SOFAStack

业务智能可观测平台
技术白皮书

产品版本：AntStack Plus 1.11.0

文档版本：20220930

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

1.什么是业务智能可观测平台

业务智能可观测平台 BOS（Business-Intelligent Observability Service）是一款具有可视化监测能力的金融级监控产品。它基于日志、指标、链路等海量数据进行多维聚合，向用户提供业务监控、应用监控、云原生监控、基础资源监控、日志查询分析、分布式链路等多角度的可视化监测功能，有丰富的可视化大盘，并提供了告警订阅功能。该服务可以帮助运维、研发、SRE（Site Reliability Engineer）等快速地发现问题、定位问题、分析问题、解决问题，为线上系统可用率提供有效保障。

BOS 支持的功能特性如下：

- **全方位实时监控：**提供业务、应用、基础资源、云原生等各种视角的监控能力，可实现关键指标秒级、普通指标分钟级的监控，具有高可靠、高时效、低延迟的特点。
- **灵活的报警规则：**可根据业务特征、时间段、重要程度等维度设置报警规则，实现不误报、不漏报。
- **便捷的自定义配置：**具有丰富的自定义产品配置功能，可便捷、高效地进行产品、报警配置。
- **开放的技术栈配置：** 可实现 Kubernetes、SOFA 技术栈应用部署即监控，通过简单技术栈配置，即可接入并监控非标业务应用。
- **可视化大盘：**丰富的可视化大盘，帮助您定制个性化的监控大盘。
- **分布式链路：**提供应用拓扑和链路查询功能，观测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。
- **日志查询和日志关联：**提供日志查询和日志关联功能。用户可以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。
- **低资源占用：**在可靠传输大量监控数据时，保证对宿主机的 CPU、内存等资源的极低占用率。
- **高可用：**提供万台设备的分钟级监控部署能力，故障自动恢复，集群可伸缩。
- **稳定高效的时序和数据存储：**在线持续聚合数据，保证数据容量可控，提供智能分级存储、存放策略。

BOS 监控的通用逻辑如下：

- **定义数据源：**首先，需要建立元数据基本模型，元数据是监控系统的根基。例如：定义应用、部署实例、机房信息、单元化信息等。在 SOFAStack 场景下，这部分信息需要从 PaaS 侧自动获取。
- **数据采集：**通过安装在机器上的 Agent 采集监控数据，这部分可以是日志，也可以是通过各种方式获取到的指标。
- **数据清洗：**将采集得到的非结构化数据，解析为监控系统可以处理的结构化信息。这部分解析能力需要事先通过监控系统预设好某种规则。例如，日志解析规则、Prometheus 协议规则等。

举个例子，有业务日志样例如下，希望通过监控获取每一分钟各商品各自创建了多少笔交易这一数据。

日志内容样例：

```
2012-11-11 11:11:11,2950211004,衣服,交易,创建,7,Y,
2012-11-11 11:11:12,2950211005,衣服,交易,创建,8,Y,
2012-11-11 11:11:13,2950211006,食品,交易,创建,9,Y,
2012-11-11 11:11:14,2950211007,家居,交易,创建,11,Y,
2012-11-11 11:11:15,2950211008,食品,交易,创建,7,N,
……
```

> **说明**
>
> 日志格式为：`日志打印时间,订单id,商品,服务名,方法名,耗时,结果`。

- **数据统计：**由于监控系统面对的是海量集群，在获取结构化数据之后，需要对数据进行数据统计。统计方式可以有很多种，比如求行数、求和、求平均、最大/最小值等统计方法。
- **预警配置：**可在统计数据上定义类型丰富的预警规则，包括绝对值、同比、环比、最近 N 分钟求和等，支持定义多项规则和逻辑运算。

2.功能架构

BOS 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。

- **告警管理**

  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。

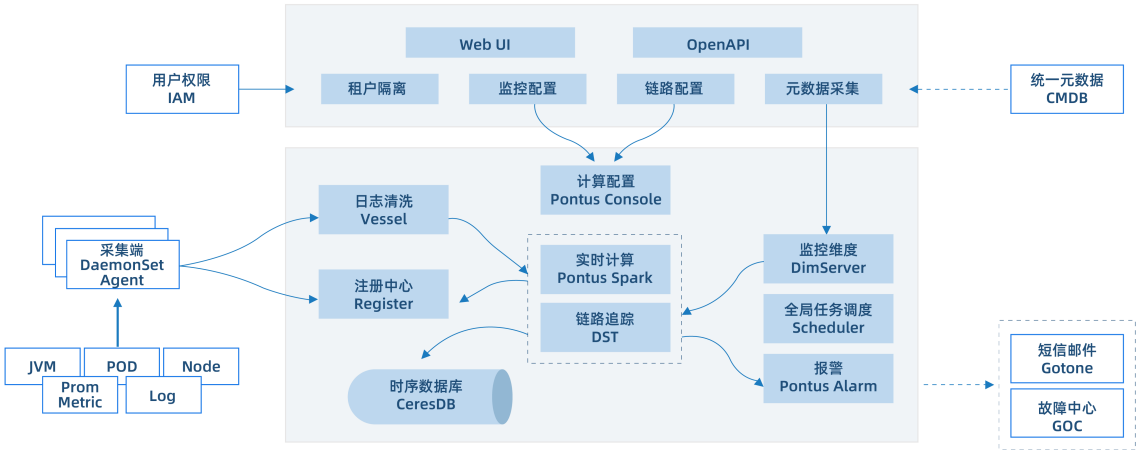
- **企业级特性**

  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。

3.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。

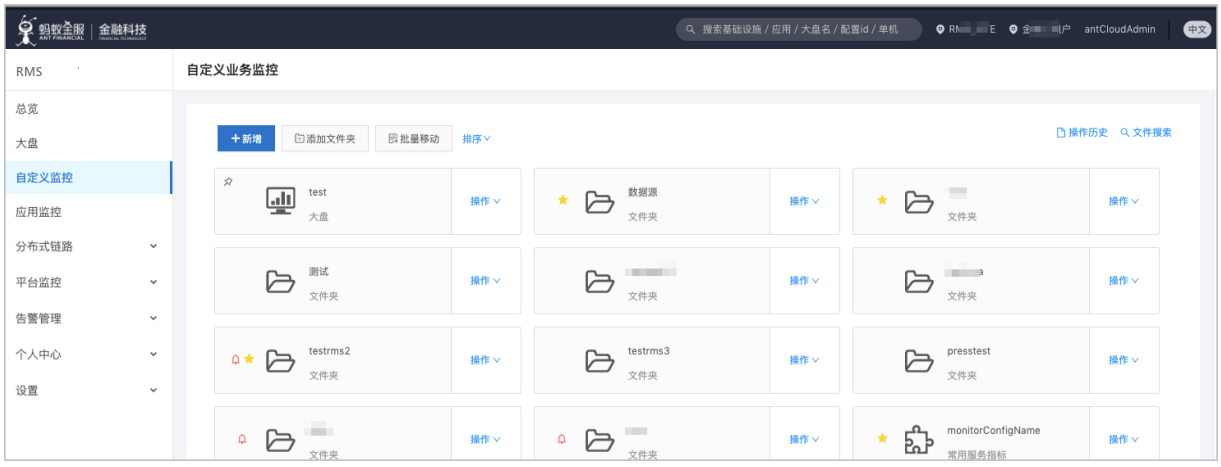
- **PontusConsole监控数据平台的管控端**
  - 主要负责内部运维性管控，系统配置，以及提供给产品使用的配置接口。
  - 提供操作维度表、指标表的标准数据服务接口。
- **Scheduler 全局任务调度器**
  - 不依赖任何中间件，实现全局任务调度能力，仅用于监控系统内部，实现功能自闭环。
  - 负责根据配置信息定时生成并分发监控的计算任务。
  - Scheduler 可以实现多机房部署，通过数据库维持高可用。
- **PontusSpark 分布式计算引擎**
  - 功能强大的 Spark 计算集群，负责接收产品层的计算配置，并对数据进行离线计算和统计。
  - 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 监控报警组件（又称 BOSAlarm）**
  - 主要负责将监控计算完成的报警信息以短信、邮件、钉钉等方式向外推送。
  - 在金融云输出的模式下，报警渠道主要通过 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：



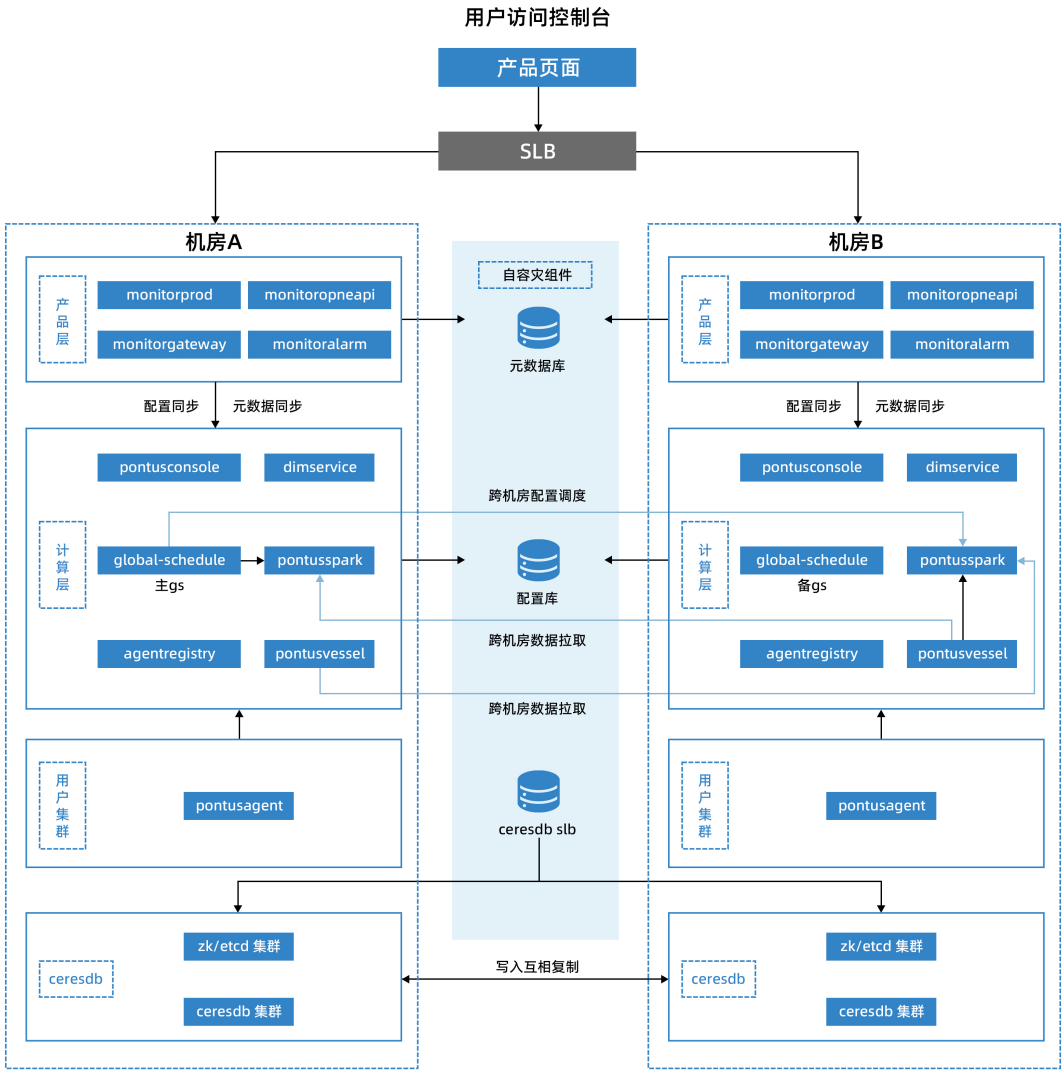
产品层负责与金融云用户权限系统 IAM 打通，读取用户身份信息，同时 BOS 的所有权限管控在 IAM 上进行收口。同时，产品层也负责打通 PaaS 元数据，将元数据同步至 DimServer 。

> **说明**
>
> PaaS 元数据是指通过金融云 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。

4.产品容灾架构

经典网络的容灾架构

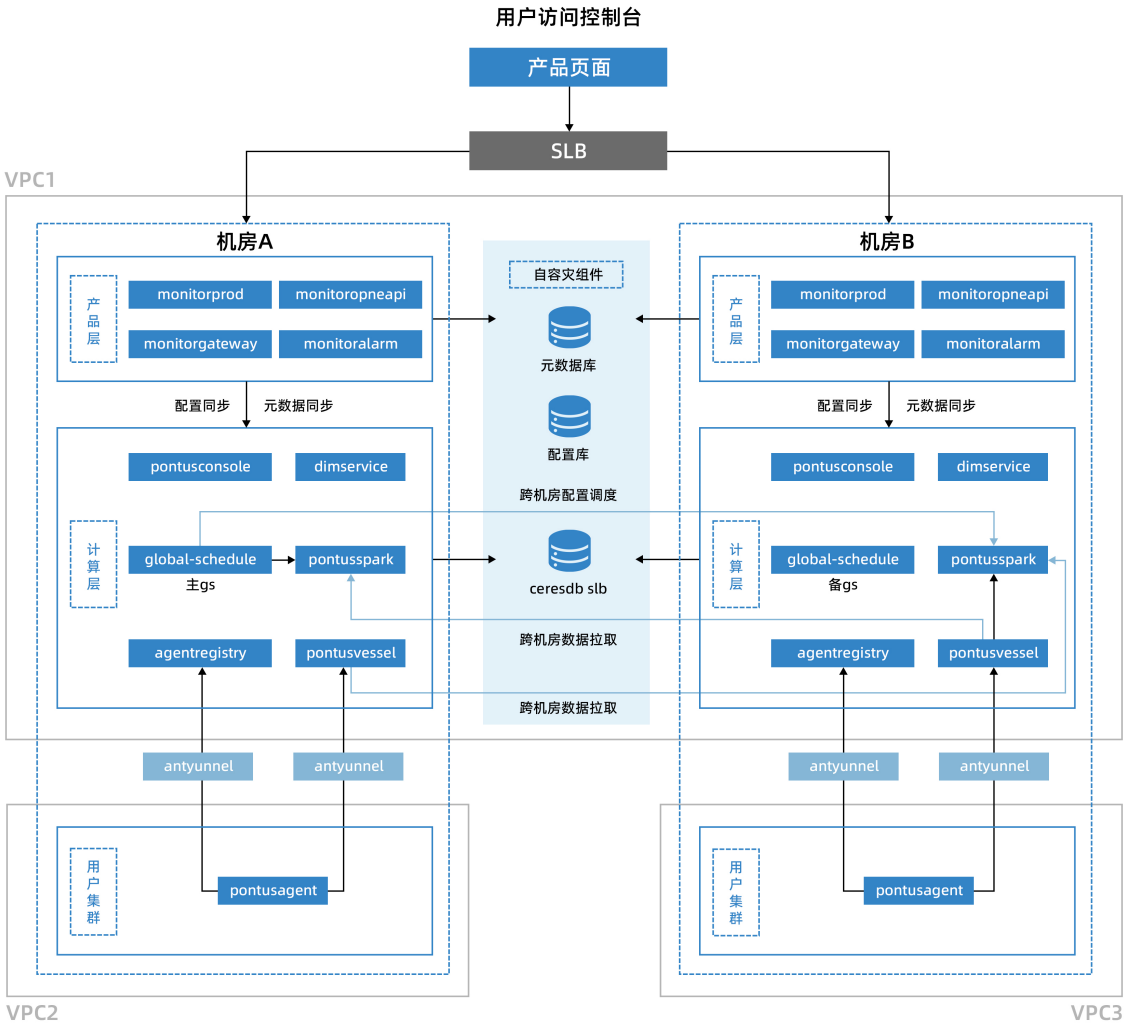
经典网络的容灾架构如下图所示：



- 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。

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。



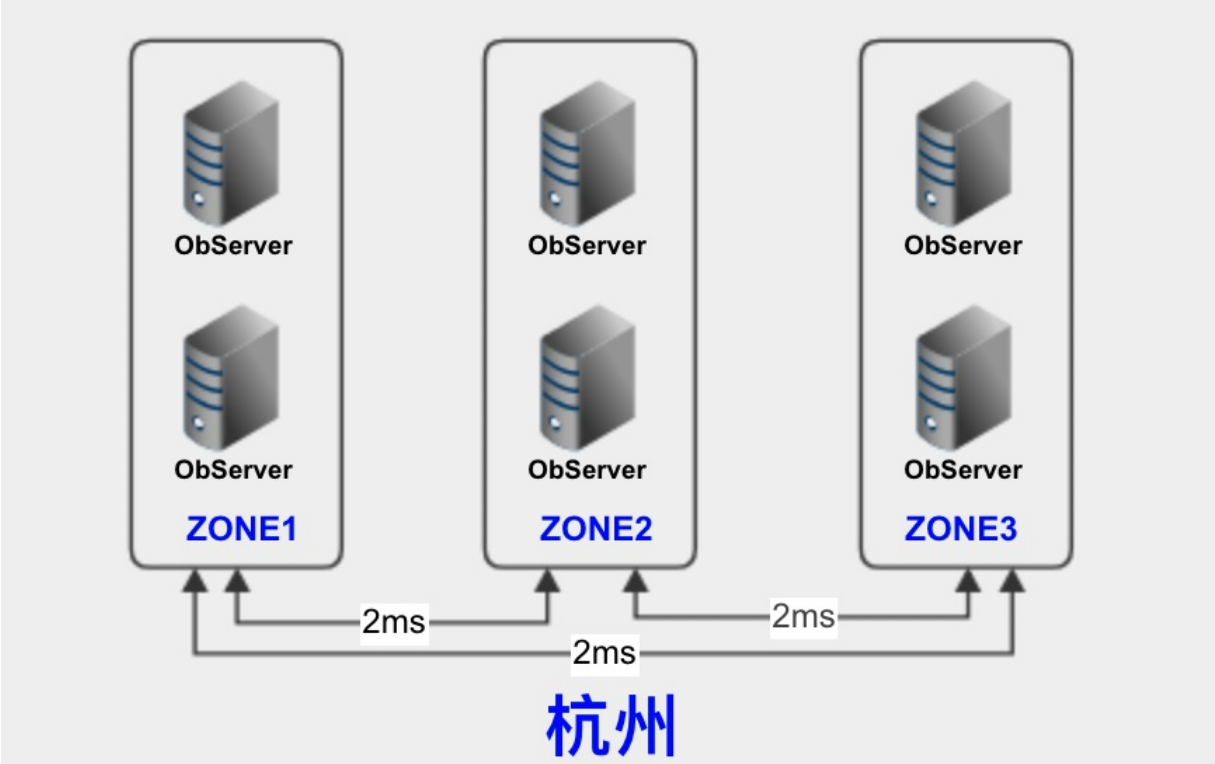
OceanBase 容灾

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。



两地三中心五副本架构

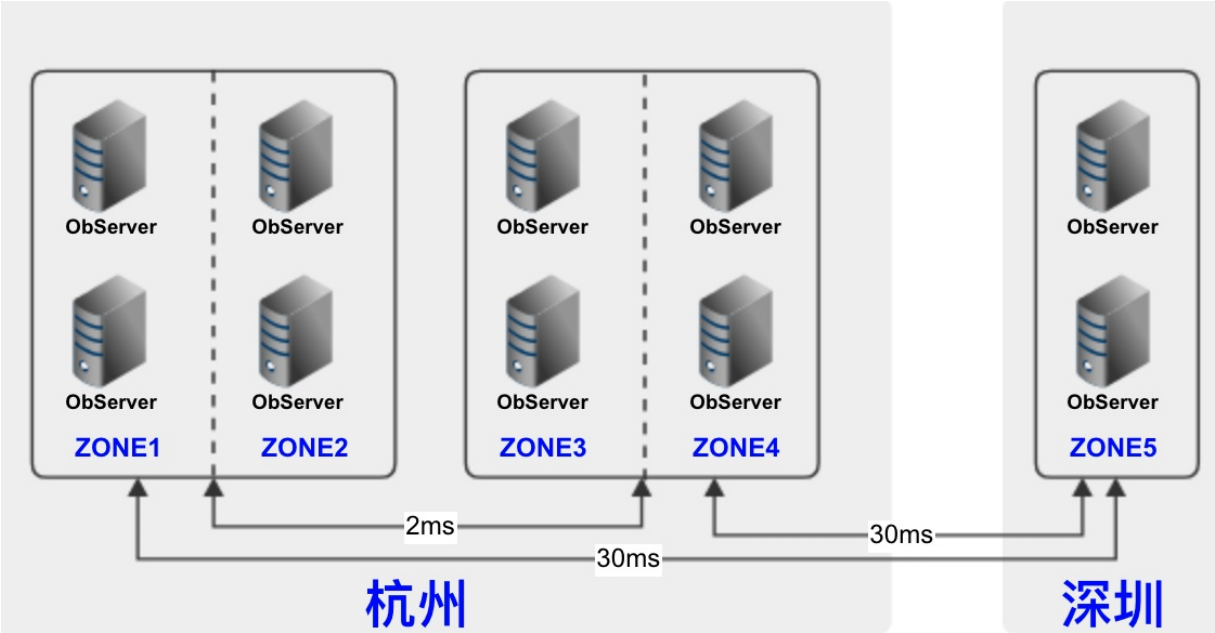
该部署架构要求机房是在两个城市，其中一个城市有两个机房，另外一个城市一个机房。该部署架构最少要求是五台机器，每台机器一个副本，一共是五副本。杭州两机房，每个机房有两个副本，一共四个副本，深圳一个机房一个副本。

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。

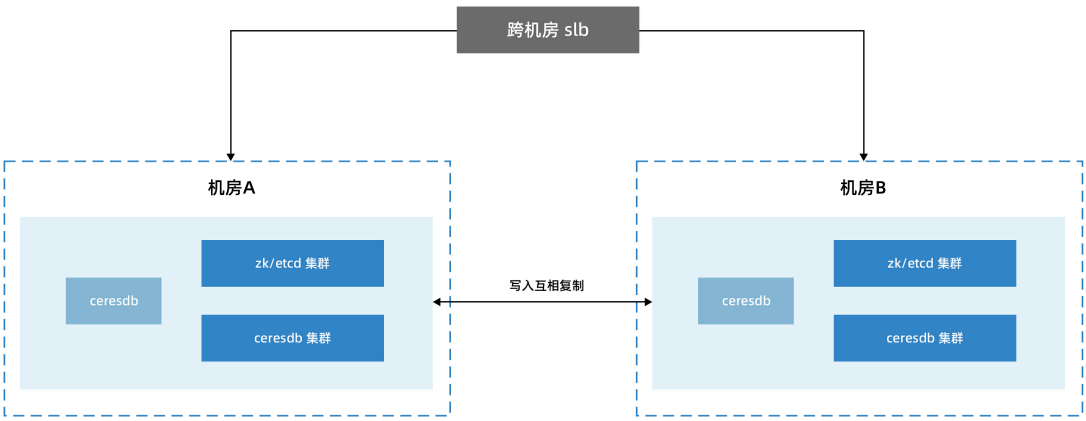
该部署要求同城两机房的延迟低于 2ms，城市间的延迟一般会大于 30ms，30ms 物理距离基本上能做到北京到深圳了，可以说要求是比较宽的，另外同城两机房的要求也是很容易满足的。

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。



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

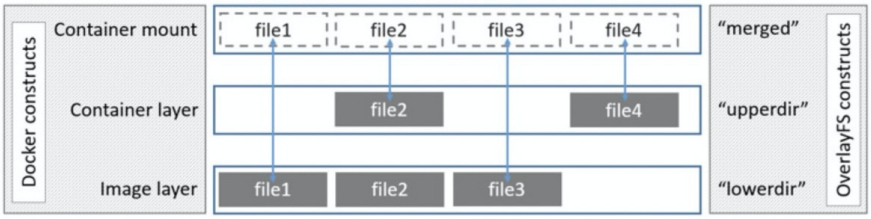
类似现在 cafemonitor 的小黑屋机制，CeresDB 的双写双读功能也需要支持熔断机制，以及熔断恢复。

5.功能原理

日志采集

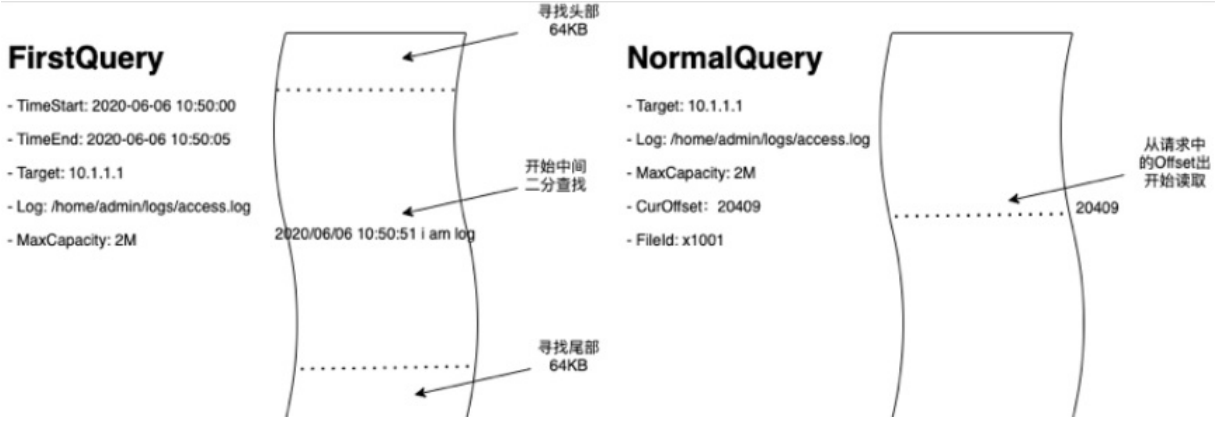
日志文件读取方式

日志采集分为宿主机日志采集和容器日志采集，宿主机日志 agent 采集直接读取就行，容器日志采集 agent 通过容器的 MergedDir 挂载来读取日志。



查找方式

Agent 采集日志有两种查找方式：FirstQuery 和 NormQuery。FirstQuery 通过时间戳进行二分查找，NormQuery 则通过 offset 来查找。



支持的日志时间格式

- normal：最常见的时间格式 `2016-12-13 10:00:14`
- apache：apache 时间格式 `08/May/2017 01:13:52`
- long：完整时间戳，精确到毫秒，格式 `1476064860000`

- Shortlong：时间戳，精确到秒，格式 `1476064860`
- Monthfirst：月份在行首，且不包括年份 `May 08 01:13:52`
- Revert：normal 格式的月份在前 `05-25-2017 18:28:42`
- monthfirst2：月份在行首，且包含年份 `May 08 2017 01:13:52`

服务器时区自动确认

对于非中国机房的服务器，系统自动默认服务器所在地时区。

支持的编码

编码为自动识别的，一般支持 GBK、UTF-8 等。

当有编码不识别时，可以在配置的“基本信息”中“日志编码”进行填写。

系统指标采集

监控系统指标采集来自 asar 或者 cAdvisor。

asar

asar 包含了丰富的采集模块，在主机侧实现对容器各项数据的采集。

```
$asar --cpu | head -n 20
Time             ---------------------------------cpu---------------------------------
Time               user     sys    wait    hirq    sirq    util    nice   steal   guest
20/10/20-15:09:33  0.23     1.15   0.00    1.14    0.02    2.53    0.00    0.00    0.00
20/10/20-15:09:38  0.13     1.09   0.00    1.14    0.02    2.38    0.00    0.00    0.00
20/10/20-15:09:43  0.16     1.22   0.00    1.13    0.02    2.53    0.00    0.00    0.00
20/10/20-15:09:48  0.52     0.96   0.00    1.14    0.02    2.64    0.00    0.00    0.00
20/10/20-15:09:53  0.13     1.09   0.00    1.13    0.02    2.38    0.00    0.00    0.00
20/10/20-15:09:58  0.15     1.24   0.00    1.13    0.02    2.54    0.00    0.00    0.00
20/10/20-15:10:03  0.77     2.79   0.00    1.20    0.03    4.79    0.00    0.00    0.00
20/10/20-15:10:08  0.16     1.19   0.00    1.14    0.02    2.50    0.00    0.00    0.00
20/10/20-15:10:13  0.16     1.37   0.00    1.15    0.03    2.84    0.13    0.00    0.00
20/10/20-15:10:18  0.21     1.00   0.00    1.13    0.02    2.36    0.00    0.00    0.00
20/10/20-15:10:23  0.29     1.23   0.00    1.14    0.02    2.68    0.00    0.00    0.00
20/10/20-15:10:28  0.22     1.32   0.00    1.15    0.02    2.71    0.00    0.00    0.00
20/10/20-15:10:33  0.23     0.96   0.00    1.14    0.02    2.34    0.00    0.00    0.00
20/10/20-15:10:38  0.18     1.12   0.00    1.13    0.02    2.45    0.00    0.00    0.00
20/10/20-15:10:43  0.15     1.24   0.00    1.14    0.02    2.55    0.00    0.00    0.00
20/10/20-15:10:48  0.45     0.95   0.00    1.13    0.02    2.55    0.00    0.00    0.00
20/10/20-15:10:53  0.13     1.14   0.00    1.13    0.01    2.42    0.00    0.00    0.00
20/10/20-15:10:58  0.16     1.23   0.00    1.14    0.02    2.55    0.00    0.00    0.00
```

容器系统指标说明

指标名称	说明
cpu_total_cores	容器核心数

cpu_util	内存使用率，由 user+sys+nice+guset+hirq+sirq 组成，能反应当前系统 CPU 的利用率
cpu_sys	内核态时间比例
cpu_steal	被偷走的 CPU 时间的占比
cpu_wait	io wait 时间的比例
cpu_user	用户态时间的比例
load_load1	1min 内 load 指标
load_load5	5min 内 load 指标
load_load15	15min 内 load 指标
mem_util	内存使用率
mem_used	total-buff-cache-free 的内存量，特别注意不包含 cache
mem_total	总内存
traffic_bytin	网卡 rx 方向的流量大小，单位 bytes/s
traffic_bytout	网卡 tx 方向的流量大小，单位 bytes/s
traffic_pktin	网卡 rx 方向的 pps
traffic_pktout	网卡 tx 方向的 pps
traffic_pktdrp	出入方向的丢包数和。在容器内（采用 veth pair）
traffic_pkterr	出入方向的错误包数和。在容器内（采用 veth pair）
tcp_active	active open/s，主动发起连接的每秒次数

tcp_lisove	listen overflow，这种是由于 listen accept 队列满导致被丢弃的数量
tcp_AtmpFail	AttemptFails，发起连接失败的每秒次数，比如半连接队列满导致 syn 包被丢弃等等
tcp_CurrEstab	Current Established connection，当前 TCP 连接数
tcp_pasive	passive open/s，被动生成连接的每秒次数
tcp_retran	retransmit ratio，根据重传 segment 和总的 segment，计算出来的重传率
udp_idgm	主机受到的 udp 包的个数/s
udp_odgm	主机发送的 udp 包的个数/s
udp_idmerr	udp 错误包的数目，包括 udp csum 错误等
udp_noport	未知 udp 端口号的包个数/s，主机侧没有进程监听在该端口所致
partition_util	使用的磁盘空间的比率。单位 %
partition_bfree	空闲的空间，单位 byte。该挂载点下剩余可用的磁盘空间
partition_bused	使用的空间，单位 byte。已经使用的磁盘空间
partition_btotl	总共的磁盘空间，单位 byte。该挂载点总共可用的磁盘空间
partition_ifree	文件系统可用的 inode 个数，ext4 文件系统有总的 inode 的使用个数限制
partition_itotl	总的可用的 inode 的个数
partition_iutil	使用的 inode 个数占总的可使用个数的占比，总体使用率。单位 %
io_wio	写 IO 每秒次数
io-rio	读 IO 每秒次数

io_rbytes	读 IO 每秒 byte 数
io_wbytes	写 IO 每秒 byte 数
io_rqsize	request sector/IO，每个 IO 请求的大小，单位为 KB
io_qusize	queue size，当前 IO 队列中的 IO 请求数量
io_await	average response time for I/O requests，每个 IO 从发起到结束平均消耗时间，单位 ms
io_svctm	service time，实际下发到设备到返回的时间，不包含队列中等待的时间

cAdvisor

详情请参考 说明文档。

JVM指标采集

Java 程序启动后，默认会在 `/tmp/hsperfdata_${username}` 目录下以该进程的 id 为文件名新建文件，并在该文件中存储 JVM 运行的相关信息，其中的 userName 为当前的用户名，`/tmp/hsperfdata_${username}` 目录会存放该用户所有已经启动的 Java 进程信息。

而 jps、jconsole、jvisualvm 等工具的数据来源就是这个文件（ `/tmp/hsperfdata_${username}/pid` ）。所以当该文件不存在或是无法读取时就会出现 jps 无法查看该进程号，jconsole 无法监控等问题。

BOS 的 JVM 监控也是通过解析 `/tmp/hsperfdata_${username}/${pid}` 来实现，比如 `/tmp/hsperfdata_admin/1234` 。

详细指标说明如下：

eden 区

指标名称	说明
eden_used（sun.gc.generation.0.space.0.used）	Eden 区使用量
eden_capacity（sun.gc.generation.0.space.0.capacity）	当前 Eden 区容量，可以继续申请扩大
eden_max（sun.gc.generation.0.space.0.maxCapacity）	Eden 区总容量，无法再扩

Old 区

指标名称	说明
old_used（sun.gc.generation.1.space.0.used）	Old 区使用量
old_capacity（sun.gc.generation.1.space.0.capacity）	当前 Old 区容量，可以继续申请扩大
old_max（sun.gc.generation.1.space.0.maxCapacity）	Old 区总容量，无法再扩

metaspace 区

metaspace 区指标有很多，绝大多数时候关注 meta_used/meta_capacity/meta_util/meta_rutil 即可。

另外 chunks 相关指标必须将 jdk 升级到 8.7.12_fp1 版本及以上。

指标名称	说明
meta_used（sun.gc.metaspace.used）	Metaspace 的使用量
meta_capacity（sun.gc.metaspace.capacity）	当前 Metaspace 的容量，可以继续申请扩大
meta_util	meta_used/meta_capaticy，不考虑碎片的使用率
meta_rutil	meta_chunks_used/meta_capacity，考虑了碎片的使用率
compressedclass_used（sun.gc.compressedclassspace.used）	包含在 sun.gc.metaspace.used 中，表示 compressed class 区域
compressedclass_capacity（sun.gc.compressedclasspace.capacity）	包含在 sun.gc.metaspace.capacity 中，表示 compressed class 区域
meta_chunks_free（com.alibaba.metaspace.freeChunksTotal）	Metaspace空闲Chunks的总容量
meta_chunks_used（com.alibaba.metaspace.usedChunksTotal）	Metaspace 使用中的 Chunks 的总容量，接近 MaxMetaspace 时触发 FGC

compressedclass_chunks_free（com.alibaba.compressedclassspace.freeChunksTotal）	包含在 com.alibaba.metaspace.freeChunksTotal 中，表示 compressed class 区域
compressedclass_chunks_used（com.alibaba.compressedclassspace.usedChunksTotal）	包含在 com.alibaba.metaspace.usedChunksTotal 中，表示 compressed class 区域

GC

指标名称	说明
ygc_count（sun.gc.collector.0.invocations）	ygc 次数
ygc_time（sun.gc.collector.0.time）	ygc 总耗时
fgc_count（sun.gc.collector.1.invocations）	fgc 总次数（CMS 不准）
fgc_time（sun.gc.collector.1.time）	fgc 总耗时（CMS 不准）
tlab_alloc（sun.gc.tlab.alloc）	在 tlab 分配的总大小

Runtime

指标名称	说明
thread_started（java.threads.started）	线程（被 start 过的）数量
thread_live（java.threads.live）	存活的线程的数量
thread_daemon（java.threads.daemon）	daemon 线程数量
thread_live_peak（java.threads.livePeak）	活的线程的峰值数量
safepoints（sun.rt.safepoints）	safepoints 次数
safepoint_time（sun.rt.safepointTime）	safepoint 时间
safepoint_sync_time（sun.rt.safepointSyncTime）	safepoint sync 阶段花费的总时间

application_time（sun.rt.applicationTime）	应用运行总时间
parks（sun.rt._sync_Parks）	park 的次数
notificcations（sun.rt._sync_Notifications）	notify 的次数
inflations（sun.rt._sync_Inflations）	锁膨胀的次数
futile_wakeups（sun.rt._sync_FutileWakeups）	无效唤醒的次数

Prometheus 指标采集

监控支持采集 Prometheus metrics 格式的数据，支持 Gauge、Counter、Histogram 和 Summary 四种类型，数据格式可以参考云原生 Prometheus metrics。

进程扫描采集

监控 Agent 通过 user、cmdline keywords 等方式来识别当前物理机或者 Node 上所运行的唯一进程，并且识别进行的存活状态，从而达到进程监控的目的。

6.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。

调用外部的服务（ Service Access Layer，SAL）

在一个面向服务架构（SOA）中，系统之间有复杂的服务依赖关系。SAL 表示某个系统调用其他系统暴露的服务的统计数值。

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。

JVM 指标

BOS 的 JVM 指标是通过解析 `/tmp/hsperfdata_${username}/${pid}` 来实现。

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。

维度

有概念上的实体且能够用一组属性描述的事物，通常可以是一个容器，物理机，应用实例进程，也可以是一些更高抽象层次的对象比如一个应用类型，一个 Mysql cluster 或者一项业务等。当维度是容器时，那么容器的 IP、hostname、容器的宿主物理机，容器所在的 IDC 都是它的子属性。

维度的属性可以简单分为两类，一类属性本身还是一个维度，它有其自身的子属性，比如容器的物理机，它可以有自己的属性；另外一类属性本身已经是一个单纯的值了，比如 IP 和 hostname。因此对维度进行深度遍历之后，必然能得到最终的各种描述性属性。

系统指标

系统指标用来监控应用本身所在虚拟机或 ECS 的资源情况。 包括 CPU、Load、Memory 等情况。您无需配置日志，打开开关即可，监控 agent 会自动获取对应数据。

应用监控（Application Monitoring）

指对系统硬件指标（如 CPU、负载、硬盘、内存等）及应用业务指标（如错误量、页面访问量、应用服务调用量等）进行监控。

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。