

Elasticsearch全观测 技术解析与应用

搜索&推荐技术应用系列



阿里云开发者电子书系列

阿里云产品专家、Elastic首席架构师、阿里云MVP带你了解

- 全观测技术诉求及能力介绍
- 全观测技术原理及应用价值
- 全观测行业应用并快速上手



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

| 目录

走进阿里云 Elasticsearch	4
全观测技术原理与技术生态	15
全观测能力呈现与应用价值	26
ES 全观测性行业应用	37
阿里云 ES 全观测性配置	49

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。

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。



什么是Elasticsearch

业内最主流的信息检索、分析引擎，DB-Engine指数排行“全球热度No.7数据库，全球热度No.1检索引擎”



信息查询（查工商信息/物流信息等）



日志检索和分析（IT运维领域）



数据分析和可视化
(业务数据/交易数据等)



订单查询（被集成到ERP/CRM等系统）



地理位置查询（LBS/地图/O2O）

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

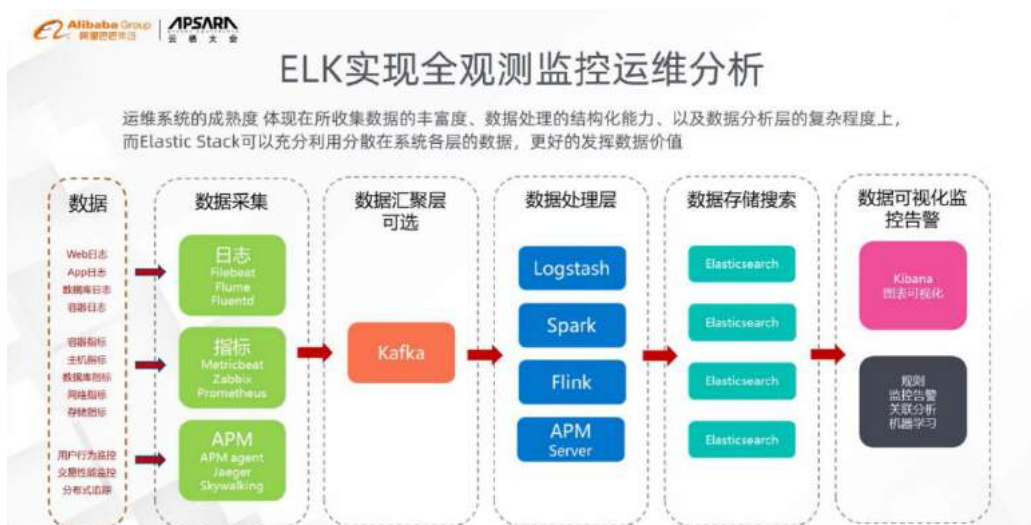
报警规则，甚至能利用机器学习的能力对未来情况进行预判。



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。

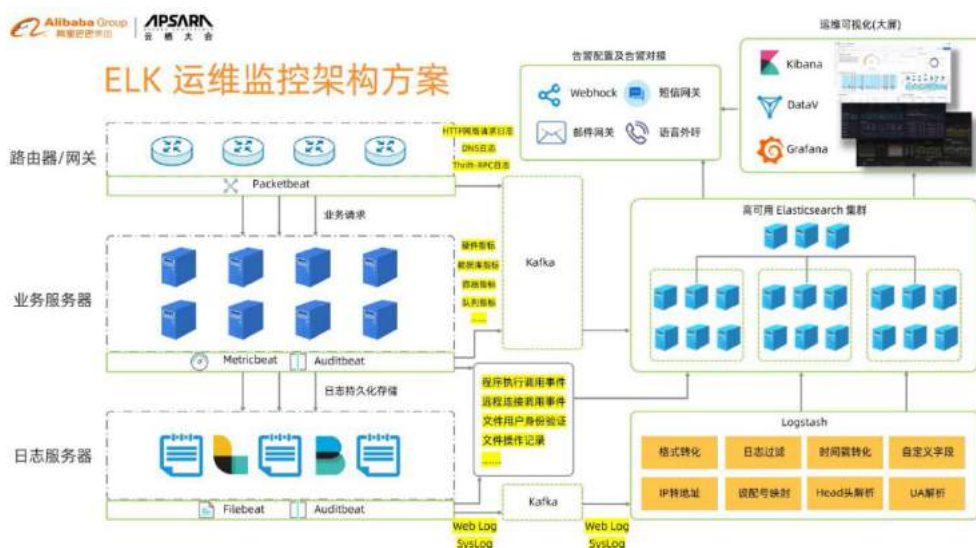
在直播、游戏等很多行业中，他们非常需要数据去做一些用户行为研究和整体链路的优化，从而为核心业务带来提升。而要让这套运维系统发挥业务价值，就取决于我们如何处理这些数据，如何更好地获取和分析数据所代表的真实含义。Elasticsearch 正具备这样的全观测能力，它区别于简单的日志处理和大数据分析，是能在同一个平台进行分析的全观测解决方案。



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，所以我们需要对系统进行优化，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。

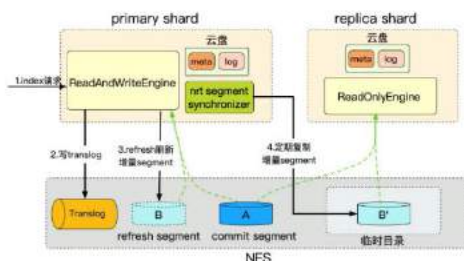
针对这些问题，阿里云团队推出了 Elasticsearch 日志增强版，专门针对 Elasticsearch 内核做了一些计算存储分离相关的改造，从而能让整体写入性能得到提升。同时，通过底层映射同一块物理盘，我们在集群伸缩的时候不需要迁移里面的数据，实现秒级弹性扩缩容。

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。



阿里云Elasticsearch日志增强版

云上ES在全观测场景下的核心技术剖析



- ✓ 索引分片一写多读，数据只保存一份
- ✓ 依赖云存储，保证数据可靠性
- ✓ IO fence机制保证数据一致性
- ✓ 内存物理复制，降低主备延迟（毫秒可见）

存储成本节约100%

物理数据仅存一份
FST off-heap 支持了更大内存磁盘比

写入性能提升100%

计算上避免了物理副本写入的多次cpu/iops开销

秒级弹性扩缩容

副本秒级快速扩缩容和迁移，轻松应对高峰流量

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。



什么时候应该用？

为日志分析与高并发场景能力打造

海量日志
日志量TB级

增量日志并发高
峰值写入10W docs/s

数据容灾需求
数据副本数>=1

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。

游戏&教育行业

阿里云

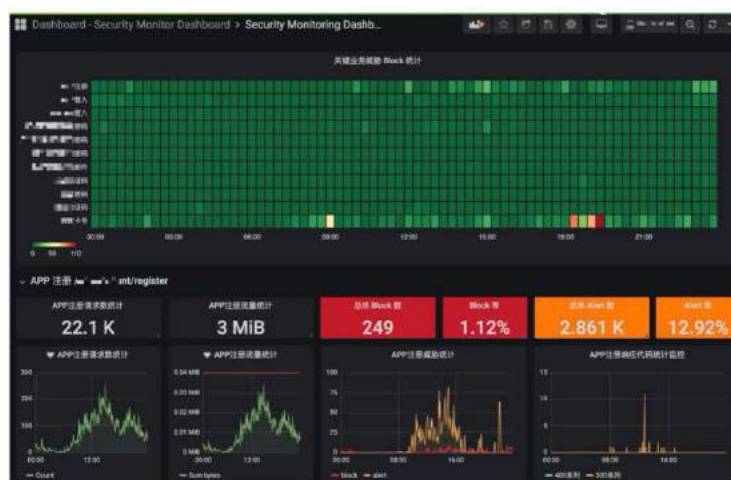
应用场景	场景痛点	产品能力
<ul style="list-style-type: none"> 1对1、1对多、1对N的在线直播：直播的过程中，会产生很多行为动作，如进入直播间、退出直播间、点赞、打赏、评论、对话等，如果动作无响应，对于用户体验就很差，需要基于日志的数据可视化，来帮助测试或研发监控、复现和定位问题； 全链路应用性能监控：终端用户、客户（三方机构 或者 内部工作室及业务团队）、平台方（平台运维部分），涉及的问题层层传递，需要保证性能监控的时效性； 业务数据、日志搜索：教育行业的教案教义试题搜索，游戏行业的日志点查、范围查询，互娱行业的业务运营数据统计分析查询等； 	<ul style="list-style-type: none"> 流量波动，集群弹性：周期性的流量波峰波谷（下班时间、周末、寒暑假），集群需要适应性的伸缩，直面供应链成本冗余或者不足的问题，以及频繁变更集群的运维成本问题高； 链路冗长，问题难定位：数据链路端到端流程太长，一旦异常定位问题成本很高； 高稳定性要求，成本高：日志系统不仅仅向终端客户暴露，日志系统可能也会面向内外部业务团队使用（如工作室、或三方机构），全链路的稳定保障要求极高； 搜索高要求：不仅仅是全文检索，基于标签的本文、甚至是非文本检索需求导致搜索复杂； 	<ul style="list-style-type: none"> 体系化产品能力：云上数据及服务高可用、数据存储加密和安全管控、MS级的数据时效性、全链路数据加快S级反馈； 数据时效性：MS级的数据时效性，全链路数据监控S级反馈； 多云灾备：多云灾备解决方案； TCO成本优化：通过场景调优、产品组合和价格策略，可以让客户TCO下降50%以上； 专家级服务：提供专家级支撑能力，输出基于客户场景的解决方案、架构优化、疑难问题解决；

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。

Use Case - 基于日志的业务数据监控

阿里云 | 奥运会全球指定云服务商



业务监控看板

业务数据的实时监控，如：
入侵监测，客流监控，
实时交易额检测等。

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。

Use Case - 基于APP日志的用户行为分析



用户行为分析

用户行为日志实时统计和监控分析。实时了解用户行为路径、产品体验度。

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。

电商&零售&金融



应用场	场景痛点	产品能力
<ul style="list-style-type: none">电商业搜索：电商场景存在海量的商品数据、订单数据，在售前和售后阶段，均对数据的精准搜索有需求，在售前阶段帮助用户快速找到意向商品和相关服务；在售后阶段帮助用户找到历史订单信息，帮助卖家根据手写退换单中的模糊信息快速找到并处理售后订单。综合数据分析：交易及零售行业在线上线下均有大量的数据产生，例如业务系统日志、交易数据、POS机数据、用户信息、用户在门店或线上的行为数据、智能设备数据等，需要对数据做多渠道收集、存储并分析。	<ul style="list-style-type: none">流量波动，集群弹性：电商零售行业周期性的流量波峰波谷(周末、大促)，集群需要适应性的伸缩，直面供应链成本冗余或者不足的问题，以及频繁变更集群的运维成本问题高；搜索质量要求：搜索作为电商零售场景核心流量入口，搜索准确率直接影响用户体验和成交转化，基础开源分词器无法满足高质量搜索需求；高稳定性要求，成本高：电商零售行业在流量高峰时，需要同时承载大量的查询和写入压力，对系统的可用性、稳定性保障要求极高；	<p>与自建相比：</p> <ul style="list-style-type: none">体系化产品能力：云上数据及服务高可用、集群一键升降配、数据存储加密和安全管控、MS级的数据时效性多云灾备：多云灾备解决方案；成本优化：通过场景调优、产品组合和价格策略，TCO下降50%以上； <p>场景匹配：</p> <ul style="list-style-type: none">专家级服务：提供专家级支撑能力，输出基于客户场景的解决方案、架构优化、疑难问题解决；全链路支持：云上ELK全产品支持，提供从数据采集、传输、处理、可视化的一站式服务；

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。

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。



分阶段构建可观测性



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。



健康检查



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。

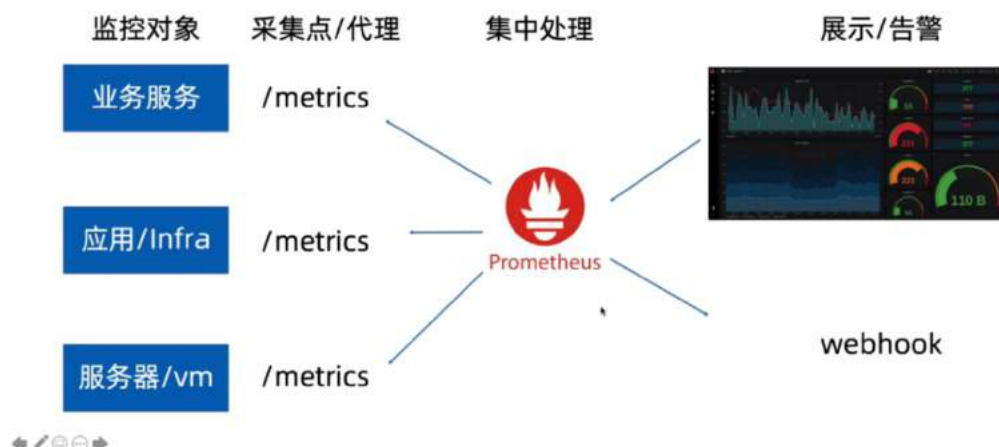


如何从指标获得观测性？有哪些方式？

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。

从指标获得可观测性



第三是日志。如何从日志获得可观测性？

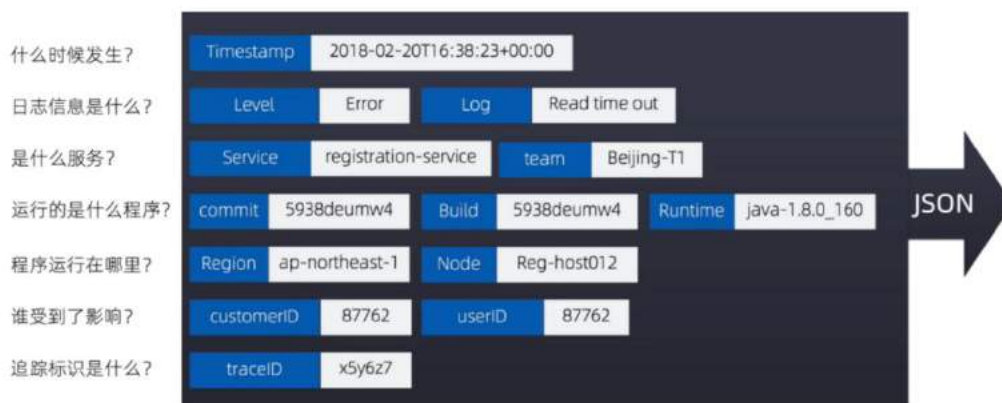
日志很多就是一串字符串，所以要从这里面获得可观测性，很重要的一步就是进行结构化的抽取工作。从下面的图中，我们可以看到各种各样的字段，比如 IP 地址、时间戳、进程号，以及具体请求的 URL。对于日志系统而言，就需要把文本的字符串变成结构化数据。

如何从日志获得可观测性



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。

如何从日志获得可观测性



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。

如何从APM中获得可观测性

- 服务间的调用堆栈
- 服务内部的调用序列
- 每一步花费的时间
- 代码相关信息
- 错误



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。

IT运维成熟度阶梯



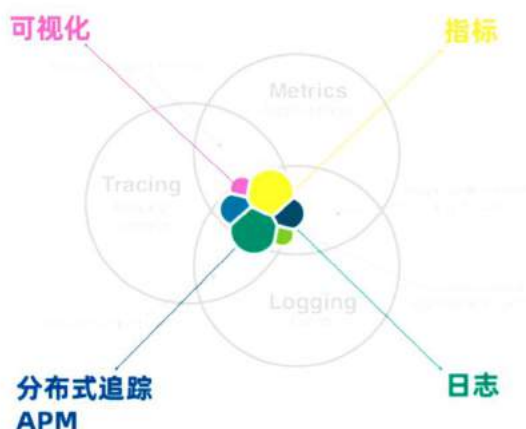
以上，我们谈了怎么可观测，以及如何去建设，下面谈谈全观测。

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。

传统运维的问题

- 数据孤岛，分散在不同部门，分析排查故障困难
- 多个厂商的多种工具，无法自动化统一分析
- 故障是立体的，日志 指标 APM都只能看到一方面的可观察性
- 只是收集，没有做到真正分析，不能发挥出大数据的价值



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。



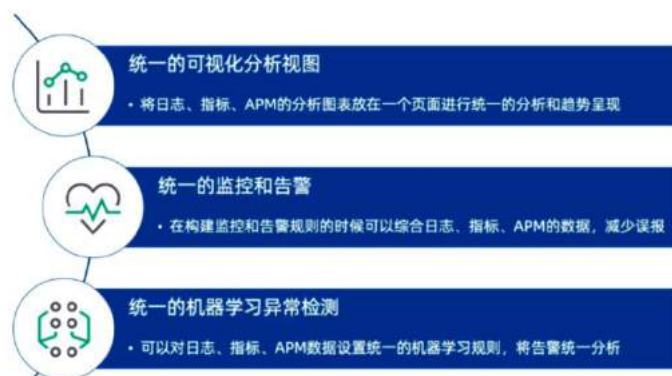
二、打通数据有哪些优势？

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。

打通数据的优势



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。

实现全观测的难点



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。



数据采集生态



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。



数据存储搜索工具生态



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，我们能用上边的各种工具对日志、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。

分享人：朱杰

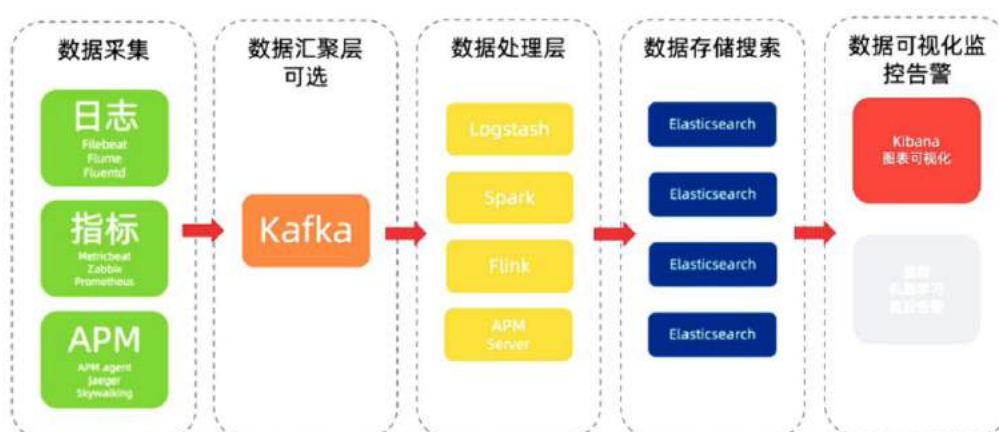
实现全观测主要有以下步骤：

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。

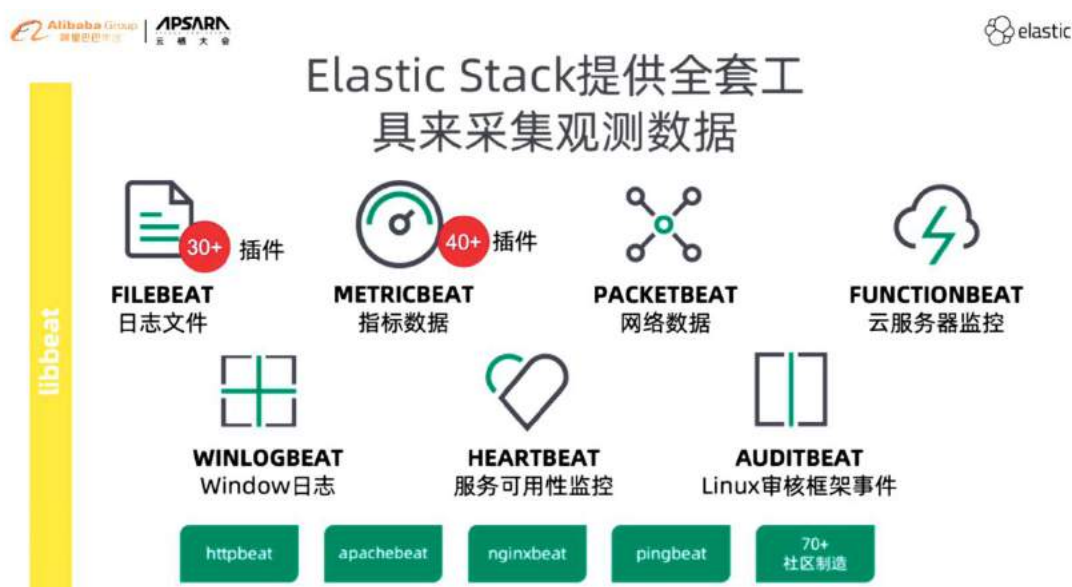


实现全观测的主要流程



一、Elastic Stack 提供的数据采集全套工具

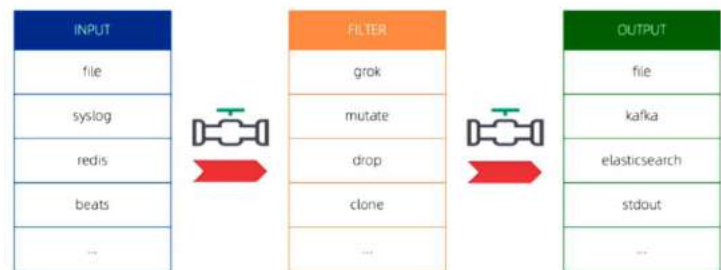
我们有丰富的 beat，比如采集日志文件的 Filebeat，有 30 多插件，能做到一定程度的开箱即用；Metricbeat 采集指标和底层的性能数据，有 40 多个插件；Packetbeat 从网络包层面采集数据；Functionbeat 主要对接在云端吐出来的指标日子；Winlogbeat 主要适配 Windows 上的日志系统；Heartbeat 主要检测服务的可用性，比如检测 API 是否在线等；最后，Auditbeat 可以连到 Linux 的 audit framework 来采集 Linux 各种各样的事件，汇总到 Elasticsearch。除此之外，我们的社区也制造了很多 beat，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

数据处理工具



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。

数据存储搜索

倒排序支持的全文搜索

列存储支持的结构化数据

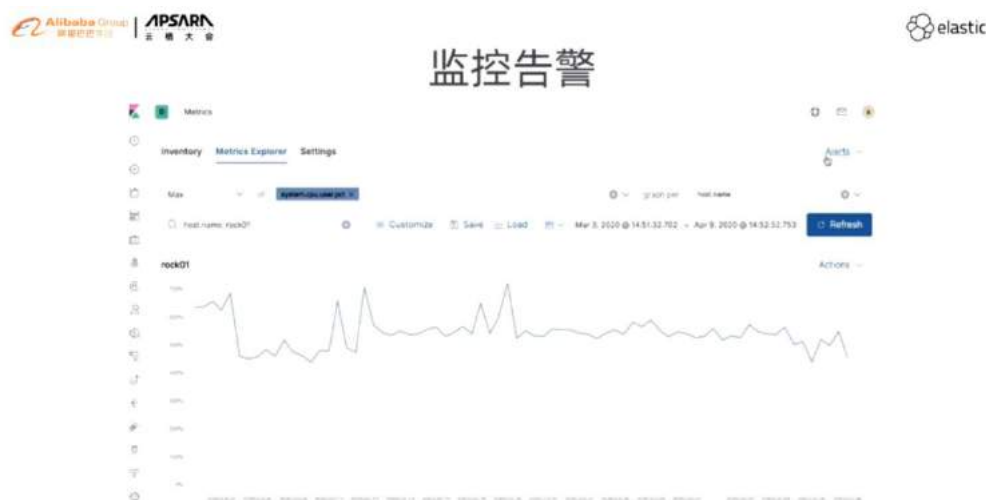
userid	first	middle	last	city	state
john123	John	James	Smith	Alamo	California
jrice	Jill	Any	Rice		
mt123	Jeff		Twain	Toledo	Ohio
sacans	Sue		Adams		
adoc	Any		Doe	Miami	Florida

BKD数支持的数值型运算

Rollup 节省存储空间

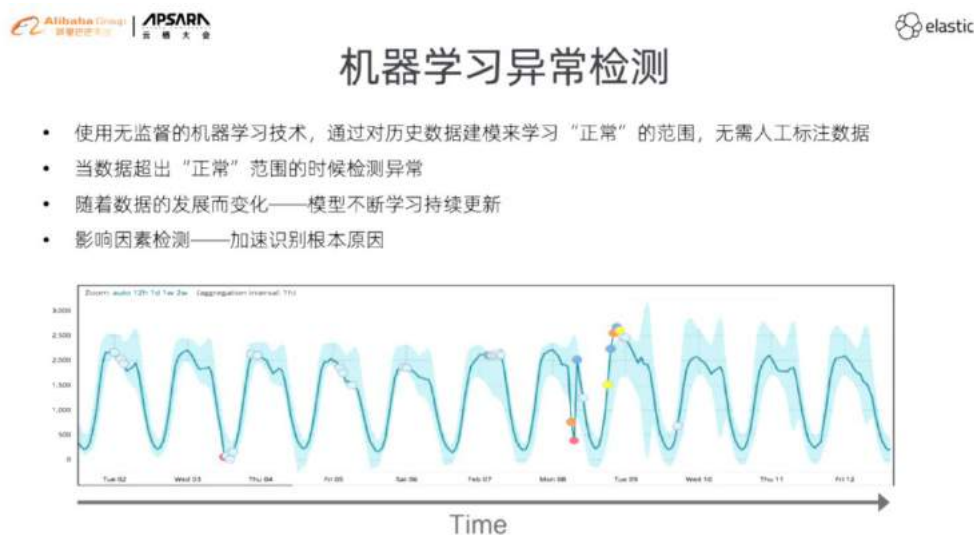
四、告警系统

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。

全观测实例

- 将日志、指标、APM 数据在一个平台统一分析
- 可以建立统一的可视化视图、对齐时间、统一过滤条件
- 建立统一的基于规则的监控和告警，关联多个数据源
- 建立统一的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。

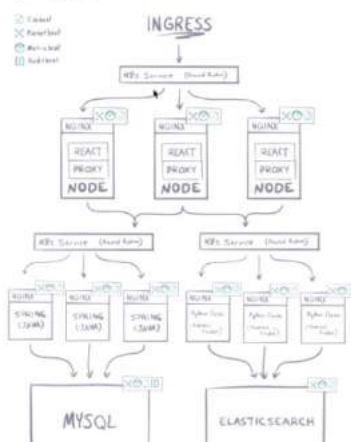
那么我们如何用上面的工具对这一架构进行监测呢？

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。

全观测实例

- 典型的微服务架构
- 全部在K8s容器化环境中部署
- 每种服务都有多个实例
- 前端：基于Nodejs提供Web服务
- 核心业务：基于Spring框架的Java服务，连接后端MySQL数据库
- API服务：基于Python Flask提供地址查询的Rest API服务，连接Elasticsearch服务器
- Filebeat采集日志
- Metricbeat采集指标
- Packetbeat采集网络包数据
- APM agent采集应用性能信息
- 所有数据汇总到Elasticsearch集群
- 使用机器学习进行监控和告警



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。

全观测定位故障



下面我们更具体地来看定位故障的每一步流程。

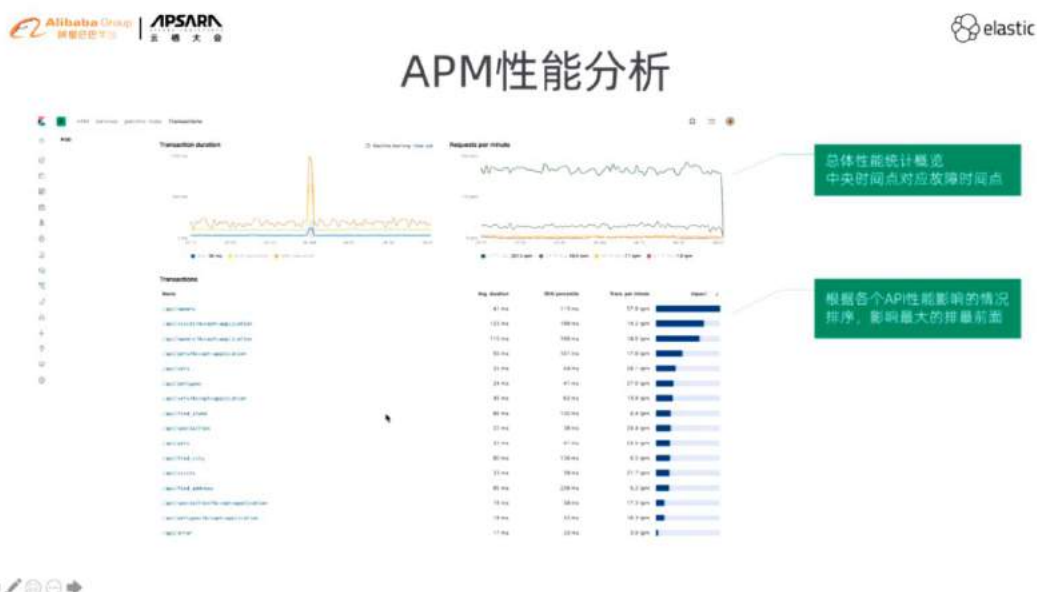
八、机器学习告警

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。

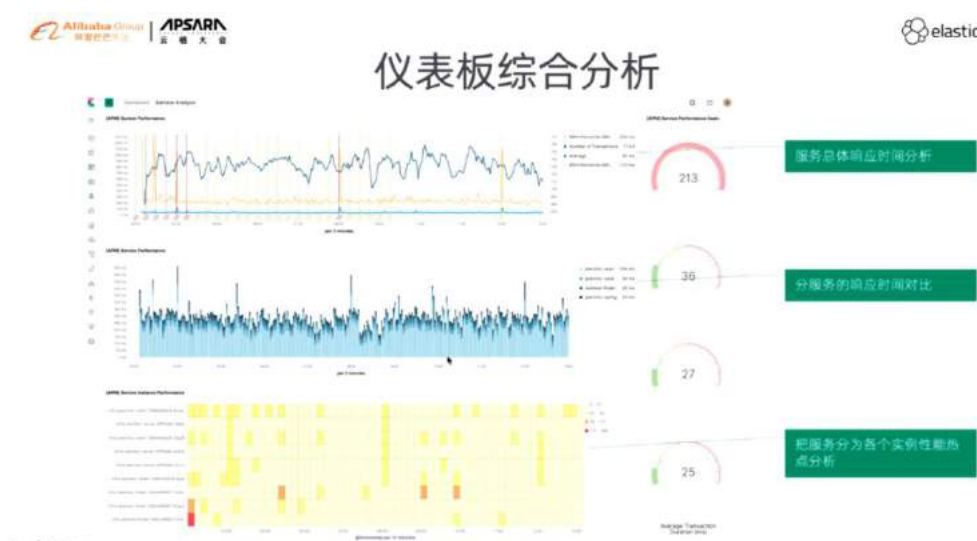


当我们点进 API 后, 还能看到它在分布式环境下如何调用堆栈。比如下图中蓝色就代表 Nodejs, 绿色代表 Java 代码的调用。可以看到, 每一个服务不仅存在多个实例, 而且是分布在不同的服务器上, 因此这个工具可以把服务流经的所有处理环节串联到一起, 实现分布式追踪。



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。

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。



案例：日志数据

Log data

```
2020-08-31T09:59:11.504+0000 [15892] [gc_heap] Heap region size: 1M
2020-08-31T09:59:11.504+0000 [15892] [gc_heap,comp] Heap address: 0x00000000.00000000, size: 1024 MB, Compressed Dups mode: 12-bit
2020-08-31T09:59:15.118+0000 [15892] [gc] Using G1
2020-08-31T09:59:15.119+0000 [15892] [gc,cds] Mark closed archive regions in map: [0x00000000ffff00000, 0x00000000ffff0000]
2020-08-31T09:59:15.119+0000 [15892] [gc,cds] Mark open archive regions in map: [0x00000000ffff00000, 0x00000000ffff0000]
2020-08-31T09:59:15.129+0000 [15892] [gc] Periodic GC disabled
2020-08-31T09:59:15.840+0000 [15892] [safepoint] Safepoint "ICBufferFull", Time since last: 713127465 ns, Reaching safepoint: 2M
2020-08-31T09:59:16.205+0000 [15892] [gc_start] GC(0) Pause Young (Normal) (G1 Evacuation Pause)
2020-08-31T09:59:16.211+0000 [15892] [gc_tech] GC(0) Using 4 workers of 4 for evacuation
2020-08-31T09:59:16.211+0000 [15892] [gc_age] GC(0) Desired survivor size 3670016 bytes, new threshold 15 (max threshold 15)
2020-08-31T09:59:16.215+0000 [15892] [gc_age] GC(0) Age table with threshold 15 (max threshold 15)
2020-08-31T09:59:16.215+0000 [15892] [gc_age] GC(0) - age 1: 4004496 bytes, 4004496 total
2020-08-31T09:59:16.215+0000 [15892] [gc_phases] GC(0) Pre Evacuate Collection Set: 0.1ms
2020-08-31T09:59:16.215+0000 [15892] [gc_phases] GC(0) Merge Heap Roots: 0.8ms
2020-08-31T09:59:16.215+0000 [15892] [gc_phases] GC(0) Evacuate Collection Set: 3.7ms
2020-08-31T09:59:16.215+0000 [15892] [gc_phases] GC(0) Post Evacuate Collection Set: 0.5ms
2020-08-31T09:59:16.215+0000 [15892] [gc_heap] GC(0) Other: 5.3ms
2020-08-31T09:59:16.215+0000 [15892] [gc_heap] GC(0) Eden regions: 51->4(51)
2020-08-31T09:59:16.215+0000 [15892] [gc_heap] GC(0) Survivor regions: 0->4(7)
2020-08-31T09:59:16.215+0000 [15892] [gc_heap] GC(0) Old regions: 0->0
2020-08-31T09:59:16.215+0000 [15892] [gc_heap] GC(0) Archive regions: 2->2
2020-08-31T09:59:16.215+0000 [15892] [gc_heap] GC(0) Humongous regions: 1->1
2020-08-31T09:59:16.215+0000 [15892] [gc_metaspaces] GC(0) Metaspaces: 10402K(11264K)->10402K(11264K) NewClass: 9097K(9728K)->9097K(9728K)
2020-08-31T09:59:16.215+0000 [15892] [gc] GC(0) Pause Young (Normal) (G1 Evacuation Pause) 520->100(10240) 9.698ms
2020-08-31T09:59:16.215+0000 [15892] [gc_tech] GC(0) User-defined Sys-0.0ms Real-0.0ms
2020-08-31T09:59:16.215+0000 [15892] [safepoint] Safepoint "G1CollectForAllocation", Time since last: 365423312 ns, Reaching safepoint: 2M
2020-08-31T09:59:17.069+0000 [15892] [gc_start] GC(1) Pause Young (Normal) (G1 Evacuation Pause)
2020-08-31T09:59:17.069+0000 [15892] [gc_tech] GC(1) Using 4 workers of 4 for evacuation
2020-08-31T09:59:17.069+0000 [15892] [gc_age] GC(1) Desired survivor size 3670016 bytes, new threshold 1 (max threshold 15)
2020-08-31T09:59:17.069+0000 [15892] [gc_age] GC(1) Age table with threshold 1 (max threshold 15)
2020-08-31T09:59:17.069+0000 [15892] [gc_age] GC(1) - age 1: 3687488 bytes, 3687488 total
2020-08-31T09:59:17.069+0000 [15892] [gc_phases] GC(1) Pre Evacuate Collection Set: 4.5ms
2020-08-31T09:59:17.069+0000 [15892] [gc_phases] GC(1) Merge Heap Roots: 0.1ms
2020-08-31T09:59:17.069+0000 [15892] [gc_phases] GC(1) Evacuate Collection Set: 0.8ms
```

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。



案例：指标数据

Metric data



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。

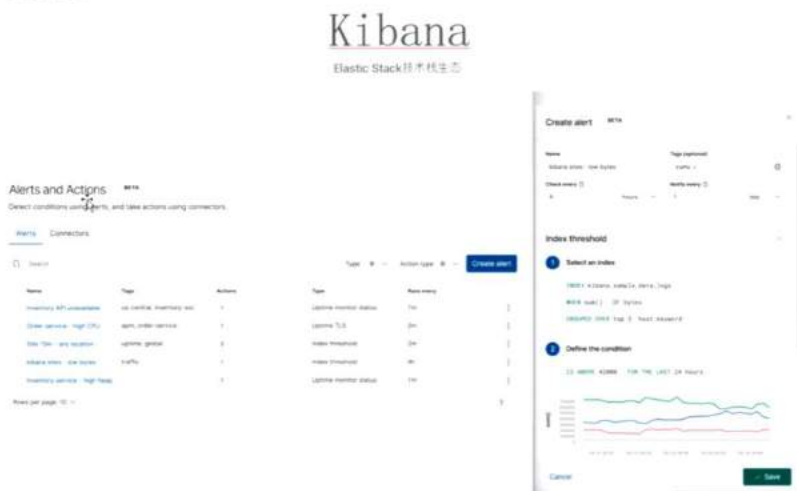


Kibana

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。



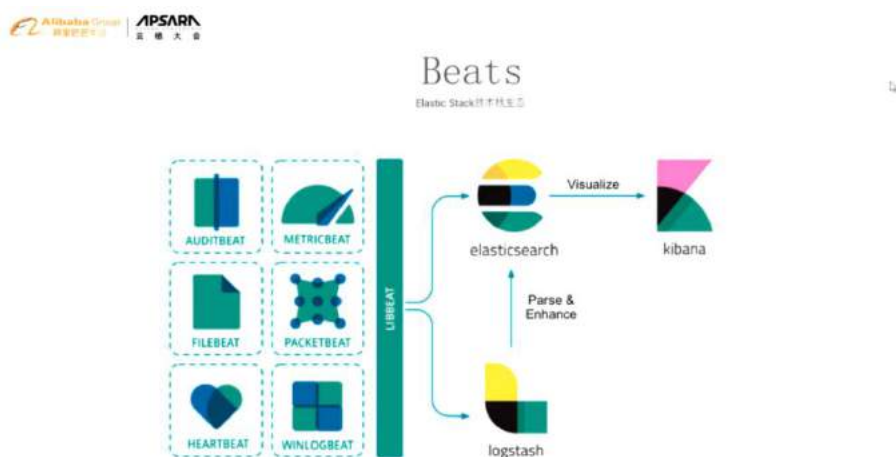
Elasticsearch

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

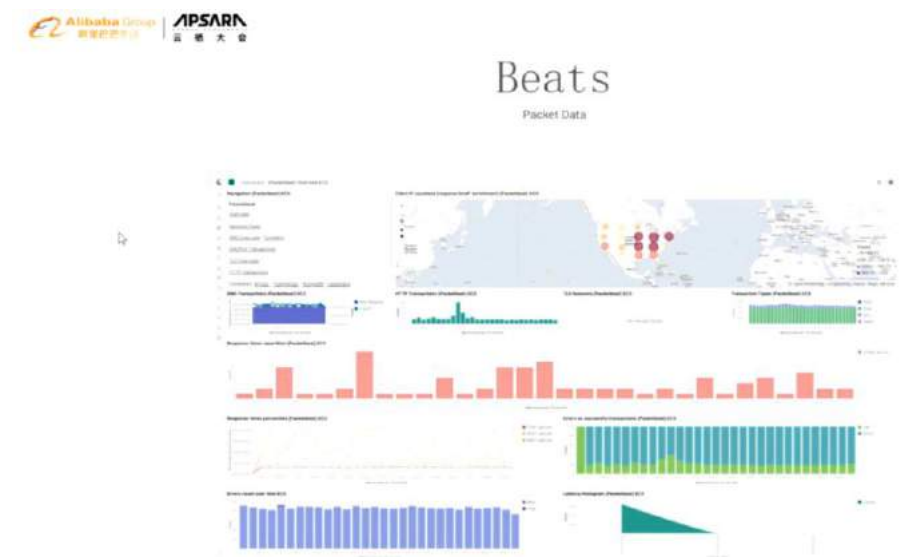
Beats 主要用于数据采集，非常轻量级，包括 Auditbeat、Metricbeat、Filebeat、Packetbeat、Heartbeat 和 Winlogbeat 几种。为了和 Logstash 做职责区分，Beats 就主要做日志数据采集，但如果遇到一些数据需要做自定义转换，就需要把采集完的数据先传到 Logstash 里进行一些处理再写到 ES 里。



Filebeat

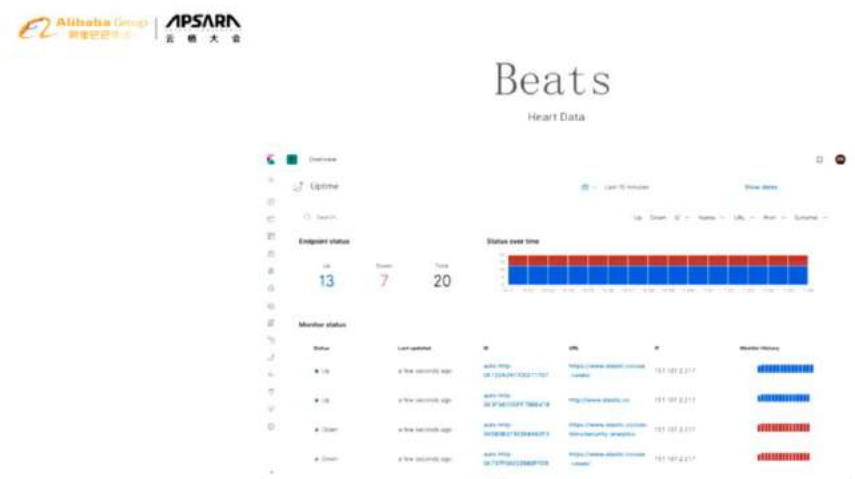
Filebeat 专门采集文本日志，比如服务器和应用上的文本 log。另外，Filebeat 已经写好了规则，非常方便。比如如果要抓取 ES 的日志，它里面直接有一个 ES 的模块，日志格式都帮忙解析完成了，只需要配一个日志路径就能完成采集。所以 Beats 家族更加“傻瓜化”，把 Logstash 的可编程性都替代掉了，直接做成最终落地。

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。

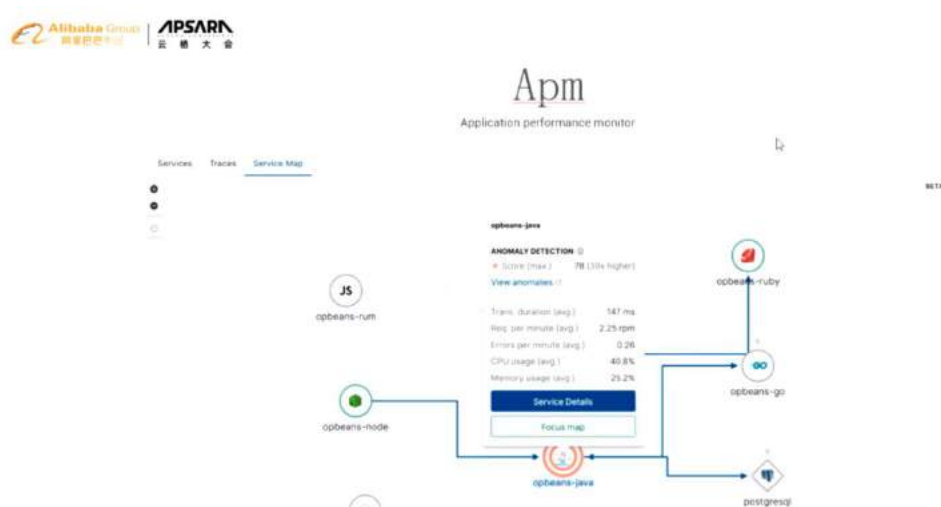


APM

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。

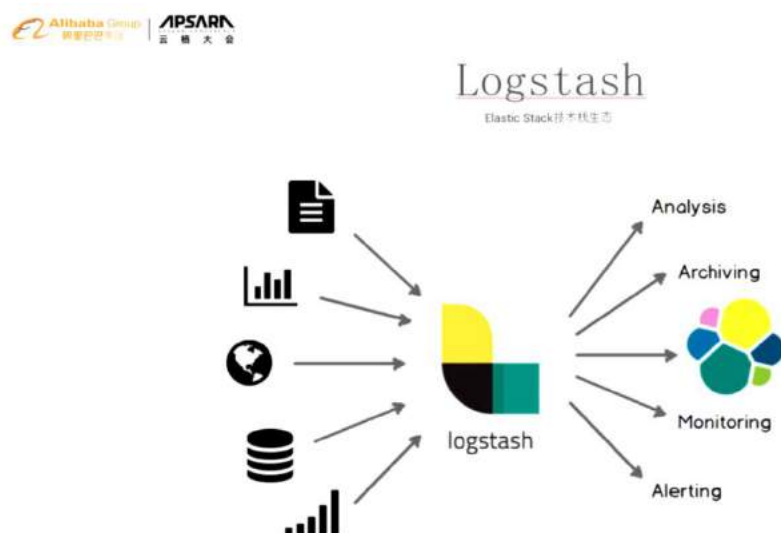


除了上面的应用程序，它服务之间也有绘制相应的图。比如下面你从 Java 调到 Go 再调到 Ruby，它都可以帮你自动来绘制出来，仅仅只需要把你的应用程序部署起来就可以了。



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。

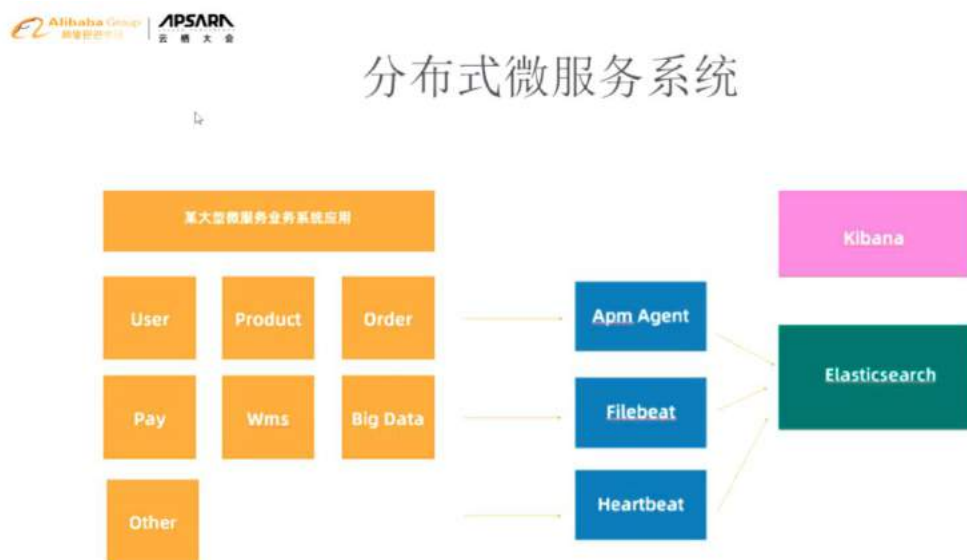
三、全观测性行业应用场景

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。



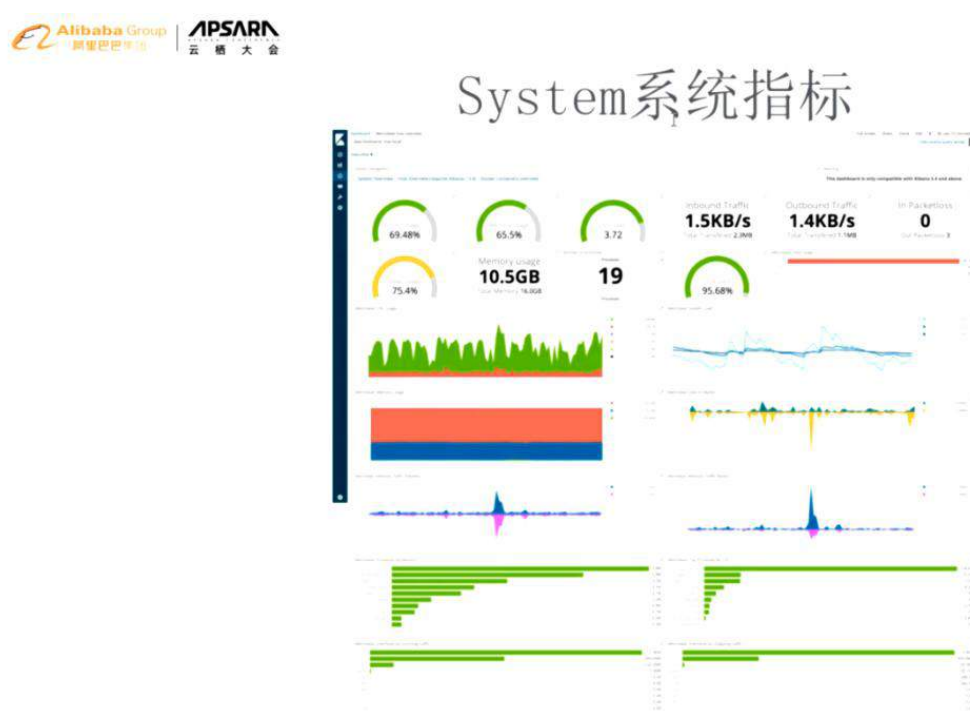
再一个就是 Kafka。我们前面讲到微服务调用，几百个微服务有那么多数据采集，数据量是非常大的，所以会引入 Kafka 承担信息缓冲，它非常重要，所以我们会对它进行完整的链路监控。



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，而这就需要有一个更加整体的技术平台。

阿里云 ES 全观测性配置

摘要：本文是借助阿里云环境进行的 ES 实战演练，主要演示了 APM 的配置和使用过程，介绍了相关界面的使用和注意事项，引入大家系统地使用 ES。

分享人：李猛

今天我们借助阿里云环境来做一个实战的演练，围绕这三点来讲解：阿里云 ES 介绍、搭建全观测性环境、全观测性总结。

一、阿里云 ES 介绍

首先介绍一下阿里云。阿里云目前在用户量与规模属于世界前三，基本上公司有条件的都会上云，而阿里云是最可靠的。阿里云在它平台上集成了很多开源的产品，使用起来非常方便，省去了搭建、服务器维护等工作。

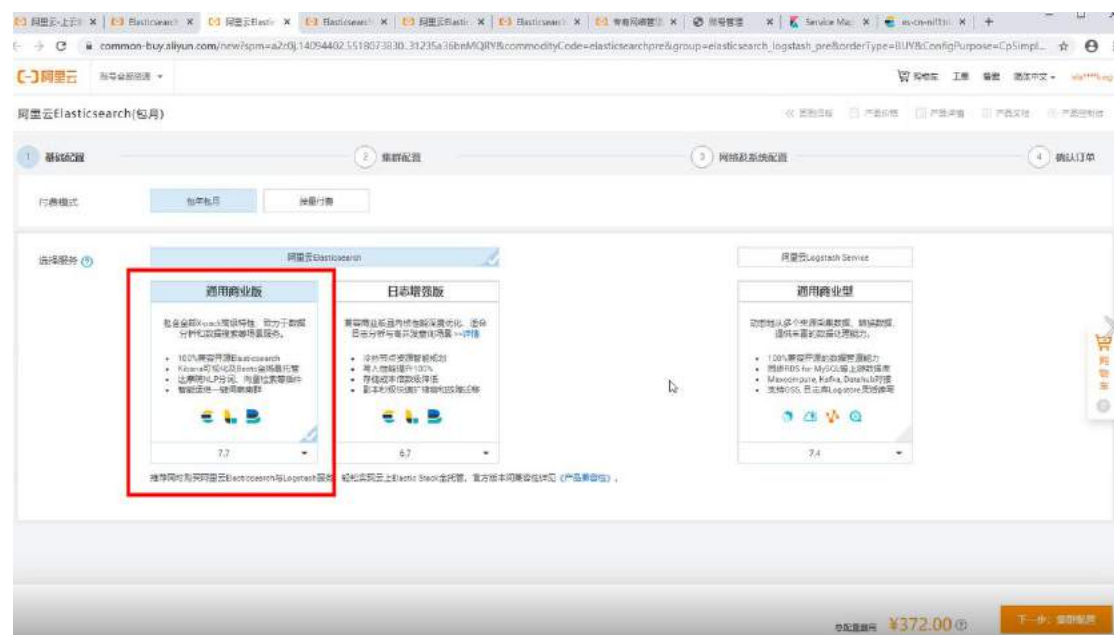
我们打开阿里云地址，在产品的大数据这个地方可以找到 Elasticsearch。



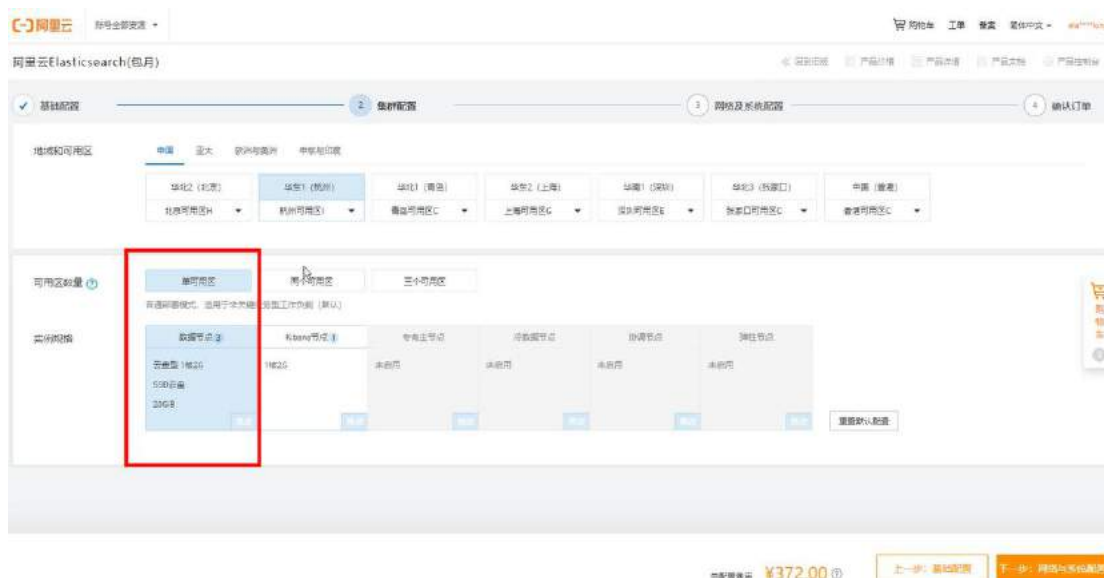
然后点进去，进入 ES 界面。现在 ES 这套产品在阿里云有一个月的试用期，搭建集群非常快，基本上几分钟就完成了。在这个里面可以看到有很多 ES 的资料和开发报告等内容。



我们点进“0 元开通 ELK”，填写一些关键的信息就可以开通了，它会引导你一步步地去配置一个 ES 环境。



第一步，在这个区域上配有三个节点，我们按照提示一步一步下去配置，非常方便。



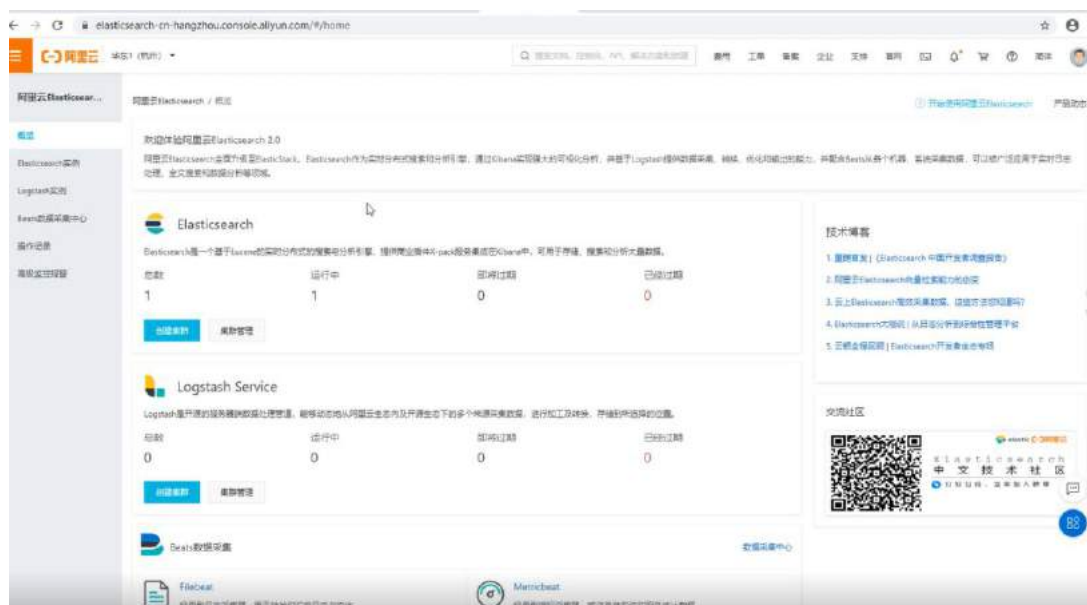
再继续下去要配虚拟网络和ES的登录密码,因为我们自己搭建ES默认是没有启动安全的。但是在云上是一定要启动,因为云上是一个公共的环境,是一个开放云,你最终要把这个地址暴露出来。

密码设置符合它的规则就好了,后面访问ES和Kibana会用到。另外选专有网络这里要注意,必须要选择同一个区域,假设你前面选择的是华东杭州,那么选择专有网络虚拟交换机也要选择同一个区域才有效。



下一步我们确认订单,支付购买就完成了,然后我们来进行搭建。

首先在整个 ES 里面，我们要搭建两个，就是 Elasticsearch 和 Kibana。



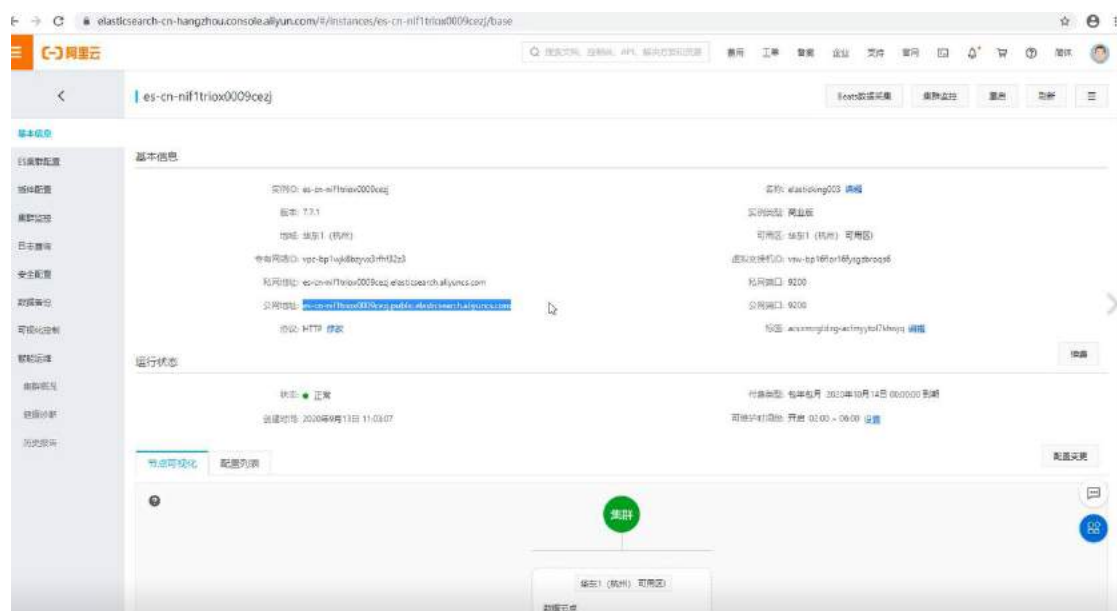
现在显示这已经有一个集群在运行，我们可以点执行管理，进去后看到这个集群的版本、实例类型、节点等等。



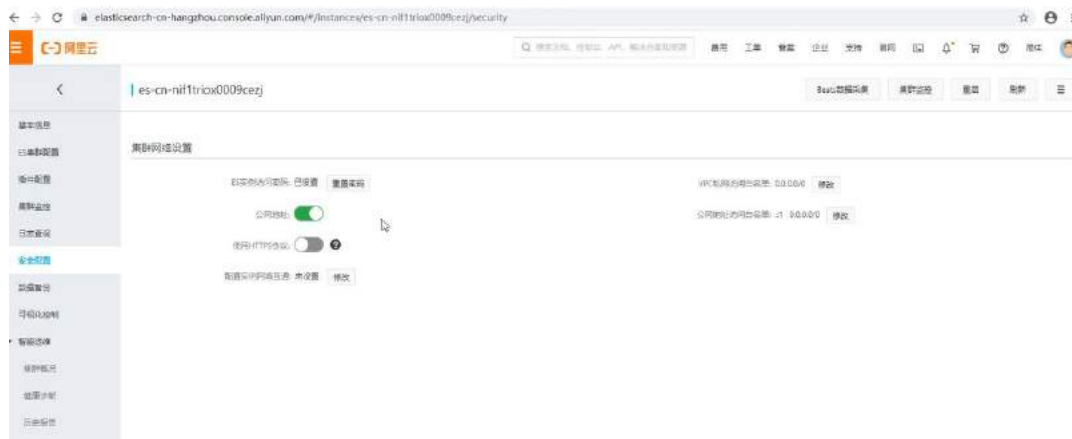
我们从实例名点进去，可以看到一些基本的信息，ES 的一些配置信息等，这些在默认的情况下都不用去修改。



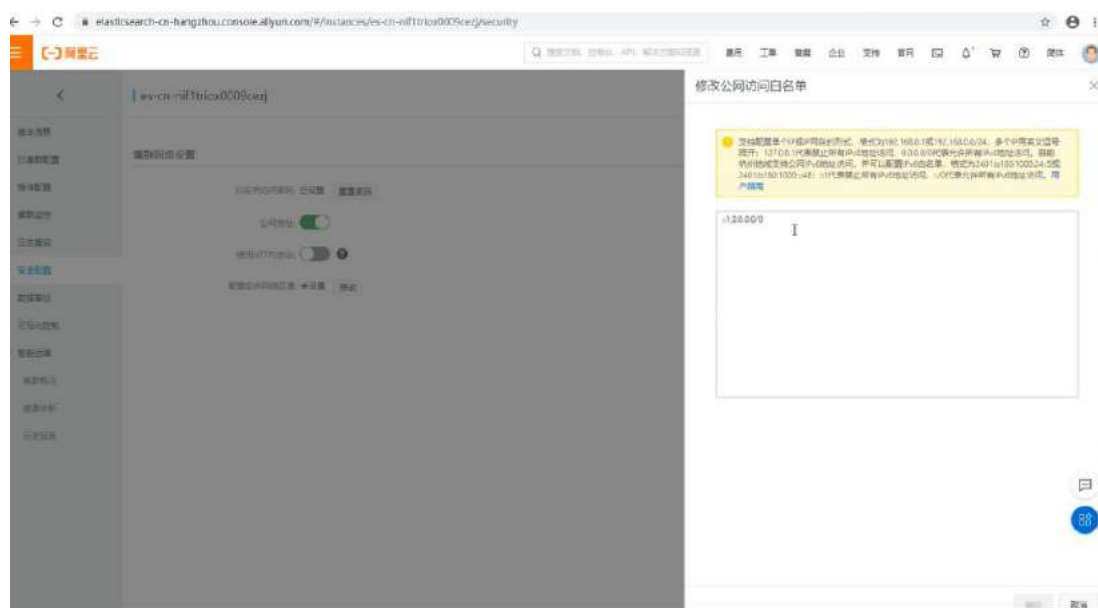
我们这个集群是搭建好的，搭建完成之后有几个地方要配置。第一个就是如果要公网访问，一定要去开通安全，它会生成这样一个临时的域名，我们可以通过它来访问 9200 端口和检查 Nodes 节点。



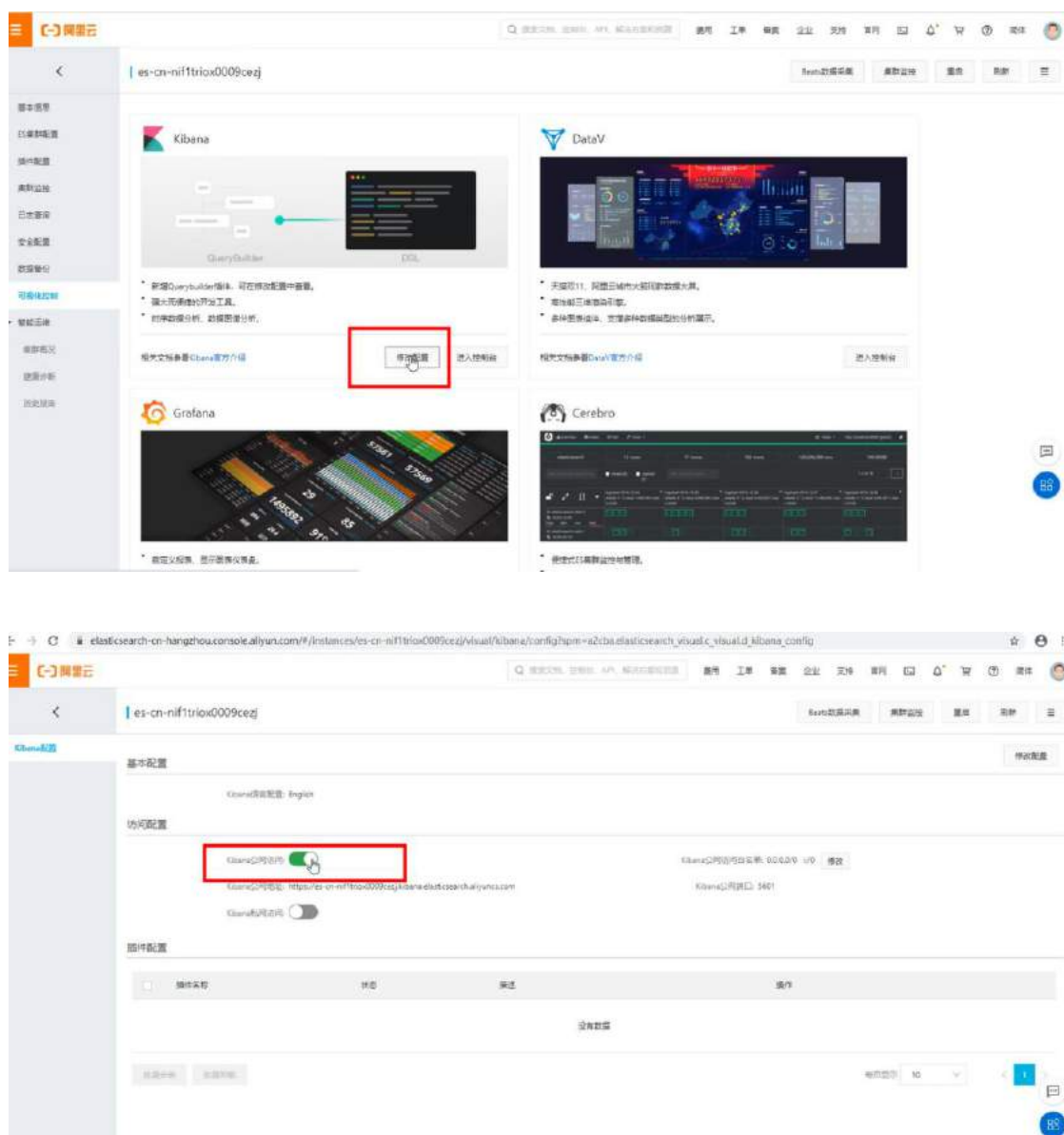
这里公网配置一定要选择打开，如果这个地方的公网地址没有启用，外网是不可访问的。



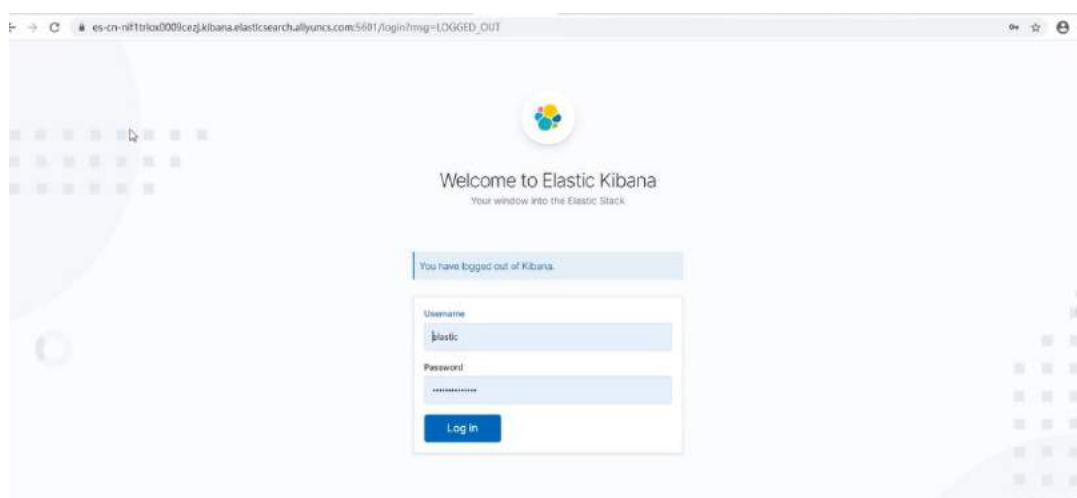
还有，公网打开之后还要修改公网访问白名单地址，根据上面的提示来写，00000 代表所有访问的 IPV4 地址不限制，如果你不设置，它默认是禁止的。



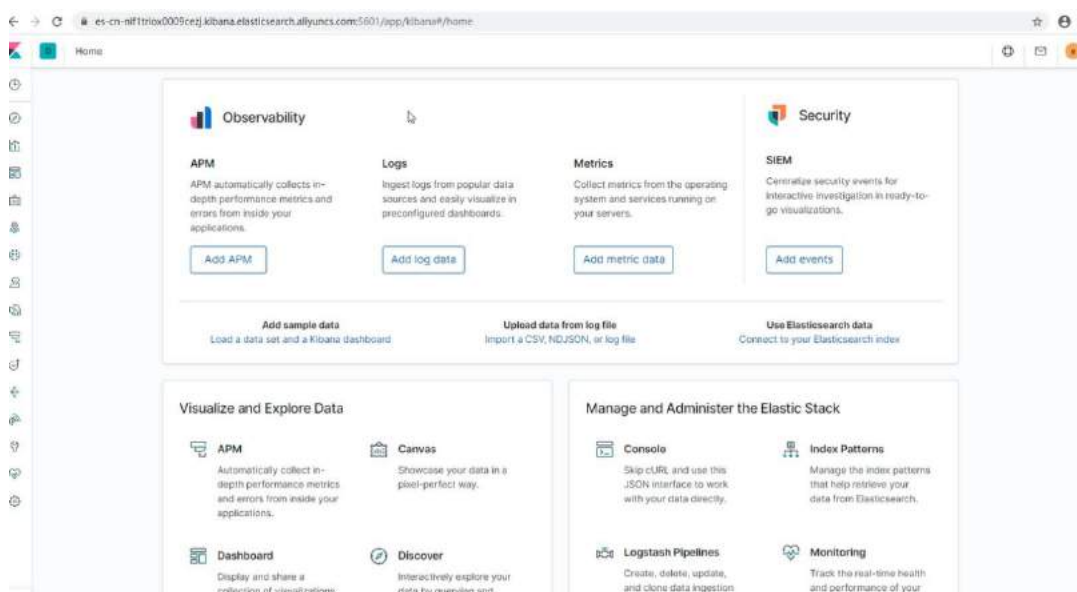
接下来是第二个配置，可视化。我们点“可视化控制”进入界面，进入后在“修改配置”里面修改公网的地址，选择“公网开放”，如果不修改它也是不能访问的。



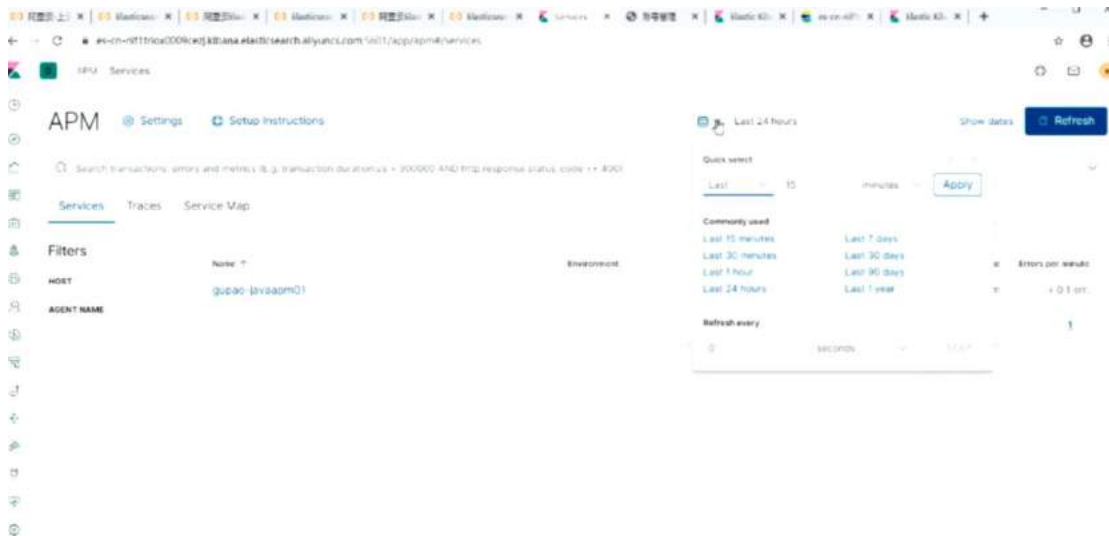
随后，我们回到上一级页面，点“修改配置”旁边的“进入控制台”，输入刚才我们设置的ES密码就可以进入Kibana的页面了。



这个界面它有一个 home，介绍到它的主要产品概念，比如第一个可观测性 observability，可以看到 APM、Logs、Metrics。我们也可以去加载一些样例数据，ES 它专门集成在里面，方便我们去做测试。

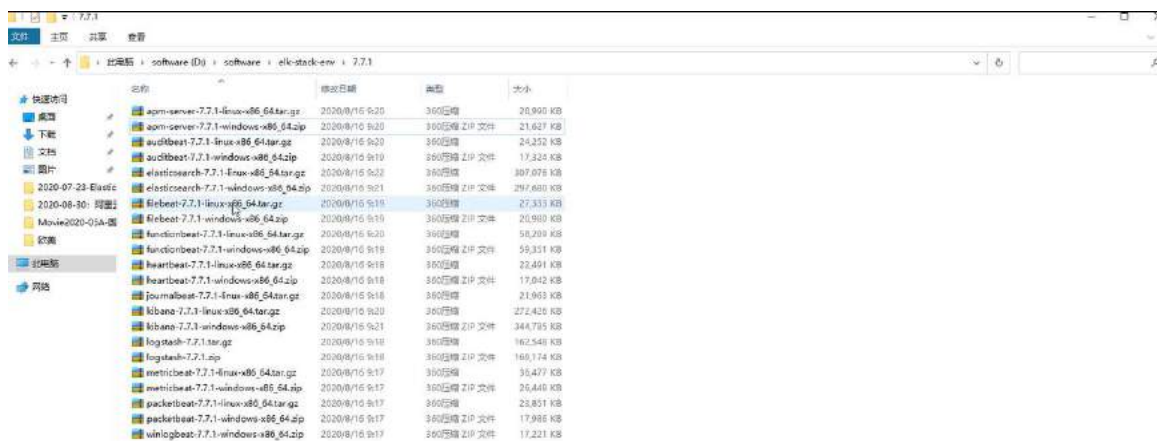


环境搭好之后，我们要做一套自己的监控。第一步是搭建 Kibana，我们已经完成了，随后要做一个应用性能采集的东西，叫 APM。我们可以在左边的菜单栏找到“APM”点进去，这个里面会记录，比如有多少个微服务，一两百个都可以采过来，然后绘制图表展示信息。



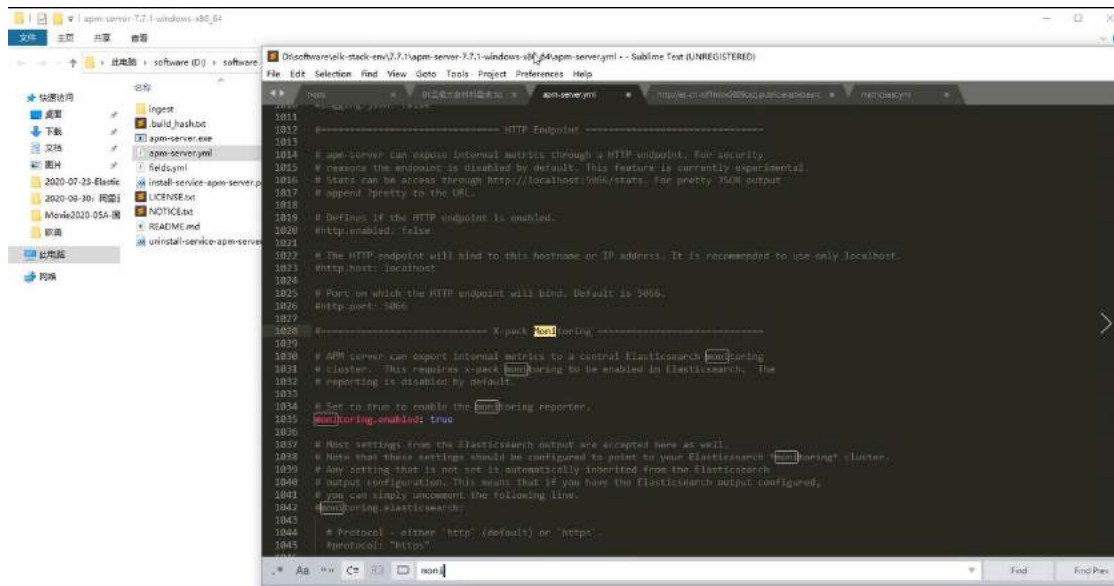
APM 要怎么应用呢？虽然阿里云上面提供了 ES、Logstash、Beats 数据采集，但是 APM 并没有给我们。假如我们的应用程序部署在别的服务器上，我们如何去进行对接？

为了演示这个案例，我们没有在阿里云申请别的服务器，就用本地机器来代替。首先，我们要部署一个 APM server，它的版本最好和服务器的 ES 版本一样。

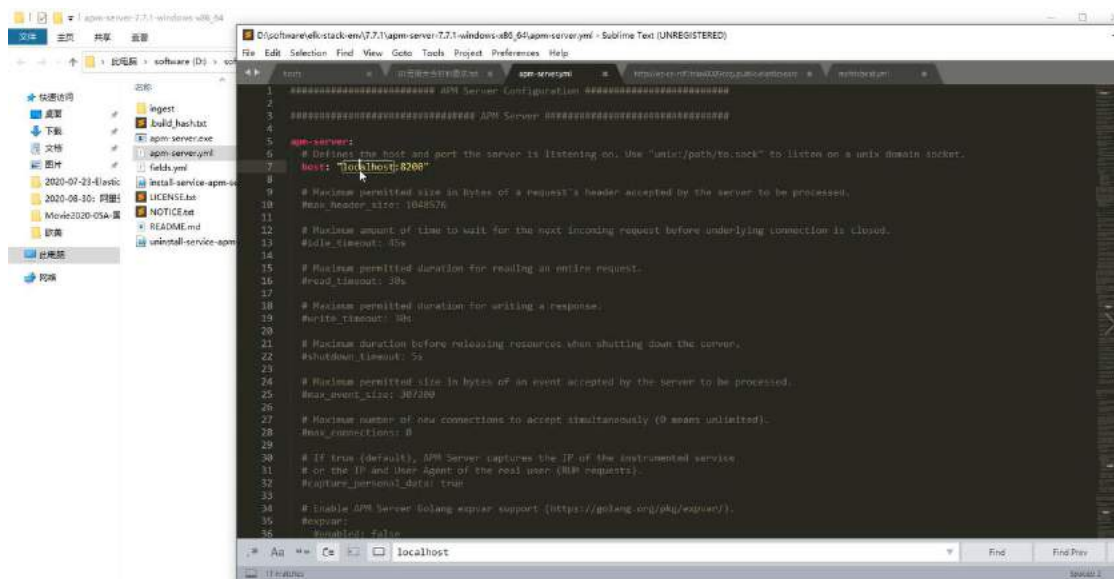


APM server 有几个地方要先进行配置，我们用文本打开 APM 文件。首先，因为数据采集之后要传到 Logstash 上面，所以这里有一个 localhost 的地址，它代表 APM server。

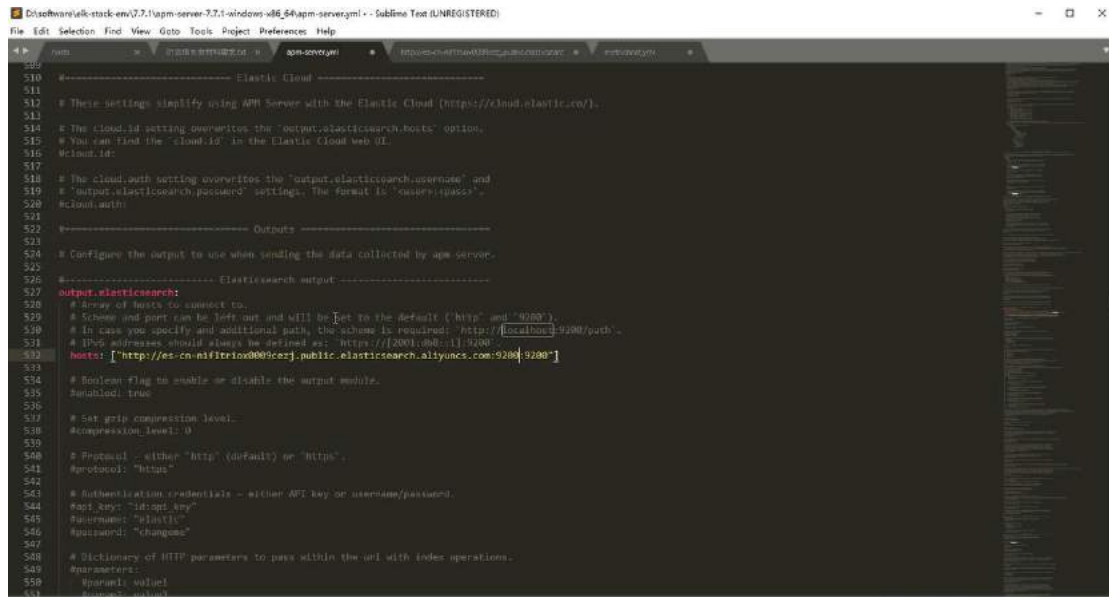
Java App agent 数据采集之后首先传给 APM server，然后 APM server 再把数据给到远程。这里因为我们本来用的本地服务器环境，所以可以默认用本地地址。



第二个就是数据最终 output 的地方，我们找到 output Elasticsearch 的 localhost，这是数据采集要指向的 ES 远程地址，我们刚刚已经说过远程的 ES 地址，把它复制过来。



大家注意原来的 9200 端口，然后协议 http 删掉。

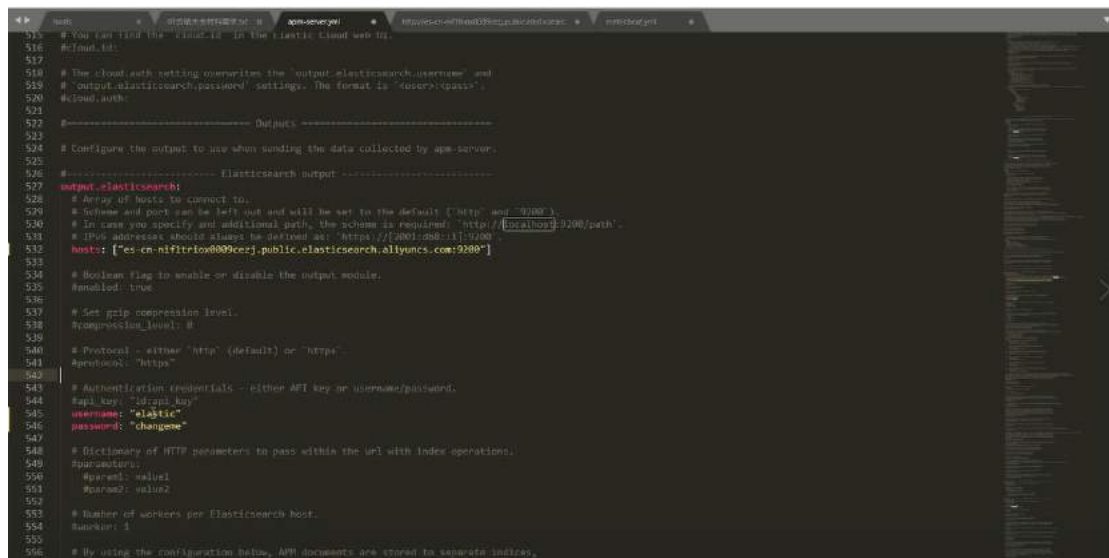


```

510 #----- Elastic Cloud -----
511 # These settings simplify using APM Server with the Elastic Cloud (https://cloud.elastic.co/).
512 # The cloud.id setting overwrites the 'output.elasticsearch.hosts' option.
513 # You can find the 'cloud.id' in the Elastic Cloud web UI.
514 #cloud.id:
515 # The cloud.auth setting overwrites the 'output.elasticsearch.username' and
516 # 'output.elasticsearch.password' settings. The format is 'user:pass'.
517 #cloud.auth:
518 #----- Outputs -----
519 # Configure the output to use when sending the data collected by apm-server.
520 #----- Elasticsearch output -----
521 output.elasticsearch:
522   # Array of hosts to connect to.
523   # Scheme and port can be left out and will be set to the default ('http' and '9200').
524   # In case you specify an additional path, the scheme is required: 'http://localhost:9200/path'.
525   # IPv6 addresses should always be defined as: 'https://[2001:db8::1]:9200'.
526   hosts: ["http://es-cn-niftrto0009cerj.public.elasticsearch.aliyuncs.com:9200:9200"]
527   # Boolean flag to enable or disable the output module.
528   enabled: true
529   # Set gzip compression level.
530   #compression_level: 0
531   # Protocol - either 'http' (default) or 'https'.
532   #protocol: "https"
533   # Authentication credentials - either API key or username/password.
534   #api_key: "id:api_key"
535   #username: "elastic"
536   #password: "changeme"
537   # Dictionary of HTTP parameters to pass within the url with index operations.
538   #parameters:
539     #param1: value1
540     #param2: value2
541   # Number of workers per Elasticsearch host.
542   #workers: 1
543   # By using the configuration below, APM documents are stored in separate indices,

```

因为是远程访问，所以还需要密码，这里把之前的用户名和密码都配置到这里。

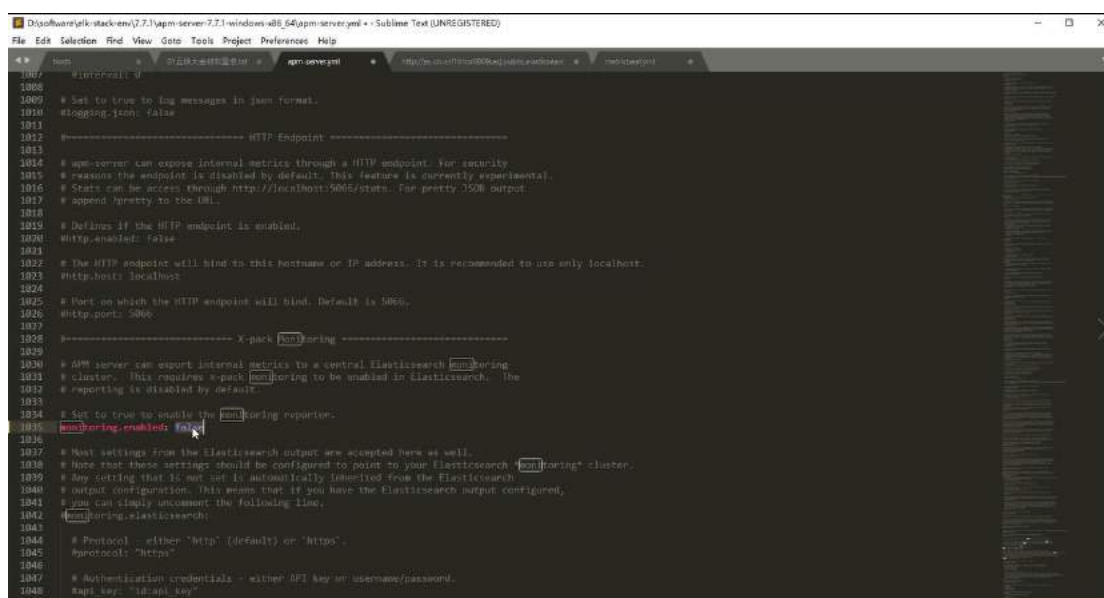


```

512 # You can find the 'cloud.id' in the Elastic Cloud web UI.
513 #cloud.id:
514 # The cloud.auth setting overwrites the 'output.elasticsearch.username' and
515 # 'output.elasticsearch.password' settings. The format is 'user:pass'.
516 #cloud.auth:
517 #----- Outputs -----
518 # Configure the output to use when sending the data collected by apm-server.
519 #----- Elasticsearch output -----
520 output.elasticsearch:
521   # Array of hosts to connect to.
522   # Scheme and port can be left out and will be set to the default ('http' and '9200').
523   # In case you specify an additional path, the scheme is required: 'http://localhost:9200/path'.
524   # IPv6 addresses should always be defined as: 'https://[2001:db8::1]:9200'.
525   hosts: ["es-cn-niftrto0009cerj.public.elasticsearch.aliyuncs.com:9200"]
526   # Boolean flag to enable or disable the output module.
527   enabled: true
528   # Set gzip compression level.
529   #compression_level: 0
530   # Protocol - either 'http' (default) or 'https'.
531   #protocol: "https"
532   # Authentication credentials - either API key or username/password.
533   #api_key: "id:api_key"
534   #username: "elastic"
535   #password: "changeme"
536   # Dictionary of HTTP parameters to pass within the url with index operations.
537   #parameters:
538     #param1: value1
539     #param2: value2
540   # Number of workers per Elasticsearch host.
541   #workers: 1
542   # By using the configuration below, APM documents are stored in separate indices,

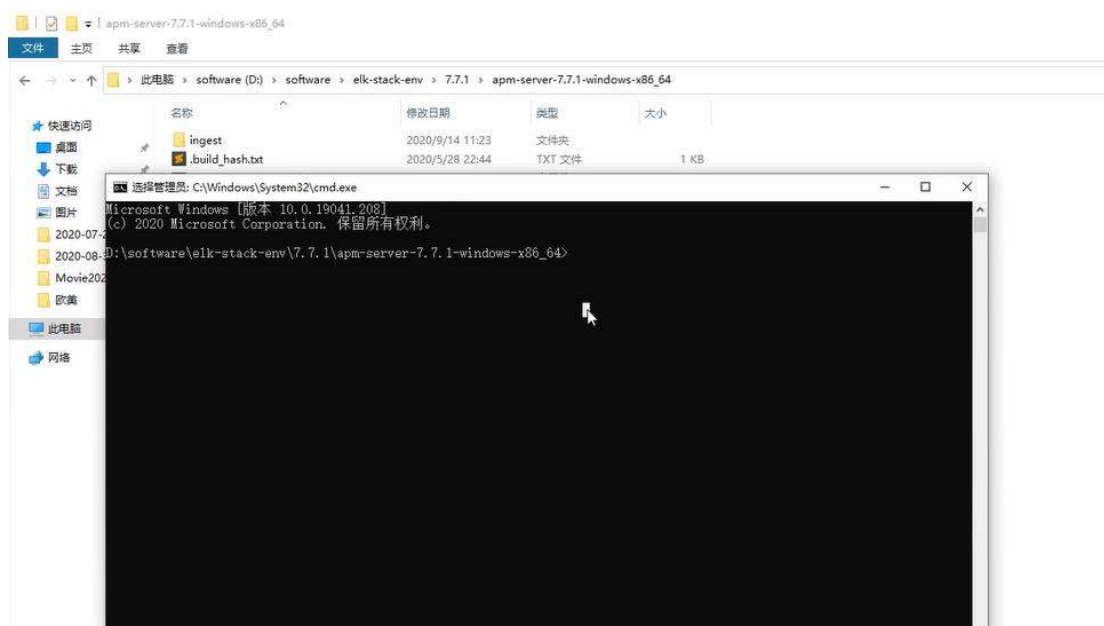
```

然后我们找到 monitor，这是监控 ES 自己的，它默认是不启用的，所以我们把它选择为 true。做完之后，APM 这一套我们就算配置完成了，然后我们就可以启动它。

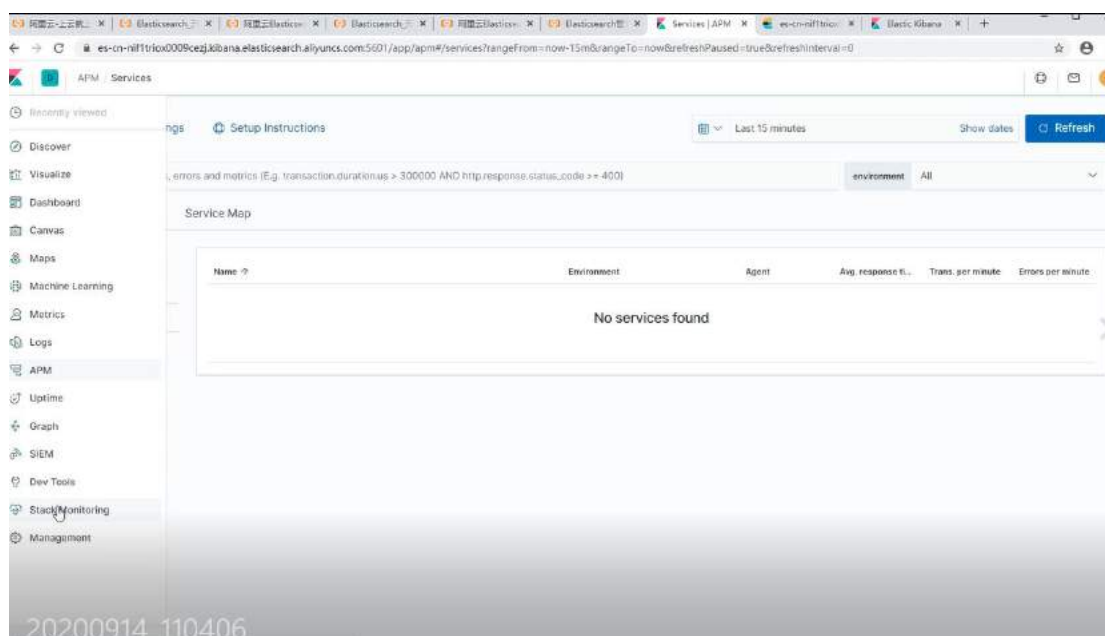


```
1007 #interval: 10s
1008
1009 # Set to true to log messages in json format.
1010 logging.json: false
1011
1012 #----- HTTP Endpoint -----
1013
1014 # apm-server can expose internal metrics through a HTTP endpoint. For security
1015 # reasons the endpoint is disabled by default. This feature is currently experimental.
1016 # Stats can be access through http://localhost:5066/vars. For pretty JSON output
1017 # append /pretty to the URL.
1018
1019 # Defines if the HTTP endpoint is enabled.
1020 http.enabled: false
1021
1022 # The HTTP endpoint will bind to this hostname or IP address. It is recommended to use only localhost.
1023 http.host: localhost
1024
1025 # Port on which the HTTP endpoint will bind. Default is 5066.
1026 http.port: 5066
1027
1028 #----- X-pack Monitoring -----
1029
1030 # APM server can export internal metrics to a central Elasticsearch monitoring
1031 # cluster. This requires x-pack monitoring to be enabled in Elasticsearch. The
1032 # reporting is disabled by default.
1033
1034 # Set to true to enable the monitoring reporter.
1035 monitoring.enabled: false
1036
1037 # Not settings from the Elasticsearch output are accepted here as well.
1038 # Note that these settings should be configured to point to your Elasticsearch monitoring cluster.
1039 # Any setting that is not set is automatically inherited from the Elasticsearch
1040 # output configuration. This means that if you have the Elasticsearch output configured,
1041 # you can simply uncomment the following line.
1042 monitoring.elasticsearch:
1043
1044 # Protocol - either 'http' (default) or 'https'.
1045 protocol: "https"
1046
1047 # Authentication credentials - either API key or username/password.
1048 api_key: "id:api_key"
1049 username: "elastic"
```

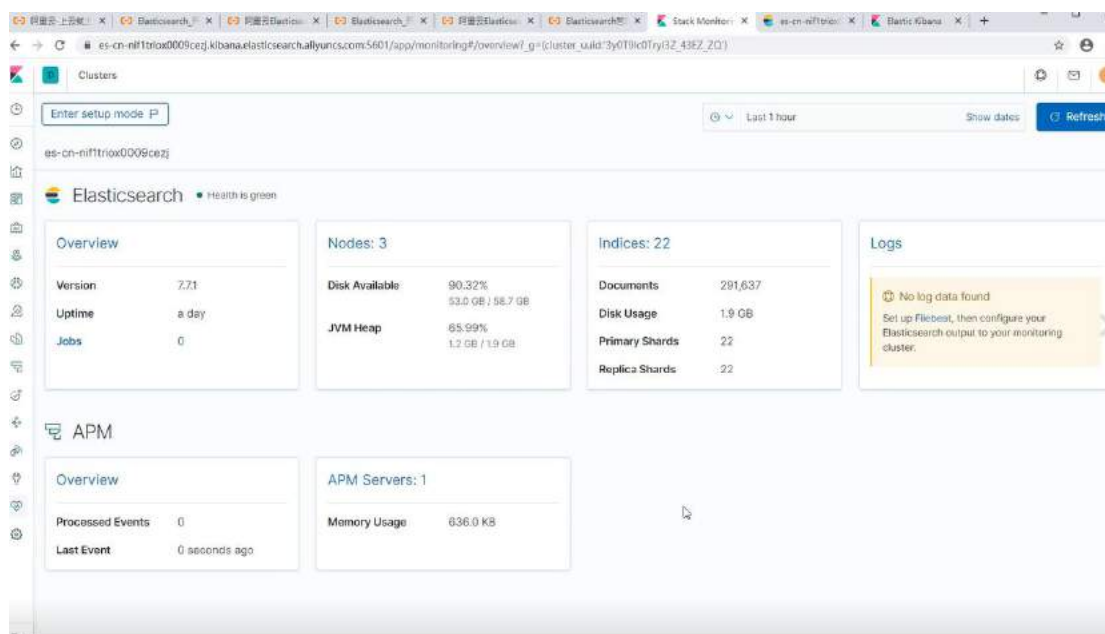
在 Windows 下它是一个 exe 文件，我们输入 CMD 打开窗口，再运行 APM server。



搭建好之后要先去检查一下 Monitor 是否部署到位，我们找到 Kibana 界面的 Stack Monitoring，点进去查看。

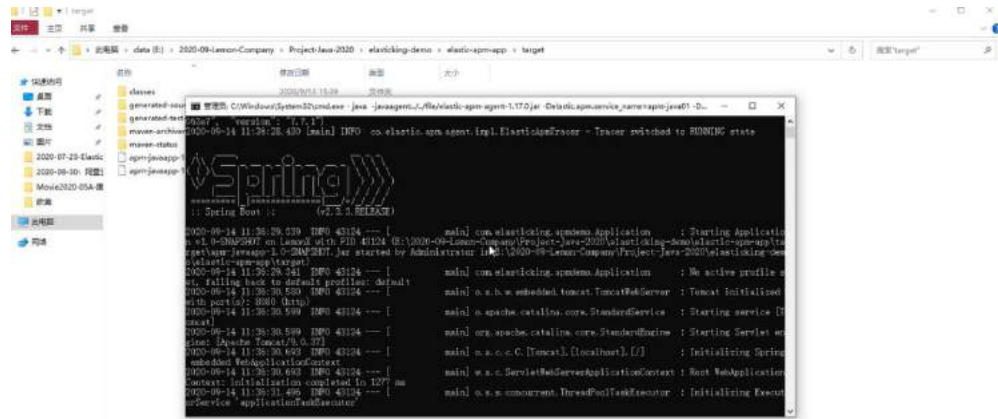


点进去之后，可以在页面上看到我们刚启动的 APM server。

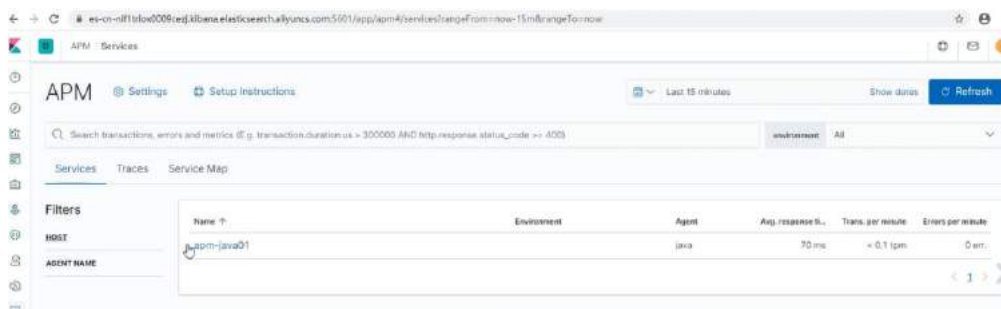


接下来，我们就可以把应用程序的日志采集上来进行分析了。为了演示，我们需要写一个 Java APP，这里已经提前写好了，我们开始编程。

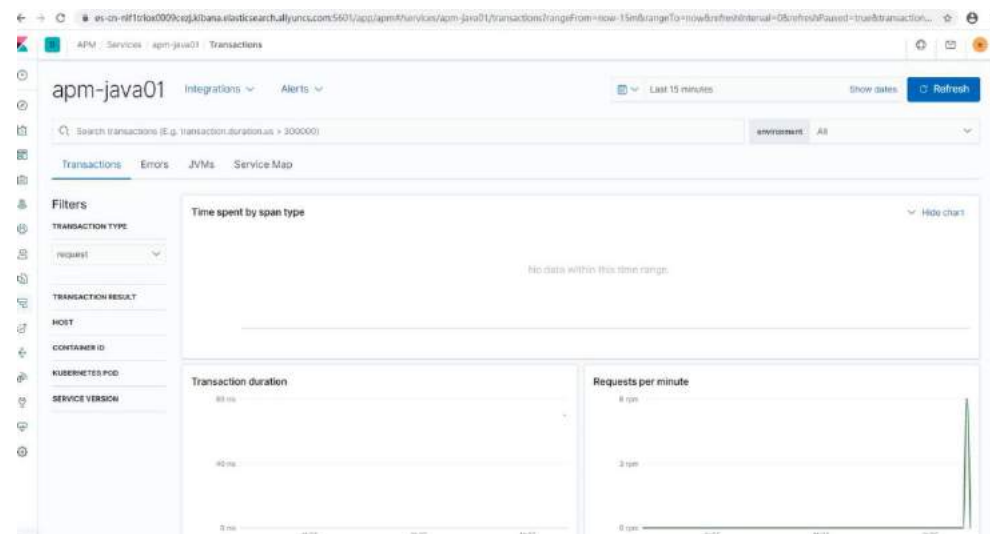
基于这些我们来运行一下，看看效果怎么样。我们回到目录打开这个文件，可以看到，这是一个典型的 Spring Boot 项目，它已经运行成功了，然后我们去访问一下。



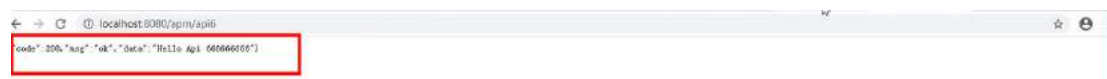
我们刷新这个界面，大概几秒钟就可以看到 apm-java01 了。



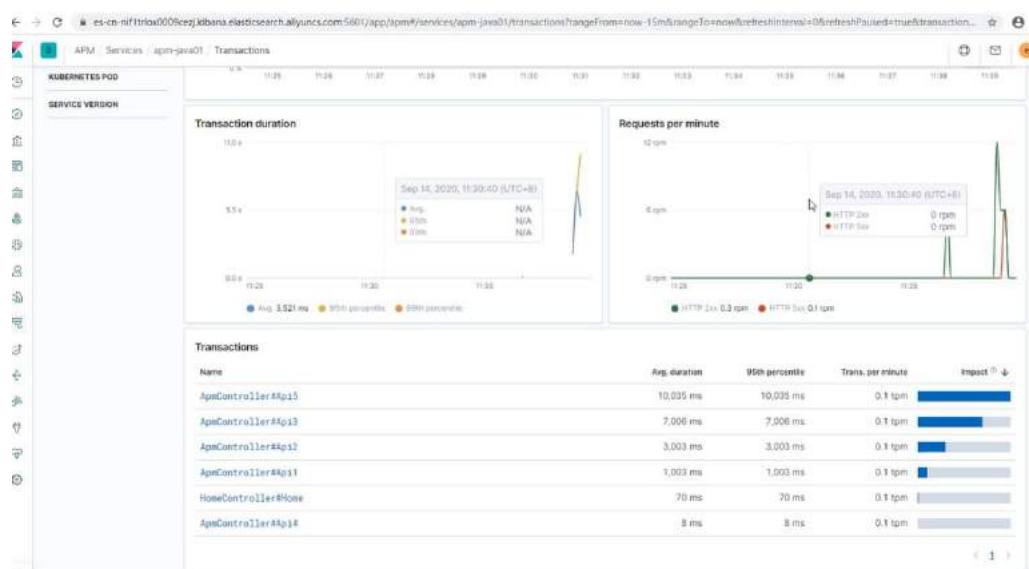
我们点进来，可以看到 APM 的窗口。



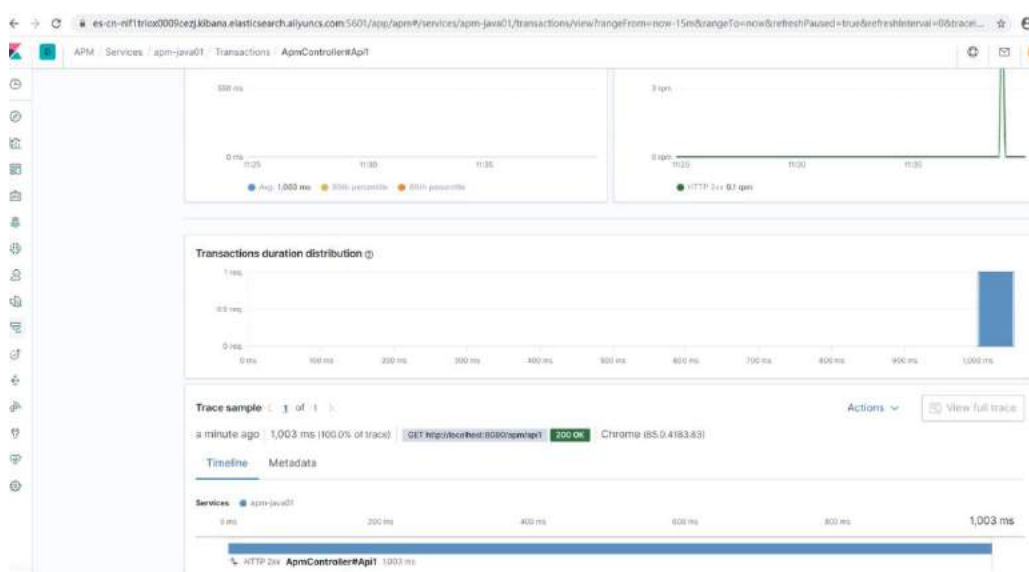
接下来我们把其他的接口都访问一遍。



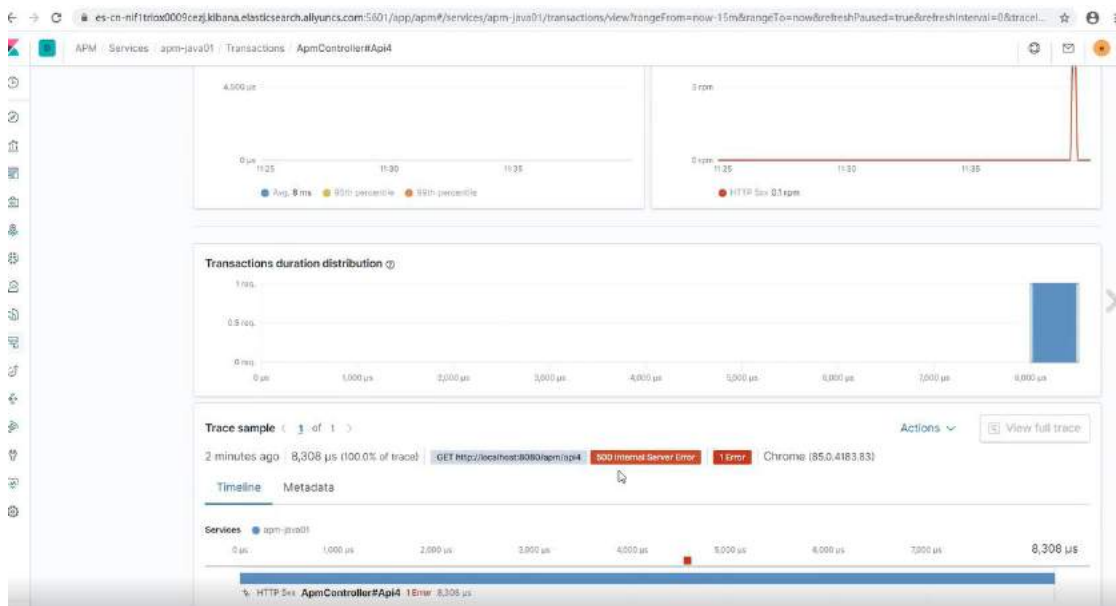
然后刷新列表,我们可以在里面发现存在的问题,比如这个接口平均响应时间居然要十几秒,说明它一分钟只能做不到 10 个处理。



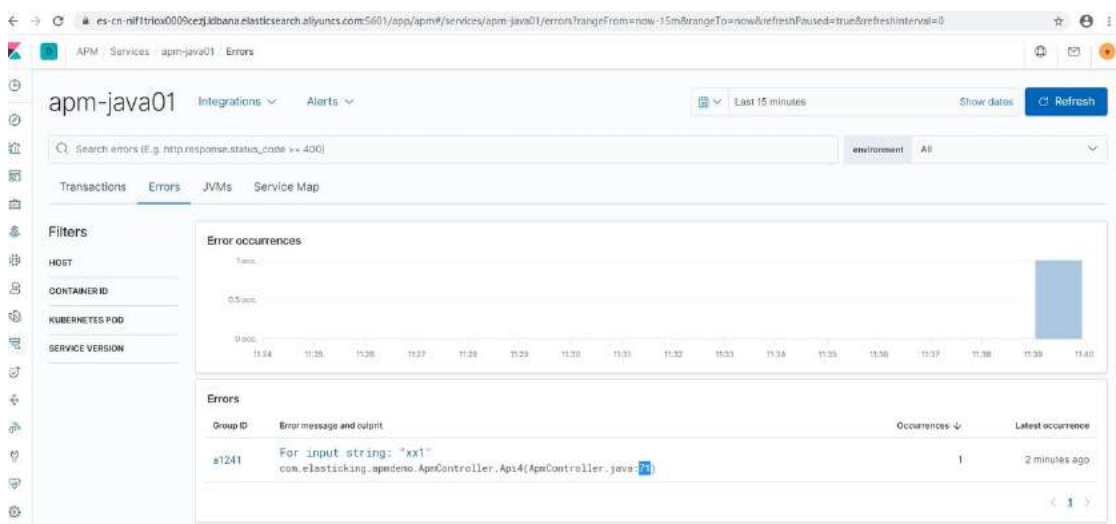
接下来我们一个一个看。首先点进 API1,可以看到它消耗的时间比较长,因为我们在程序里面故意做了一个延迟,API2 和 API3 也是一样,我们都在里面做了延迟的操作。



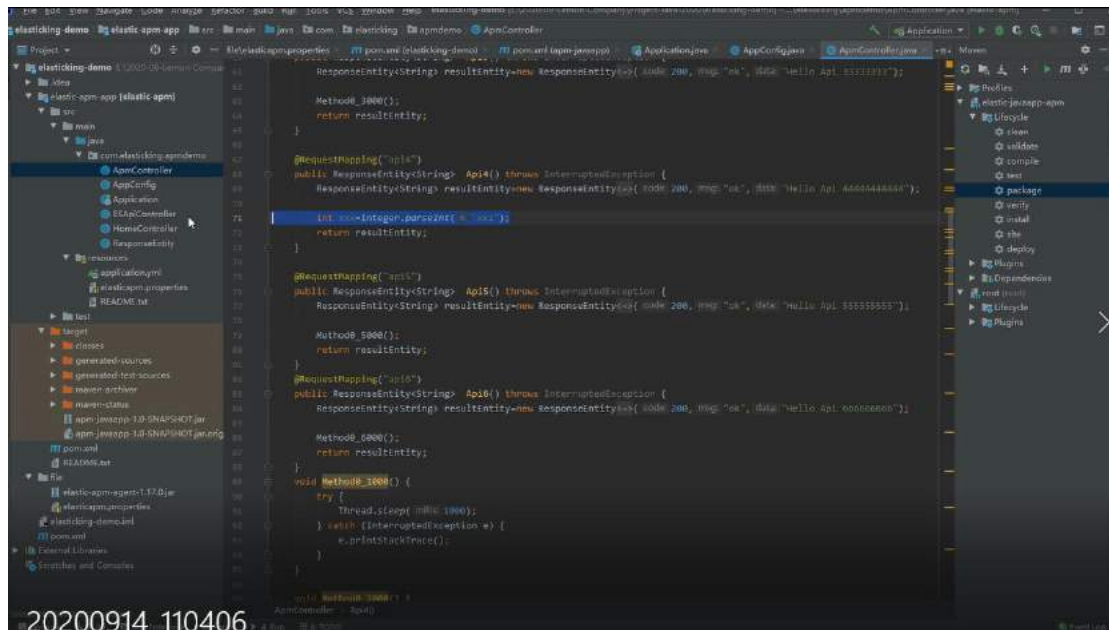
API4 因为我们写的是报错, 所以大家可以注意到它的状态是不一样的, 它这里是返回 500。



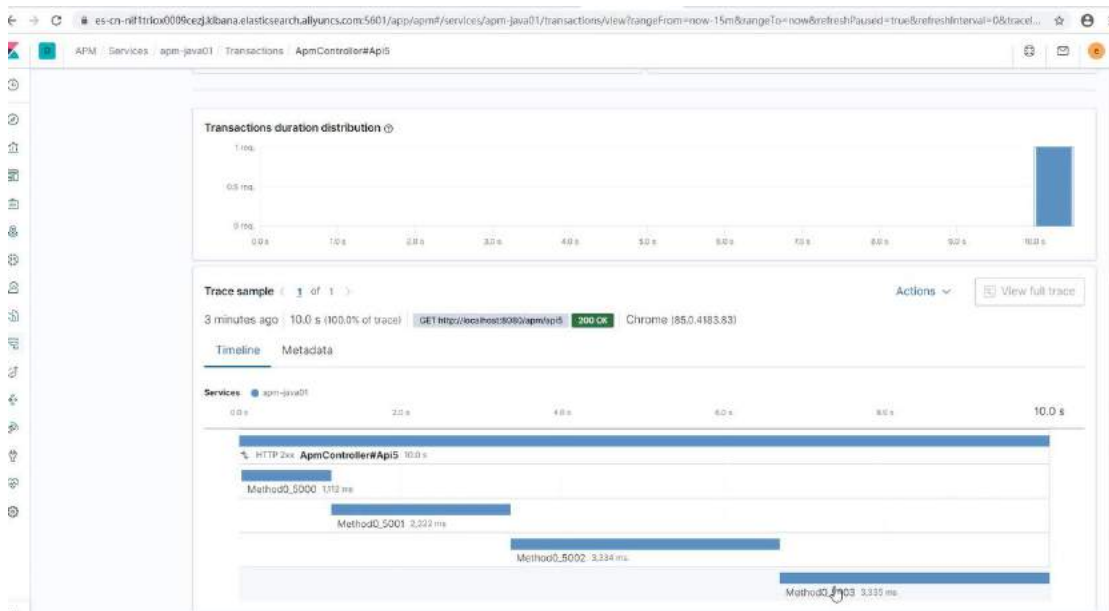
作为开发人员, 我们最关心的就是程序接口的响应速度和错误, 而 ES 已经帮我们做了太多工作。我们在 Errors 这个页面可以找到刚刚的错误, 它甚至把代码行定位出来了, 在第 71 行。



我们回到 71 行代码片段看一下, 确实是这里错误。



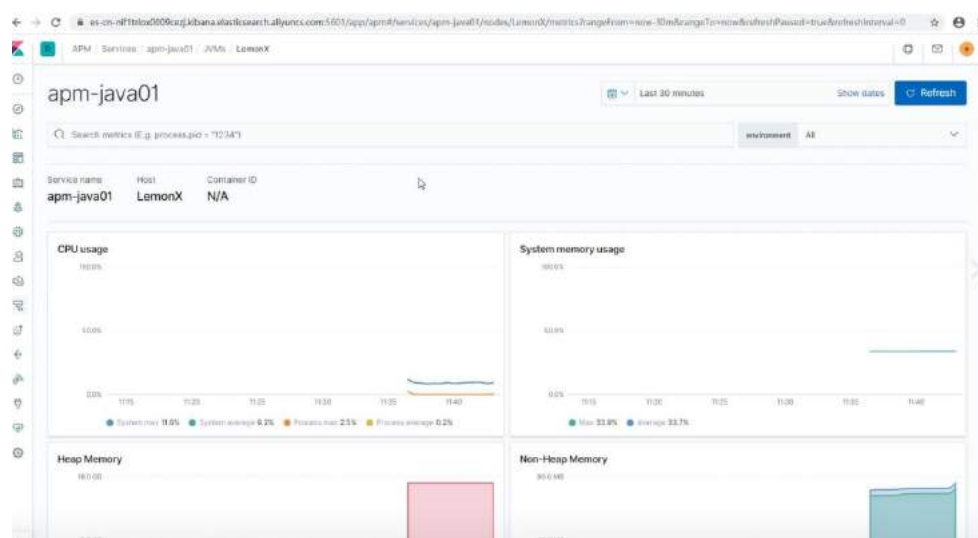
接下来再看看 API5，API5 我们是用来说检查全链路追踪的。5000、5001、5002、5003，基本上把全链路都抓出来，看为什么时间消耗这么多。



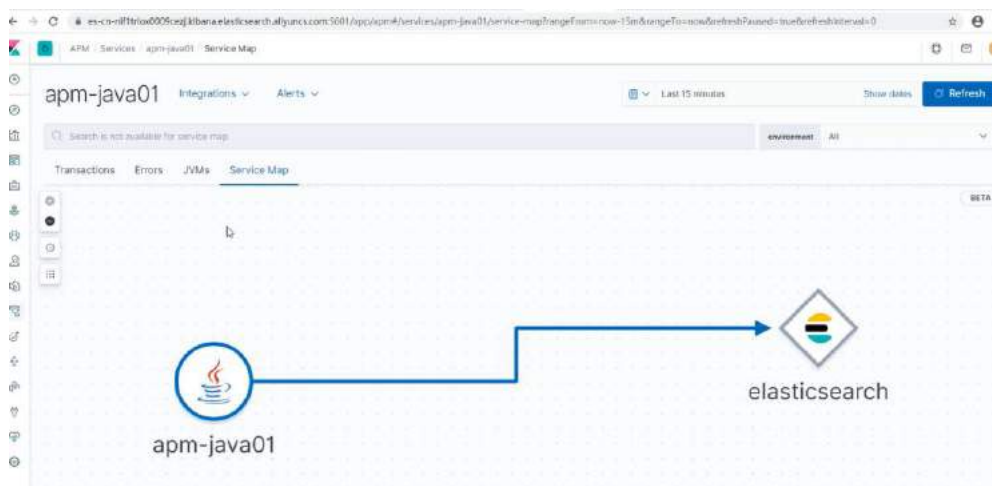
以上就是 Transactions 的部分，下面讲讲 JVMs，我们从这里点进来。



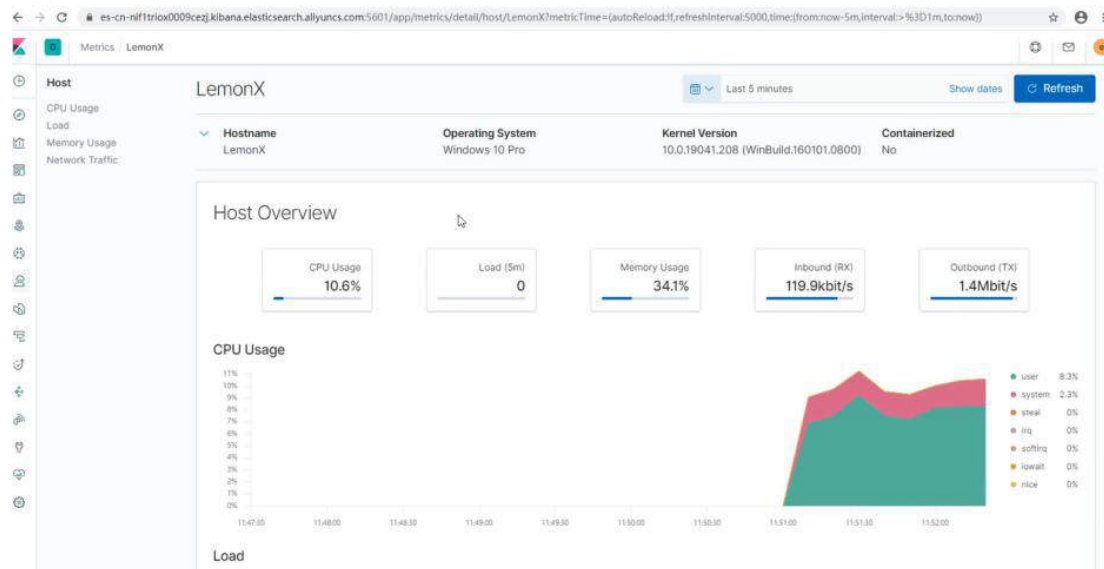
在里面可以看到历史的数据、追踪的数据，CPU 消耗的内存、堆栈等等。



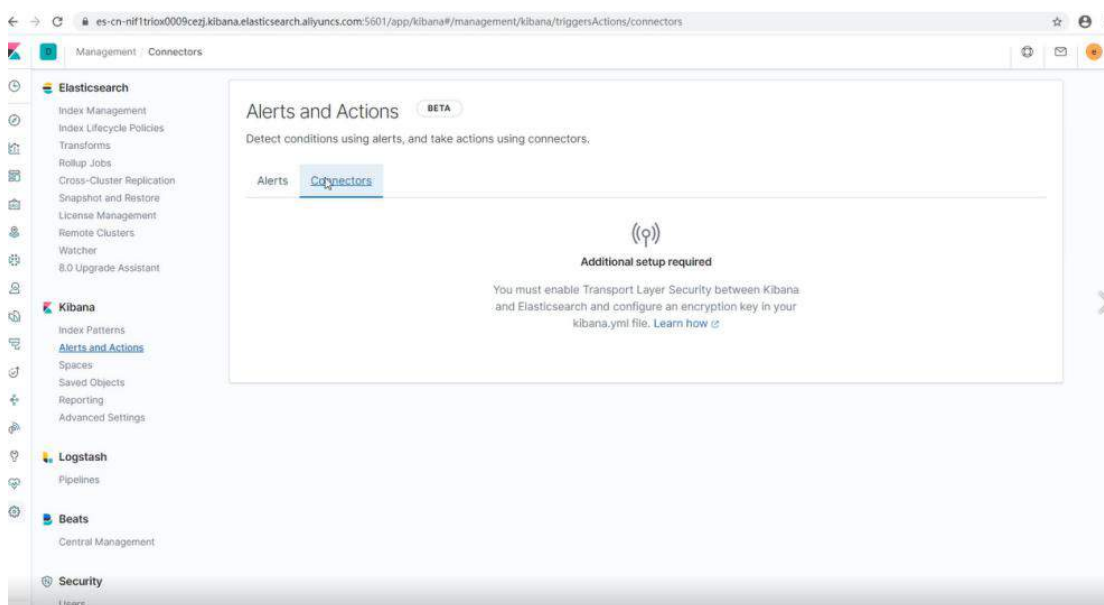
接下来再介绍一个强大的 APM，Service Map。我们点进去可以看到服务的调用关系，比如我们的 java 程序调用了 ES，假如我们的服务调用了很多服务，这个服务又调用了下一个服务，你就能直观地知道你的应用是怎样的情况，非常方便。



以上就是关于 APM 的介绍，当然 ES 的监控其实有很多，APM 只是其中一个，它还包括了日志指标，Filebeat、Metricbeat、Heartbeat 等指标采集，它们的配置和 APM 基本一样，配置完成后我们在 ES 界面上就能看到相应的数据。



最后给大家说一下 Alert 告警的位置，我们在 Management 的 Alerts and Actions 里可以找到，然后自己进行配置。





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