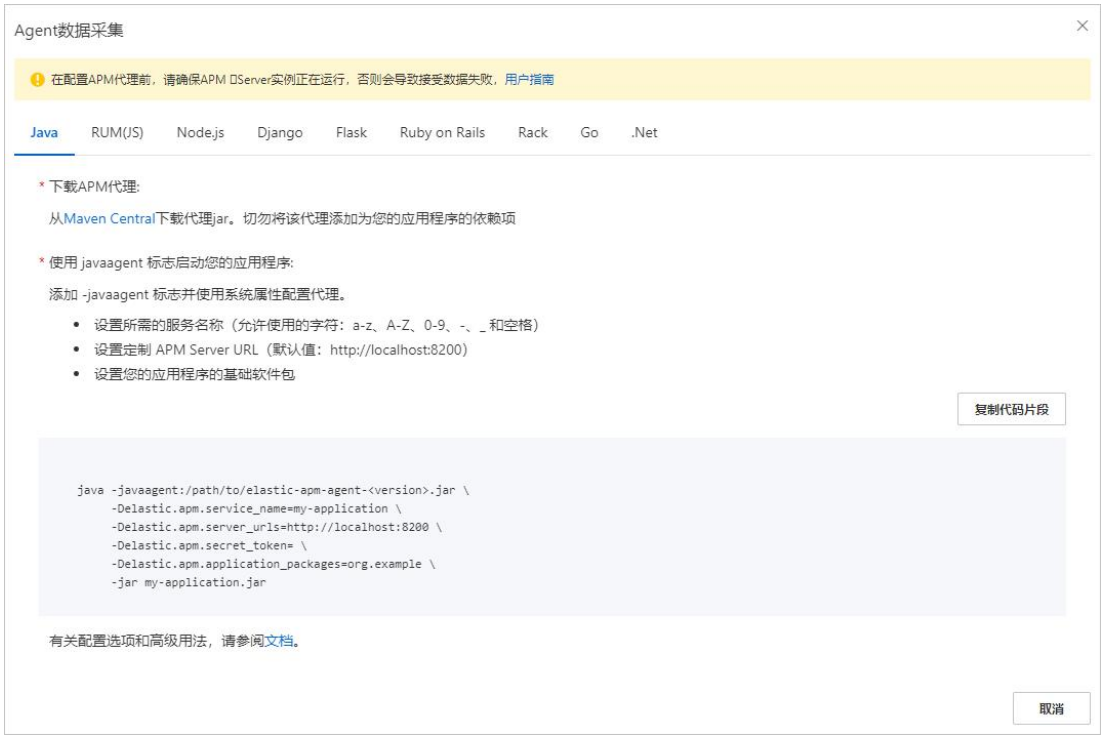
1. 步骤一：创建 APM Server 实例
2. 步骤二：配置 APM Agent
3. 步骤三：在 Kibana 中进行应用性能分析

步骤一：创建 APM Server 实例

1. 登录阿里云 Elasticsearch 控制台。
2. 进入 APM 应用性能监控控制台。
   i 在顶部菜单栏处，选择地域。
   ii 在左侧导航栏，单击 **APM 应用性能监控**。
3. 在 **APM Server 管理**页面，参见创建 APM Server 实例，创建 APM Server 实例。其中专有网络需要与待监控的客户端服务（APM Agent）所在的专有网络保持一致。实例创建成功后，您可以单击对应实例右侧的**实例管理**，查看实例的访问地址等基本信息。

步骤二：配置 APM Agent

1. 在 **APM Server 管理**页面，单击 **APM 数据采集**。
2. 在 **Agent 数据采集**对话框中，单击与您客户端相同语言的页签，根据提示配置数据采集。



配置完成后，代理程序会结合应用程序收集性能指标和错误，最终将收集到的所有数据发送至 Server 端。以 Java 为例，配置方法如下：

i 参见 High Level REST Client（7.x），配置应用程序访问阿里云 Elasticsearch，在 pom 依赖中添加 APM Agent 依赖。

```
<dependency>
  <groupId>co.elastic.apm</groupId>
  <artifactId>elastic-apm-agent</artifactId>
  <version>1.27.0</version>
</dependency>
```

ii 在 Maven Central 中下载应用兼容的代理文件。

iii 设置启动参数，并使用 javaagent 参数启动应用。

```
java -javaagent:/root/elastic-apm-agent-1.27.0.jar \
     -Delastic.apm.service_name=my-application \
     -Delastic.apm.server_urls=http://apm-cn-tl32fxqq****.apm.elasticsearch.aliyuncs.com::8200 \
     -Delastic.apm.secret_token= \
     -Delastic.apm.application_packages=org.example \
     -jar elastic.jar
```

- apm-agent-java-0.1.2.jar：需要替换为您上一步下载的代理文件的名称，包含文件类型后缀（.jar）。
- Delastic.apm.server_urls：将该参数值替换为您 APM Server 实例的访问地址。实例创建成功后，单击对应实例右侧的实例管理，即可获取实例的访问地址。

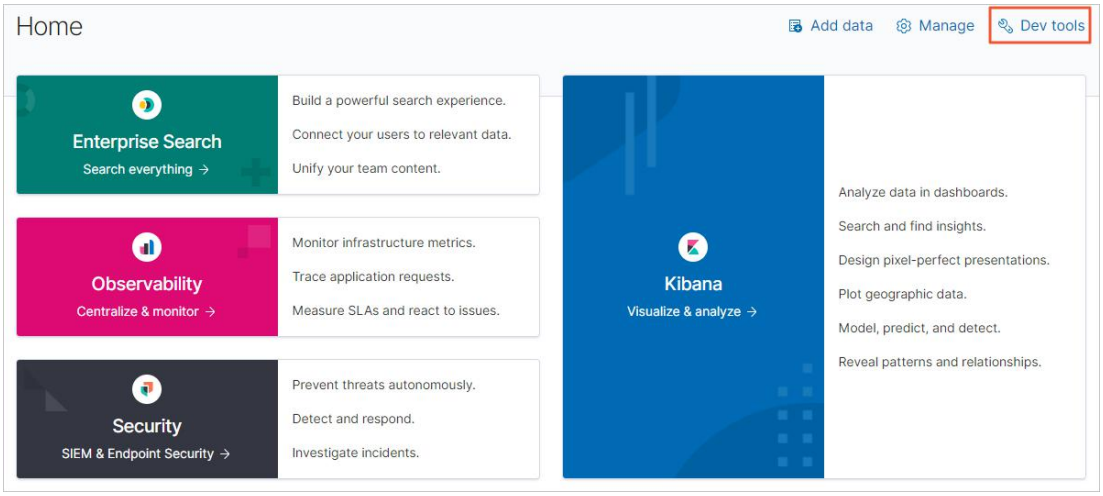
步骤三：在 Kibana 中进行应用性能分析

1. 登录步骤一：创建 APM Server 实例时，所关联的 Elasticsearch 实例的 Kibana 控制台。

   登录 Kibana 控制台的具体操作，请参见登录 Kibana 控制台。

说明：本文以阿里云 Elasticsearch 7.10 版本为例，其他版本操作可能略有差别，请以实际界面为准。

2. 根据页面提示进入 Kibana 主页，单击右上角的 **Dev tools**。



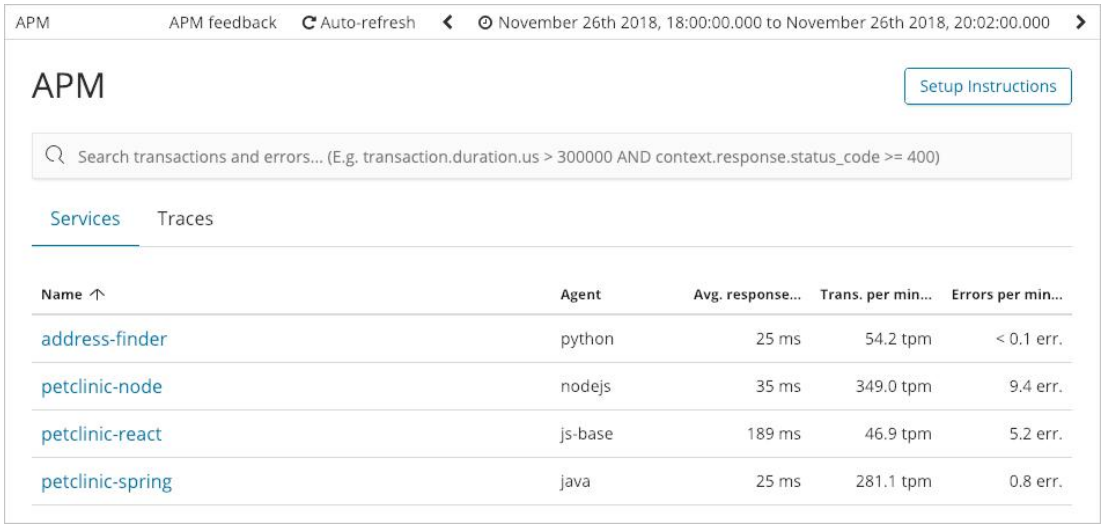
3. 在 Console 页签，运行以下脚本，开启自动创建 APM onboarding 索引。

说明：

- APM Server 实例创建成功后会自动启动实例服务，启动后 Elasticsearch 会自动创建 APM 相关索引，您可在 Kibana 控制台中查看。除此之外，由于阿里云 Elasticsearch 对自动创建索引的命名规范有要求，APM onboarding 索引的命名不符合当前规范，因此您还需要手动开启自动创建 APM onboarding 索引。
- 以下脚本中的 7.10.2 为您创建的 APM Server 实例的版本号。

```
PUT _cluster/settings
{
  "persistent": {
    "action.auto_create_index":"+.*,+apm-7.10.2-onboarding-*,-*"
  }
}
```

4. 查看 APM 中包含的所有服务。



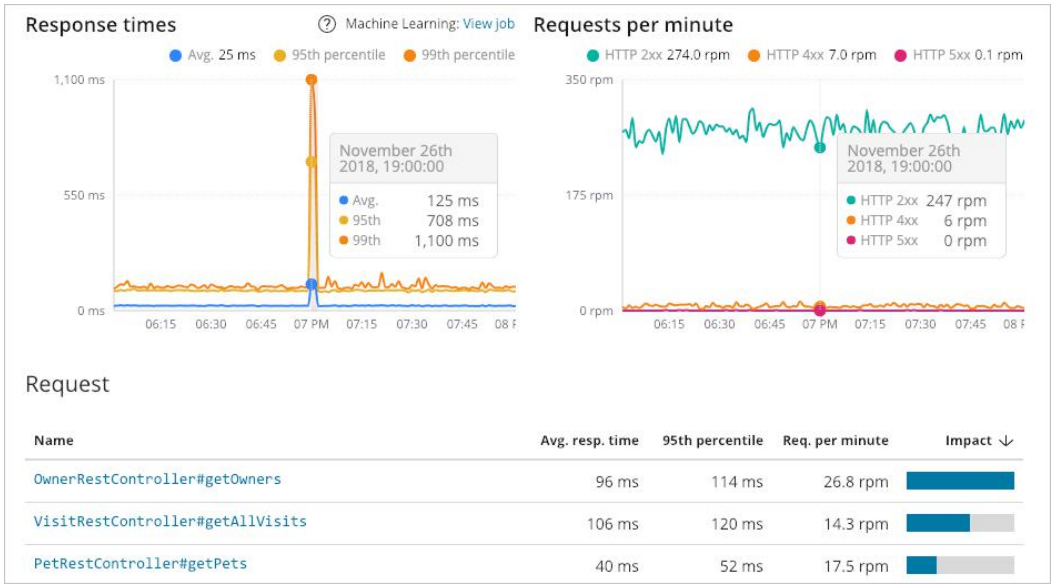
i 在页面左上角单击☰图标，展开左侧导航栏。

ii 在左侧导航栏，单击 Observability 下的 APM。

iii 在 Services 页签下查看 APM 的所有服务。

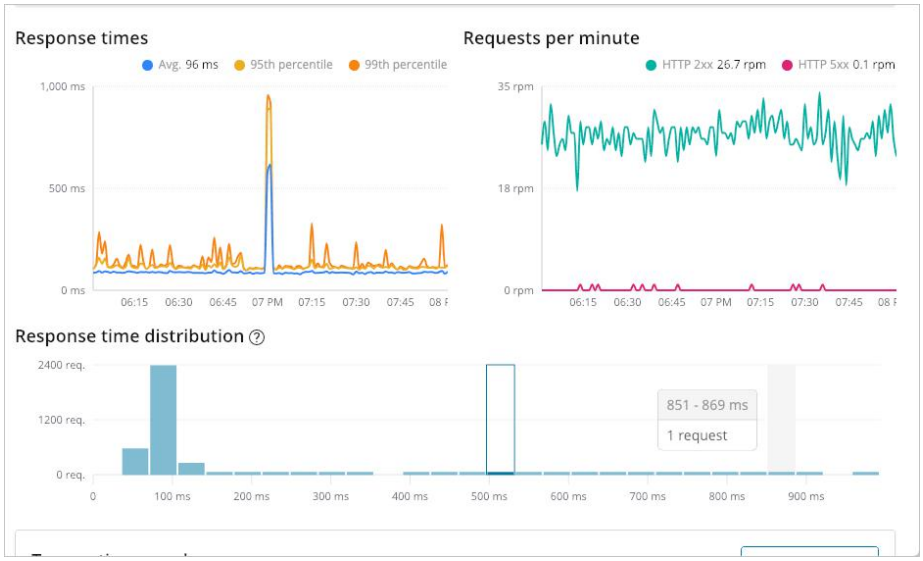
5. 单击对应服务，查看服务的具体信息。

以 petclinic-spring 服务为例，服务信息如下图（每项服务都具有类似的布局）。



o 左上角显示了响应时间的平均值、p95 和 p99 值，以显示发生异常的具体时间。同时您可以在图表上隐藏不关注的数值曲线，以便更好地了解异常值对整个服务影响。将鼠标移到任

- 何一个图表上时，可以得到一个弹出窗口显示当时的具体摘要。从图中可以看出，响应时长的突增没有导致任何服务器发生 500 响应错误。
- 通过查看下方请求明细，您可以看到应用程序中每个请求都来自不同的节点，您可以通过使用各种代理 API 来扩充默认节点。支持按列标题、响应时长和影响列排序，影响列考虑了对应请求的延迟和热度数据。以影响最高的 getOwners 请求为例 ，它的平均延迟并不高，为 96 毫秒。单击该请求名称，可以看到对应请求的详细信息，图中反应最慢的请求也不到一秒钟。



- 向下滚动，可以看到该请求处理的操作瀑布视图。其中有很多 SELECT 语句正在进行，使用 APM 能够看到正在执行的实际查询。图 1. 请求处理的操作瀑布视图

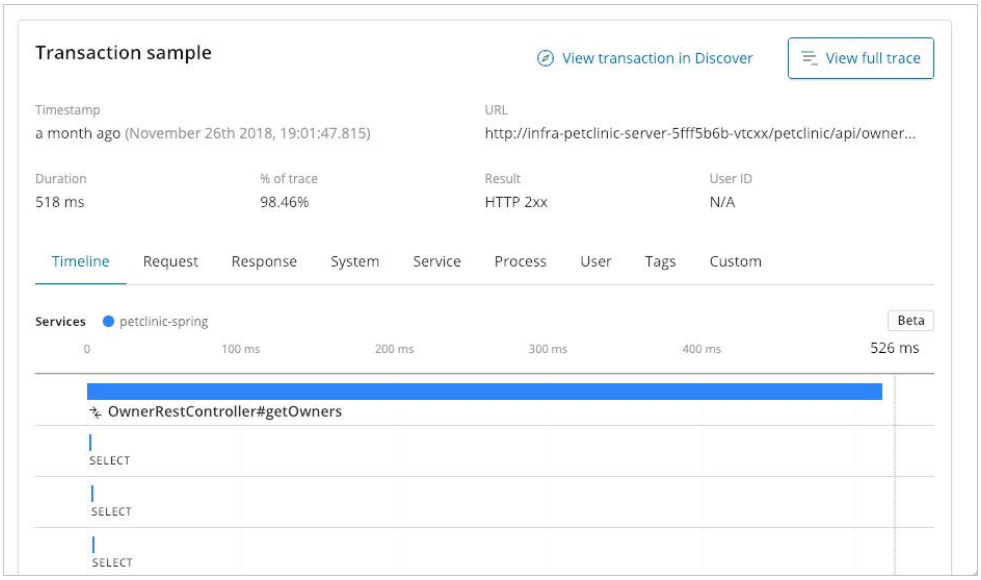
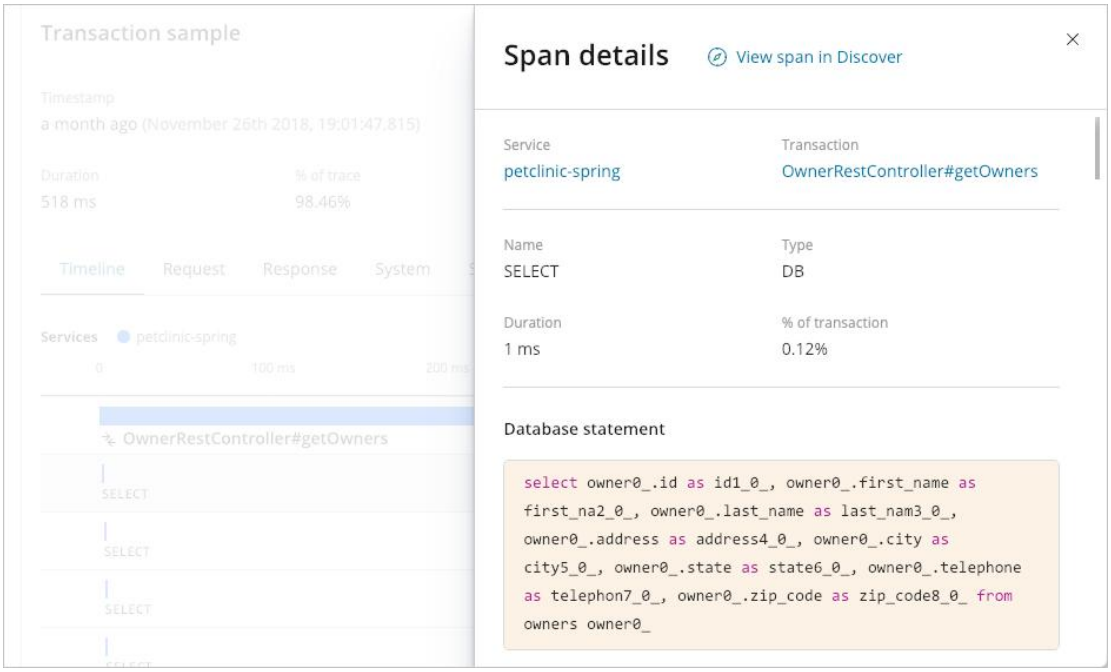


图 2. 正在执行的实际查询



由于该应用程序中的多层微服务均安装了 APM Agent，您可以单击 View full trace，查看该请求的详细信息并显示全链路中参与处理的所有组件，从而可以实现一个从浏览器层开始的请求的分布式追踪。图 3. 请求的详细信息

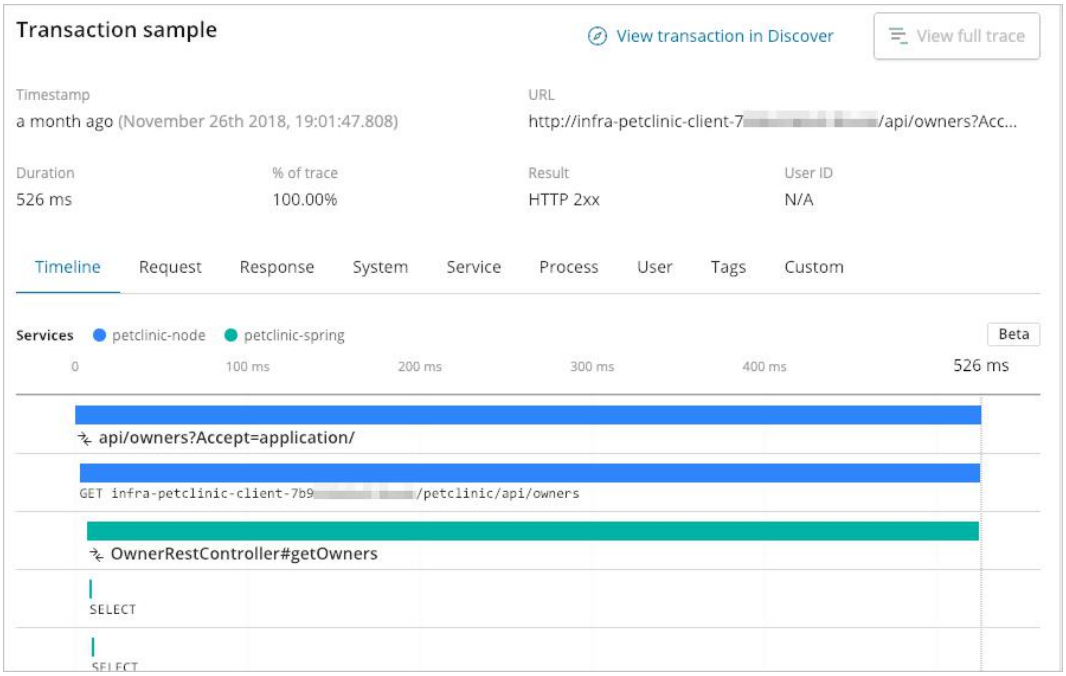
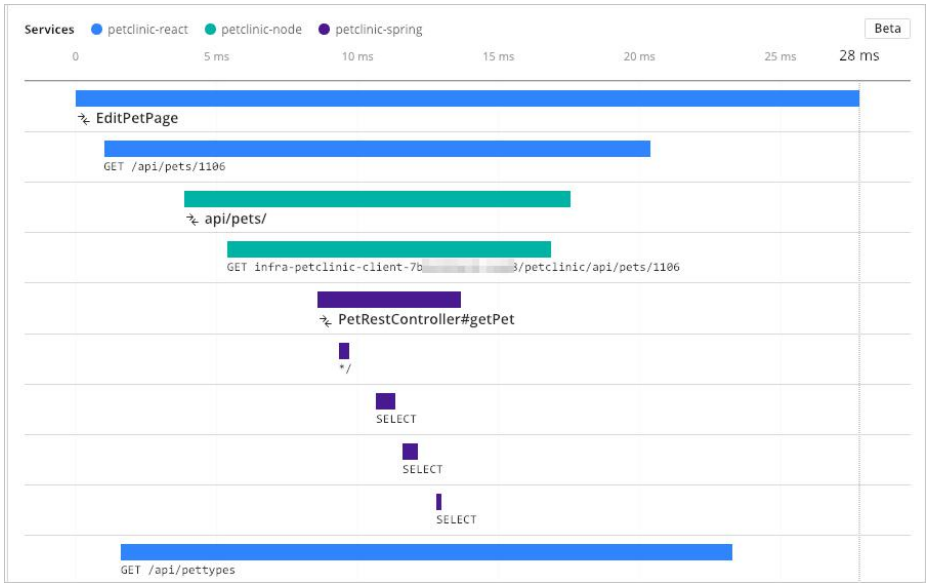
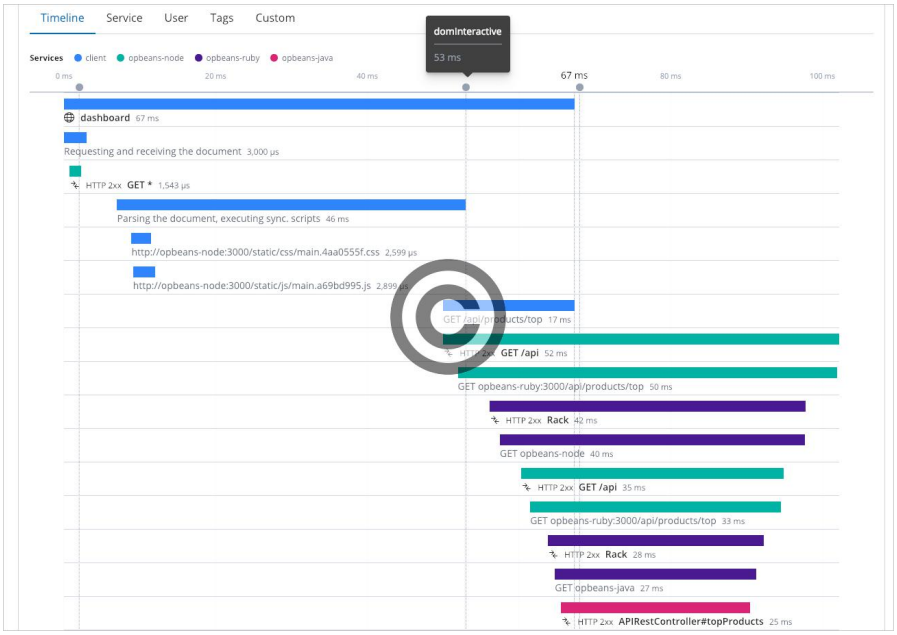


图 4. 全链路中的所有组件



真实用户监测：为了从分布式跟踪中获得最大价值，您需要尽可能多地监测组件和服务，包括使用真实用户监测 RUM（Real User Monitoring）。快速的服务响应时间并不意味着请求在浏览器中很快就能完成， APM 能够帮助业务衡量最终用户在浏览器中的终端体验。下图的分布式跟踪显示了四种不同的服务，包含了网络浏览器等多项。



参考文档：利用 Elasticsearch 和 Elastic APM 监测应用

通过 Elastic 实现 Kubernetes 容器全观测

摘要：Elastic 可观测性是通过 Kibana 可视化能力，将日志、指标及 APM 数据结合在一起，实现对容器数据的观测和分析。当您的应用程序以 Pods 方式部署在 Kubernetes 中，可以在 Kibana 中查看 Pods 生成日志、主机和网络上的事件指标及 APM 数据，逐步缩小排查范围进行故障排查。本文介绍具体的实现方法。

前提条件

- 创建阿里云 Elasticsearch 实例，版本为 6.8，并开启白名单和自动创建索引功能。
  具体操作，请参见创建阿里云 Elasticsearch 实例、配置 Elasticsearch 公网或私网访问白名单和配置 YML 参数。
- 创建 ACK 集群，并运行 Pod 服务。本文的测试场景使用的 Kubernetes 版本为 1.18.8-aliyun.1，ECS 规格为 2C8G。
  具体操作，请参见创建 Kubernetes 托管版集群。
- 配置 kubectl 客户端，可通过 kubectl 连接 Kubernetes 集群。
  具体操作，请参见通过 kubectl 管理 Kubernetes 集群。

背景信息

本文介绍如何使用 Elastic 实现对 Kubernetes 容器的全方位检测，具体内容如下：

- 通过 Metricbeat 实现指标采集
- 通过 Filebeat 实现日志采集
- 通过 Elastic APM 实现应用程序性能监测

关于 Metricbeat、Filebeat 及 APM 更多的特性说明，请参见 Infrastructure monitoring、Log monitoring 和 Elastic APM。

通过 Metricbeat 实现指标采集

在 Kubernetes 上部署 Metricbeat，有以下两种方式：

- DaemonSet：保证每个节点运行一个 Pod，可以实现对 Host 指标、System 指标、Docker 统计信息以及 Kubernetes 上运行的所有服务指标的采集。
- Deployment：部署单个 Metricbeat 实例，该实例用于检索整个集群的唯一指标，如 kubernetes event 或者 kube-state-metrics。

注意：

- 本文以同时使用 DaemonSet 和 Deployment 的部署方式为例介绍如何部署 Metricbeat 容器，您也可以仅使用 DaemonSet 方式或 Deployment 方式进行部署。
- Metricbeat 依赖 kube-state-metrics 监控，部署前需要确保已完成 kube-state-metrics 的部署。阿里云 ACK 容器默认在 arms-prom 命名空间下部署了 kube-state-metrics 监控。

1. 通过 kubectl 访问云容器，下载 Metricbeat 配置文件。

```
curl -L -O https://raw.githubusercontent.com/elastic/beats/6.8/deploy/kubernetes/metricbeat-kubernetes.yaml
```

2. 修改 Metricbeat 配置文件。

注意：官方下载的 YML 文件中，DaemonSets 和 Deployments 的资源使用 extensions/v1beta1，而 v1.18 及以上版本的 Kubernetes，DaemonSets、Deployments 和 Replicasets 资源的 extensions/v1beta1 API 将被废弃，请使用 apps/v1。

i 修改 **kind: Deployment** 和 **kind: DaemonSet** 下的配置信息。

- 修改环境变量，具体内容如下：

```
env:
- name: ELASTICSEARCH_HOST
  value: es-cn-nif23p3mo0065****.elasticsearch.aliyuncs.com
- name: ELASTICSEARCH_PORT
  value: "9200"
- name: ELASTICSEARCH_USERNAME
  value: elastic
- name: ELASTICSEARCH_PASSWORD
  value: ****
- name: KIBANA_HOST
  value: es-cn-nif23p3mo0065****-kibana.internal.elasticsearch.aliyuncs.com
- name: KIBANA_PORT
  value: "5601"
```

注意：下载的 Metricbeat 配置文件中默认未定义 Kibana 相关变量，可以通过容器 env 传入变量信息。

参数	说明
ELASTICSEARCH_HOST	阿里云 Elasticsearch 实例的私网地址。
ELASTICSEARCH_PORT	阿里云 Elasticsearch 实例的私网端口。
ELASTICSEARCH_USERNAME	阿里云 Elasticsearch 的用户名，默认值 elastic。
ELASTICSEARCH_PASSWORD	elastic 用户的密码。
KIBANA_HOST	Kibana 私网地址。
KIBANA_PORT	Kibana 私网端口。

- 增加 spec.selector 配置信息，具体内容如下：

```
## kind: DaemonSet
spec:
  selector:
    matchLabels:
      k8s-app: metricbeat
  template:
    metadata:
      labels:
        k8s-app: metricbeat

## kind: Deployment
spec:
  selector:
    matchLabels:
      k8s-app: metricbeat
  template:
    metadata:
      labels:
        k8s-app: metricbeat
```

ii. 分别在 **name: metricbeat-daemonset-config** 和 **name: metricbeat-deployment-config** 下，配置 Kibana Output 信息，调用文件中配置的环境变量。

```
output.elasticsearch:
      hosts: ['${ELASTICSEARCH_HOST:elasticsearch}:${ELASTICSEARCH_PORT:9200}']
      username: ${ELASTICSEARCH_USERNAME}
      password: ${ELASTICSEARCH_PASSWORD}
setup.kibana:
      host: "https://${KIBANA_HOST}:${KIBANA_PORT}"
setup.dashboards.enabled: true
```

iii 修改 **metricbeat-daemonset-modules** 配置，定义 **system** 模块监控的 cpu、load、memory、network 等系统指标，以及 **kubernetes** 模块可以获取的监控指标。

说明：关于 Metricbeat 更多的模块配置及指标说明，请参见 System module 和 Kubernetes module。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: metricbeat-daemonset-modules
  namespace: kube-system
  labels:
    k8s-app: metricbeat
data:
  system.yml: |-
    - module: system
      period: 10s
      metricsets:
        - cpu
        - load
        - memory
        - network
        - process
        - process_summary
        - core
        - diskio
        - socket
      processes: ['.*']
      process.include_top_n:
        by_cpu: 5      # include top 5 processes by CPU
        by_memory: 5   # include top 5 processes by memory

    - module: system
```

```
        period: 1m
        metricsets:
          - filesystem
          - fsstat
        processors:
        - drop_event.when.regexp:
            system.filesystem.mount_point: '^/(sys|cgroup|proc|dev|etc|host|lib)
($|/)'
    kubernetes.yml: |-
      - module: kubernetes
        metricsets:
          - node
          - system
          - pod
          - container
          - volume
        period: 10s
        host: ${NODE_NAME}
        hosts: ["localhost:10255"]
```

iv修改 **metricbeat-deployment-modules** 配置，获取 kube-state-metric 监控指标和 event 服务指标。

注意：Metricbeat 服务创建在 kube-system 的 namespace 下，kube-state-metrics 默认创建在 arms-prom 的 namespace 下，由于 namespace 不同，所以 hosts 的命名格式为 kube-state-metrics.<namespace>:8080。如果 Metricbeat 服务与 kube-state-metrics 模块都创建在同一 namespace 下，则 hosts 的命名格式为 kube-state-metrics:8080。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: metricbeat-deployment-modules
  namespace: kube-system
  labels:
    k8s-app: metricbeat
data:
  # This module requires `kube-state-metrics` up and running under `kube
-system` namespace
  kubernetes.yml: |-
    - module: kubernetes
      metricsets:
        - state_node
        - state_deployment
        - state_replicaset
        - state_pod
        - state_container
      period: 10s
      host: ${NODE_NAME}
      hosts: ["kube-state-metrics.arms-prom:8080"]
    # Uncomment this to get k8s events:
    - module: kubernetes
      metricsets:
        - event
```

v 配置 RBAC 权限声明，保证 Metricbeat 可以获取到 Kubernetes 集群资源信息。

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: metricbeat
subjects:
- kind: ServiceAccount
  name: metricbeat
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: metricbeat
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: metricbeat
  labels:
    k8s-app: metricbeat
rules:
- apiGroups: [""]
  resources:
  - nodes
  - namespaces
  - events
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups: ["extensions"]
  resources:
  - replicasets
  verbs: ["get", "list", "watch"]
- apiGroups: ["apps"]
  resources:
```

```
  - statefulsets
  - deployments
  verbs: ["get", "list", "watch"]
- apiGroups:
  - ""
  resources:
  - nodes/stats
  verbs:
  - get
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: metricbeat
  namespace: kube-system
  labels:
    k8s-app: metricbeat
---
```

3. 部署 Metricbeat，并查看资源状态。

   通过 kubectl 执行以下命令：

```
kubectl apply -f metricbeat-kubernetes.yaml
kubectl get pods -n kube-system
```

注意：请确保 Pods 资源均处于 Running 状态，否则在 Kibana 平台有可能会看不到相应数据。

4. 在 Kibana 查看监测数据。

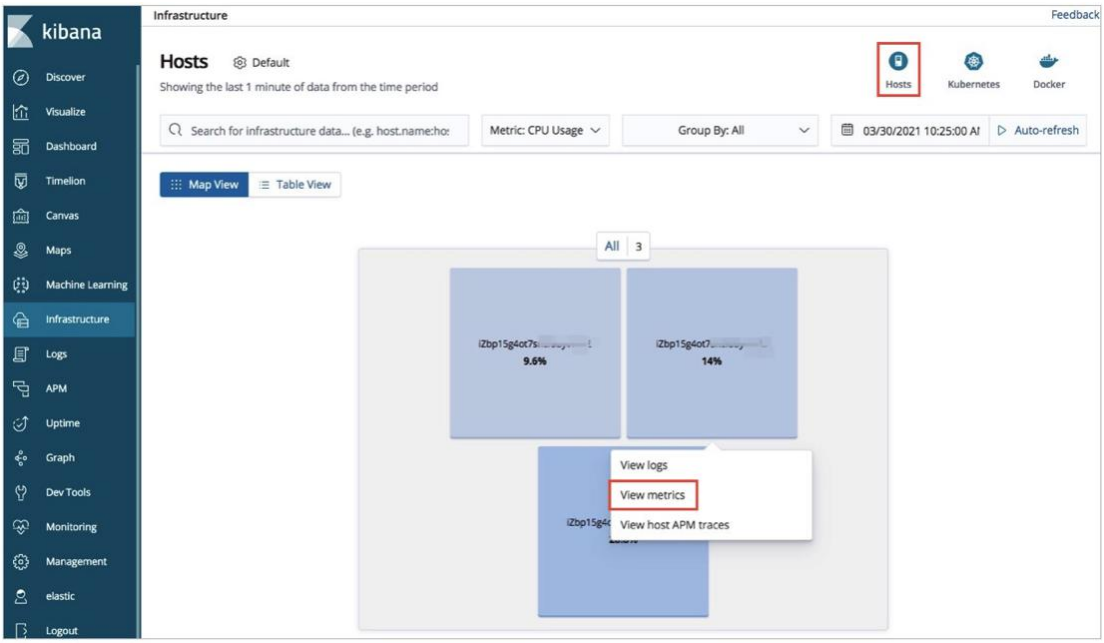
   i 登录目标阿里云 Elasticsearch 实例的 Kibana 控制台。

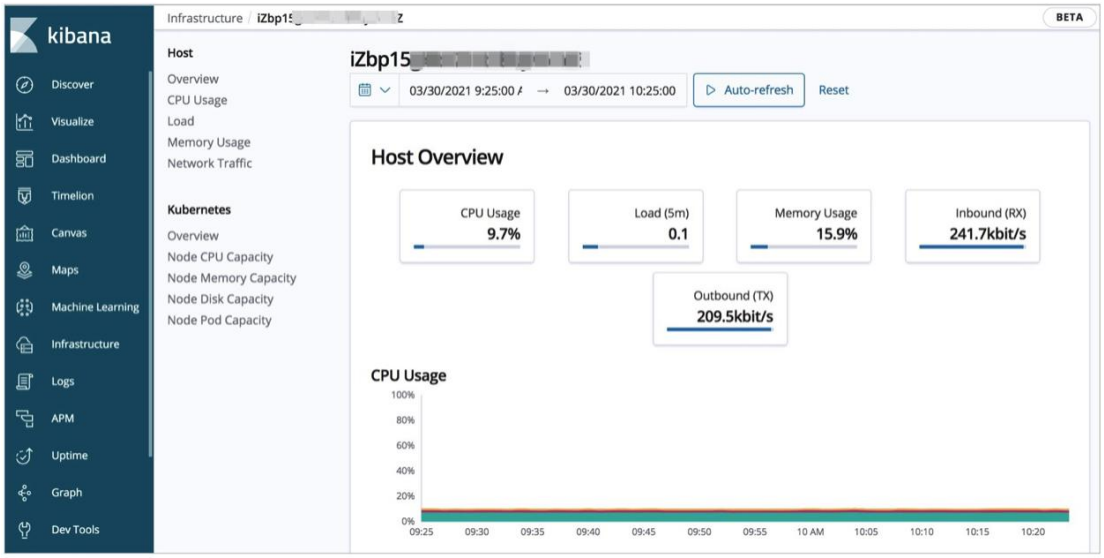
   具体操作步骤请参见登录 Kibana 控制台。

   ii 单击左侧菜单 Infrastructure。

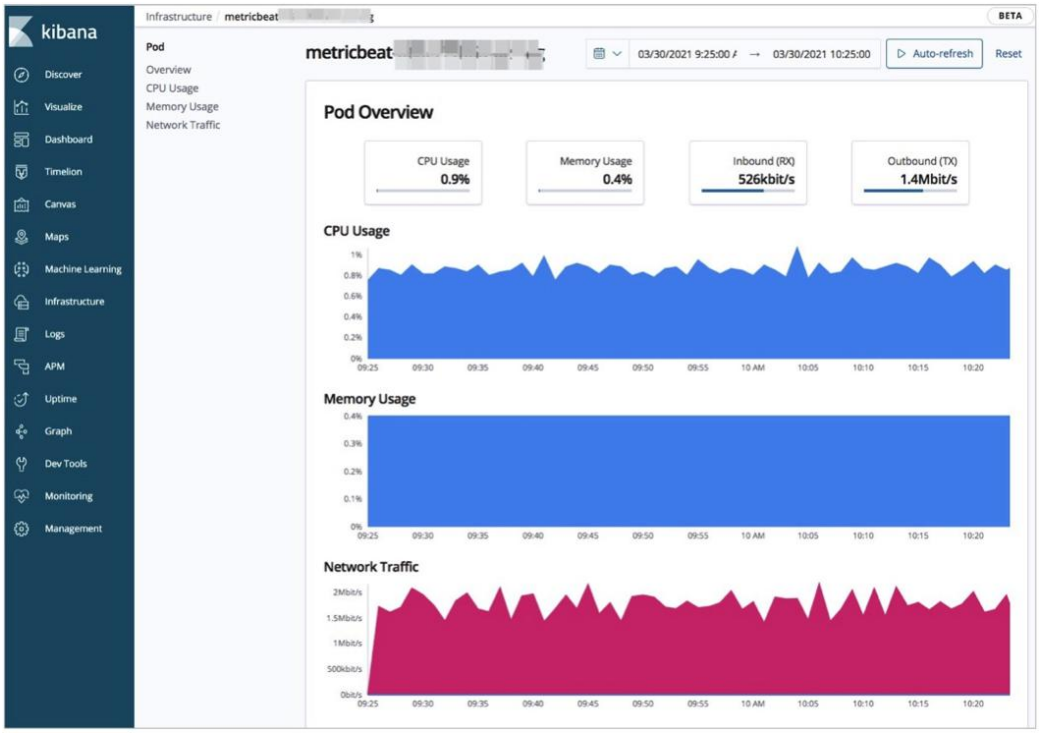
iii 查看 Hosts、Kubernetes Pods 对应的 Metrics 信息。

- 查看 Hosts 对应的 Metrics 信息：单击右上角 **Hosts**，在 **Map View** 页签下，单击指定 Host，选择 **View metrics**，就可以查看对应的 CPU、Load、Memory 等指标数据。



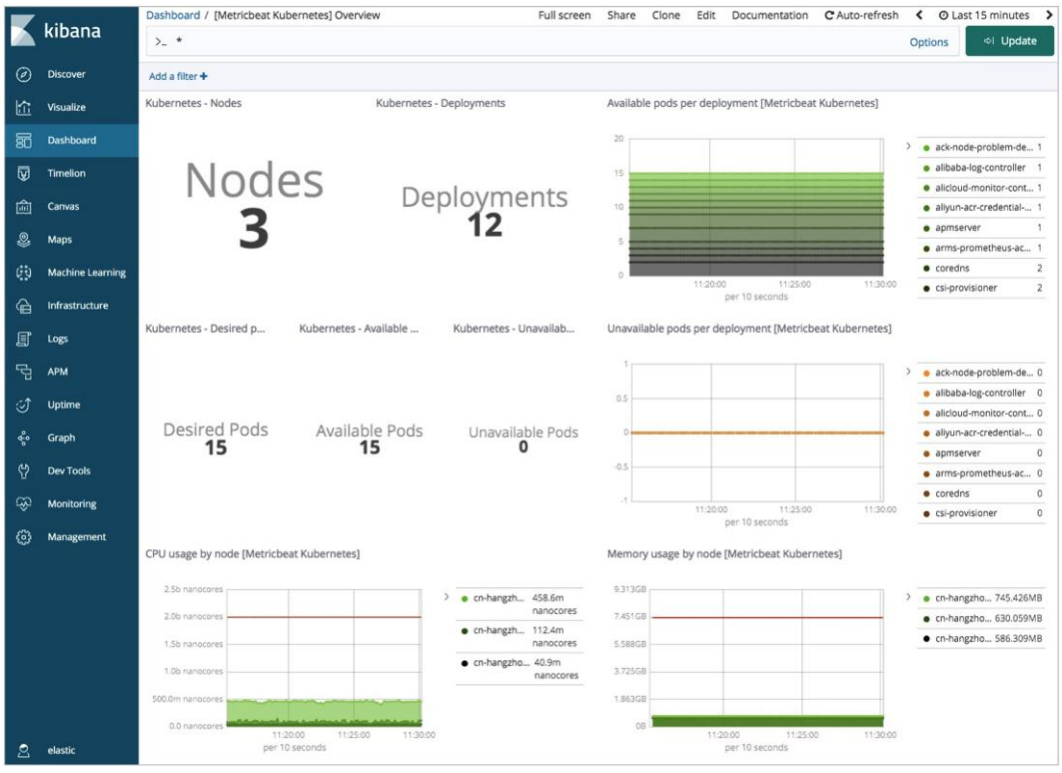


- 查看 Kubernetes Pods 对应的 Metrics 信息：单击右上角 **Kubernetes**，在 **Map View** 页签下，单击指定 Pod，选择 **View metrics**，就可以查看对应的 CPU、Memory、Network 等指标数据。



iv查看 Kubernetes 集群资源总览数据。

单击左侧菜单 **Dashboard**，选择[Metricbeat Kubernetes] Overview，就可以查看集群资源总览数据。



通过 Filebeat 实现日志采集

本文示例中，Filebeat 容器使用 DaemonSet 控制器部署，确保集群上每一个节点都有一个正在运行的实例采集数据，并且配置文件中的资源部署在 kube-system 命名空间下。您如果需要改变，可以手动更改配置文件。

1. 下载 Filebeat 配置文件。

   通过 kubectl 访问云容器，下载 Filebeat 配置文件。

```
curl -L -O https://raw.githubusercontent.com/elastic/beats/6.8/deploy/kubernetes/filebeat-kubernetes.yaml
```

2. 修改 Filebeat 配置文件。

   i 修改 **kind: DaemonSet** 下的环境变量。

```
env:
- name: ELASTICSEARCH_HOST
  value: es-cn-nif23p3mo0065****.elasticsearch.aliyuncs.com
- name: ELASTICSEARCH_PORT
  value: "9200"
- name: ELASTICSEARCH_USERNAME
  value: elastic
- name: ELASTICSEARCH_PASSWORD
  value: ****
- name: KIBANA_HOST
  value: es-cn-nif23p3mo0065****-kibana.internal.elasticsearch.aliyuncs.com
- name: KIBANA_PORT
  value: "5601"
- name: NODE_NAME
  valueFrom:
     fieldRef:
        fieldPath: spec.nodeName
```

参数	说明
ELASTICSEARCH_HOST	阿里云 Elasticsearch 实例的私网地址。
ELASTICSEARCH_PORT	阿里云 Elasticsearch 实例的私网端口。
ELASTICSEARCH_USERNAME	阿里云 Elasticsearch 的用户名，默认值 elastic。
ELASTICSEARCH_PASSWORD	elastic 用户的密码。
KIBANA_HOST	Kibana 私网地址。
KIBANA_PORT	Kibana 私网端口。
NODE_NAME	Kubernetes 集群 Host。

ii 修改 **name: filebeat-config** 对应的 ConfigMap 配置信息，配置 Kibana Output 信息，调用文件中配置的环境变量。

```
output.elasticsearch:
        hosts: ['${ELASTICSEARCH_HOST:elasticsearch}:${ELASTICSEARCH_
PORT:9200}']
        username: ${ELASTICSEARCH_USERNAME}
        password: ${ELASTICSEARCH_PASSWORD}
setup.kibana:
        host: "https://${KIBANA_HOST}:${KIBANA_PORT}"
```

iii 配置 Kubernetes，采集容器日志。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: filebeat-inputs
  namespace: kube-system
  labels:
    k8s-app: filebeat
data:
  kubernetes.yml: |-
    - type: docker
      containers.ids:
      - "*"
      processors:
        - add_kubernetes_metadata:
            host: ${NODE_NAME}
            in_cluster: true
---
```

3. 在 Kubernetes 中部署 Filebeat，并查看资源状态。

   通过 kubectl 执行以下命令：

```
kubectl apply -f filebeat-kubernetes.yaml
kubectl get pods -n kube-system
```

注意：请确保 Pods 资源均处于 Running 状态，否则在 Kibana 平台有可能会看不到相应数据。

4. 在 Kibana 查看实时日志。

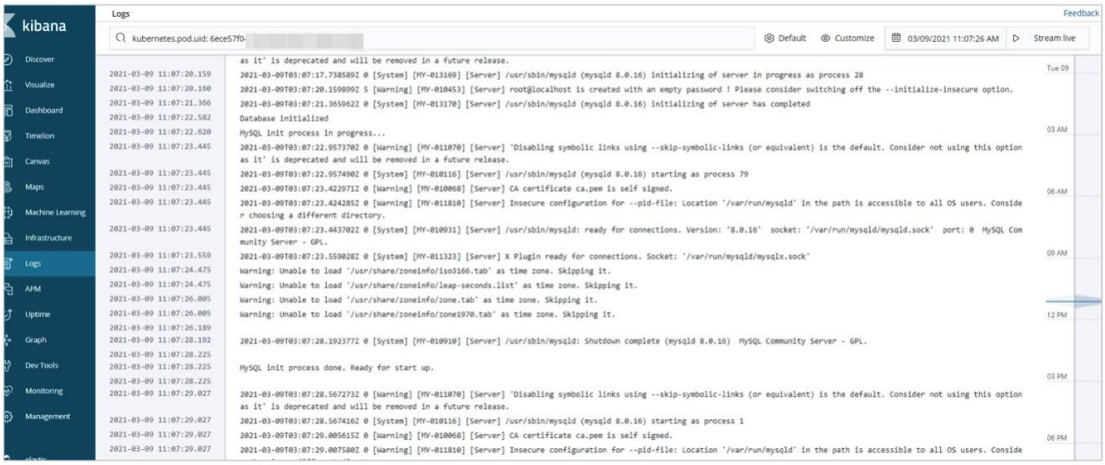
   i 登录目标阿里云 Elasticsearch 实例的 Kibana 控制台。

   具体操作步骤请参见登录 Kibana 控制台。

   ii 查看 Hosts、Kubernetes Pods 对应的日志。

   o 查看 Hosts 对应的日志信息：单击右上角 **Hosts**，在 **Map View** 页签下，单击指定 Host，选择 **View logs**，就可以查看 Host 实时日志。

- 查看 Kubernetes Pods 对应的日志信息：单击右上角 **Kubernetes**，在 **Map View** 页签下，单击指定 Pod，选择 **View logs**，就可以查看 Pod 实时日志。



通过 Elastic APM 实现应用程序性能监测

Elastic APM 是基于 Elastic Stack 构建的应用程序性能监控系统。它提供实时监控软件服务和应用程序的功能，采集传入请求的响应时间和数据库查询、高速缓存调用及外部 HTTP 请求等的详细性能信息，帮助您更快速的查明并修复性能问题。Elastic APM 还可以自动收集未处理的错误、异常及调用栈，帮助您识别出新错误，并关注对应错误发生的次数。

关于 Elastic APM 更多的介绍，请参见 Elastic APM Overview。

1. 部署 APM Server 容器。

   本文示例使用 Kubernetes 部署，通过 ConfigMap 控制器定义 apm-server.yml 文件，并初始化 Pods 启动，通过 service 实现服务自动发现和负载均衡。

   i 配置 apm-server.yml 文件。

   完整的配置文件内容如下：

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: apm-deployment-config
  namespace: kube-system
  labels:
    k8s-app: apmserver
data:
  apm-server.yml: |-
      apm-server.host: "0.0.0.0:8200"
      output.elasticsearch:
          hosts: ['${ELASTICSEARCH_HOST:elasticsearch}:${ELASTICSEARC
H_PORT:9200}']
          username: ${ELASTICSEARCH_USERNAME}
          password: ${ELASTICSEARCH_PASSWORD}
      setup.kibana:
          host: "https://${KIBANA_HOST}:${KIBANA_PORT}"


---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apmserver
  namespace: kube-system
  labels:
    k8s-app: apmserver
spec:
  selector:
    matchLabels:
      k8s-app: apmserver
  template:
    metadata:
```

```
      labels:
        k8s-app: apmserver
    spec:
      serviceAccountName: apmserver
      hostNetwork: true
      dnsPolicy: ClusterFirstWithHostNet
      containers:
      - name: apmserver
        image: docker.elastic.co/apm/apm-server:6.8.14
        args: [
          "-c", "/etc/apm-server.yml",
          "-e",
        ]
        env:
        - name: ELASTICSEARCH_HOST
          value: es-cn-oew20i5h90006****.elasticsearch.aliyuncs.com
        - name: ELASTICSEARCH_PORT
          value: "9200"
        - name: ELASTICSEARCH_USERNAME
          value: elastic
        - name: ELASTICSEARCH_PASSWORD
          value: ****
        - name: KIBANA_HOST
          value: es-cn-oew20i5h90006****-kibana.internal.elasticsearch.aliyu
ncs.com
        - name: KIBANA_PORT
          value: "5601"
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        securityContext:
          runAsUser: 0
```

```
        resources:
          limits:
            memory: 50Mi
          requests:
            cpu: 20m
            memory: 30Mi
        volumeMounts:
        - name: config
          mountPath: /etc/apm-server.yml
          readOnly: true
          subPath: apm-server.yml
      volumes:
      - name: config
        configMap:
          defaultMode: 0600
          name: apm-deployment-config
---
apiVersion: v1
kind: Service
metadata:
  name: apmserver
  namespace: kube-system
  labels:
    k8s-app: apmserver
spec:
  clusterIP: None
  ports:
  - name: http-metrics
    port: 8200
    targetPort: 8200
  selector:
    k8s-app: apmserver
---
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: apmserver
  namespace: kube-system
  labels:
    k8s-app: apmserver
---
```

注意：

- Deployment 部署资源中，使用 docker.elastic.co/apm/apm-server:6.8.14 镜像部署 Pods 容器，镜像版本需要和阿里云 Elasticsearch 实例版本一致。
- 通过 service 对集群暴露 8200 服务端口，保证 APM Agents 可与 APM Server 通信。

参数	说明
ELASTICSEARCH_HOST	阿里云 Elasticsearch 实例的私网地址。
ELASTICSEARCH_PORT	阿里云 Elasticsearch 实例的私网端口。
ELASTICSEARCH_USERNAME	阿里云 Elasticsearch 的用户名，默认值 elastic。
ELASTICSEARCH_PASSWORD	elastic 用户的密码。
KIBANA_HOST	Kibana 私网地址。
KIBANA_PORT	Kibana 私网端口。
NODE_NAME	Kubernetes 集群 Host。

ii 部署 APM Server 容器，并查看资源状态。

通过 kubectl 执行以下命令：

```
kubectl apply -f apm-server.yml
kubectl get pods -n kube-system
```

注意：请确保 Pods 资源均处于 Running 状态，否则在 Kibana 平台有可能会看不到相应数据。

2. 配置 APM Agents。

   本文示例是通过 Spring Boot 实现一个简单的 Web 应用并打包为 JAR 包，并将 JAR 包和从 Maven Central 下载的最新 Java Agent 上传到服务器。详细信息，请参见 Spring Boot 和 Maven Central。

   i 登录 Kubernetes 节点，在工作目录中创建 Dockerfile 文件，文件名为 myapply。Dockerfile 文件内容如下：

```
FROM frolvlad/alpine-oraclejdk8
MAINTAINER peterwanghao.com
VOLUME /tmp
ADD spring-boot-0.0.1-SNAPSHOT.jar spring-boot-0.0.1-SNAPSHOT.jar
ADD elastic-apm-agent-1.21.0.jar elastic-apm-agent-1.21.0.jar
EXPOSE 8080
ENTRYPOINT ["java","-javaagent:/elastic-apm-agent-1.21.0.jar","-Delastic.apm.service_name=my-application","-Delastic.apm.server_url=http://apmserver:8200","-Delastic.apm.application_packages=com.example","-jar","/spring-boot-0.0.1-SNAPSHOT.jar"]
```

ENTRYPOINT 定义容器启动时运行的 Java 命令及参数如下：

参数	说明
-javaagent	指定 APM Agent 代理 JAR 包。
-Delastic.apm.service_name	APM Service Name，允许以下字符：a-z、A-Z、0-9、-、_及空格。
-Delastic.apm.server_url	APM Server URL，http://apmserver:8200 是在 apm-server.yml 文件中的 service 定义。
-Delastic.apm.application_packages	应用程序的基础软件包。
-jar	指定应用 JAR 包。

ii 通过 docker build 命令和 Dockerfile 定义的 myapply 文件构建镜像。

在当前路径下，执行以下命令：

```
docker build -t myapply .
```

iii 将构建好的镜像加载到其他容器节点。

iv配置 Pods 部署文件，文件名为 my-application.yaml。

文件内容如下：

```
---
apiVersion: v1
kind: Pod
metadata:
  name: my-apply
  namespace: kube-system
  labels:
    app: my-apply
spec:
  containers:
    - name: my-apply
```

```
        image: myapply:latest
        ports:
          - containerPort: 8080
        imagePullPolicy: Never
---
apiVersion: v1
kind: Service
metadata:
  name: my-apply
  namespace: kube-system
  labels:
    app: my-apply
spec:
  type: NodePort
  ports:
  - name: http-metrics
    port: 8080
    nodePort: 30000
  selector:
    app: my-apply

---
```

说明：image 为构建好的镜像文件。

iv通过 kubectl 执行以下命令，部署 Pods。

```
kubectl apply -f my-application.yaml
```

v 待所有 Pods 资源均处于 Running 状态后，使用 Curl 访问主机的 30000 端口。执行命令如下：

```
curl http://10.7.XX.XX:30000
```

说明：10.7.XX.XX 为 Kubernetes 的节点 IP 地址。

能够访问对应主机后，APM Agents 部署成功。

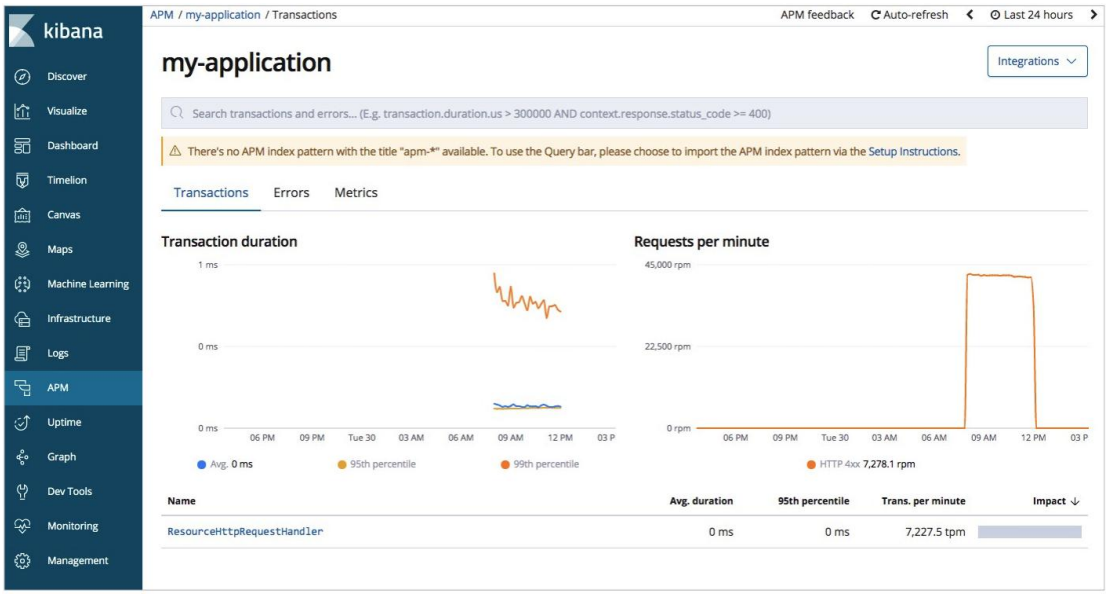
3. 在 Kibana 控制台查看 APM 监控数据。

i 登录目标阿里云 Elasticsearch 实例的 Kibana 控制台。

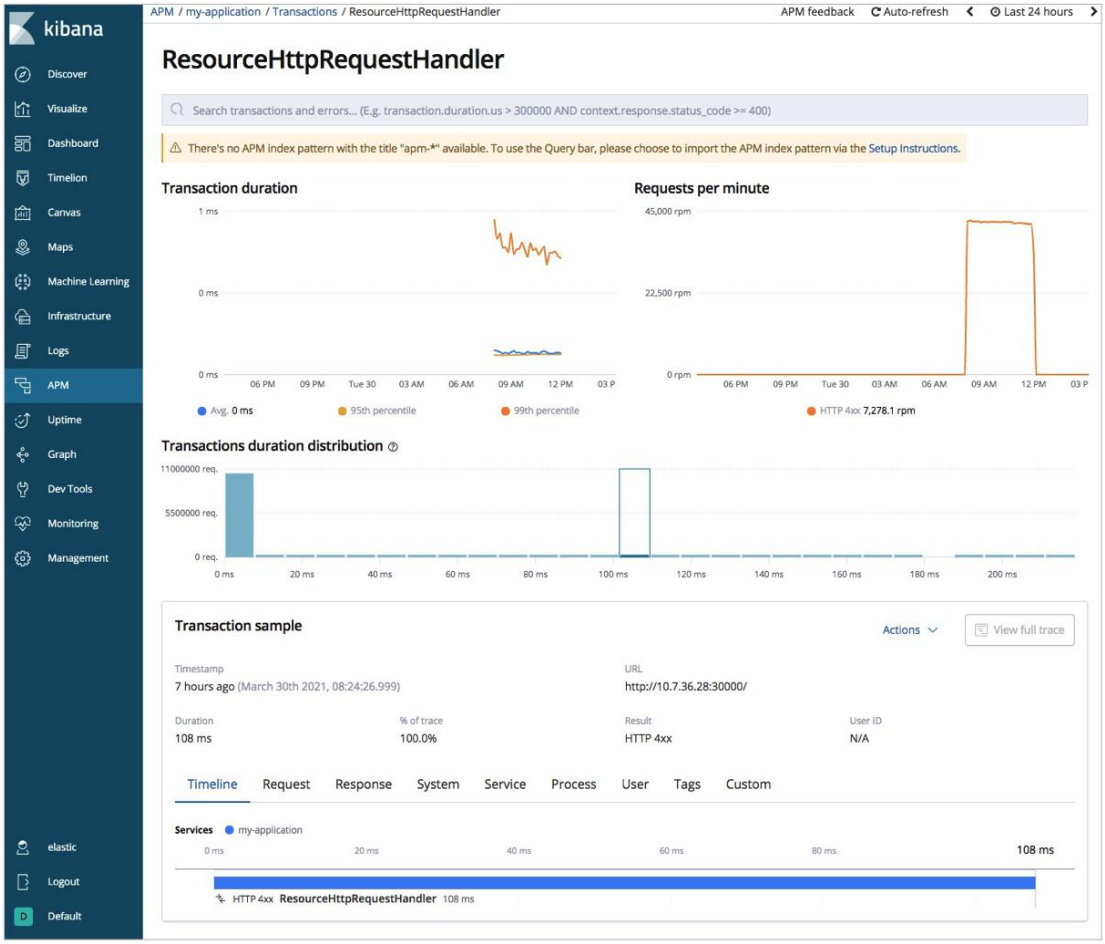
具体操作步骤请参见登录 Kibana 控制台。

ii 单击左侧菜单 APM。

iii 单击目标应用程序，本文示例为 my-application，可以看到服务的整体性能数据。

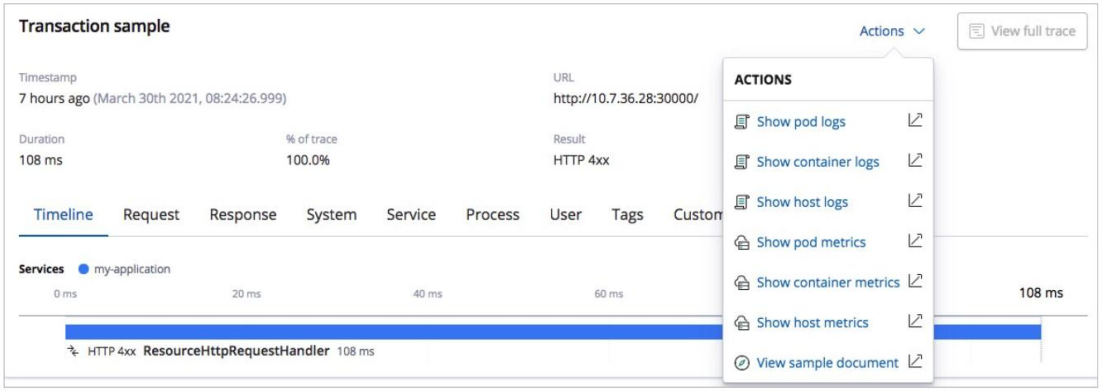


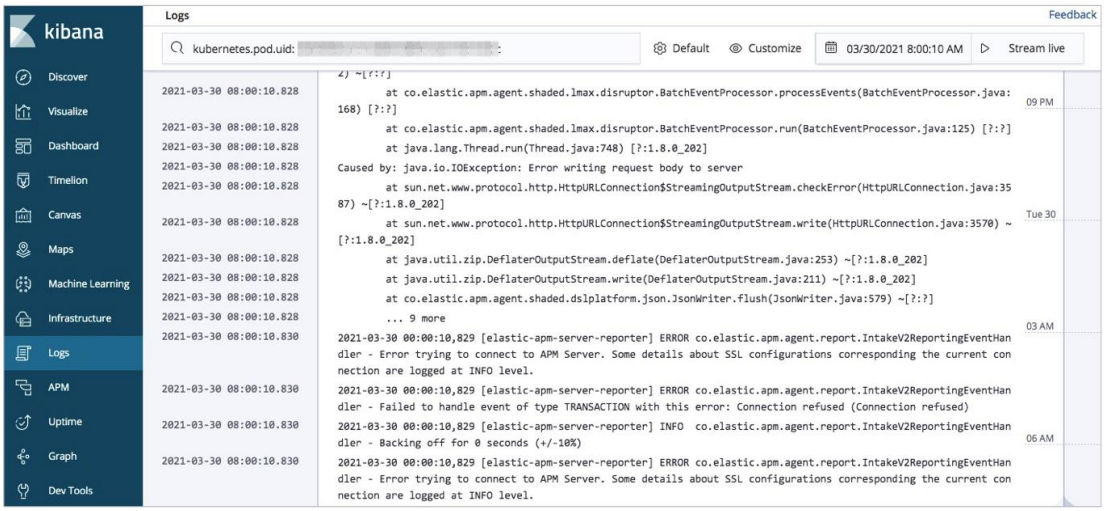
iv 单击对应的请求接口，可以看到具体的请求信息。



v 查看 Hosts、Pods 相关日志和指标数据。

单击 Actions，选择 Show pod logs、Show pod metrics 等，查看对应的日志和指标数据。





常见问题

- 问题：Kubernetes 配置文件中 **resources.requests** 下资源设置较大，Pods 无法启动成功。

  解决方案：Metricbeat、Filebeat、APM 配置文件中都需要设置 **resources.requests**，建议根据 Kubernetes 集群规格适当调整设置的值。

- 问题：在部署 Metricbeat、Filebeat、APM 容器时，一直报错。报错内容类似：`no matches for kind "DaemonSet" in version "extensions/v1beat1"`。

  解决方案：官方下载的 YML 文件中，Daemonsets 和 Deployments 的资源使用 extensions/v1beta1，而 v1.18 及以上版本的 Kubernetes，Daemonsets、Deployments 和 Replicasets 资源的 extensions/v1beta1 API 将被废弃，请使用 apps/v1。

钉钉扫码加入
“Elasticsearch 技术交流群”

扫码了解
“更多阿里云 Elasticsearch”

扫码订阅
“Elasticsearch 技术博客”

阿里云开发者“藏经阁”
海量免费电子书下载