SOFAStack

业务智能可观测平台

运维指南

产品版本：AntStack Plus 1.11.0

文档版本：20221021

蚂蚁集团
ANT GROUP

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

商标声明

免责声明

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.运维概述

1.1. 功能架构

业务智能可观测平台 BOS（Business-Intelligent Observability Service） 通过采集各种指标、日志和链路等数据，并进行海量数据的清洗、计算。以此来支撑产品层的应用监控、业务监控、平台监控、基础设施监控、告警管理和分布式链路等能力。作为一个企业级产品，BOS 提供资源租户隔离、访问鉴权控制和监控配置模板等企业级特性。并且提供数据高可用、服务高可靠、双机房容灾部署等平台特性。

基于 BOS 丰富数据和强大功能，可以支撑容灾巡检、故障重放、弹性扩缩、微服务治理和全链路压测等场景。

- **应用监控**

BOS 能同时监控容器应用和经典应用，并通过 LDC、IDC 和单机实例等多视角、多维度逐层下钻分析，实时展现服务实例、依赖的中间件和基础资源运行状态、使用趋势和告警信息，发掘应用故障所在的层级和对象，保证应用的流畅运行。

- **业务监控**

  BOS 提供了灵活的、基于业务场景的自定义业务监控，通过业务监控可将不同监控图表展示到同一个屏幕上，通过不同的大盘模板、统计模板等形式来展示可观测数据，例如，分钟级多 Key、TopN 等，让用户可以全面、深入地掌握业务数据。

- **中间件监控**

  BOS 默认集成了对消息、Mesh、数据库等中间件的监控，并支持在应用监控中查看应用所调用各中间件的情况。

- **基础资源监控**

  支持对物理机、虚拟机、Kubernetes 集群和原生容器等资源的监控。

- **日志管理**

  BOS 提供日志查询和日志关联功能。用户不仅可以对日志执行查询操作，还可以进行历史查询和上下文查询，以及查看 Error 指标关联的错误日志和链路关联业务日志，更加方便高效地进行问题分析定位。

- **分布式链路**

  分布式链路帮助运维人员、开发人员和架构师看清楚复杂的大规模微服务架构下的应用及服务之间的复杂调用关系、性能指标、出错信息与关联日志，从而实现故障根因分析、服务治理、应用开发调试、性能管理、性能调优、架构管控、故障定责等运维开发工作。

- **告警**

  针对各资源对象，BOS 允许用户灵活地配置自定义告警规则，并支持多种订阅方式，如邮件、短信、钉钉等。当监控数据满足阈值条件时，第一时间通知对应的运维人员，帮助其发现异常及原因。

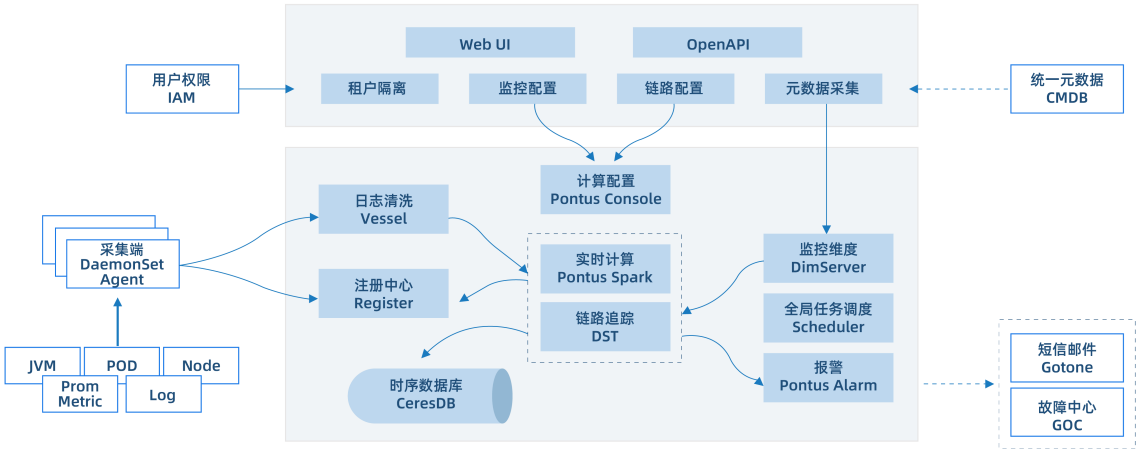
- **企业级特性**

  在企业级特性层面，BOS 包含的能力如下：

  - 提供多层级业务和资源隔离机制，比如租户、workspace 等，保证监控数据的安全性。
  - 支持监控具有多 VPC 服务实例的应用，并以应用中心视角展示监控数据。
  - 支持监控配置模板导入与导出，使测试环境的应用监控配置可在生产环境一键激活，避免了重复配置，也保证了发布流程的标准化和可控性。

1.2. 系统架构

BOS 的基本实现原理是在所有的应用主机上部署 Agent（代理），从而采集主机和应用的各种指标、日志和链路等数据，并将其存储在 CeresDB 中，通过在 PontusSpark 中进行分析计算，实现在不同产品场景中的可视化展现和告警通知。



采集层

- **DaemonSet Agent（又称 Pontus-Agent）日志采集代理**
  - 负责所有的数据的最终采集，包括原始日志拉取和指标型数据采集。
  - 具有插件化的能力，统一本地调度能力，负责拉起插件和解析数据，解析完了之后做一次聚合计算。然后放入本地的缓存中，等待 Vessel 来拉取数据。
  - 提供一次性的任务执行能力，主要用于性能分析。
  - 启动时会和 Register 进行通信。通信之后会获取 Agent 运行所需要的所有配置，包含运行时采集配置，Vessel 建连的配置等。
- **Register 注册中心**
  - 负责与 Agent 之间保持网络心跳，收集所有 Agent 的状态，版本透视等信息。
  - 负责 Agent 与 Vessel 的配置生成与下发，即负责管控所有的采集配置。
  - 负责管控 Agent 与 Vessel 之间的连接与会话（session），Register 感知 Vessel 集群负载，负责通知 Agent，应该与哪个 Vessel 地址进行建连。
- **Vessel 流量清洗组件**
  - 抽象来看，Vessel 其实就是一个独立部署的 Agent，这个 Agent 通过 remote input 的形式，拿到了日志或者其他的原始数据。
  - 一个 Vessel 服务可以托管 N 个 Agent，即从 N 个 Agent 中收集数据。
  - Vessel 可以通过一系列的解析规则，最终将非结构化的数据变成结构化的数据并返回，也就是所谓的数据清洗。
  - 所有上层组件（包括计算层与链路层组件）都是通过 Vessel 来获取数据。

计算层

- **DimServer维度数据服务（又称元数据服务）**
  - 包括维度数据的读和写的服务。在这一层上无业务语义，业务语义为上层产品层注入。
  - 计算和采集层都会用到维度数据（元数据），通过 DimServer 提供的 Java Client 来实现数据通讯。
  - 为适配任意维度数据结构，DimServer 实现了维度数据 Table 化，即使用若干张宽表来处理所有可能的元数据表结构，这类宽表被称为维度表。

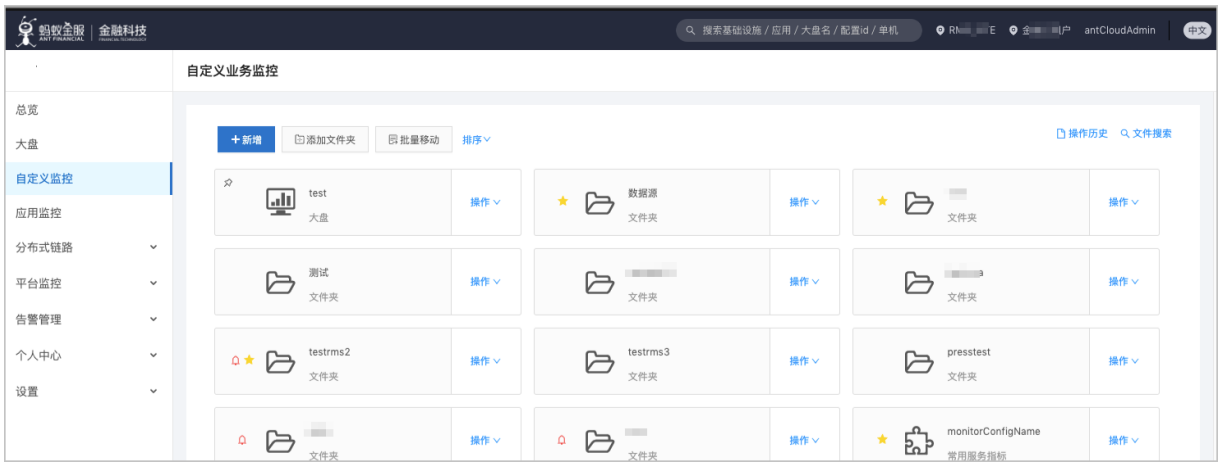
- **PontusConsole 监控数据平台的管控端**
  - 主要负责内部运维性管控，系统配置，以及提供给产品使用的配置接口。
  - 提供操作维度表、指标表的标准数据服务接口。
- **Scheduler 全局任务调度器**
  - 不依赖任何中间件，实现全局任务调度能力，仅用于监控系统内部，实现功能自闭环。
  - 负责根据配置信息定时生成并分发监控的计算任务。
  - Scheduler 可以实现多机房部署，通过数据库维持高可用。
- **PontusSpark 分布式计算引擎**
  - 功能强大的 Spark 计算集群，负责接收产品层的计算配置，并对数据进行离线计算和统计。
  - 从 Register 中获取监控数据采集的 Vessel 地址，并通过 Vessel 获取必要的监控数据。
- **PontusAlarm 监控告警组件（又称 BOSAlarm）**
  - 主要负责将监控计算完成的告警信息以短信、邮件、钉钉等方式向外推送。
  - 在 SOFAStack 输出的模式下，告警渠道主要通过 Gotone 服务提供。
- **MonitorGateway 分布式链路追踪**
  - 主要负责分布式链路处理的相关功能。
  - 对于链路信息的收集，提供三种方式，分别为主动日志采集、SLS 日志采集以及链路日志上报。

数据层

主要是由时序数据库 CeresDB 来承载。CeresDB 是蚂蚁自研的时序数据库，时序引擎是一种存储和管理时间序列数据的分布式数据库，为时间序列提供高性能读写、预处理计算、可视化查询等功能。

产品层

产品层（应用名为 monitorprod）承载了 BOS 的所有页面以及用户交互、配置逻辑，是监控对客的主要门户。如下图所示：



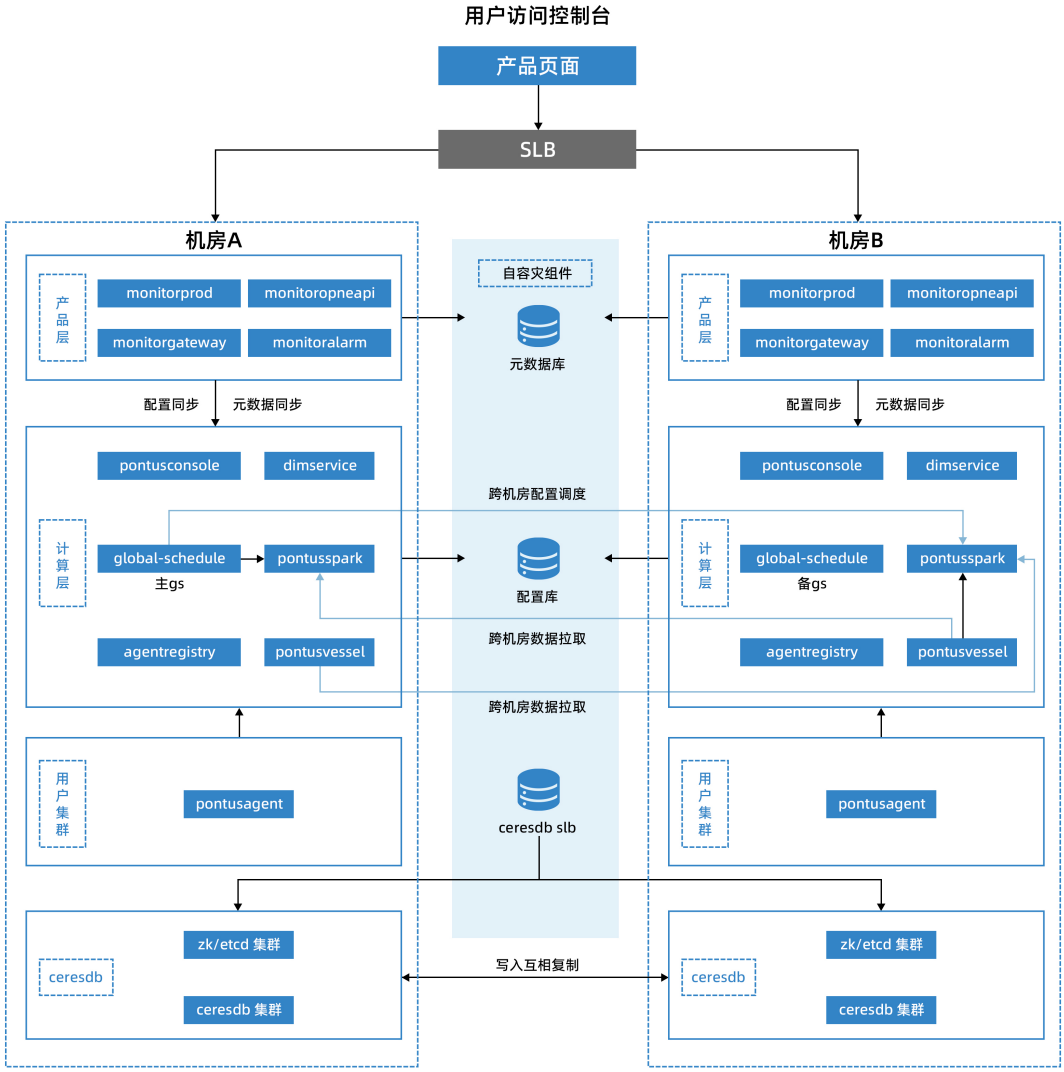
产品层负责与用户权限系统 IAM 打通，读取用户身份信息，同时 BOS 的所有权限管控在 IAM 上进行收口。同时，产品层也负责打通 PaaS 元数据，将元数据同步至 DimServer 。

> 说明
>
> PaaS 元数据，是指通过 PaaS 系统发布的描述应用的基本信息，例如：应用 IP、机房、单元化等信息。

1.3. 产品容灾架构

1.3.1. 经典网络的容灾架构

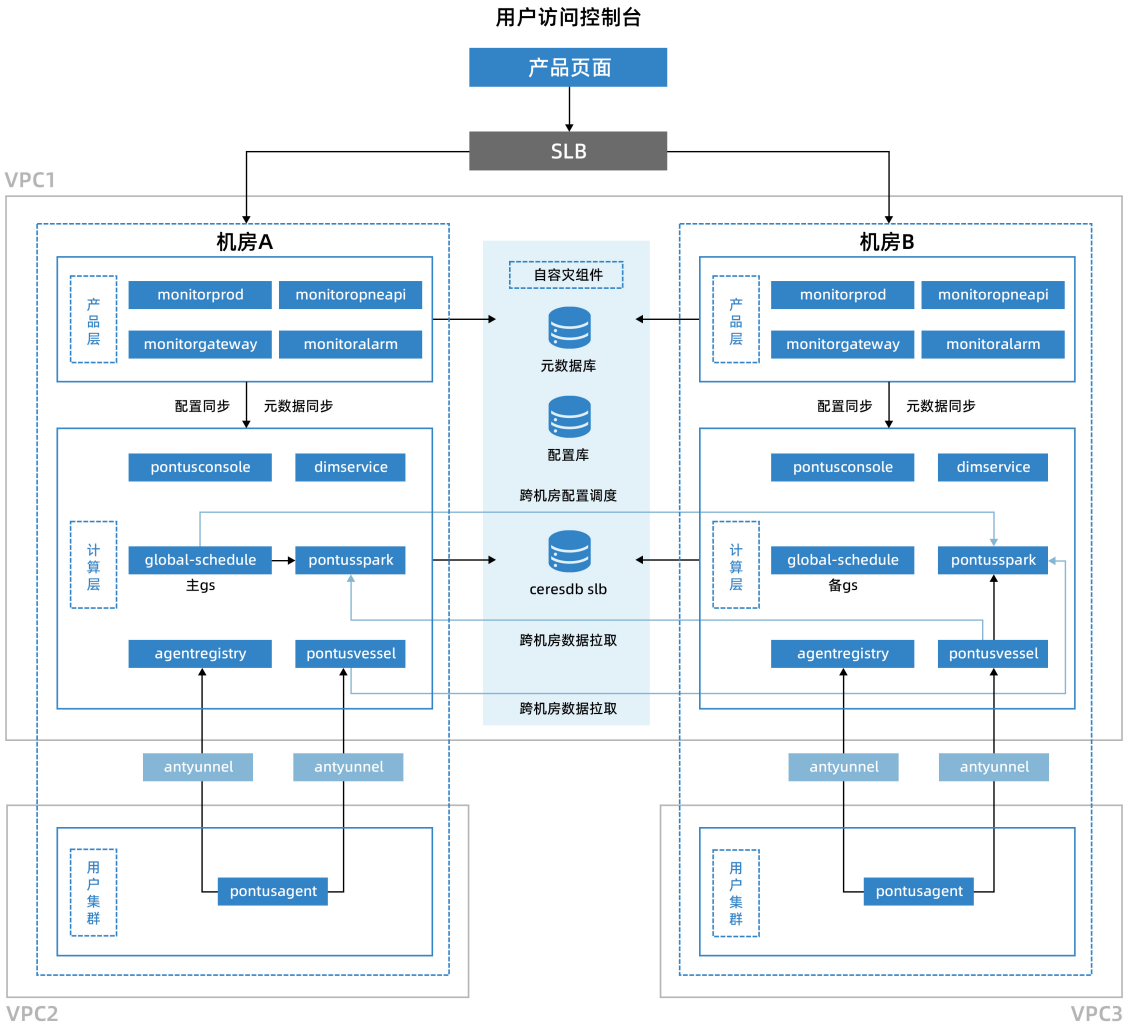
经典网络的容灾架构如下图所示：



- 两个机房均部署全量的 BOS 产品组件。
- 数据层的容灾主要依赖 OB、CeresDB 等存储产品的容灾能力，详请参考 OceanBase 容灾。
- 对于 Scheduler 来说同一时间只有一个机房的 Scheduler 会起作用，作为“主”，提供分布式调度服务。该主 Scheduler 与数据库之间保持心跳，如果机房宕机，则备机房的 Scheduler 可以从数据库中识别心跳暂停，从而启用灾备模式，继续提供分布式调度服务。

1.3.2. VPC 网络的容灾架构

对于 VPC 网络，与经典网络最大的区别在于网络隔离，因此会有独立的 anytunnel slb 对外提供 Vessel 和 Registery 的代理。整体容灾架构与经典网络基本一致。



1.3.3. OceanBase 容灾

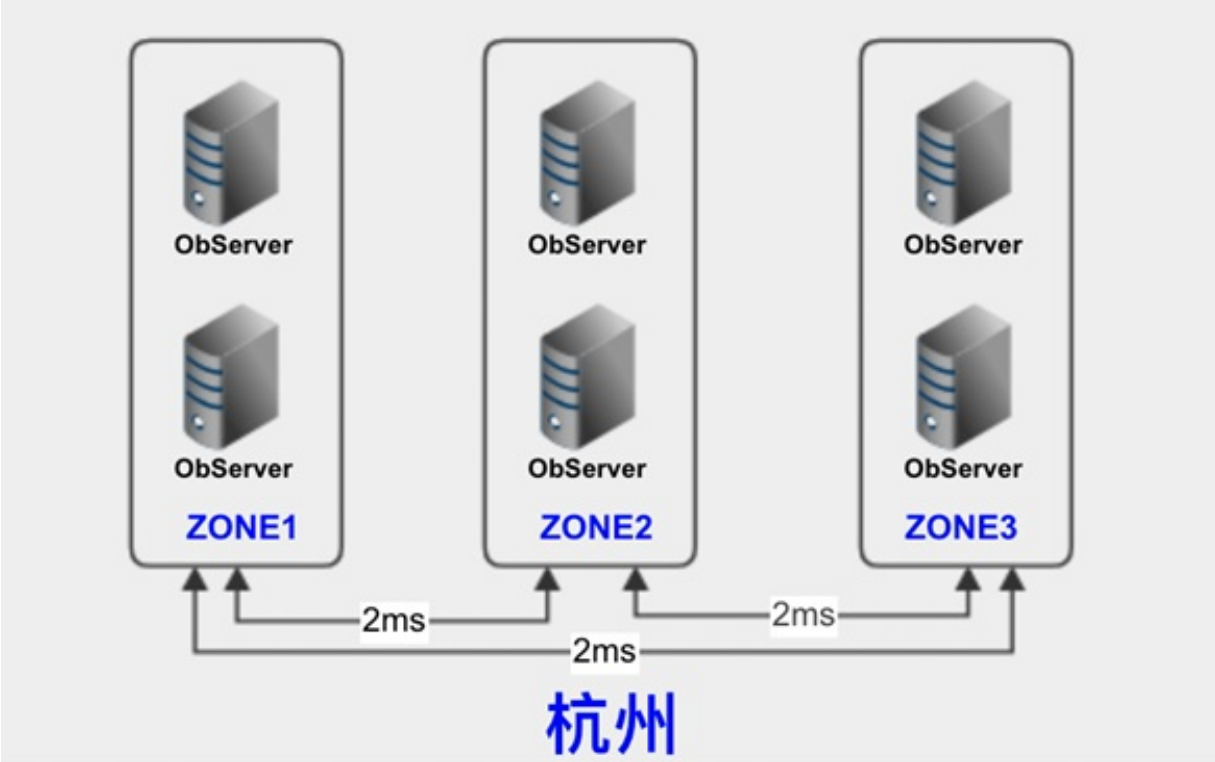
1.3.3.1. 同城三机房/同机房三 Zone 部署架构

该部署架构也是 OceanBase 线上最小化部署的架构，需要 3 台机器，每台机器一个副本，是 OceanBase 早期最常用的一种部署架构。

同机房三 zone 部署能够支持单机容灾，同城三机房部署除了能支持单机容灾外还能支持单机房的容灾。

该部署架构各个 zone（或者机房）间的延迟很低，OceanBase 的主可以在任意的一台 ObServer 上，但是出于运维和高可用的要求，一般会把主放到一个 zone 中（主放到 3 个 zone 中只要有单机故障就会有影响，放到一台机器上，单机故障至少有 2/3 的概率对业务无感知），大促情况下再把主打散来获得性能上的成倍提升。该部署架构不支持二次容灾，只能抗一次单机故障或者一次单机房故障，故障之后再出故障就会面临停服的风险。

另外该部署架构机房建设成本比较高，要求每个 zone 之前的延迟一般不高于 2ms（三机房部署每个机房间的延迟最好也是低于 2ms），集团内部由于阿里云强大的机房建设能力能做到这一点，外部用户（如银行、国际专有域等）很少能满足同城三机房的要求，因此就有后面的几种部署架构。



1.3.3.2. 两地三中心五副本架构

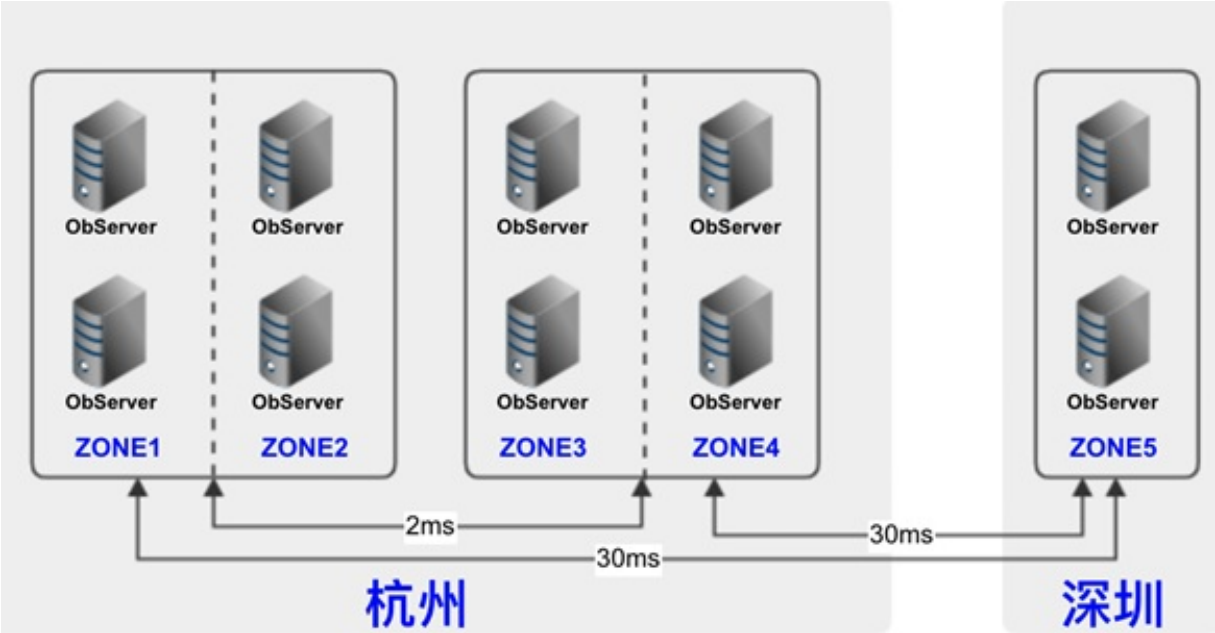
该部署架构要求机房是在两个城市，其中一个城市有两个机房，另外一个城市一个机房。该部署架构最少要求是五台机器，每台机器一个副本，一共是五副本。杭州两机房，每个机房有两个副本，一共四个副本，深圳一个机房一个副本。

该部署架构的成本高于三副本部署架构，但是单城市单副本的机器可以部署 OceanBase 的日志副本，只负责 paxos 投票，不存静态数据，这样对机器要求大大降低，log 节点甚至可以部署在 ECS 上。

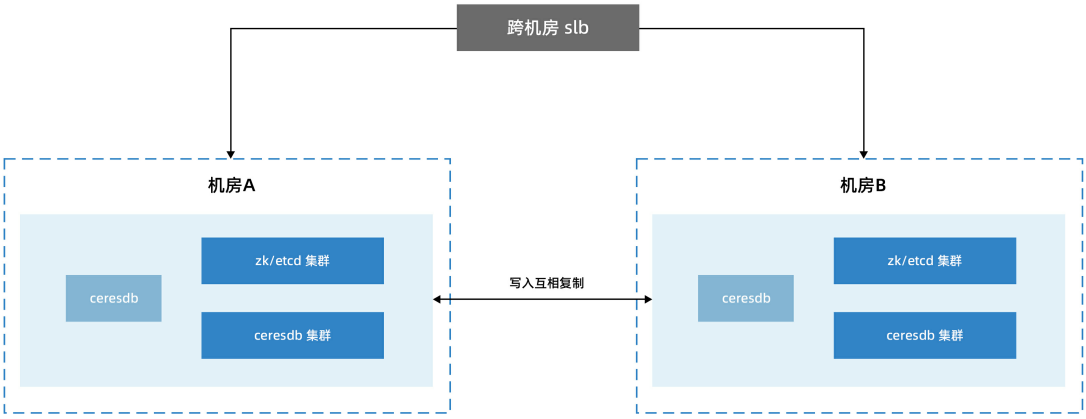
该部署要求同城两机房的延迟低于 2ms，城市间的延迟一般会大于 30ms，30ms 物理距离基本上能做到北京到深圳了，可以说要求是比较宽的，另外同城两机房的要求也是很容易满足的。

该架构支持单机容灾、机房容灾。五副本架构，paxos 协议要同步三份才算成功，大部分情况下是同城四副本中的三个，当一个机房全部挂掉后，paxos 要跟远程的副本进行强同步，这时候业务耗时会陡增，每次事务提交增加 30ms。在这种情况下，就需要借助 OceanBase 提供的副本降级功能，从五副本降级为三副本，从而恢复业务。

该部署架构支持“简单”二次容灾，最差情况下是单机房挂掉还能支持一次单机故障（前提是已经从五副本变为三副本），不支持单机房故障后再出现机房级故障，也不支持城市级别的容灾。



1.3.4. Ceresdb 双机房容灾



写入策略

- 任何发送到 A 机房的写入都会同步复制写入到 B 机房，同样的道理任何发送到 B 机房的写入都会同步复制到 A 机房。
- 错误处理为：任何一个机房写入成功，即认为写入成功。

查询策略

由于写入只需要一方写入成功，即可认为写入成功，那么实际上，双机房的数据无法保证一致，因此查询的准确性在灾难发生的期间无法得到保证，我们只是尽量保证查询出来的结果对于用户来说是一致的。

查询的流程如下：

- 落到任何一个机房的查询，都会并发查询另外一个机房的查询。
- 两个查询如果有任何一个出错，那么使用其中一个。
- 两个查询如果都成功，不做合并，而是按照固定的选择策略进行选择（保证客户端看到的数据是一致的），选择的策略目前按照：优先选择数据更多的；数据一样多的情况下，优先选择 A 机房。

熔断机制

类似现在 cafemonitor 的小黑屋机制，CeresDB 的双写双读功能也需要支持熔断机制，以及熔断恢复。

2.高危操作

2.1. 运维操作的危险级别定义

所有运维操作按业务场景和操作的危险程度被分为三级：G1、G2、G3。各级别定义如下：

- **G1**：L1|L2 可以根据文档完成操作，不需要做变更申请，该操作按文档执行完全安全，该操作不会对业务产生影响。
- **G2**：L1|L2 驻场人员可以根据文档指引，需要做变更申请，向产品侧咨询确认方可执行操作。该操不会对业务产生影响。
- **G3**： L1|L2 驻场人员可以根据文档指引，需要做变更申请，向产品侧以及客户侧咨询确认方可执行操作。该操作有可能对业务产生影响。

> **说明**
>
> 原则上，驻场运维人员对任何变更操作都应该做变更申请，仅限于少数的应急场景中可以不做申请直接操作。

2.2. 运维操作和文件

本节将列出您在运维时需格外注意的高危运维操作和不能随意清理的高危文件。

高危操作

高危操作即上文提到的 G3 级别的运维操作，进行该类操作时请按照相应的指导进行。

技术栈变更

技术栈变更会影响监控的全局配置，严重者会导致监控不可用，且恢复困难，属于高危操作。

如需进行技术栈变更，请联系 BOS 产品侧，提供技术栈变更的具体需求。由 BOS 产品侧对可能造成的影响进行评估，出具符合需求的技术栈配置。

高危文件

高危文件是指不能被随意清理的文件，如需清理，需要与产品，客户进行确认。

应用	路径	用途
ceresdbtrace	容器内：/data/	存放时序数据
ceresdbmr	容器内：/data/	存放时序数据
agent	宿主机：/home/agent/config	Agent 配置信息
agent	宿主机：/home/agent/data	Agent 采集数据暂存

3.运维准备

产品运维过程中需要准备以下登录入口、账号、权限及工具。

名称	登录入口	权限
云游	yunyou.example.com。“example”对应根域名，需要根据实际环境确认。	产品管理员
监控	Corewatch.example.com。“example”对应根域名，需要根据实际环境确认。	管理员

4.应急维护

4.1. 应急维护的定义和流程

应急维护的定义

在出现重大故障时，最短时间内进行业务恢复的处理定义为应急处理操作。

所谓重大故障，是指发生突然、影响面广、涉及范围大、并可对网络的安全运行与服务质量造成严重后果的故障，如可用性故障。

应急维护的流程和原则

如遇到下文覆盖的故障，建议按照文中指导进行故障排查和修复；若故障排查和修复遇到困难，可联系技术支持寻求帮助。如遇到下文未覆盖的故障，请直接联系技术支持。

4.2. 应用宕机

4.2.1. monitorprod 应用宕机

monitorprod 应用宕机主要分为以下两种情况。

场景 1

【现象】

BOS 页面无法打开，无法访问，报网络连接失败，或者浏览器页面报 503 错误。

【可能原因】

- Java 内存溢出导致应用宕机
- 组件的 bug
- 底层 PaaS 异常调度

【解决方法】

1. 登录云游页面，找到 monitorprod 应用的容器，检查容器状态是否健康。
2. 若容器状态异常，可直接通过云游发起容器重启操作。
3. 若容器可以登录，通过 `ps -ef|grep java|grep -v xflush` ，检查容器 Java 进程是否存在。
    - 若 Java 进程不存在，则查看 `/home/admin/logs` 下是否有 .Hprof 结尾的文件，若有此文件，则保存此文件，然后参考第 2 步重启服务。
    - 若 Java 进程存在，可以执行 `jmap -heap <pid>` 命令，查看各内存区域的设置；执行 `jmap-histo:live <pid>` 命令，查看每个 class 的实例数目、内存使用率和类全名；保存收集到的 JVM 相关数据、信息，联系技术支持人员排查，然后参考第 2 步重启服务。

【验证方法】

页面是否可以正常打开，正常访问。

场景 2

【现象】 monitorprod 3 实例，其中一个实例端口检测失败。

【可能原因】

monitorprod-1 实例 Java 进程被系统杀死（OOM Killer）。

出现 OOM Killer 原因如下：

由于物理机本身内存为 8G，同时物理机上还有其他容器和进程，所以实际上 monitorprod 是和物理机上其他进程共享 8G 资源。本次 OOM 主要是因物理机上的其他进程 systemd 内存消耗过高（1.5G）而导致 monitorprod java 进程得分最高被系统杀死。可以概括为如下几点：

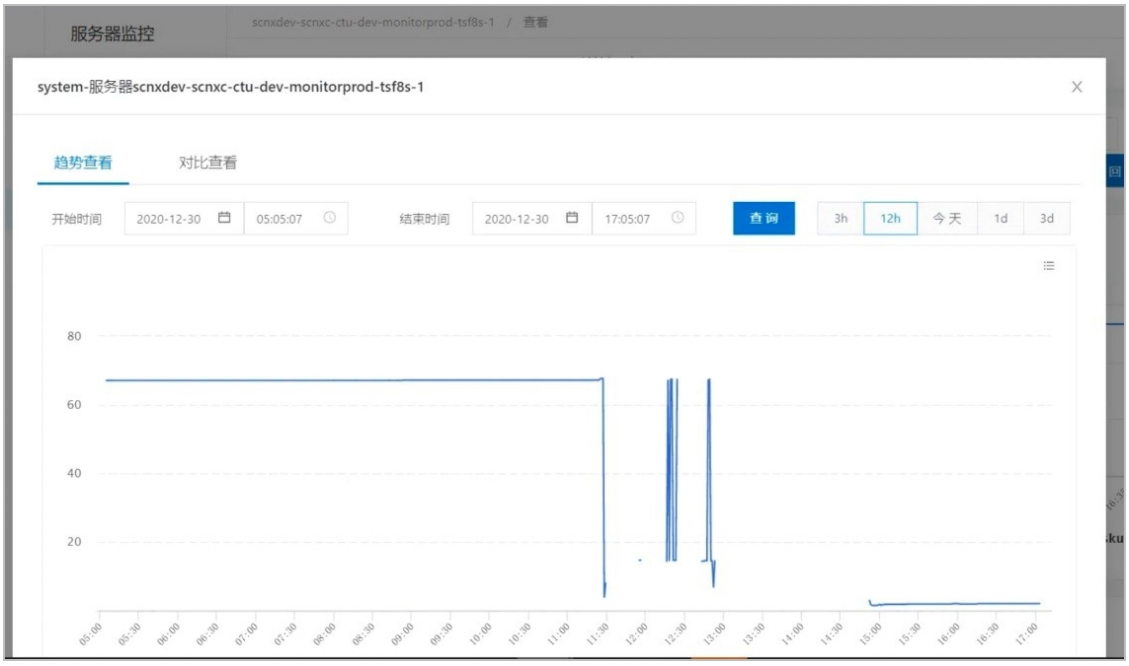
- monitorprod 的 8G 资源是共享型而非独占型。
- monitorprod-1 实例所在物理机 systemd 进程占用 1.5G 内存。
- pontusagent 容器发现自杀现象，从侧面说明 Agent 本身消耗的内存到达自身设置的上限。

正常情况下系统各进程内存使用，如下图所示：

```
top - 16:49:36 up 12 days,  5:42,  1 user,  load average: 0.18, 0.22, 0.30
Tasks: 172 total,   1 running, 129 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.8 us,  2.0 sy,  0.0 ni, 96.1 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 99.5/7971192  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||]
KiB Swap:  0.0/0        [                                                                                                          ]

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
22521 admin     20   0 9991252   6.2g      0 S   3.3 81.4 809:18.27 java
 9869 1100      20   0 3605136 113084      0 S   0.0  1.4   1:22.95 java
 2673 root      20   0  742720  78512   5564 S   0.0  1.0  46:51.15 asar
 3286 root      20   0 1208296  57444   3756 S   3.7  0.7 507:02.84 sigma-slave
 2837 root      20   0 2395800  49132   1088 S   0.3  0.6  91:08.23 java
 3058 root      20   0 1533804  44852   5596 S   1.0  0.6 164:10.84 dockerd
 3981 root      20   0   59680  24300    720 S   0.7  0.3  74:30.23 kube-proxy
 8413 root      20   0 1179764  19988      0 S   0.0  0.3  50:22.90 pontus-agentd
 1397 root      20   0  177524  17204  16804 S   0.3  0.2   3:47.13 systemd-journal
22590 nobody    20   0  186484  11952    992 S   0.0  0.1   2:55.52 tengine
22589 nobody    20   0  202868  11948    992 S   0.0  0.1   2:51.40 tengine
22593 nobody    20   0  153716  11948    984 S   0.0  0.1   2:55.76 tengine
22592 nobody    20   0  153716  11924    992 S   0.0  0.1   2:54.57 tengine
 2554 root      20   0  741512  11836   1520 S   0.0  0.1   2:53.64 rsyslogd
 8697 root      20   0  937908  11564   3208 S   0.3  0.1  48:21.40 SOURCE_PREDEFIN
22178 root      20   0  120500   9480    364 S   0.0  0.1   2:42.72 supervisord
 3092 root      20   0 1346740   9148   1952 S   0.0  0.1   3:06.77 docker-containe
19207 root      20   0  156720   8524   7188 S   0.0  0.1   0:00.07 sshd
 3896 root      20   0  134880   8416     92 S   0.0  0.1  29:18.67 pdi-daemonset
 2689 root      20   0  288272   6256   4976 S   0.0  0.1   0:36.34 asar-output
 2868 root      10 -10  148600   4508      0 S   1.3  0.1 183:32.03 AliYunDun
19256 root      20   0  164088   4280   3524 R   0.3  0.1   0:00.14 top
 8410 root      20   0  381668   3804      0 S   0.0  0.0   2:05.38 master-pontus-a
19210 root      20   0  117516   3156   2836 S   0.0  0.0   0:00.00 bash
    1 root      20   0   54108   3152   1556 S   0.0  0.0   6:49.16 systemd
19749 1100      20   0   16100   3104   2776 S   0.0  0.0   0:00.03 diver-monitor.s
 3049 root      20   0  377600   2568      0 S   0.0  0.0   0:46.58 alilocal
 8916 root      20   0   14848   2448      0 S   0.0  0.0   0:38.22 collectAgent
 3042 root      20   0   14616   2292      0 S   0.0  0.0   0:44.75 tmpfs
 3035 root      20   0  436644   1884      0 S   0.0  0.0   0:27.10 alinet
 2475 dbus      20   0   60452   1644    852 S   0.0  0.0   3:09.89 dbus-daemon
 2592 root      20   0  114936   1584    544 S   0.0  0.0   0:13.87 sshd
 3104 root      20   0   21976   1548      0 S   0.0  0.0   5:42.98 images_clean
22209 root      20   0   29096   1464    788 S   0.0  0.0   0:15.99 crond
 2785 root      20   0   34600   1452      0 S   0.0  0.0   6:19.25 AliYunDunUpdate
22587 root      20   0   90228   1400      0 S   0.0  0.0   0:00.00 tengine
```

从 monitorprod-1 实例内存使用趋势图（见下图）可以看到，monitorprod-1 内存使用稳定，不是因其而引起的 OOM。



monitorprod-1 实例所在物理机上各个进程内存使用情况如下图所示，其中 systemd 内存使用异常。

```
Tasks: 157 total,   1 running, 113 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.6 us,  0.0 sy,  0.0 ni, 98.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  7971208 total,  4523964 free,  2124460 used,  1322784 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  4799440 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0 1768724   1.5g   3152 S   0.0 19.8  37:14.41 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:02.83 kthreadd
    3 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:0H-kb
    8 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_wq
    9 root      20   0       0      0      0 S   0.0  0.0   2:49.22 ksoftirqd/0
   10 root      20   0       0      0      0 I   0.0  0.0  68:04.69 rcu_sched
   11 root      20   0       0      0      0 I   0.0  0.0   0:00.00 rcu_bh
   12 root      rt   0       0      0      0 S   0.0  0.0   1:44.94 migration/0
   14 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/0
   15 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/1
   16 root      rt   0       0      0      0 S   0.0  0.0   1:54.14 migration/1
   17 root      20   0       0      0      0 S   0.0  0.0   3:01.26 ksoftirqd/1
   19 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/1:0H-kb
   20 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/2
   21 root      rt   0       0      0      0 S   0.0  0.0   1:39.78 migration/2
   22 root      20   0       0      0      0 S   0.0  0.0   6:27.30 ksoftirqd/2
   24 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/2:0H-kb
   25 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/3
   26 root      rt   0       0      0      0 S   0.0  0.0   1:47.39 migration/3
   27 root      20   0       0      0      0 S   0.0  0.0   1:06.58 ksoftirqd/3
   29 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/3:0H-kb
   30 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kdevtmpfs
   31 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 netns
   32 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kauditd
   33 root      20   0       0      0      0 S   0.0  0.0   0:03.70 khungtaskd
   34 root      20   0       0      0      0 S   0.0  0.0   0:01.27 oom_reaper
   35 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 writeback
   36 root      20   0       0      0      0 S   0.0  0.0   0:00.42 kcompactd0
   37 root      25   5       0      0      0 S   0.0  0.0   0:00.00 ksmd
   39 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 crypto
   40 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kblockd
   45 root      20   0       0      0      0 S   0.0  0.0 371:58.42 kswapd0
   46 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/u9:0
   69 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kthrotld
[root@iZi0e05yjt3ypjxqlqcyquZ ~]#
```

【解决方法】

- monitorprod 纵向扩容。让 monitorprod 独享 8G 内存，而非共享。
- systemd 此类问题在域外处理难度起来较高，纵向扩容在一定程度上可以预防类似问题引入的故障。

4.2.2. pontusconsole 宕机

【现象】

BOS 页面可以打开，但是监控数据加载失败，页面可以显示图表，但是里面没有数据。

【可能原因】

- Java 内存溢出导致应用宕机
- 组件的 bug
- 底层 PaaS 异常调度

【解决方法】

1. 登录云游页面，找到 pontusconsole 应用的容器，检查容器状态是否健康。

2. 若容器状态异常，可直接通过云游发起容器重启操作。
3. 若容器可以登录，通过 `ps -ef|grep java|grep -v xflush` ，检查容器 Java 进程是否存在。
4. 若 Java 进程不存在则查看 `/home/admin/logs` 下是否有 .Hprof 结尾的文件，若有此文件，则保存此文件，然后参考第 2 步重启服务。
5. 若 Java 进程存在，可以执行 `jmap -heap <pid>` 命令，查看各内存区域的设置；执行 `jmap-histo:live <pid>` 命令，查看每个 class 的实例数目、内存使用率和类全名；保存收集到的 JVM 相关数据、信息，联系技术支持人员排查，然后参考第 2 步重启服务。

【验证方法】

页面是否可以正常打开，并正常加载数据。

4.2.3. monitorgateway 宕机

【现象】

BOS 页面可以打开，监控数据加载正常，但是链路追踪中无法展示链路数据。

【可能原因】

- Java 内存溢出导致应用宕机
- 组件的 bug
- 底层 PaaS 异常调度

【解决方法】

1. 登录云游页面，找到 monitorgateway 应用的容器，检查容器状态是否健康。
2. 若容器状态异常，可直接通过云游发起容器重启操作。
3. 若容器可以登录，通过 `ps -ef|grep java|grep -v xflush` ，检查容器 Java 进程是否存在。
4. 若 Java 进程不存在则查看 `/home/admin/logs` 下是否有 .Hprof 结尾的文件，若有此文件，则保存此文件，然后参考第 2 步重启服务。
5. 若 Java 进程存在，可以执行 `jmap -heap <pid>` 命令，查看各内存区域的设置；执行 `jmap-histo:live <pid>` 命令，查看每个 class 的实例数目、内存使用率和类全名；保存收集到的 JVM 相关数据、信息，联系技术支持人员排查，然后参考第 2 步重启服务。

【验证方法】

页面是否可以正常打开，链路数据可以正常访问。

4.2.4. bosalarm 宕机

【现象】

BOS 页面可以打开，功能可以正常使用，但是收不到报警。

【可能原因】

- Java 内存溢出导致应用宕机

- 组件的 bug
- 底层 PaaS 异常调度

【解决方法】

1. 登录云游页面，找到 bosalarm 应用的容器，检查容器状态是否健康。
2. 若容器状态异常，可直接通过云游发起容器重启操作。
3. 若容器可以登录，通过 `ps -ef|grep java|grep -v xflush` ，检查容器 Java 进程是否存在。
4. 若 Java 进程不存在则查看 `/home/admin/logs` 下是否有 .Hprof 结尾的文件，若有此文件，则保存此文件，然后参考第 2 步重启服务。
5. 若Java 进程存在，可以执行 `jmap -heap <pid>` 命令，查看各内存区域的设置；执行 `jmap-histo:live <pid>` 命令，查看每个 class 的实例数目、内存使用率 和类全名；保存收集到的 JVM 相关数据、信息，联系技术支持人员排查，然后参考第 2 步重启服务。

【验证方法】

可以正常收到报警。

4.2.5. agentregistry 宕机

【现象】

BOS 页面可以打开，计算配置无法下发，无计算结果，图表显示数据为空。

【可能原因】

- Java 内存溢出导致应用宕机
- 组件的 bug
- 底层 PaaS 异常调度

【解决方法】

1. 登录云游页面，找到 agentregistry 应用的容器，检查容器状态是否健康。
2. 若容器状态异常，可直接通过云游发起容器重启操作。
3. 若容器可以登录，通过 `ps -ef|grep java|grep -v xflush` ，检查容器 Java 进程是否存在。
4. 若 Java 进程不存在则查看 `/home/admin/logs` 下是否有 .Hprof 结尾的文件，若有此文件，则保存此文件，然后参考第 2 步重启服务。
5. 若 Java 进程存在，可以执行 `jmap -heap <pid>` 命令，查看各内存区域的设置；执行 `jmap-histo:live <pid>` 命令，查看每个 class 的实例数目、内存使用率 和类全名；保存收集到的 JVM 相关数据、信息，联系技术支持人员排查，然后参考第 2 步重启服务。

【验证方法】

RMS 可以正常使用，且数据可以正常展示。

4.3. 断电恢复

BOS 所涉及的组件都是通过容器方式部署，断电重启之后，容器会自动恢复。如果服务没有恢复，可以到云游控制台上逐个重启 BOS 相关应用，不涉及启动顺序。

4.4. 网络链接失败

网络链接失败，可能导致 BOS 相关服务不可用，需要对底层 PaaS 进行排查。

4.5. 数据丢失

BOS 的数据存储仅依赖关系型数据库（RDS 或 OceanBase）和 CeresDB。其中 RDS 或 OceanBase 自身保证数据完备性，存储的数据分两部分：

- **配置型数据**：一般不做变更，不会丢失数据。如果数据丢失，需要重新配置。
- **元数据**：丢失之后，5 分钟会自动重新同步，自动恢复。

CeresDB 中存储的是计算结果，即计算所得的监控时序数据。CeresDB 自身可以做主备同步，达到高可用。如果数据丢失，不影响监控整体使用，只是无法查询历史上丢失的记录。

5.日常运维

5.1. 监控

BOS 大部分内容可以通过自身服务完成监控。登录 BOS 页面，查找对应的应用信息，基于监控项查看应用监控信息。

异常说明

应用	分类	日志路径	告警阈值	采集间隔	告警间隔
monitorprod	错误日志	/home/admin/logs/monitorprod/common-error.log	最近 5 分钟内报错数超 100 条	1 分钟	3 分钟
monitorgateway	错误日志	/home/admin/logs/monitorgateway/common-error.log	最近 5 分钟内报错数超过 100 条	1 分钟	3 分钟
monitoropenapi	错误日志	/home/admin/logs/monitoropenapi/common-error.log	最近 5 分钟内报错数超过 100 条	1 分钟	3 分钟
pontusconsole	错误日志	/home/admin/pontusconsole/logs/error.log	最近 5 分钟内报错数超过 100 条	1 分钟	3 分钟
pontusvessel	错误日志	/home/admin/logs/vessel	最近 5 分钟内报错数超过 100 条	1 分钟	3 分钟
agentregistry	错误日志	/home/admin/gaearegistry/logs/error.log	最近 5 分钟内报错数超过 100 条	1 分钟	3 分钟
globalscheduler	错误日志	/home/admin/pontuscenter/logs/pontus_error.log	最近 5 分钟内报错数超过 100 条	1 分钟	3 分钟

日志说明

日志是排查 BOS 问题的关键措施和手段，可以对日志进行监控以便及时发现运行问题和故障。各个应用的日志根目录为：

应用	日志路径
monitorprod	/home/admin/logs/monitorprod/
monitorgateway	/home/admin/logs/monitorgateway
monitoropenapi	/home/admin/logs/monitoropenapi
pontusconsole	/home/admin/pontusconsole/logs
pontusvessel	/home/admin/logs/vessel
agentregistry	/home/admin/gaearegistry/logs
globalscheduler	/home/admin/pontuscenter/logs

设置监控项

场景	监控应用	告警项	告警阈值区间
	dimservice	CPU 使用率	大于 90%
		内存使用率	大于 90%
		磁盘使用率	大于 90%
		端口监控（8080）	不通
	globalscheduler	CPU 使用率	大于 90%
		内存使用率	大于 90%

组件监控		磁盘使用率	大于 90%
		端口监控（5061）	不通
	bosalarm	CPU 使用率	大于 90%
		内存使用率	大于 90%
		磁盘使用率	大于 90%
		端口监控（14876）	不通
	pontusconsole	CPU 使用率	大于 90%
		内存使用率	大于 90%
		磁盘使用率	大于 90%
		端口监控（8080）	不通
	pontusspark	CPU 使用率	大于 90%
		内存使用率	大于 90%
		磁盘使用率	大于 90%
		端口监控（5062）	不通
	pontusvessel	CPU 使用率	大于 90%
		内存使用率	大于 90%
		磁盘使用率	大于 90%
		端口监控（19320）	不通

agentregistry	CPU 使用率	大于 90%
	内存使用率	大于 90%
	磁盘使用率	大于 90%
	端口监控（9025）	不通
monitoropenapi	CPU 使用率	大于 90%
	内存使用率	大于 90%
	磁盘使用率	大于 90%
	端口监控（8080）	不通
monitorprod	CPU 使用率	大于 90%
	内存使用率	大于 90%
	磁盘使用率	大于90%
	端口监控（8080）	不通
monitorgateway	CPU 使用率	大于 90%
	内存使用率	大于 90%
	磁盘使用率	大于90%
	端口监控（7200）	不通
ceresdbtrace	CPU 使用率	大于 90%
	内存使用率	大于 90%

		磁盘使用率	大于 90%
		端口监控（5000）	不通
	ceresdbmr	CPU 使用率	大于 90%
		内存使用率	大于 90%
		磁盘使用率	大于 90%
		端口监控（5000）	不通

5.2. 巡检

BOS 的巡检通过高可用容灾平台进行，不区分业务巡检和组件巡检。业务巡检即组件巡检。

应用	分类	说明
dimservice	基础巡检-端口	检查 8080 端口是否正常
dimservice	基础巡检-进程	检查进程是否存活
globalscheduler	基础巡检-端口	检查 5061 端口是否正常
globalscheduler	基础巡检-进程	检查进程是否存活
bosalarm	基础巡检-端口	检查 14876 端口是否正常
bosalarm	基础巡检-进程	检查进程是否存活
pontusconsole	基础巡检-端口	检查 8080 端口是否正常
pontusconsole	基础巡检-进程	检查进程是否存活
pontusspark	基础巡检-端口	检查 5062 端口是否正常

pontusspark	基础巡检-进程	检查进程是否存活
pontusvessel	基础巡检-端口	检查 19320 端口是否正常
pontusvessel	基础巡检-进程	检查进程是否存活
agentregistry	基础巡检-端口	检查 9025 端口是否正常
agentregistry	基础巡检-进程	检查进程是否存活
monitoropenapi	基础巡检-端口	检查 8080 端口是否正常
monitoropenapi	基础巡检-进程	检查进程是否存活
monitorprod	基础巡检-端口	检查 8080 端口是否正常
monitorprod	基础巡检-进程	检查进程是否存活
monitorgateway	基础巡检-端口	检查 7200 端口是否正常
monitorgateway	基础巡检-进程	检查进程是否存活
ceresdbtrace	基础巡检-端口	检查 5000 端口是否正常
ceresdbtrace	基础巡检-进程	检查进程是否存活
ceresdbmr	基础巡检-端口	检查 5000 端口是否正常
ceresdbmr	基础巡检-进程	检查进程是否存活

6.常见故障处理

6.1. 技术栈相关异常

6.1.1. 监控无数据—之前有数据，从某一时间点数据消失

【场景描述】之前有监控数据，从某一时刻开始监控数据消失。

【产生原因】

可能的原因如下：

- 技术栈配置被修改（采集、计算、展现配置）
- 元数据存在问题
- 计算侧异常

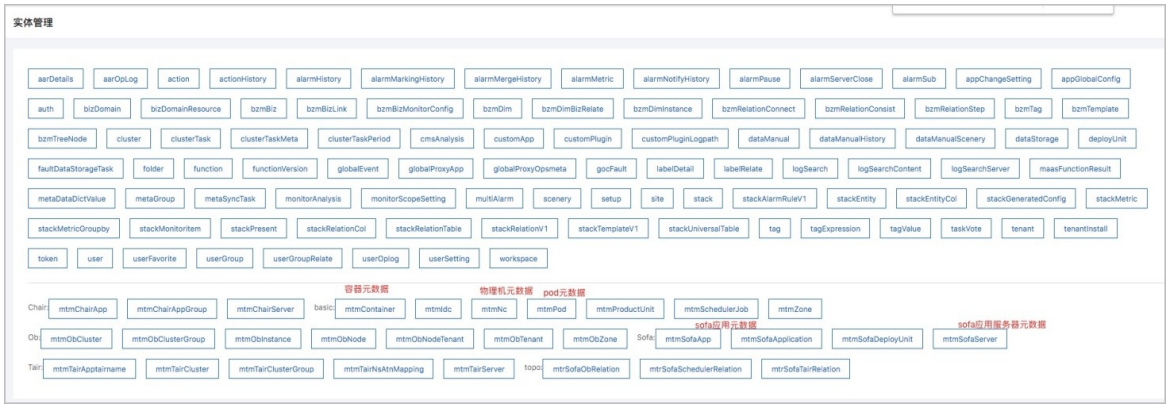
【解决方案】

1. 确认技术栈配置是否被修改。

   可以与正常的环境对比，通常一个环境的全局技术栈是不允许被修改的，一般不会发生这种情况。

2. 确认元数据是否存在问题。

   可以通过 `http://rms 地址/boss/manager` 页面查看元数据。



3. 确认计算侧异常。

   如何初步的确认计算异常：

   - 如果类似的其它资源也出现了无数据问题，且元数据不存在问题。
   - 如果出现大面积的无数据。
   - 如果上述两个检测都不存在问题。

6.1.2. 监控无数据—大面积无数据问题

【场景描述】站点大面积无监控数据。

【产生原因】

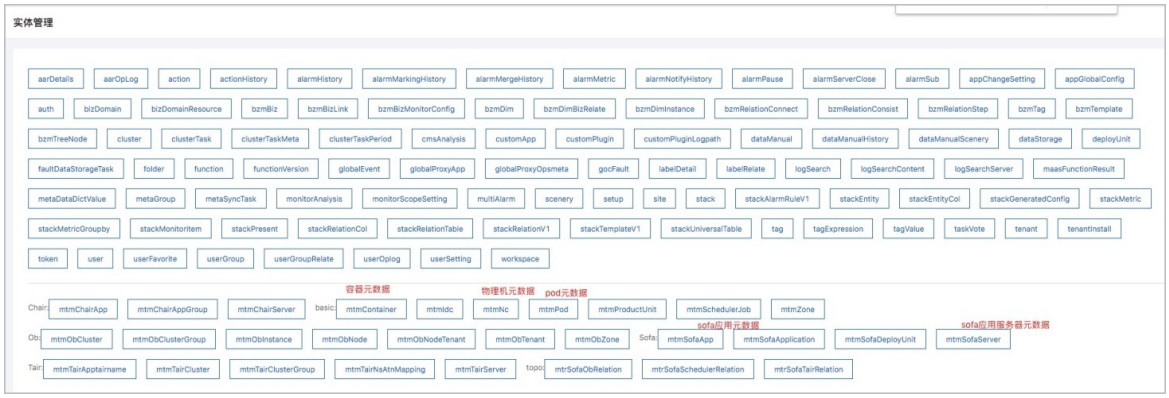
可能的原因如下：

- 元数据存在问题
- 计算侧异常

【解决方案】

1. 确认元数据是否存在问题。

   可以通过 `http://rms地址/boss/manager` 页面查看元数据。



2. 确认计算侧异常。

   如何初步的确认计算异常：

   - 如果类似的其它资源也出现了无数据问题，且元数据不存在问题。
   - 如果出现大面积的无数据。
   - 如果上述两个检测都不存在问题。

6.1.3. 监控无数据—云原生无数据

【场景描述】云原生无数据。

【产生原因】

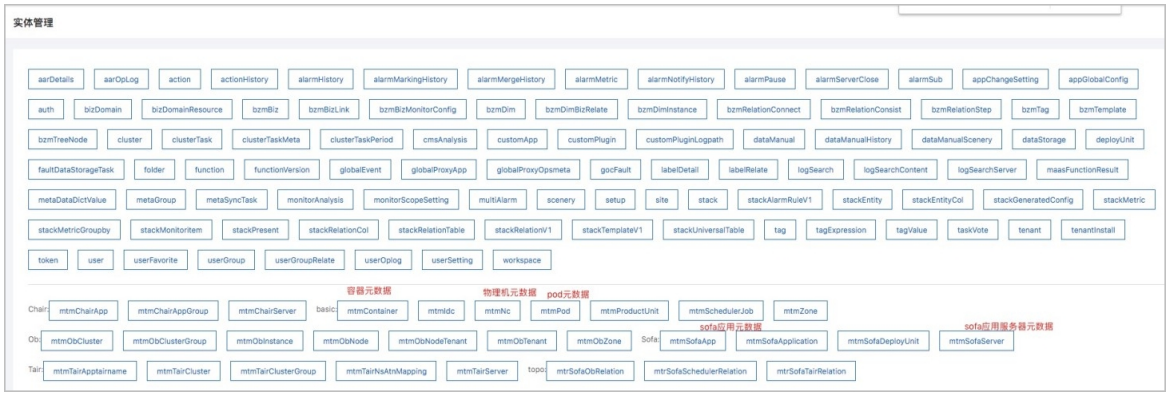
可能的原因如下：

- 元数据存在问题
- 采集存在问题

【解决方案】

1. 确认元数据是否存在问题。

   可以通过 `http://rms 地址/boss/manager` 页面查看元数据。



2. 确认采集是否存在问题。
   - 确认采集是否正常：参考计算侧排查文档。
   - 确认 kubelet 本身 metrics 是否存在问题：
     - 请求 kubelet 的 /metrics 接口查看数据：curl http://节点 IP:10255/metrics
     - 请求 kubelet 的 /metrics/casvisor 接口查看数据: curl http://节点 IP:10255/metrics/casvisor

6.1.4. 监控无数据—个别应用无数据问题

【场景描述】仅少数应用没有数据。

【产生原因】

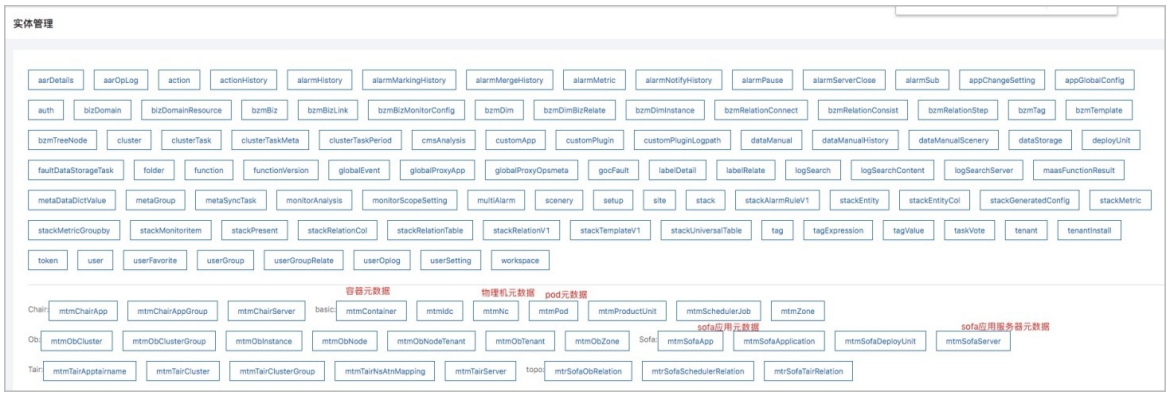
可能的原因如下：

- 元数据存在问题
- 应用本身存在问题

【解决方案】

1. 确认元数据是否存在问题。

   可以通过 `http://rms地址/boss/manager` 页面查看元数据。



2. 确认应用本身是否存在问题。

   日志类监控问题，应用本身不存在对应的日志。

6.1.5. 监控无数据—新建技术栈监控项

【场景描述】新建一个技术栈监控项，但没有监控数据。

【产生原因】

- 技术栈监控项配置存在问题
- 元数据存在问题
- 计算存在问题

【解决方案】

查看后台元数据是否存在。

6.2. Agent 异常

【场景描述】Agent 出现异常。

【产生原因】

- node 节点 Agent 未部署
- Agent 进程丢失

【解决方案】

常用命令如下：

```
# 查看监控agent部署情况
kubectl --kubeconfig=tingxi-cluster.kubeconfig get ds -A

# 连接至node节点查看 error 日志
ssh -i /root/cafe/cafecluster.key 192.168.0.222
tailf /home/agent/logs/error.log

# 查看agent的pod的状态是否正常
kubectl --kubeconfig=tingxi-cluster.kubeconfig get po -n pontusagent

# 查看agent的pod的日志
kubectl --kubeconfig=tingxi-cluster.kubeconfig -n pontusagent logs pontusagent-5txzz

# 采集日志路径
kubectl --kubeconfig=tingxi-cluster.kubeconfig exec -it fanglatest-g42l7-4fjhh bash
/home/admin/logs/helloworld-app/common-default.log
```

【验证方法】BOS 控制台上能正确的自定义监控，并且有数据。

6.3. 产品层异常

6.3.1. 元数据初始化—技术栈导入报错

【场景描述】

登录到 monitorprod 容器，查看 /home/admin/logs/monitorprod/site-install-info.log 日志，出现 `[http-nio-8080-exec-4][T:] - import stack error, CRITICAL!`

具体报错如下：

```
 at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:16
6)
 at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:198)
 at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:96)
 at org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:493)
 at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:140)
 at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:81)
 at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:87)
 at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:342)
 at org.apache.coyote.http11.Http11Processor.service(Http11Processor.java:800)
 at org.apache.coyote.AbstractProcessorLight.process(AbstractProcessorLight.java:66)
 at org.apache.coyote.AbstractProtocol$ConnectionHandler.process(AbstractProtocol.java:806)
 at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.java:1498)
 at org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProcessorBase.java:49)
 at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
 at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
 at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
 at java.lang.Thread.run(Thread.java:748)
2020-12-03 16:12:43,042 [http-nio-8080-exec-4][T:] - import stack error, CRITICAL!
java.lang.NullPointerException
 at com.alipay.monitorprod.web.siteboot.stack.StackImportFacadeImpl.handleStack(StackImport
FacadeImpl.java:88)
        at com.alipay.monitorprod.web.siteboot.stack.StackImportFacadeImpl.stackExport(Stac
kImportFacadeImpl.java:70)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.jav
a:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at com.alipay.sofa.web.mvc.core.servlet.handler.support.HandlerMethodInvoker.invoke
HandlerMethod(HandlerMethodInvoker.java:154)
        at com.alipay.sofa.web.mvc.core.servlet.handler.support.AbstractMethodHandlerAdapte
r.invokeHandlerMethod(AbstractMethodHandlerAdapter.java:408)
        at com.alipay.sofa.web.mvc.core.servlet.handler.support.AbstractMethodHandlerAdapte
r.handle(AbstractMethodHandlerAdapter.java:388)
        at com.alipay.web.mvc.car.CarHandlerAdapter.handle(CarHandlerAdapter.java:83)
        at org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.j
ava:967)
        at com.alipay.sofa.runtime.web.smvc.servlet.CarDispatcherServlet.doDispatch(CarDisp
atcherServlet.java:194)
        at org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.ja
va:901)
        at com.alipay.sofa.runtime.web.smvc.servlet.CarDispatcherServlet.doService(CarDispa
tcherServlet.java:164)
        at org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet
.java:970)
        at org.springframework.web.servlet.FrameworkServlet.doPost(FrameworkServlet.java:87
2)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:707)
        at org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:8
46)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:790)
```

【产生原因】导入了错误的技术栈。

【解决方案】

使用正确的技术栈。

6.3.2. 健康检查失败

【场景描述】健康检查失败。

【产生原因】/

【解决方案】

运行 `curl localhost:8080/checkService` 。

如果提示端口未启动，则大概率是 spring 启动失败。可以在 /home/admin/logs/spring/spring.log 里查看对应的日志。

6.3.3. 应用-系统指标-无数据

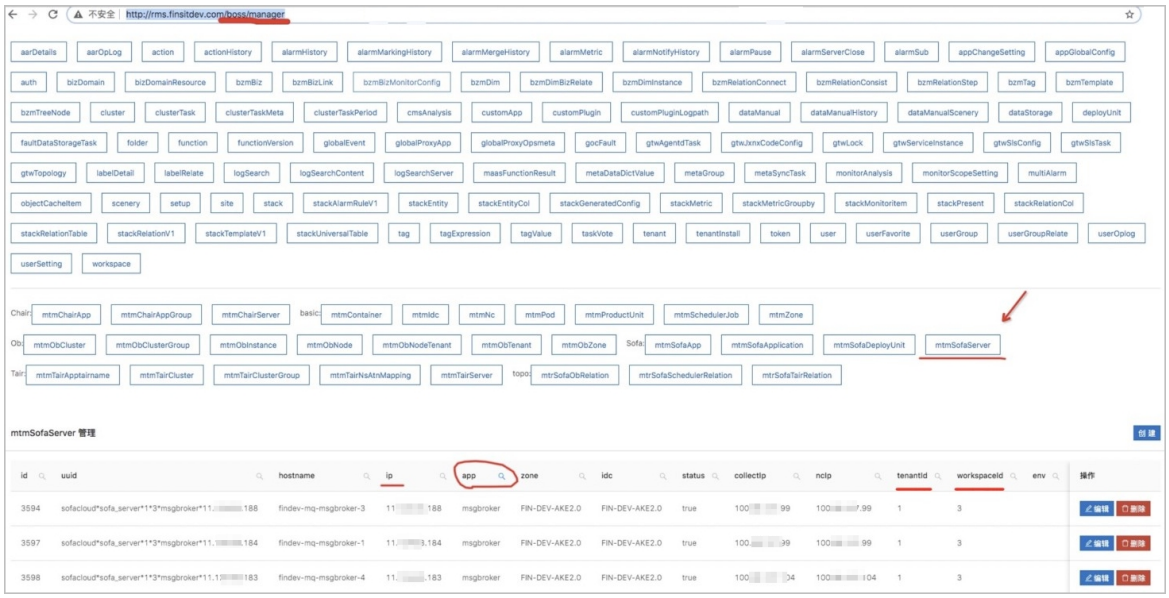
【场景描述】个别应用无数据。

【产生原因】

仅个别应用（而非全部应用）无数据表明，计算配置、计算配置同步、数据采集、计算、存储整个链路没有问题。问题很可能出现在元数据和 Agent。

【解决方案】

1. 查看元数据中该应用是否存在实例。
2. 打开 `/boss/manager` 页面。
3. 以 msgbroker 应用为例，查看 sofaserver 表，查看该 workspace 下是否存在对应实例，如下图所示。



   - 如果存在实例，排查采集是否有问题。具体步骤如下：

a. 打开 debug 日志。

```
curl localhost:19510/log/debug/start
```

b. 查询内存数据。

```
curl 127.0.0.1:19510/getCacheInfo?target=${ip}\&table=${table_name}\&getFinal=1
```

c. 查看数据拉取情况。

```
grep 'do process general data' -a /home/agent/logs/debug.log | grep ${ip} | grep sofacloud@@system@@DEFAULT | tail -n 5
```

d. 查看任务执行情况

```
grep 'exec pipeline job' /home/agent/logs/info.log | grep ${ip}  | grep sofacloud@@jvmgc@@DEFAULT@@3 | tail -n 5
```

e. 执行 asar。

```
asar --cgroup_cpu
```

- 如果不存在实例，则无数据属于正常现象，说明应用未实际发布。

6.4. 监控配置异常

【场景描述】无法选择日志文件。

【产生原因】

- 元数据有问题。
- Agent 安装有问题。

【解决方案】

1. 首先确认此应用的元数据内是否存在对应的 opsmeta。
2. 如果存在对应的 opsmeta，确认此应用的 server 的 `/home/admin/logs` 目录下是否有日志文件。

3. 如果手动指定了根目录，目前根目录前缀仅允许 `/home/admin` 和 `/home/logs` 。
4. 如果没有，那么打开浏览器控制台，查看加载日志目录接口 (agent/browser) 是否有报错：
   - CEM[ConnectTimeoutException] 报错：
     - 对应机器 Agent 未安装：物理机场景为对应机器的 Agent 未安装；容器场景为 deamonset 未 apply。
     - 对应机器的 Agent 连不通（端口为 19310）。
   - 如果是其他报错，需要查看具体的报错内容。

【验证方法】可以选择到日志文件。

6.5. 监控数据展现异常

6.5.1. 监控数据为一排查

【场景描述】监控数据异常。

【产生原因】

- 产品层获取 pontusConsole 接口出现异常。
- 计算层有问题。

【解决方案】

1. 先排查 universalQuery 接口，查看返回的 message 是否有问题。
2. 如果 universalQuery 接口返回正常，排查计算层是否存在问题。

【验证方法】监控数据恢复正常。

6.5.2. 监控数据为空排查

【场景描述】监控数据为空。

【产生原因】产品层无法调通 pontus 的 API 接口。

【解决方案】

1. 查看 pontus 开头的日志中是否有明显错误。

2. 如果问题，则查看 env 里的 RMS_PONTUS_MODEL_INNER_SLB 的值，是否带上了 http 和 8080（比如 `http://pontus-pontusconsole-service:8080` ），并试一下该值是否能通。

【验证方法】监控数据恢复正常。

6.5.3. 监控数据为 0 排查

【场景描述】监控数据为 0。

【产生原因】

- 监控配置错误。
- 计算或者采集配置未下发到 pontus 计算层。

【解决方案】

1. 检查是否存在日志，监控配置是否正确，是否均为 0。
2. 查看 pontus 开头的日志内是否存在明显的错误。
3. 检查是否已将采集和计算的配置下发至 pontusconsole 的数据库。
   - **采集**

     ```
     select * from pontus2_collect_conf where name like "SM@@%" limit 1\G;

     select * from pontus2_collect_conf where id = '852000192'\G;
     ```

   - **计算**

     ```
     select * from pontus2_transform limit 1 \G
     ```

【验证方法】监控数据恢复正常。

6.6. 监控告警异常

6.6.1. 告警规则已配置但无告警排查

【场景描述】告警规则已配置，但无告警。

【产生原因】/

【解决方案】

1. 确认告警规则配置是否正确。
2. 确认告警初始化已配置。详情请参见《部署指南》告警初始化章节。
3. 查看 alarmconfig 库的 alarm_rule 表，是否有数据。

   如果没数据，查看 monitorprod 应用的 `/home/admin/logs/monitorprod/custom-plugin-sync.log` 。判断 monitorprod 到 monitoropenapi 的 alarm 配置同步是否有问题。

   如果有问题，且提示网络问题或地址访问问题，则需要查看 monitorprod 的 `/home/admin/release/run/target/boot/logs/http-util.log` 这一日志中的报错信息。

4. 查看 bosalarm 组件的 `/home/admin/AlarmExecutorLocal/logs` 下的 pontus_error.log，是否有异常日志。
5. 若无异常日志，可能是 bosalarm 将告警信息推送至产品层时出现问题。

   此时，可以查看 bosalarm 的 `/home/admin/AlarmExecutorLocal/conf/config.properties` 中的 neo.gateway.endpoint 和 neo.gateway.endpoint.url.fmt，检查 endpoint 是否能够连通且 fmt 是否已完成配置。

6.7. 元数据异常

6.7.1. 元数据同步问题

【场景描述】元数据同步问题。

【产生原因】

- 元数据接口调用失败。
- 元数据接口返回不准确（比如paas界面上有应用，但是接口返回的数据里却没有）。

【解决方案】

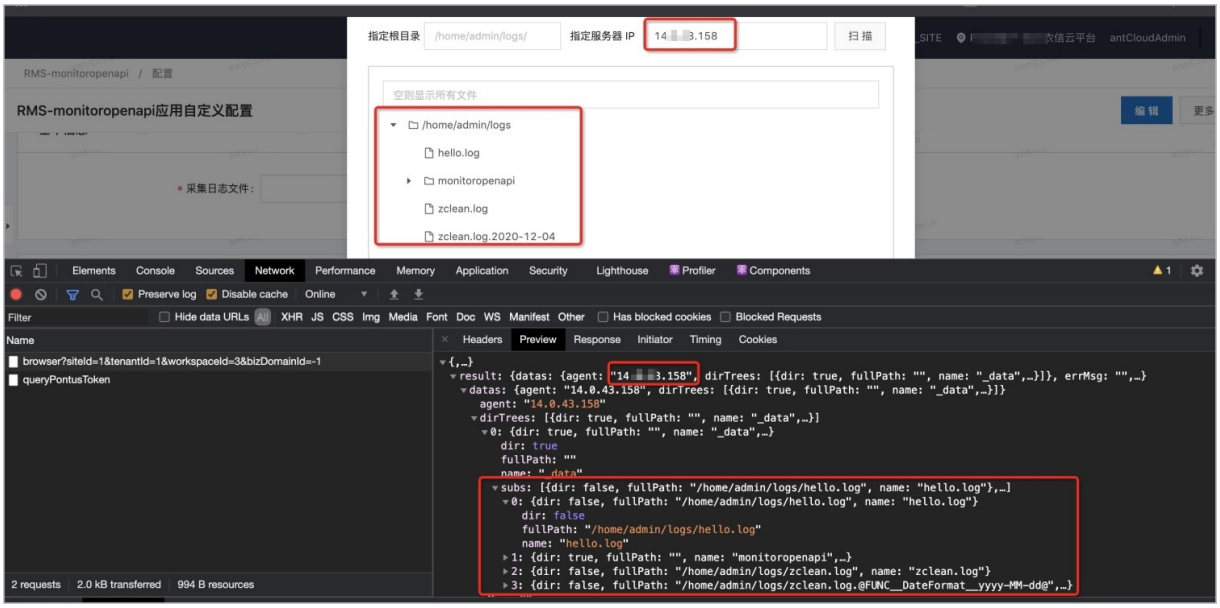
1. 查看 globalproxy-error.log。
2. 查看 cmdb-unimeta-error.log。
3. 若无 error，则查看 cmdb-unimeta-default.log。
4. 若仍无 error，则查看 task-info.log，检查是否在生成任务时发生失败。

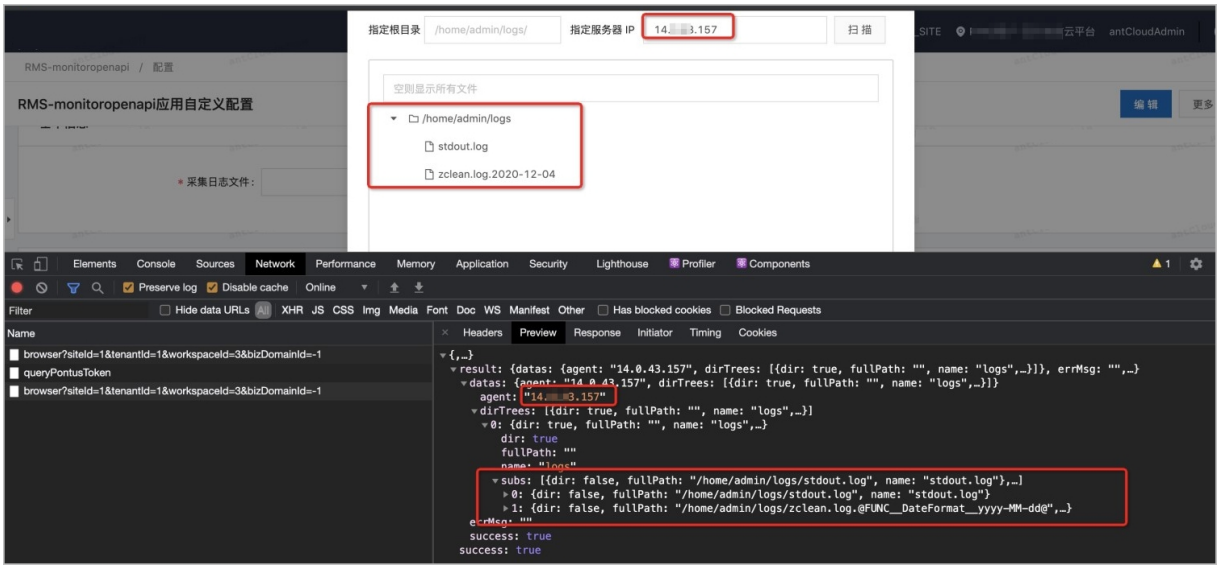
【验证方法】元数据同步正确。

6.8. 计算层异常

6.8.1. 扫描日志路径结果错误

【场景描述】扫描日志路径结果错误。


指定根目录 /home/admin/logs/
指定服务器 IP 14.[redacted]3.158
扫描
RMS-monitoropenapi / 配置
RMS-monitoropenapi应用自定义配置
编辑
采集日志文件:
空则显示所有文件
/home/admin/logs
hello.log
monitoropenapi
zclean.log
zclean.log.2020-12-04
Elements Console Sources Network Performance Memory Application Security Lighthouse Profiler Components
Preserve log Disable cache Online
Filter Hide data URLs XHR JS CSS Img Media Font Doc WS Manifest Other Has blocked cookies Blocked Requests
Name
browser?siteId=1&tenantId=1&workspaceId=3&bizDomainId=-1
queryPontusToken
Headers Preview Response Initiator Timing Cookies
result: {datas: {agent: "14.[redacted]3.158", dirTrees: [{dir: true, fullPath: "", name: "_data",…}]}, errMsg: "",…}
datas: {agent: "14.0.43.158", dirTrees: [{dir: true, fullPath: "", name: "_data",…}]}
agent: "14.0.43.158"
dirTrees: [{dir: true, fullPath: "", name: "_data",…}]
0: {dir: true, fullPath: "", name: "_data",…}
dir: true
fullPath: ""
name: "_data"
subs: [{dir: false, fullPath: "/home/admin/logs/hello.log", name: "hello.log"},…]
0: {dir: false, fullPath: "/home/admin/logs/hello.log", name: "hello.log"}
dir: false
fullPath: "/home/admin/logs/hello.log"
name: "hello.log"
1: {dir: true, fullPath: "", name: "monitoropenapi",…}
2: {dir: false, fullPath: "/home/admin/logs/zclean.log", name: "zclean.log"}
3: {dir: false, fullPath: "/home/admin/logs/zclean.log.@FUNC__DateFormat__yyyy-MM-dd@",…}
2 requests 2.0 kB transferred 994 B resources


指定根目录 /home/admin/logs/
指定服务器 IP 14.[redacted]3.157
扫描
RMS-monitoropenapi / 配置
RMS-monitoropenapi应用自定义配置
编辑
采集日志文件:
空则显示所有文件
/home/admin/logs
stdout.log
zclean.log.2020-12-04
Elements Console Sources Network Performance Memory Application Security Lighthouse Profiler Components
Preserve log Disable cache Online
Filter Hide data URLs XHR JS CSS Img Media Font Doc WS Manifest Other Has blocked cookies Blocked Requests
Name
browser?siteId=1&tenantId=1&workspaceId=3&bizDomainId=-1
queryPontusToken
browser?siteId=1&tenantId=1&workspaceId=3&bizDomainId=-1
Headers Preview Response Initiator Timing Cookies
result: {datas: {agent: "14.0.43.157", dirTrees: [{dir: true, fullPath: "", name: "logs",…}]}, errMsg: "",…}
datas: {agent: "14.0.43.157", dirTrees: [{dir: true, fullPath: "", name: "logs",…}]}
agent: "14.[redacted]3.157"
dirTrees: [{dir: true, fullPath: "", name: "logs",…}]
0: {dir: true, fullPath: "", name: "logs",…}
dir: true
fullPath: ""
name: "logs"
subs: [{dir: false, fullPath: "/home/admin/logs/stdout.log", name: "stdout.log"},…]
0: {dir: false, fullPath: "/home/admin/logs/stdout.log", name: "stdout.log"}
1: {dir: false, fullPath: "/home/admin/logs/zclean.log.@FUNC__DateFormat__yyyy-MM-dd@",…}
errMsg: ""
success: true
success: true

```
sh-4.2#
sh-4.2#
sh-4.2# ls
hello.log  monitoropenapi  zclean.log  zclean.log.2020-12-04
sh-4.2# pwd
/home/admin/logs
sh-4.2# ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.5.1  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 02:42:c0:a8:05:01  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 14.[redacted]3.157  netmask 255.255.255.0  broadcast 14.0.43.255
        ether 00:16:3e:01:02:f4  txqueuelen 1000  (Ethernet)
        RX packets 231957767628  bytes 278281184245089 (253.0 TiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 107597240814  bytes 272082758588655 (247.4 TiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 47445489  bytes 21277152977 (19.8 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 47445489  bytes 21277152977 (19.8 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

sh-4.2#
```

【产生原因】/

【解决方案】

1. 登录节点，查看 info.log 日志。

   日志路径为 `/home/agent/log/info.log` 。

   ```
   grep 'process file browse, path change' /home/agent/log/info.log
   ```

2. 查看配置参数。

   ```
   curl localhost:19510/getAppConf
   ```

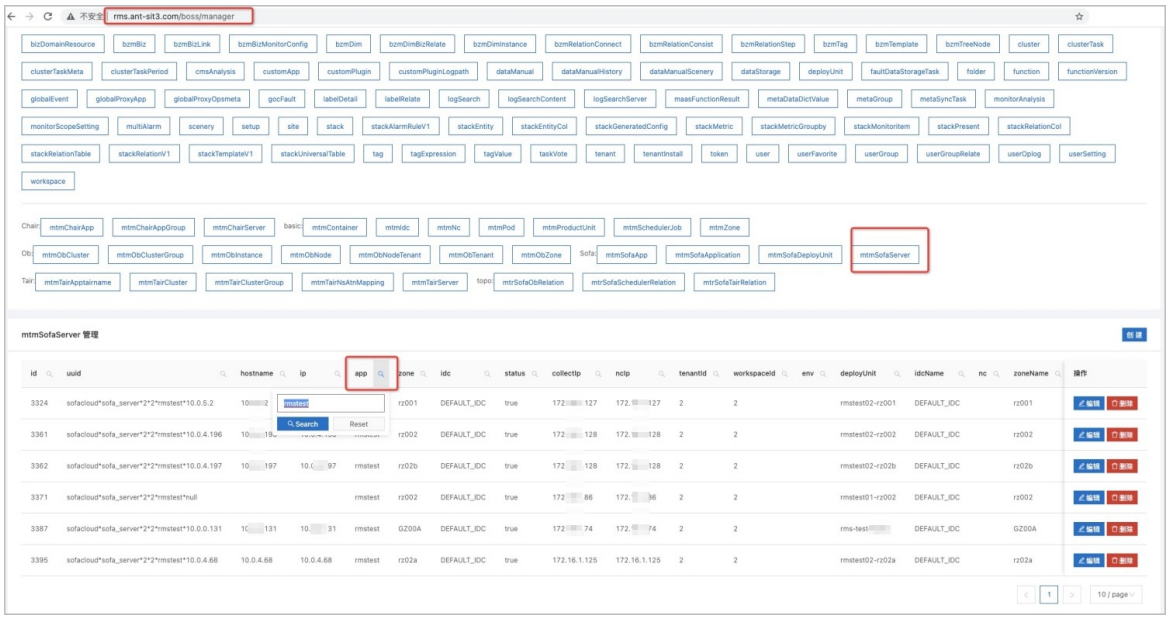
3. 查看 Agent 采集到的容器信息。

   ```
   curl 127.0.0.1:19510/conf/getContainerList
   ```

4. 查看节点 Pods。

   ```
   curl localhost:10255/pods
   ```

5. 查元数据：`监控域名/boss/manager` 。



6.8.2. 查询计算侧配置

【场景描述】/

【产生原因】/

【解决方案】

1. 登录 pontusconsole，获取日志连接串：

```
env | grep db
```

```
[root@pontus-pontusconsole-0 /home/admin]
# env | grep db
ceresdb_url=ceresdbmr-ceresdbng.intra.env88e.shuguang.com
db_pontusconfig_database=pontusconfig
ceresdb_port=5000
db_pontusconfig_url=pontusconfig.xdb.intra.env88e.shuguang.com:3867
db_pontusconfig_user=pontusconfig
ceresdb_access_key=
db_pontusconfig_password=oHFKdXIBYWMqnOC2
ceresdb_user=
jdbc_env=dev
```

2. 登录数据库。

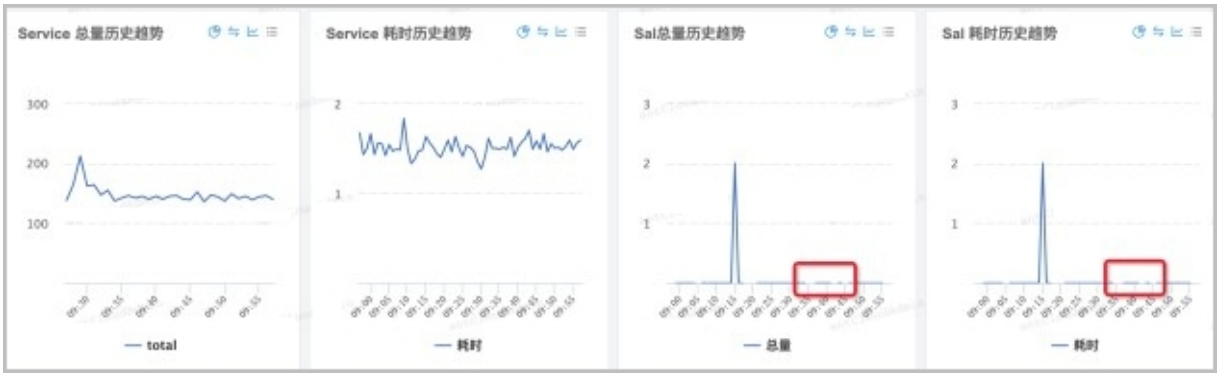
```
mysql -hpontusconfig.xdb.intra.env88e.shuguang.com -P3867-Dpontusconfig -upontusconfig
-poHFKdXIBYWMqnOC2
```

3. 查询 table。

```
select * from pontus2_transform where name like '%sofacloud@@ibmmqput@@queue%';
```

6.8.3. 监控数据断点

【场景描述】发现非零数据都是连续的，出现断点的位置值均为 0。



【产生原因】

可能的原因如下：

产品会根据计算的任务完成 (_spark_job_status_record) 情况进行补零操作。

【解决方案】

1. 登录 pontusconsole，查询对应时间点计算侧的任务完成状态。

```
curl -X POST localhost:8080/api/1.0.0/metrics/snap  -H 'Content-Type: application/json'
-d
'{"table":"_spark_job_status_record","start":1605430860000,"end":1605430860000,"conditi
ons":{"table":"sofacloud@@cal@@app@@DEFAULT@@1@@3"}}'
```

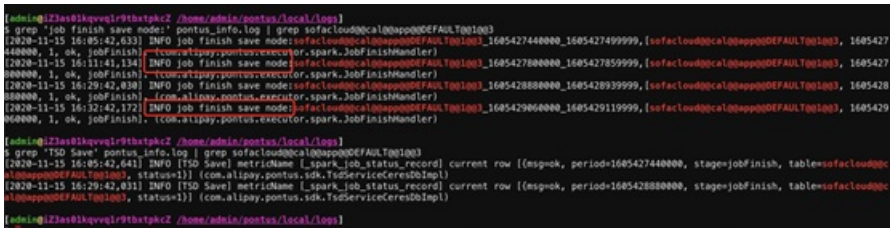
返回结果为空。

2. 登录 gs 查询任务执行情况。

```
grep sofacloud@@cal@@app@@DEFAULT@@1@@3_1605430860000
/home/admin/pontuscenter/logs/event_loop.log
```

任务执行是正常的，查到这里大概率没有任务执行状态 (_spark_job_status_record) 正常写入 ceresdb。

3. 登录 pontusspark 对应 table 发现没有写入的日志。



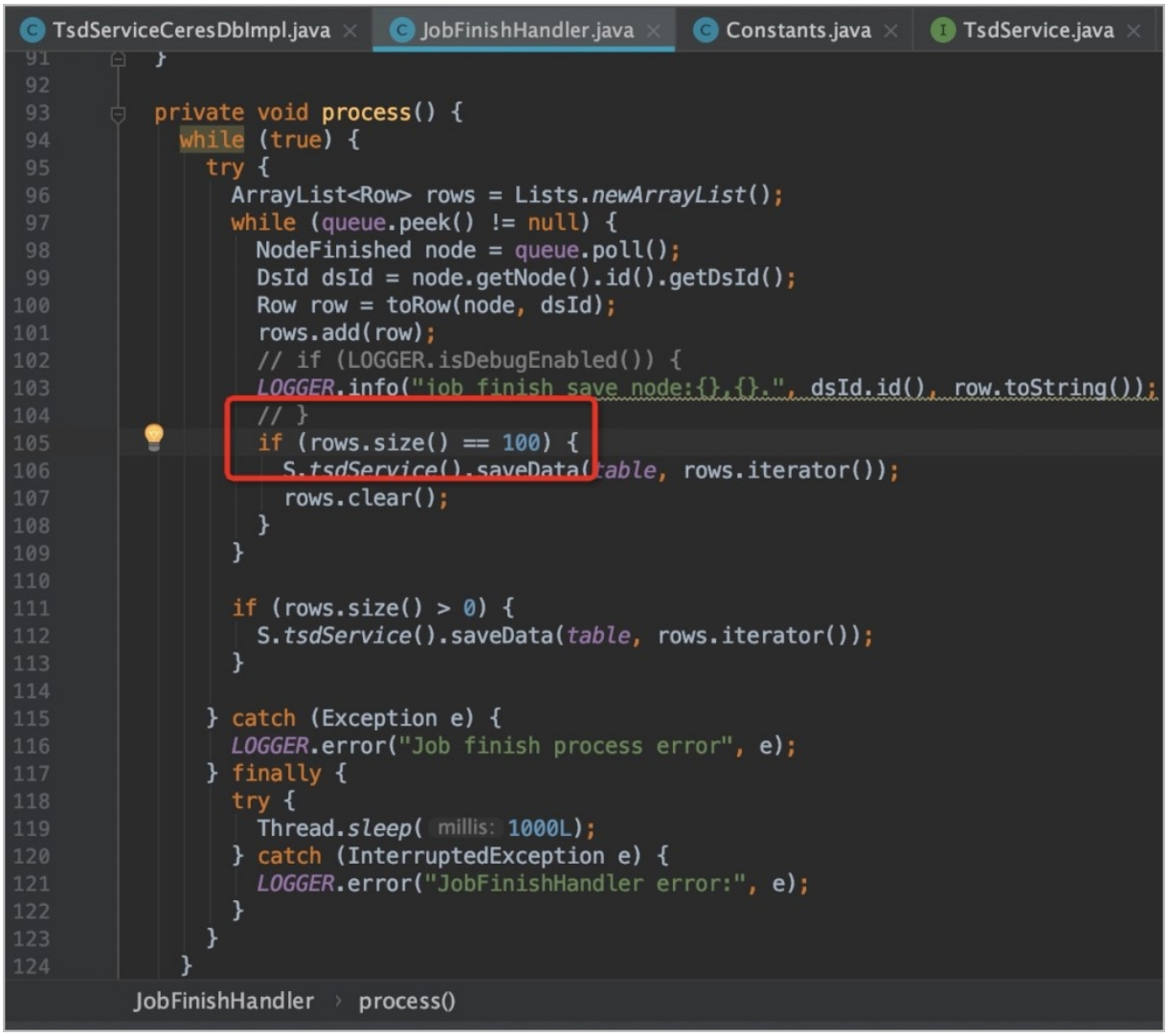
4. 查看 error 发现，有 NPE。

```
sh-4.2# tail -f pontus_error.log
[2020-11-15 18:02:25,611] ERROR Job finish process error (com.alipay.pontus.executor.spark.JobFinishHandler)
java.lang.ClassCastException: com.alipay.pontus.core.scheduler.eventLoop.event.NodeEvent$NodeDiscard cannot be cast to com.alipay.pontus.core.scheduler.eventLo
op.event.NodeEvent$NodeFailed
        at com.alipay.pontus.executor.spark.JobFinishHandler.toRow(JobFinishHandler.java:133)
        at com.alipay.pontus.executor.spark.JobFinishHandler.process(JobFinishHandler.java:100)
        at java.lang.Thread.run(Thread.java:748)
[2020-11-15 18:02:26,616] ERROR Job finish process error (com.alipay.pontus.executor.spark.JobFinishHandler)
java.lang.ClassCastException: com.alipay.pontus.core.scheduler.eventLoop.event.NodeEvent$NodeDiscard cannot be cast to com.alipay.pontus.core.scheduler.eventLo
op.event.NodeEvent$NodeFailed
        at com.alipay.pontus.executor.spark.JobFinishHandler.toRow(JobFinishHandler.java:133)
        at com.alipay.pontus.executor.spark.JobFinishHandler.process(JobFinishHandler.java:100)
        at java.lang.Thread.run(Thread.java:748)
^C
```

5. 查看任务执行状态的代码 `com.alipay.pontus.executor.spark.JobFinishHandler`，分析问题。

【补充说明】根因分析：

由于 _spark_job_status_record 是批量更新，脏配置 NPE 会导致同一批次都更新失败。




```
}

private void process() {
  while (true) {
    try {
      ArrayList<Row> rows = Lists.newArrayList();
      while (queue.peek() != null) {
        NodeFinished node = queue.poll();
        DsId dsId = node.getNode().id().getDsId();
        Row row = toRow(node, dsId);
        rows.add(row);
        // if (LOGGER.isDebugEnabled()) {
        LOGGER.info("job finish save node:{},{}.", dsId.id(), row.toString());
        // }
        if (rows.size() == 100) {
          S.tsdService().saveData(table, rows.iterator());
          rows.clear();
        }
      }

      if (rows.size() > 0) {
        S.tsdService().saveData(table, rows.iterator());
      }

    } catch (Exception e) {
      LOGGER.error("Job finish process error", e);
    } finally {
      try {
        Thread.sleep( millis: 1000L);
      } catch (InterruptedException e) {
        LOGGER.error("JobFinishHandler error:", e);
      }
    }
  }
```

修复建议：

```
private void process() {
  while (true) {
    try {
      ArrayList<Row> rows = Lists.newArrayList();
      while (queue.peek() != null) {
        NodeFinished node = queue.poll();
        DsId dsId = node.getNode().id().getDsId();
        Row row = toRow(node, dsId);
        rows.add(row);
        // if (LOGGER.isDebugEnabled()) {
        LOGGER.info("job finish save node:{},{}.", dsId.id(), row.toString());
        // }
        if (rows.size() == 1) {
          S.tsdService().saveData(table, rows.iterator());
          rows.clear();
        }
      }

      if (rows.size() > 0) {
        S.tsdService().saveData(table, rows.iterator());
      }

    } catch (Exception e) {
      LOGGER.error("Job finish process error", e);
    } finally {
      try {
        Thread.sleep( millis: 1000L);
      } catch (InterruptedException e) {
        LOGGER.error("JobFinishHandler error:", e);
      }
    }
  }
```

JobFinishHandler › process()

6.8.4. 所有监控数据缺失

【场景描述】所有的监控数据在一段时间内出现缺失情形

【产生原因】

初步推测是计算层（gs+spark）有故障。

【解决方案】

1. 检查gs任务执行状态。
    i. 检查进程是否存在。执行以下命令：

    ```
    ps aux | grep java
    ```

    如果不存在，可自主启动，与开发人员联系以便排查问题。

    启动路径为：`/home/admin/bin/deploy.sh` 。

ii. 在页面单击任一个查询，例如：

复制对应的 dsId，检查 event loop 日志：

`less /home/admin/pontuscenter/logs/event_loop.log`，查看该配置是否正常执行。

2. 检查 spark 任务执行状态。

   执行以下命令，检查进程是否存在：

```
ps aux | grep java
```

6.8.5. 日志监控正常但系统指标缺失

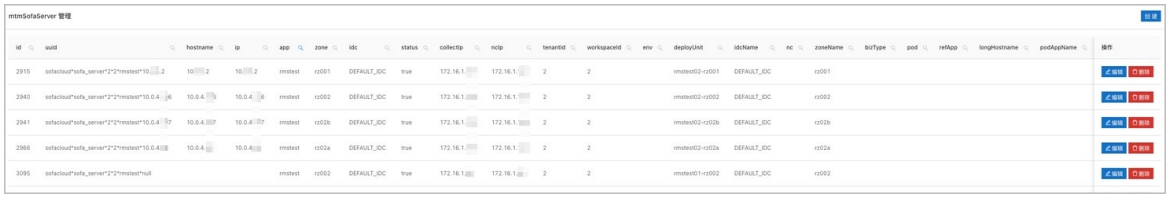
【场景描述】日志监控正常，但系统指标缺失。

【产生原因】

初步认为计算层和 vessel 均没有问题，问题可能出现在 Agent 及其 registry。

【解决方案】

1. 通过 `boss/manager` 查看应用元数据。



   选择 mtmSofaServer，在 App 中搜索需要查询的 App，找到对应的容器 IP（上图中即为 `172.16.1.127`）。

2. 查看 Agent 配置是否正常执行。

   在 AKE 集群找到某个容器（应用的 node 可能无法访问，可通过核心集群的容器检查），尝试在 Captain 登录 node，或者远程 debug：

   在 AKE 任意找一容器，进入之后访问需排查应用的 node：

```
curl 172.16.1.127:19510/conf/getDBConf
curl 172.16.1.127:19510/getAppConf
```

上述两个接口由 Agent 提供，如果看起来正常，可暂时认为 Agent 没有问题。接下来检查 registry 的 IP。

3. 检查 agentregistry 状态。

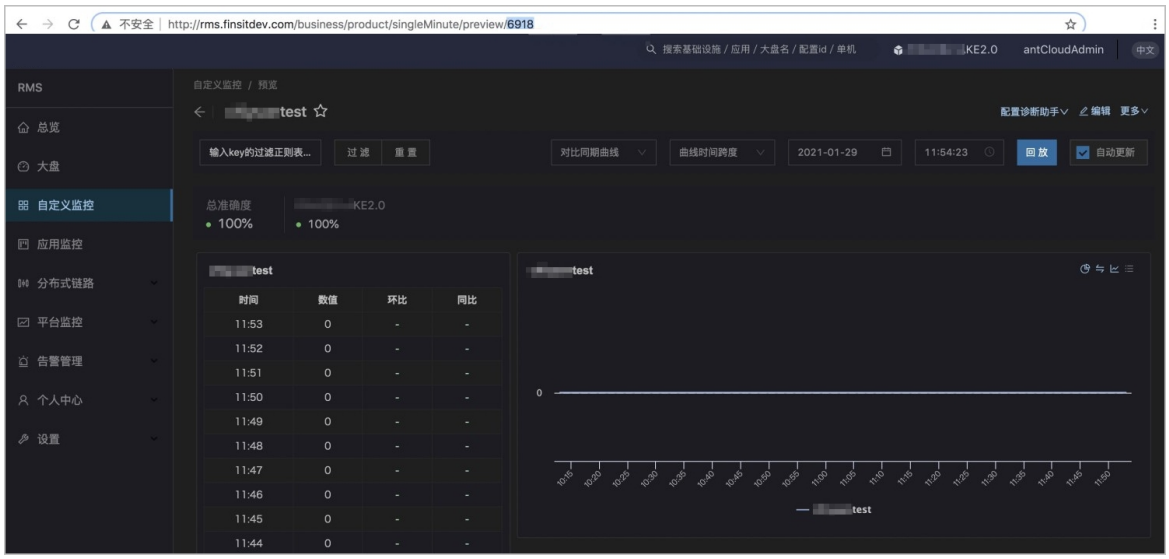
   对于步骤 2 找到 registry 的 IP（上图即为 `192.168.37.57`），此 IP 是 registry 的负载均衡 IP。前往云游 local 中查找 registry，可以确认其容器 IP。在 captain 中找到该容器，查看 agentregistry 是否正常工作。

6.8.6. 采集任务配置是否成功下发到计算侧问题排查

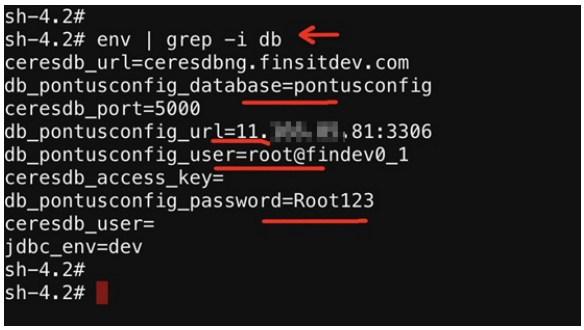
关于采集任务配置是否成功下发到计算侧问题排查方法如下：

1. 获取采集配置 ID。

   打开自定义监控配置，请求中最后的数据即为自定义配置 ID。



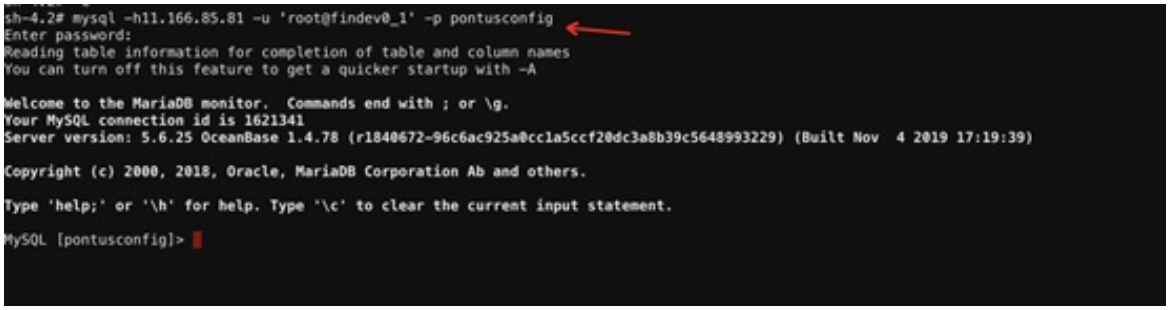
2. 进入 pontusconsole 容器，获取 DB 信息。

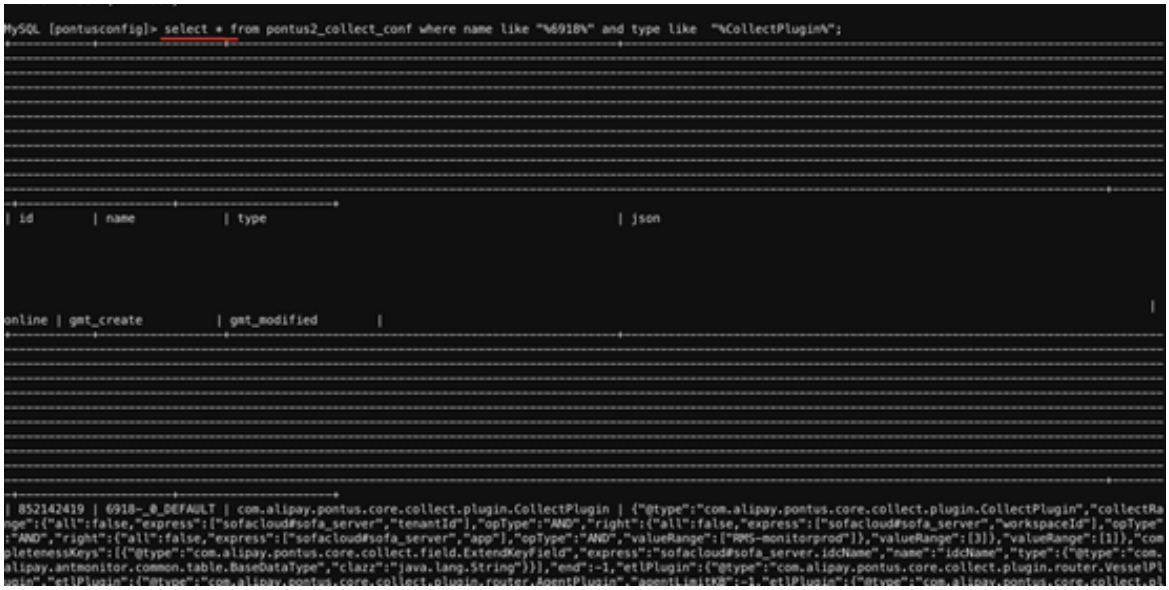


3. 进入 DB 查看是否有对应的配置：

```
 select * from pontus2_collect_conf where name like "%6918%" and type like
"%CollectPlugin%";
```

，其中 6918 是第 1 步获取的 ID。





- 如果存在对应配置，则说明配置下发计算成功。
- 如果不存在，则说明同步失败。

6.9. 存储层异常

6.9.1. ceresdb 启动失败

【场景描述】

健康检查日志报错 `telnet 5000` 失败，去容器中发现 5000 端口上没有进程。

> **说明**
>
> 该环境下 vessel 启动也会出现类似的问题，进程一直无法启动成功。

【产生原因】

```
(gdb) bt
#0  0x00007ff9782ea4ed in __lll_lock_wait () from /lib64/libpthread.so.0
#1  0x00007ff9782e5dcb in _L_lock_883 () from /lib64/libpthread.so.0
#2  0x00007ff9782e5c98 in pthread_mutex_lock () from /lib64/libpthread.so.0
#3  0x00005578ac2188bd in malloc_mutex_lock_final (mutex=0x5578ad37c360 <init_lock>)
    at /app/ceresbase/target/release/build/jemalloc-sys-7aa051630e0130cd/out/jemalloc/inclu
de/jemalloc/internal/mutex.h:141
#4  _rjem_je_malloc_mutex_lock_slow (mutex=mutex@entry=0x5578ad37c360 <init_lock>) at /app/
ceresbase/target/release/build/jemalloc-sys-7aa051630e0130cd/out/jemalloc/src/mutex.c:84
#5  0x00005578ac1ddef0 in malloc_mutex_lock (tsdn=0x0, mutex=0x5578ad37c360 <init_lock>)
    at /app/ceresbase/target/release/build/jemalloc-sys-7aa051630e0130cd/out/jemalloc/inclu
de/jemalloc/internal/mutex.h:205
#6  0x00005578abce6ba1 in malloc_init_hard () at /app/ceresbase/target/release/build/jemall
oc-sys-7aa051630e0130cd/out/jemalloc/src/jemalloc.c:1506
#7  0x00005578ac1e6356 in malloc_init () at /app/ceresbase/target/release/build/jemalloc-sy
s-7aa051630e0130cd/out/jemalloc/src/jemalloc.c:217
#8  imalloc (dopts=<synthetic pointer>, sopts=<synthetic pointer>) at /app/ceresbase/target
/release/build/jemalloc-sys-7aa051630e0130cd/out/jemalloc/src/jemalloc.c:1986
#9  calloc (num=1, size=32) at /app/ceresbase/target/release/build/jemalloc-sys-7aa051630e0
130cd/out/jemalloc/src/jemalloc.c:2138
#10 0x00007ff9776fd550 in _dlerror_run () from /lib64/libdl.so.2
#11 0x00007ff9776fd058 in dlsym () from /lib64/libdl.so.2
#12 0x00007ff9784f8b41 in sysconf (__name=30) at libsysconf-alipay.c:92
#13 0x00005578ac21961b in os_page_detect () at /app/ceresbase/target/release/build/jemalloc
-sys-7aa051630e0130cd/out/jemalloc/src/pages.c:396
#14 _rjem_je_pages_boot () at /app/ceresbase/target/release/build/jemalloc-sys-7aa051630e01
30cd/out/jemalloc/src/pages.c:563
#15 0x00005578abce6ae1 in malloc_init_hard_a0_locked () at /app/ceresbase/target/release/bu
ild/jemalloc-sys-7aa051630e0130cd/out/jemalloc/src/jemalloc.c:1291
#16 0x00005578abce6c06 in malloc_init_hard () at /app/ceresbase/target/release/build/jemall
oc-sys-7aa051630e0130cd/out/jemalloc/src/jemalloc.c:1517
#17 0x00005578ac1e6356 in malloc_init () at /app/ceresbase/target/release/build/jemalloc-sy
s-7aa051630e0130cd/out/jemalloc/src/jemalloc.c:217
#18 imalloc (dopts=<synthetic pointer>, sopts=<synthetic pointer>) at /app/ceresbase/target
/release/build/jemalloc-sys-7aa051630e0130cd/out/jemalloc/src/jemalloc.c:1986
#19 calloc (num=1, size=32) at /app/ceresbase/target/release/build/jemalloc-sys-7aa051630e0
130cd/out/jemalloc/src/jemalloc.c:2138
#20 0x00007ff9776fd550 in _dlerror_run () from /lib64/libdl.so.2
#21 0x00007ff9776fd058 in dlsym () from /lib64/libdl.so.2
#22 0x00007ff9784f8b41 in sysconf (__name=30) at libsysconf-alipay.c:92
#23 0x00007ff97583eba8 in init_lib () from /lib64/libselinux.so.1
#24 0x00007ff9787098f3 in _dl_init_internal () from /lib64/ld-linux-x86-64.so.2
#25 0x00007ff9786fb15a in _dl_start_user () from /lib64/ld-linux-x86-64.so.2
#26 0x0000000000000001 in ?? ()
#27 0x00007fff17b17100 in ?? ()
#28 0x0000000000000000 in ?? ()
```

问题的本质是 libsysconf 在 hook sysconf 时，依赖了 dlsym，dlsym又依赖了 jemalloc，而 jemalloc 依赖了 sysconf，导致了循环依赖。

【解决方案】

目前在宿主机 0.1.7 版本以上，该问题均已经修复，宿主机需要安装并使用新版本：

- CENTOS6：libsysconf-alipay-0.1.9-20191219222502.alios6-x86_64
- CENTOS7：libsysconf-alipay-0.1.9-20191219222502.alios7-x86_64

在宿主机安装完成后，需要重新发布应用，重建底层 container 后，就会生效。

6.10. 链路跟踪系统异常

6.10.1. 有 TraceLog 日志，但查询不出来

【场景描述】有 TraceLog 日志，但查询不出来。

【产生原因】/

【解决方案】

1. BOS agentd 通道：
   i. 查看节点上的 agentd 状态。
   ii. 如果是 VPC 场景，还需要查看 pontus agent registry 和 pontus vessel 的状态。
2. 主动上报模式：

   查看节点是否能正常连通 monitorgateway。
3. SLS 通道：

   使用链路服务时，需要确认相应环境 ECS 是否已经安装了日志采集客户端，执行如下命令：

   ```
   sudo/etc/init.d/ilogtaild status
   ```

   如果尚未安装，可参照 分布式链路快速开始 或运维脚本来安装。对于新申请的 ECS 也需要定期安装采集客户端。（新版可参考 阿里云文档。）

> **重要**
>
> 机器组标识符 (user_defined_id) 的格式为 `${tenantName}_${workspaceName}` 。安装时需要注意这点, 其他的步骤没有变化。

6.10.2. 链路数据收集延迟

【场景描述】链路数据收集延迟。

【产生原因】Tracer 日志产生量超过限制。

【解决方案】

需要关注 Tracer 日志的产生量，目前支持的 rpc/db/rest 等每类日志 25M/s 的生成量。

如果业务调用量很大，需联系管理员扩 shard。

6.10.3. 查询链路详情存在类型为 MOCK 的节点

【场景描述】查询链路详情存在类型为 MOCK 的节点。

【产生原因】主要是因为未采集到该节点服务器日志。

【解决方案】

- 检查缺失节点 ECS 的日志采集客户端是否正常安装。
- 检查缺失节点的调用类型是否在系统支持范围内。

6.10.4. 多维查询没有结果，或者搜索链路为空

【场景描述】多维查询没有结果，或者搜索链路为空。

【产生原因】超出查询有效期或者日志采集客户端未正常安装。

【解决方案】

检查调用时间是否为有效时间内（默认缓存 7 天），超出时间的可能会查询不到数据。

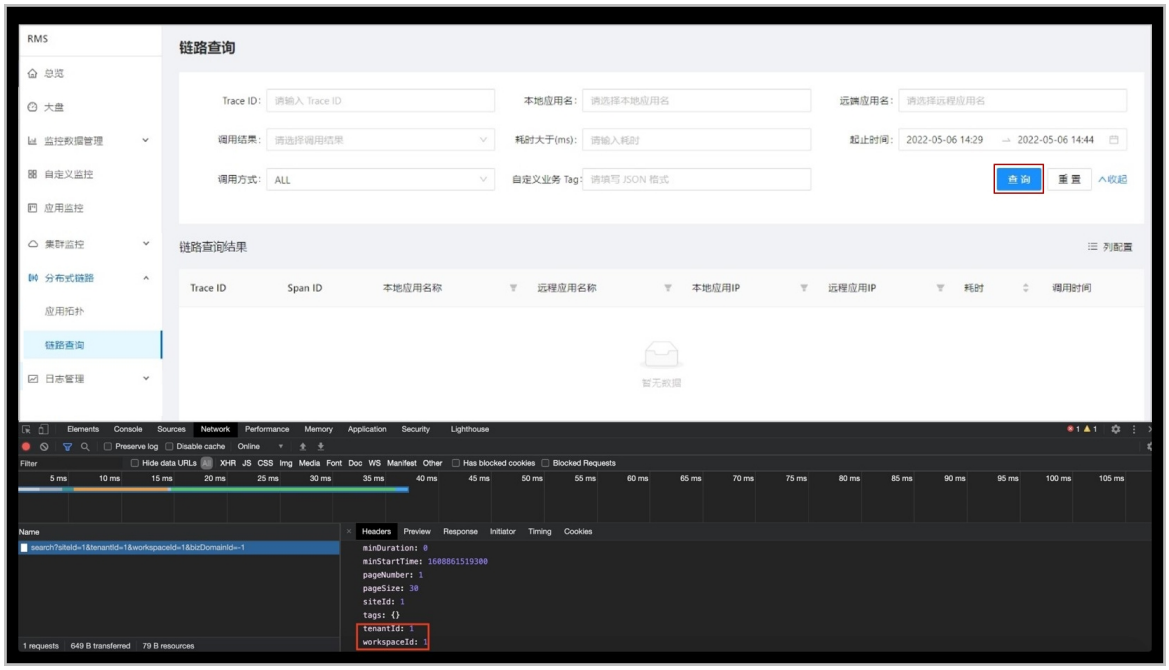
另外检查日志采集客户端是否正常安装。

6.10.5. Mock 数据自测与问题排查

6.10.5.1. 自测方法

1. 登录控制台，选定一个用户空间（非租户视角），单击左侧导航栏中的 **分布式链路 > 链路查询**。

2. 按 F12 打开浏览器控制台，单击 **Network** 页面，单击 **查询**，然后查看请求参数中的 tenantId 和 workspaceId。

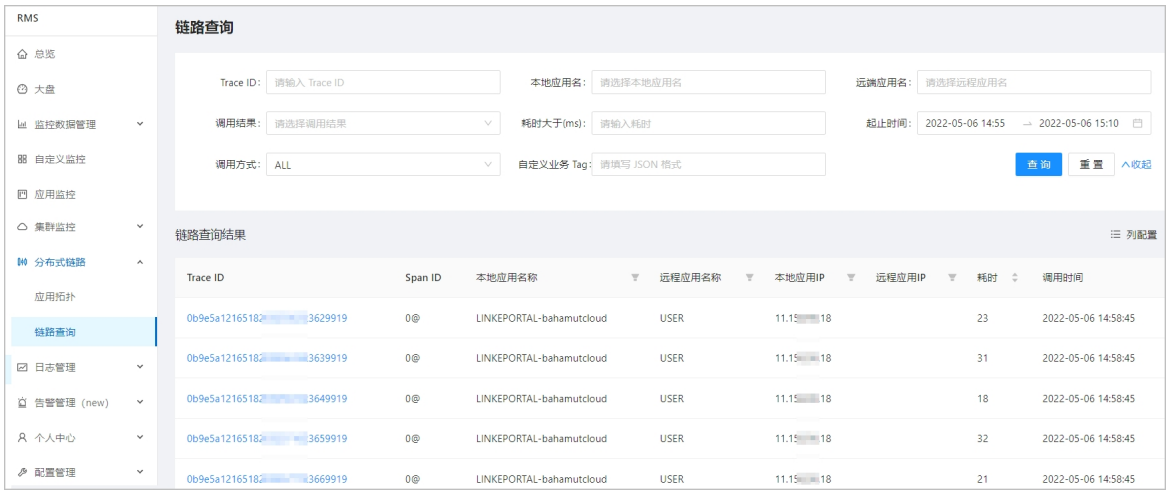


3. 触发 Mock 接口，留意 monitorgateway IP，tenantId 和 workspaceId。HTTP 响应码应为 200。

```
curl -X POST \
    http://{monitorgateway IP}:7202/mock/normal \
    -H 'Content-Type: application/json' \
    -d '{
  "tenantId": 1,
  "workspaceId": 1
}'
```

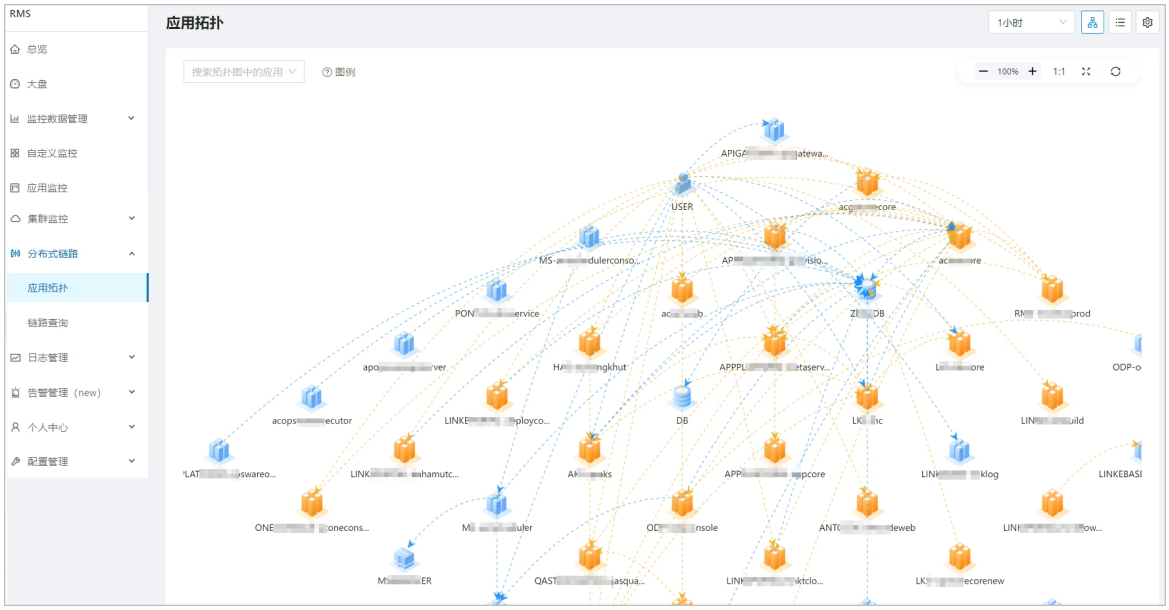
4. 刷新 BOS 链路查询页面，应该会有数据。

> **说明**
>
> 若刷新后页面上无数据，等一分钟后再试，如果仍然没有数据，则说明有问题。



5. 单击 **分布式链路 > 应用拓扑**，应该会有数据。

> **说明**
>
> 若无数据，等一分钟后再试，如果仍然没有数据，则说明有问题。



6.10.5.2. 问题排查

链路查询无数据

【场景描述】链路查询无数据。

【产生原因】

可能是 ceresdb-trace 数据库部署或者 monitorgateway 关联的参数有问题。

【解决方案】

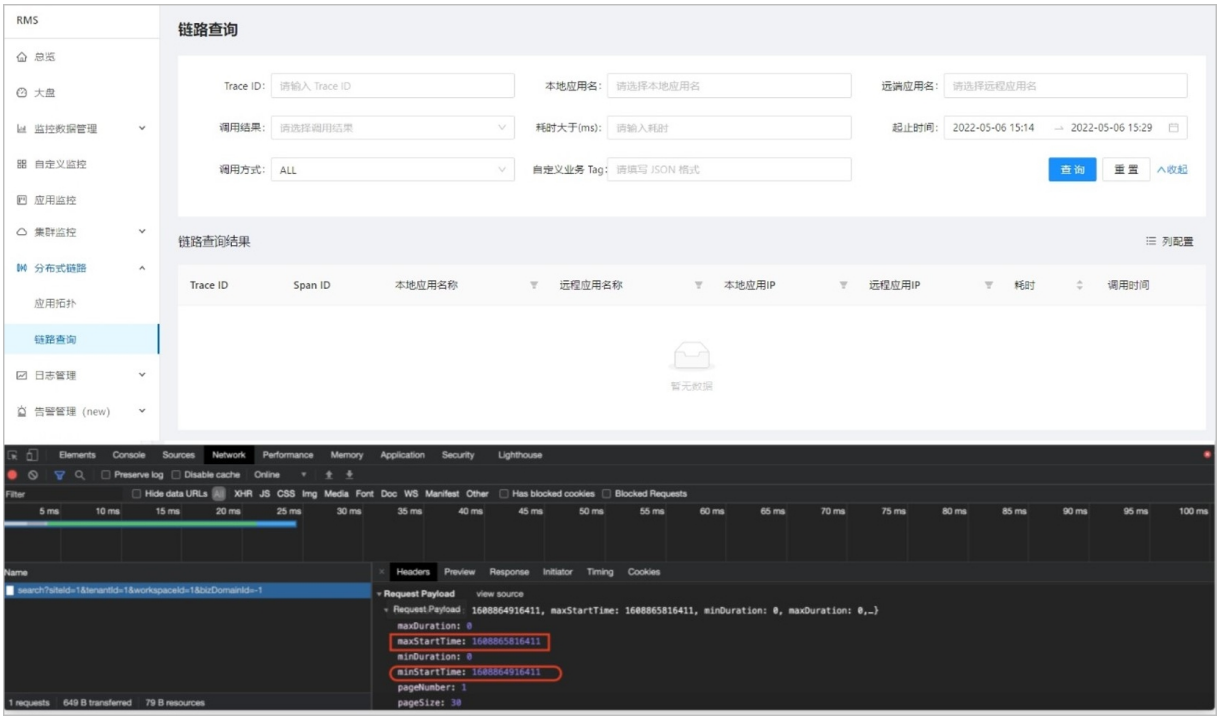
1. 检查 monitorgateway 的环境变量。

```
sh-4.2# env | grep CERESDB_TRACE_URL
CERESDB_TRACE_URL=:                    :5000
```

2. 检查monitorprod的环境变量。

```
sh-4.2# env | grep CERESDB_TRACE_URL
CERESDB_TRACE_URL=http://              :5000
```

3. 检查 ceresdb-trace 是否有数据，需要注意的是，将表名 span_20201127 中的日期要换成**当前日期**，lte 时间戳和 gte 时间戳需与下图中的 maxStartTime 和 minStartTime 保持一致。
   - lte = maxStartTime
   - gte = minStartTime




```
curl -X POST \
    http://CERESDB-TRACE-IP:5000/api/trace/query/span_20201127 \
    -H 'Content-Type: application/json' \
    -d '{
  "from": 0,
  "limit": 1000,
  "query": {
    "bool": {
      "must": [
        {
          "range": {
            "startTime": {
              "lte": 1606463353821,
              "gte": 1606462453821
            }
          }
        }
      ]
    }
  },
  "size": 30
}'
```

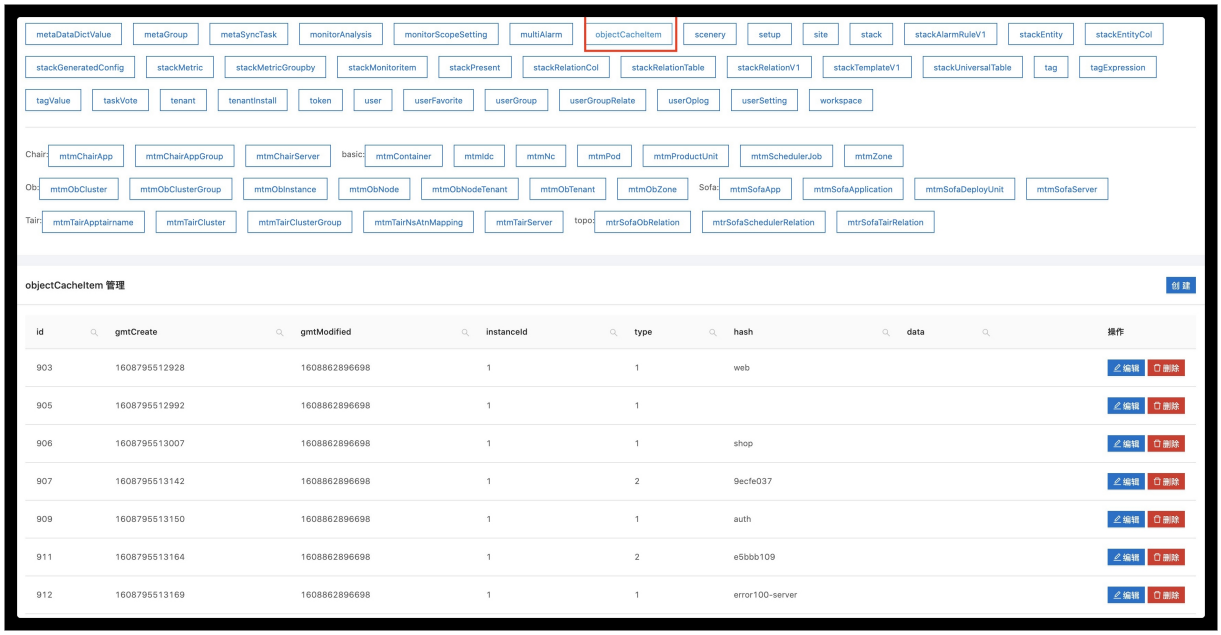
链路拓扑无数据

【场景描述】链路拓扑无数据。

【产生原因】可能是 objectCacheItem 内数据存在问题。

【解决方案】

查看数据库，可通过产品层的 /boss/manager 接口查看 objectCacheItem 内是否有数据，可参考下图。


metaDataDictValue
metaGroup
metaSyncTask
monitorAnalysis
monitorScopeSetting
multiAlarm
objectCacheItem
scenery
setup
site
stack
stackAlarmRuleV1
stackEntity
stackEntityCol
stackGeneratedConfig
stackMetric
stackMetricGroupby
stackMonitoritem
stackPresent
stackRelationCol
stackRelationTable
stackRelationV1
stackTemplateV1
stackUniversalTable
tag
tagExpression
tagValue
taskVote
tenant
tenantInstall
token
user
userFavorite
userGroup
userGroupRelate
userOplog
userSetting
workspace
Chair:
mtmChairApp
mtmChairAppGroup
mtmChairServer
basic:
mtmContainer
mtmIdc
mtmNc
mtmPod
mtmProductUnit
mtmSchedulerJob
mtmZone
Ob:
mtmObCluster
mtmObClusterGroup
mtmObInstance
mtmObNode
mtmObNodeTenant
mtmObTenant
mtmObZone
Sofa:
mtmSofaApp
mtmSofaApplication
mtmSofaDeployUnit
mtmSofaServer
Tair:
mtmTairApptairname
mtmTairCluster
mtmTairClusterGroup
mtmTairNsAtnMapping
mtmTairServer
topo:
mtrSofaObRelation
mtrSofaSchedulerRelation
mtrSofaTairRelation
objectCacheItem 管理
创 建
id
gmtCreate
gmtModified
instanceId
type
hash
data
操作
903 1608795512928 1608862896698 1 1 web
905 1608795512992 1608862896698 1 1
906 1608795513007 1608862896698 1 1 shop
907 1608795513142 1608862896698 1 2 9ecfe037
909 1608795513150 1608862896698 1 1 auth
911 1608795513164 1608862896698 1 2 e5bbb109
912 1608795513169 1608862896698 1 1 error100-server
编辑
删除

7.基础术语

基本概念表中的基本概念按中文拼音进行排序。

报表（Report）

指包含单个或多个数据源的视图，集中展示各数据源的监控结果。

大盘（Dashboard）

指包含多个报表的一个页面。

单机视角

从应用实例（单机）维度进行指标数据的聚合，从单机的空间分布和时间分布上对应用监控指标数据进行对比分析。

订阅

订阅后，通知组内的成员会收到监控告警通知。

服务指标

服务指标将应用相关的 Error、Service、SAL、CAL、DAL 等服务指标进行聚合透出，从 IDC（机房）、LDC（单元化）、单机 的空间分布和时间分布上进行对比分析，一个入口总览分析应用相关的所有监控数据。

IDC 视角

从 IDC（机房）维度进行指标数据的聚合，从单机的空间分布和时间分布上对应用监控指标数据进行对比分析。

监控产品（Monitoring Products）

指为实现自定义监控而提供的一些工具，可用于配置监控指标、生成数据源。

监控对象

由一个或者多个维度描述的被监控目标，比如说一个容器就可以用单个容器的维度描述。一个应用的一个逻辑 Zone 就需要用两个维度来描述（App + Zone）。监控对象实际上是监控数据的归属。

框架配置

基于 SOFA/SOFABOOT 框架开发的应用，PV、Service、SAL、SQL、Cal、CE Thread 等监控项会自动基于默认框架下的日志路径采集指标，仅需开启监控即可。

LDC 视角

从逻辑机房（LDC）维度进行指标数据的聚合，从单机的空间分布和时间分布上对应用监控指标数据进行对比分析。

配置模板

BOS 提供监控配置模板功能，支持将应用和自定义监控告警以配置模板（JSON 文件）导出，然后一键导入其他环境，如该环境已部署同名应用，则监控告警配置即生效，帮助投产运维更加高效便捷和一致。

数据源（Data Source）

指通过监控产品配置的监控指标。

通知人（Notificant）

指报警消息的接收人，接收渠道为手机短信。

通知组（Notification Group）

每个通知组可包含一个或多个通知人。在管理通知时，可通过通知组订阅，将报警内容发送给通知组内所有的通知人。

预警（Alert）

指通过配置的报警规则，触发报警通知。

指标

一个被监控对象数量特征的概念和数值。通常可以用若干个指标来描述一个监控对象随时间变化的情况，达到监控谁发生了什么事的最终效果。

自定义监控配置

非 SOFA/ SOFABoot 框架开发的应用，需为每个监控项配置采集日志路径和列值，比如 Error、Dal。如果是 SOFA/SOFABoot 框架开发的应用，但不希望使用默认日志路径监控的，也可以切换成 **自定义监控配置**。