SOFAStack

业务智能可观测平台
部署指南

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。

未经蚂蚁集团事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

商标声明

蚂蚁集团 ANT GROUP及其他蚂蚁集团相关的商标均为蚂蚁集团所有。本文档涉及的第三方的注册商标，依法由权利人所有。

免责声明

由于产品版本升级、调整或其他原因，本文档内容有可能变更。蚂蚁集团保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在蚂蚁集团授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过蚂蚁集团授权渠道下载、获取最新版的用户文档。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

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.入场前置条件 07
2.拓扑和规格说明 08
3.单机房监控部署 09
3.1. 版本信息 09
3.2. 部署顺序 09
3.3. 关键参数说明 09
3.4. 产品集部署 14
3.4.1. 导出云游解决方案 14
3.4.2. 导入解决方案 16
3.5. 部署 Agent 16
3.5.1. 部署 Agent 概述 16
3.5.2. 获取 agentregistry LB 17
3.5.3. 安装核心态 Agent 18
3.5.4. 安装用户态 Agent 20
3.5.4.1. 容器发布部署安装 21
3.5.4.1.1. Chart 安装 21
3.5.4.1.2. YAML 安装 21
3.5.4.1.2.1. LHC 部署场景 21
3.5.4.1.2.2. AKS 部署场景 24
3.5.4.2. 经典发布部署安装 24
4.双机房监控部署 27
4.1. 双机房无容灾部署 27
4.1.1. 概述 27
4.1.2. 版本信息 27
4.1.3. 部署顺序 27
4.1.4. 产品部署 28

4.1.5. Agent 部署 28
4.2. 双机房容灾部署 29
4.2.1. 概述 29
4.2.2. 版本信息 29
4.2.3. 部署顺序 30
4.2.4. 产品部署 30
4.2.5. Agent 部署 34
5.产品初始化 35
5.1. 元数据初始化 35
5.1.1. setup 配置 35
5.1.2. 原生监控初始化（可选） 36
5.2. 告警初始化 37
5.2.1. 基本配置 37
5.2.2. 基于 Gotone 的告警配置（可选） 38
5.3. 自监控配置导入 38
5.3.1. 概述 39
5.3.2. 前置条件 39
5.3.3. 准备工作 40
5.3.4. 配置导入 40
6.产品升级 42
7.产品验证 43
7.1. 核心态监控验证 43
7.1.1. 应用监控 43
7.1.2. 自定义监控 45
7.1.3. 告警计算验证 50
7.1.4. 告警推送验证 54
7.1.5. 云原生监控 58
7.1.6. 分布式链路 59

7.1.6.1. 概述 59
7.1.6.2. 快速验证法 60
7.1.6.3. Mock 数据验证法 60
7.1.6.4. 问题排查 62
7.2. 用户态监控验证 64
8.附录：技术栈备份与恢复 66

1.入场前置条件

- 确保环境内所有机器的时钟完全同步，包括云游的集群和 PaaS 部署的集群。
- 确认云游和 PaaS 的版本是否需要升级。
  - 云游版本要求大于等于 4.1.0。
  - 如果需要监控 CAFE 部署的用户应用，则需要部署 PaaS 微内核版本，版本要求大于等于 2.1.2。
- 在云游基线中，确认环境中是否已部署 ZooKeeper 产品。
- 网络要求：
  - 中枢集群 VPC 需要与各业务集群 VPC 网络连通。
  - BOS 监控产品所在网络需要与云游 Local 所在网络连通，BOS 需要调用云游 Local 接口来获取中枢元数据。
  - 用户集群节点的 19310 端口（即 Agent 通讯端口）需要能够被中枢集群访问，Agent Registry 的 9025 及 8080 端口需要能够被用户集群节点访问。

2.拓扑和规格说明

BOS 目前提供两种拓扑可供选择：

拓扑	说明
标准单机房部署	包含有 BOS 的全量产品能力。
无链路拓扑	仅包含监控能力，用于单软件场景输出。

对于 BOS，需要根据不同的场景选择不同的规格，具体如下。

- **POC 规格：** 全部应用为单点应用

  适用场景：主要用于概念验证（Proof of Concept，POC）类项目。
- **精简版（低性能）：** 监控最低性能的输出规格

  适用场景：单软件场景输出。
- **基础版（中性能）：** 监控常规版本的输出规格

  适用场景：用于包含 PaaS、中间件等全家桶模式的输出。
- **性能版（高性能）：** 监控高性能版本的输出规格

  适用场景：主要用于有大量计算的场景。

3.单机房监控部署

3.1. 版本信息

产品	所属产品集	拓扑	规格	版本
CERESDBMR	3.21.X	按需选择	按需选择	1.2.12
CERESDBTRACE	3.21.X	按需选择	按需选择	1.1.8
PONTUS	3.21.X	按需选择	按需选择	3.12.0
BOS	3.21.X	按需选择	按需选择	3.21.X

3.2. 部署顺序

按如下顺序进行部署：

1. CERESDBMR
2. CERESDBTRACE
3. PONTUS
4. BOS

3.3. 关键参数说明

产品参数说明

应用	参数	参数说明
ceresdbng	ENABLE_REDUNDANT	是否开启冗余读写（容灾），只有双机房容灾时才需要改为 true。开启之后，必须设置 REDUNDANT_SLAVE_METRIC_ENDPOINT。
ceresdbng	REDUNDANT_SLAVE_METRIC_ENDPOINT	指定另一个机房的 ceresdbng 的 VIP 地址（如：192.168.1.1:5000），如果 ENABLE_REDUNDANT 为 false，设置无效。
ceresdbng	REDUNDANT_METRIC_SYNC_TYPE	双机房读写 sync type，值为 Async/Sync，默认值为 Async。如果 ENABLE_REDUNDANT 为 false，设置无效。

ceresdbng	CLUSTER_NAME	当双机房共用一套 ZK 时： • 如果当前为主机房，则填写 ceresdb_mCluster。 • 如果当前为备机房，则填写 ceresdb_sCluster。 • 如果没有容灾机房，则直接确认即可。
ceresdbtrace	CLUSTER_NAME	datanode agent 注册在 ZK 上的二级目录。 • 如果当前为主机房，则填写 ceresdb_trace_mCluster。 • 如果当前为备机房，则填写 ceresdb_trace_sCluster。 • 如果没有容灾机房，则直接确认即可，参数无用。
ceresdbtrace	ZK_PEERS	datanode_agent 和 trace proxy 注册的 ZK 地址。 若有容灾机房且容灾方案使用的是跨机房 ZK，则填写跨机房 ZK 集群的 IP 列表，例如：127.0.0.1:2181、127.0.0.2:2181、127.0.0.3:2181。 如果是单机房模式，需要填写 `#if(${prod.CERESDBMR.METRICS_ZK_PEERS}=='zk') ${res.ZOOKEEPER.app.zookeeper.containerIps} #else ${prod.CERESDBMR.METRICS_ZK_PEERS} #end`
ceresdbtrace	TRACE_ENABLE_FAILOVER	是否开启异步双写，只有在双机房容灾场景下才需要配置为 true，否则为 false。
ceresdbtrace	TRACE_SLAVE_NODE_HOSTS	备机房中 ceresdbtrace 的三节点 host 列表。 • 若无容灾机房，直接确认即可。 • 若已经填写 ZK_PEERS，则直接确认即可。 • 若没有 ZK，需要填写该参数。

monitorprod	RMS_YUNYOU_VIP	本机房内云游的 VIP 地址。 **说明** 由于云游自身未保留这个信息，如需获取该信息，请咨询交付或驻场同学。 实际填写时，需要在最前面带上协议（如 HTTP 或 HTTPS），例如 http://192.168.37.150:81。 **注意** 该地址必须是可供 RMS 容器访问的地址。
monitorgateway	JXNX_ENABLED	是否开启 Mesh 链路功能。值为 false。
monitorgateway	AGENTD_ENABLED	是否开启 rms agentd 收集日志功能，默认开启，值为 true。
monitorgateway	AGENTD_USEPROXY	是否开启 Agentd proxy 模式，默认关闭，VPC 场景下开启。 **说明** VPC 环境填写 true，非 VPC 环境填写 false。
monitorgateway	REDIS_HOST	redis host，无需填写。
monitorgateway	REDIS_PASSWORD	redis 密码，无需填写。
pontusvessel	DataCenter	IDC 名称 • 单机房部署时，直接确认就行。 • 双机房部署时，机房信息要和元数据保持一致。 该参数会影响双机房数据采集，必须正确配置。
产品公共参数	db_type	数据库类型。 无论是 OB、RDS 还是 MySQL，该参数值目前均为 mysql。

产品公共参数	RMS_VPC_GLOBAL	RMS_VPC_GLOBAL 公共参数。默认是 false。 如果采用 VPC 隔离模式，则设为 true。

初始化镜像参数说明

应用	Key	参数说明

rmsinit	stacks	需要导入的技术栈，json 格式。
		• 如果不需要初始化技术栈，则填 []。
		• 如果需要升级站点，建议不要做任何初始化操作，填 []。
		• 对于初始化技术栈，仅在新站点部署时使用。
		警告
		初始化操作属于高危操作，请谨慎操作！
		对于新站点，一般填写以下值：
		`[`
		`{"name":"sofacloud", "version":20210430190900},`
		`{"name":"basic", "version":20210430190900}`
		`]`
		说明
		○ **name**：必填。表示技术栈的名称，例如 sofacloud。
		○ **version**：必填。表示技术栈的版本，例如当前 sofacloud 支持的版本为 20210430190900。

rmsinit	template_configs	需要导入的模板，json 格式。直接填写以下值： `{"app":[{"name":"RMS-application-monitor","version":20210421}],"custom":[{"name":"RMS-custom-monitor","version":20210421}]}` **说明** • **name**：必填。表示模板配置的名称，例如 RMS-application-monitor。 • **version**：必填。表示模板配置的版本，例如 RMS-application-monitor 模板配置的版本为 20210421。

说明

有关技术栈备份与恢复操作，请参见 附录：技术栈备份与恢复。

重要参数验证

注意

务必验证本节所列的重要参数。

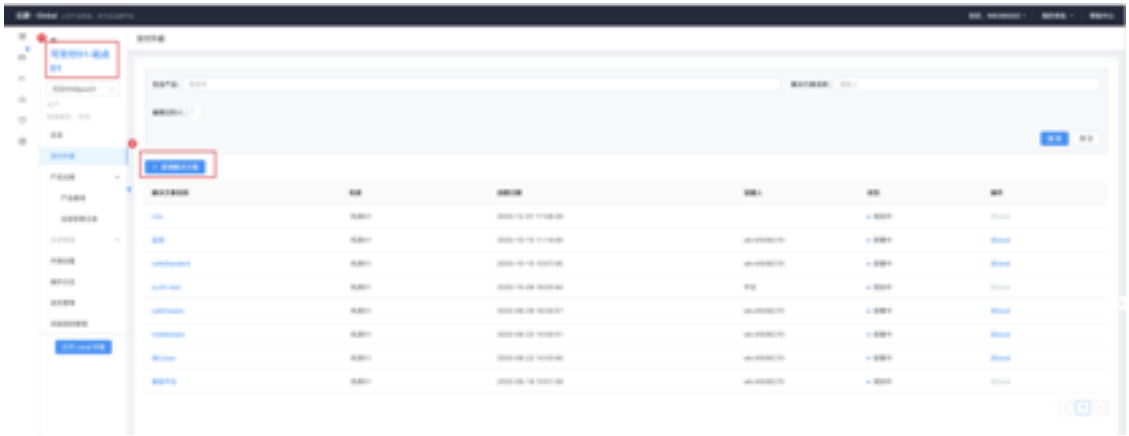
产品	应用	参数	参数类型	参数说明
CERESDBMR	ceresdbmr	ZK_PEERS		
BOS		db_type	产品公共变量	数据库类型。 无论是 OB、RDS 还是 MySQL，该参数值目前均为 mysql。
BOS	monitorpord	registry_addr	应用参数	monitorpord 依赖 REGISTRY 地址获取路由。

BOS	monitorpord	RMS_REGISTRY_INNER_SLB	应用参数	经典场景，Agent 安装所配置的 REGISTRY 地址。
BOS	monitorpord	RMS_OPSWARE_EXECUTOR_INNER_SLB	应用参数	经典场景，Agent 安装需要依赖 OPSWARE_EXECUTOR 服务。

3.4. 产品集部署

3.4.1. 导出云游解决方案

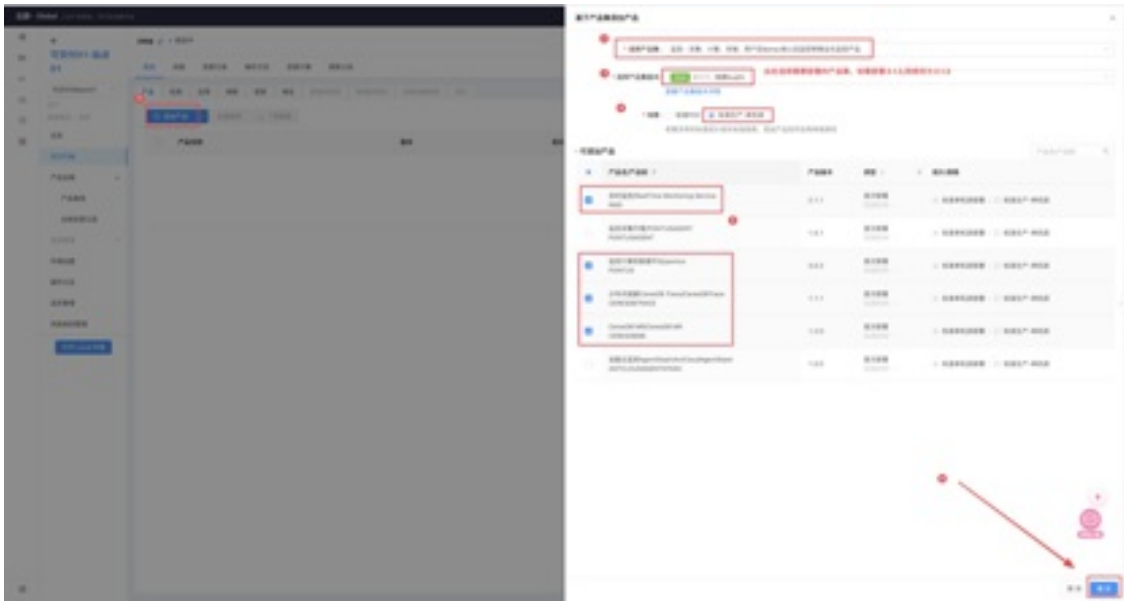
1. 在云游 Global，选择相应的环境并新增解决方案。



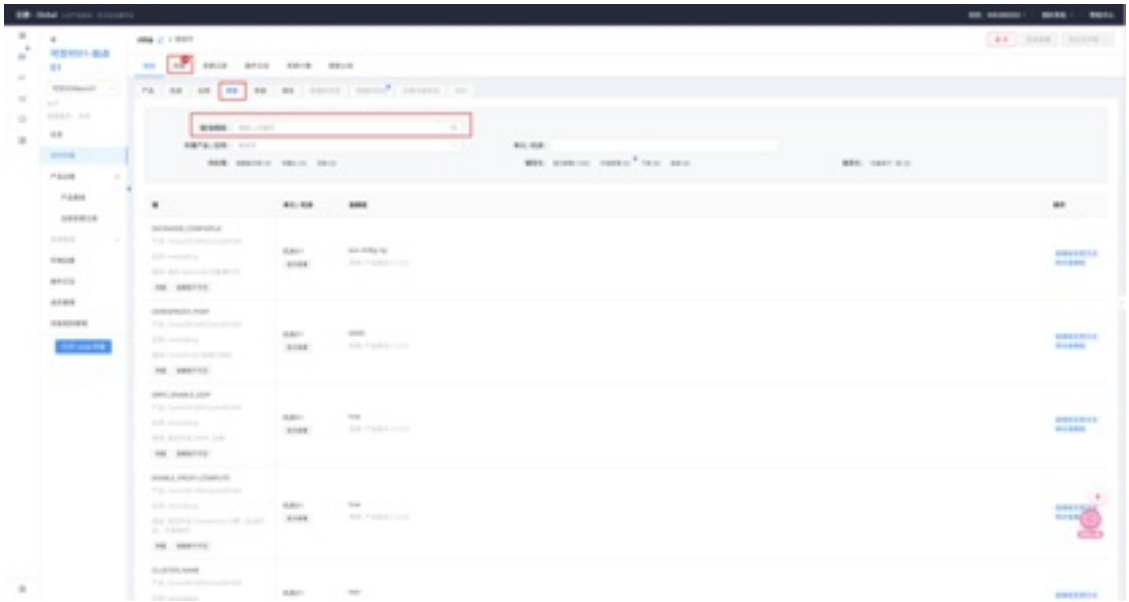
2. 选择要部署的产品集。

   需要注意的是，一般只需要选择 4 个关键的产品部署即可。

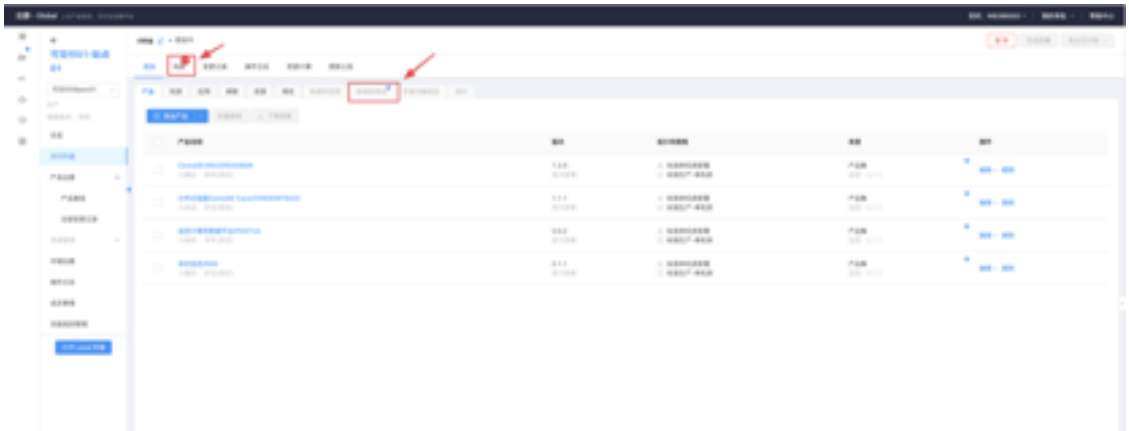
   - CERESDBMR
   - CERESDBTRACE
   - PONTUS
   - BOS



3. 根据需要，将所需参数修改为现场环境的值。参数的配置，参见 关键参数说明。



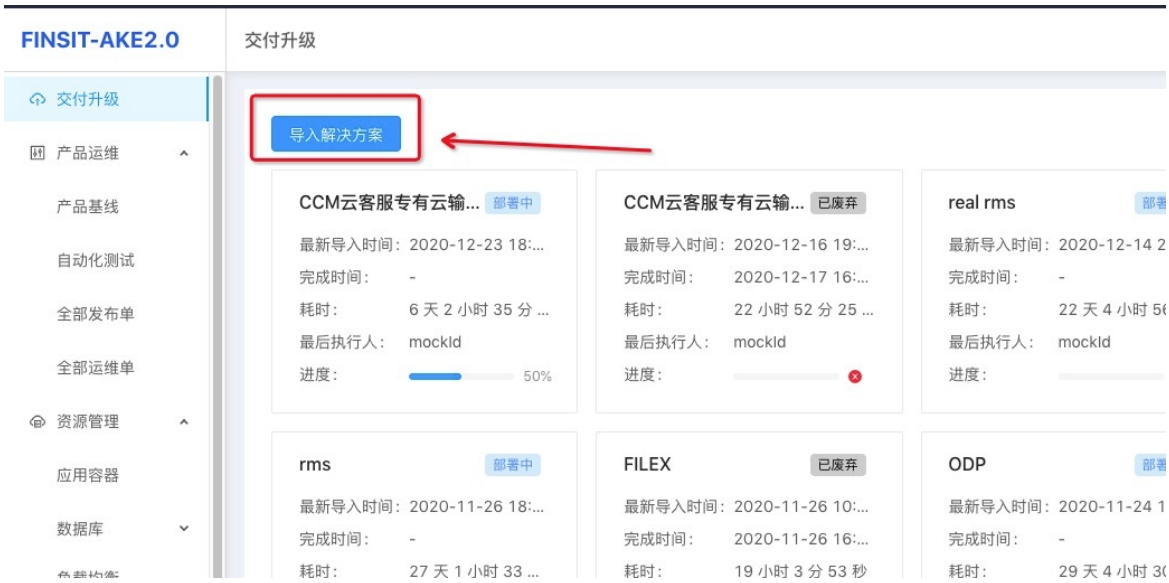
4. 数据库规划，确保无风险。



5. 导出解决方案。

3.4.2. 导入解决方案

1. 打开云游 Local，将刚导出的解决方案导入。



2. 确认参数配置，以及负载均衡规划。

   需要注意如下内容：

   - rms-init 中技术栈导入的相关参数
   - 各个产品中的双机房配置参数
   - 网络模式参数

   > **说明**
   >
   > 有关具体参数说明，请参见 关键参数说明。

3. 生成的发布单中单击各个组件开始发布。逐个单击应用，按照 部署顺序 进行部署。

3.5. 部署 Agent

监控产品部署完成之后，还需要在被监控的对象物理机或者节点（Node）上安装监控的 Agent。部署对象可以分为两部分：

- 云游负责部署的核心态应用集群。
- CAFÉ 负责部署的用户态应用集群。

3.5.1. 部署 Agent 概述

监控产品部署完成之后，还需要在被监控的对象物理机或者节点（Node）上安装监控的 Agent。部署对象可以分为两部分：

- 云游负责部署的核心态应用集群。
- CAFÉ 负责部署的用户态应用集群。

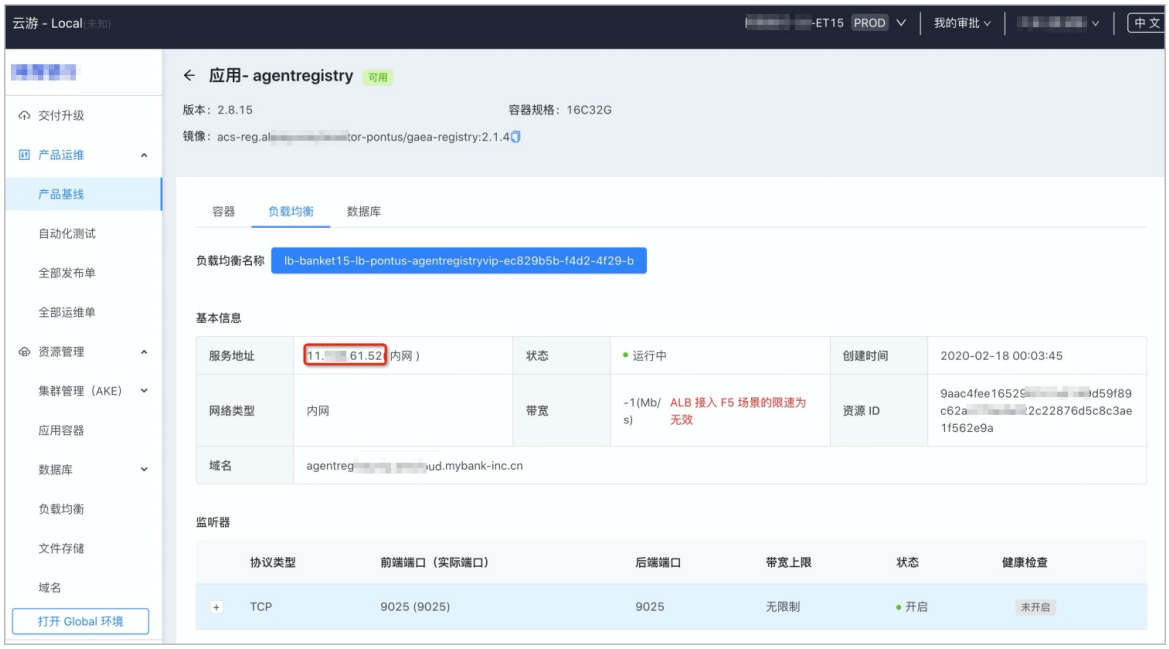
3.5.2. 获取 agentregistry LB

监控 Agent 需要连接到监控产品的 agentregistry 上进行服务注册，才能进入 BOS 的监控范围。因此，需要先获取 agentregistry 的 LB。

1. 登录云游 Local 控制台。
2. 在左侧导航栏上，单击 **产品基线**。
3. 在搜索框中输入 **pontus**，然后单击 **监控计算和数据平台**。

4. 单击 agentreistry 下方的 **负载均衡**。

5. 获取 agentregistry vip。

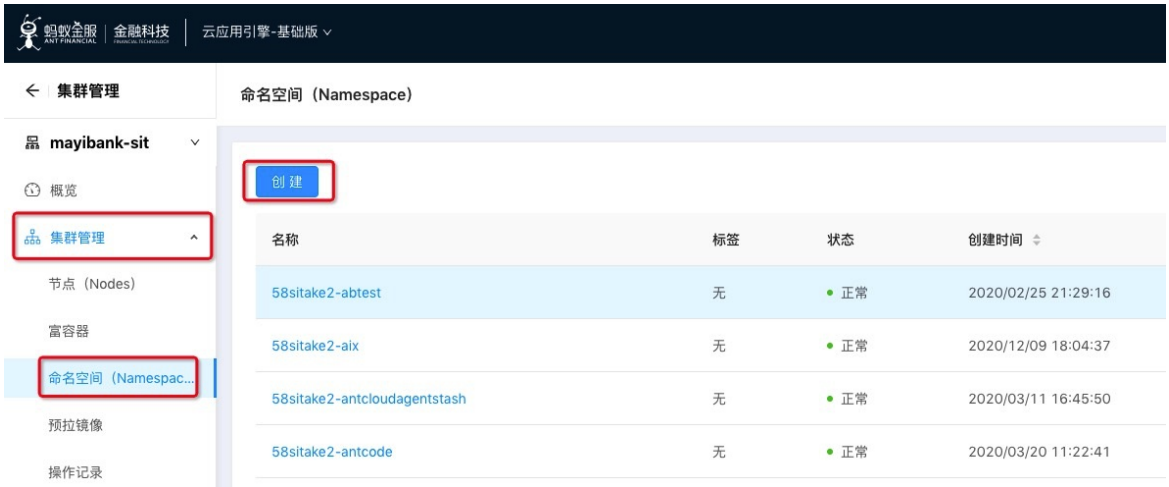


3.5.3. 安装核心态 Agent

1. **创建命名空间。**

   i. 打开 Captain，单击 **集群管理 > 命名空间**。

   ii. 单击 **创建**，创建 pontusagent 命名空间。



2. **准备 YAML。**

```
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: pontusagent
  namespace: pontusagent
  selfLink: /apis/apps/v1/namespaces/pontusagent/daemonsets/pontusagent
  uid: 4efff82d-25a7-11eb-9ae6-00163e010754
  generation: 9
  creationTimestamp: '2020-11-13T11:56:48Z'
  labels:
    app: pontusagent
```

```
    app: pontusagent
  annotations:
    deprecated.daemonset.template.generation: '17'
spec:
  selector:
    matchLabels:
      app: pontusagent
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: pontusagent
    spec:
      volumes:
        - name: rootfs
          hostPath:
            path: /proc
            type: ''
      containers:
        - name: pontusagent
          image: 'acs-reg.alipay.com/monitor-pontus/pontusagent:3.1.2.g'
          env:
            - name: EnableAgentRegistry
              value: 'false'
            - name: GaeaRegistryServers
              value: ${agentregistry vip}     ## 注意这里的 ${registry vip}的替换，
            - name: FilterDaemonSet
              value: 'true'
          resources:
            limits:
              cpu: '0.5'
              memory: 500Mi
            requests:
              cpu: '0'
              memory: '0'
          volumeMounts:
            - name: rootfs
              readOnly: true
              mountPath: /rootfs
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          imagePullPolicy: IfNotPresent
          securityContext:
            privileged: true
            procMount: Default
      restartPolicy: Always
      terminationGracePeriodSeconds: 30
      dnsPolicy: ClusterFirst
      automountServiceAccountToken: false
      hostNetwork: true
      hostPID: true
      securityContext: {}
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
```

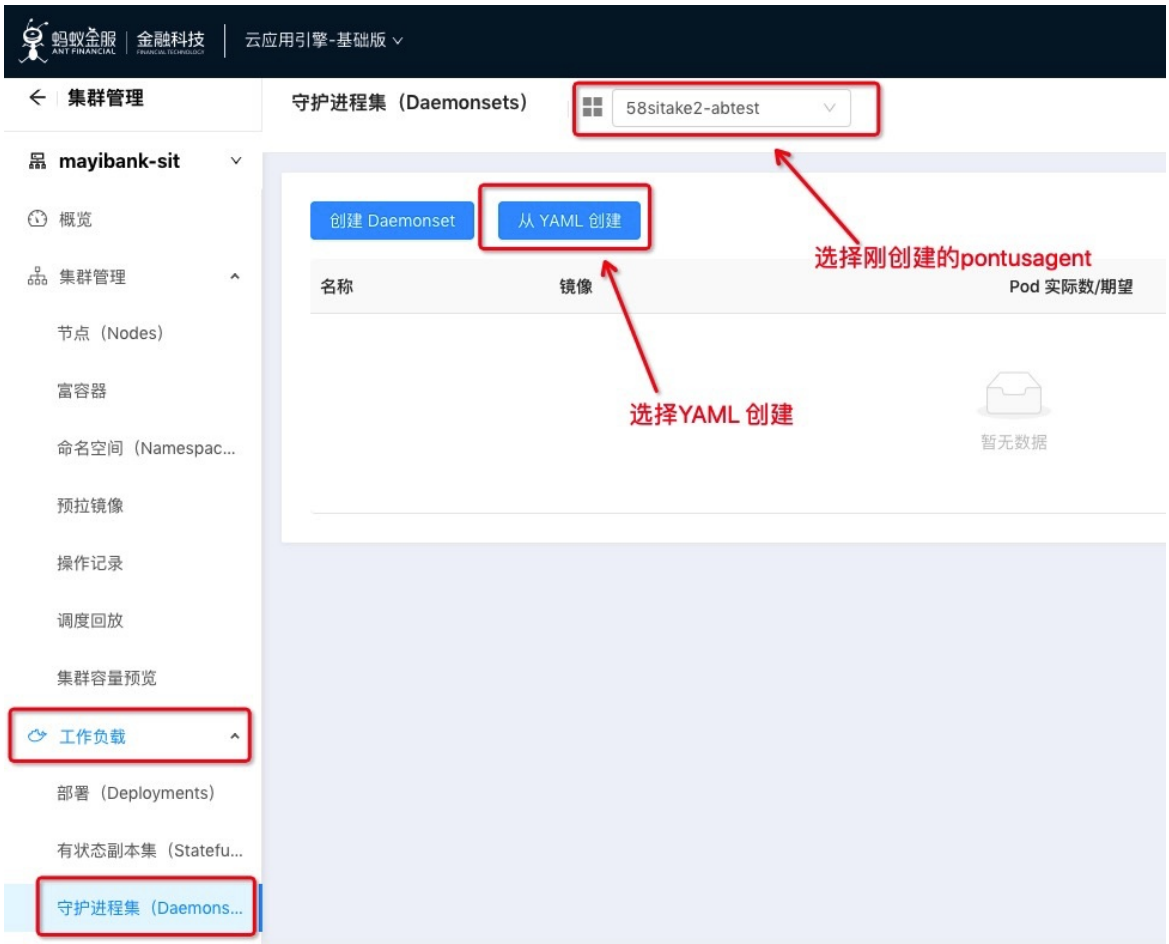
```
            nodeSelectorTerms:
              - matchExpressions:
                  - key: machine.sigma.alipay.com/no-daemonset
                    operator: DoesNotExist
      schedulerName: default-scheduler
      tolerations:
        - operator: Exists
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 5
  revisionHistoryLimit: 10
```

注意

需要将 `{{registryvip}}` 替换为前面获取到的 registry LB 的真实地址。

3. **创建 Daemonsets。**

   找到对应类型的集群，使用上一步中的 Agent YAML 创建守护进程集 (Daemonsets)。



3.5.4. 安装用户态 Agent

3.5.4.1. 容器发布部署安装

3.5.4.1.1. Chart 安装

用户态 Agent 通过 CAFE 实现安装，分为 LHC 场景、 AKS 场景、经典发布部署场景。

1. 登录 SOFAStack 控制台。
2. 在左侧导航栏单击 **集群管理 > 集群**，进入集群列表页。
3. 单击 **组件管理** 页签。找到 pontus-pontusagent 组件。
   - 如果该组件存在，则通过 Chart 安装。

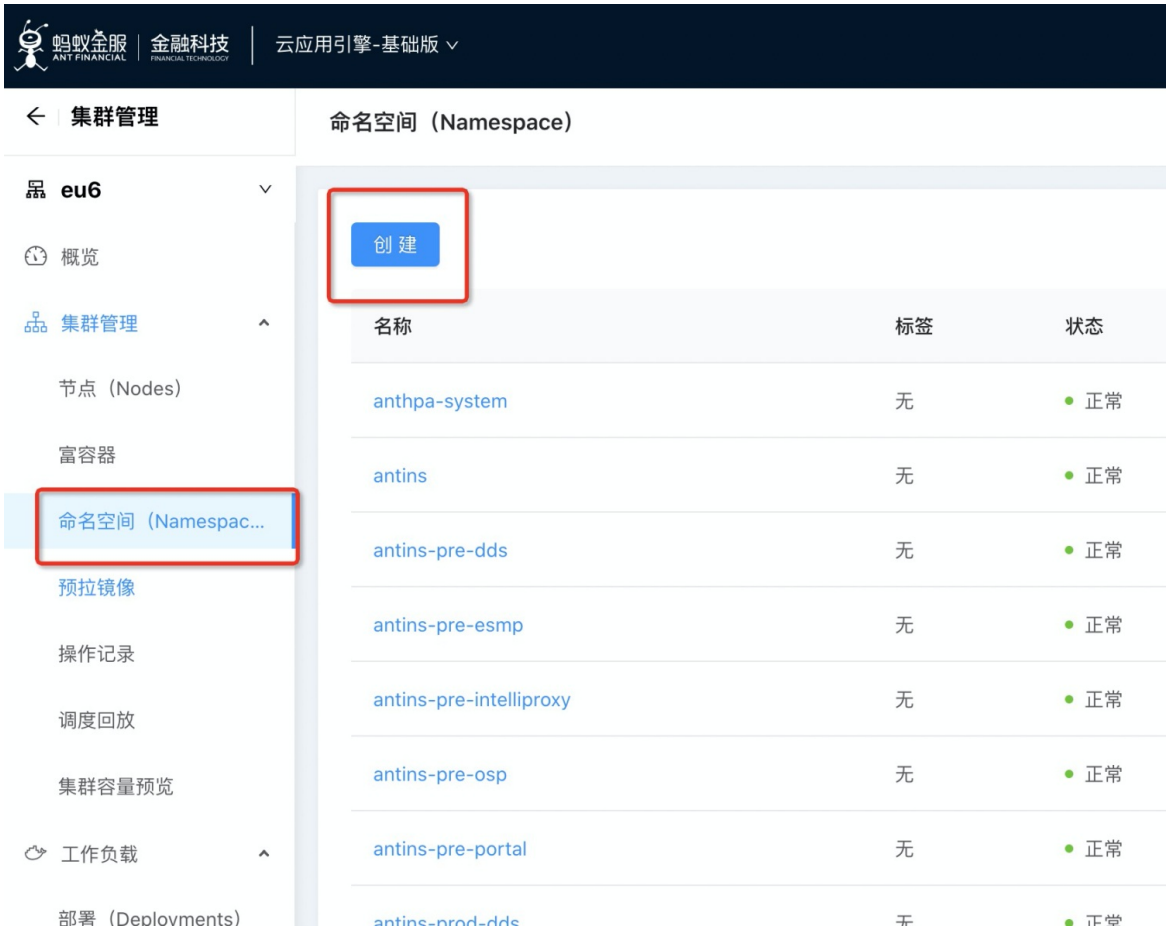
   - 如果该组件不存在，则通过 YAML 安装。详见 YAML 安装。

3.5.4.1.2. YAML 安装

3.5.4.1.2.1. LHC 部署场景

1. **创建命名空间。**

   打开 Captain，找到用户态集群，创建 pontusagent 命名空间。



2. **准备 YAML。**

```
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: pontusagent
  namespace: pontusagent
  selfLink: /apis/apps/v1/namespaces/pontusagent/daemonsets/pontusagent
  uid: 4efff82d-25a7-11eb-9ae6-00163e010754
  generation: 9
  creationTimestamp: '2020-11-13T11:56:48Z'
  labels:
    app: pontusagent
  annotations:
    deprecated.daemonset.template.generation: '17'
spec:
  selector:
    matchLabels:
      app: pontusagent
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: pontusagent
    spec:
      volumes:
```

```
        - name: rootfs
          hostPath:
            path: /proc
            type: ''
      containers:
        - name: pontusagent
          image: '{{镜像地址}}/pontusagent:3.1.2.i'  ## 注意这里的镜像地址替换
          env:
            - name: EnableAgentRegistry
              value: 'false'
            - name: GaeaRegistryServers
              value: ${agentregistry vip}     ## 注意这里的 ${registry vip} 的替换
            - name: FilterDaemonSet
              value: 'false'
          resources:
            limits:
              cpu: '1'
              memory: 1Gi
            requests:
              cpu: '0'
              memory: '0'
          volumeMounts:
            - name: rootfs
              readOnly: true
              mountPath: /rootfs
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          imagePullPolicy: IfNotPresent
          securityContext:
            privileged: true
            procMount: Default
      restartPolicy: Always
      terminationGracePeriodSeconds: 30
      dnsPolicy: ClusterFirst
      automountServiceAccountToken: false
      hostNetwork: true
      hostPID: true
      securityContext: {}
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: machine.sigma.alipay.com/no-daemonset
                    operator: DoesNotExist
      schedulerName: default-scheduler
      tolerations:
        - operator: Exists
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 5
  revisionHistoryLimit: 10
status:
  currentNumberScheduled: 162
```

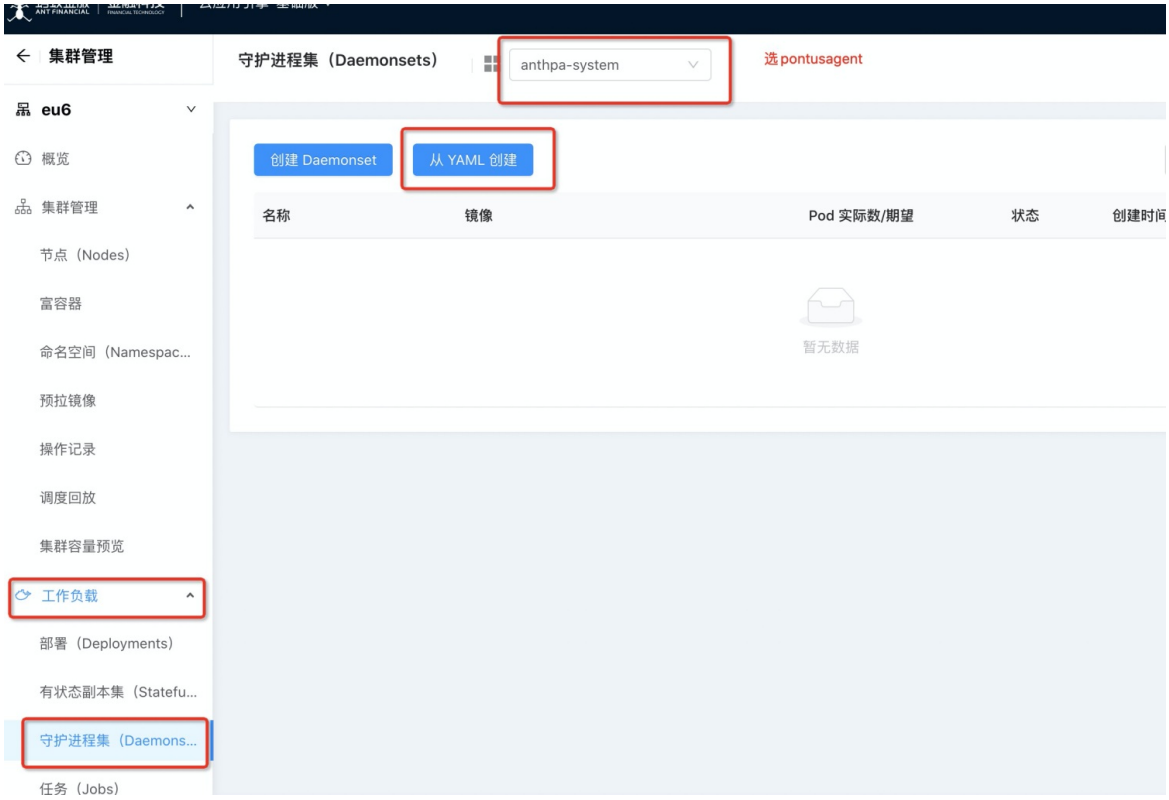
```
  numberMisscheduled: 0
  desiredNumberScheduled: 162
  numberReady: 157
  observedGeneration: 9
  updatedNumberScheduled: 1
  numberAvailable: 157
  numberUnavailable: 5
```

注意

- 需要将 `${agentregistry vip}` 替换为前面获取到的 agentregistry LB 的真实地址。确保用户态集群可以访问该 VIP，首先验证是否可以通过 telnet 访问 ${agentregistry vip} 9025。
- 关于镜像地址，每个环境的用户态都不一样，需要替换成对应环境的镜像地址，并将 pontusagent 镜像上传到对应的镜像中心上（前面核心态的镜像中心一般都是统一的，无需更换）。

3. **创建 Daemonsets。**

   找到对应类型的集群 Agent YAML 创建守护进程集（Daemonsets）。



3.5.4.1.2.2. AKS 部署场景

AKS 部署场景的步骤与 LHC 部署场景的相同，详情请参见 LHC 部署场景。

3.5.4.2. 经典发布部署安装

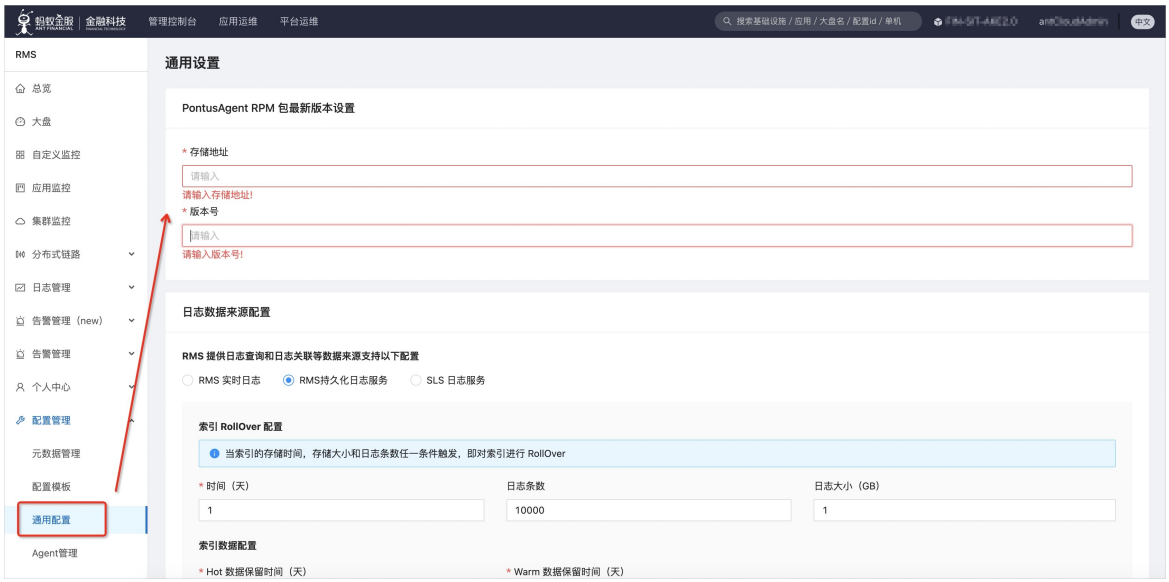
1. 上传 Agent 的 RPM 包到 OSS，获取 OSS 的下载链接，需要将 RPM 包设置为公共读。

> **说明**
>
> Agent 域内下载地址：http://yum.tbsite.net/taobao/7/x86_64/test/pontus-agentd/pontus-agentd-3.1.2.f-20210810232642.x86_64.rpm

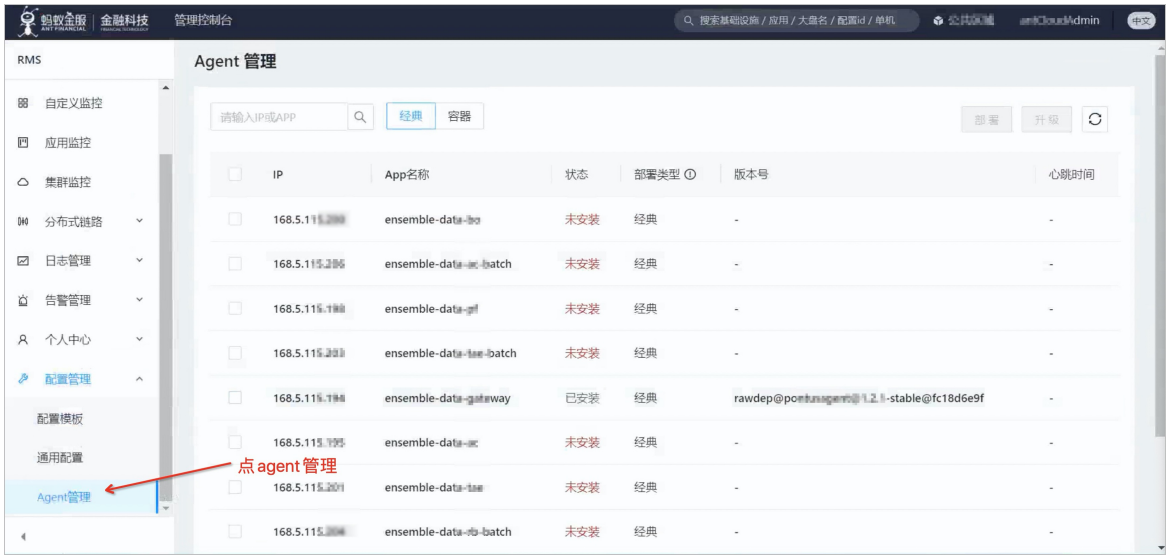
2. 登录控制台，在左侧导航栏选择 **通用设置**。
3. 在 **通用设置** 页，完成以下配置。
   - **存储地址**：OSS 下载链接地址。

     > **注意**
     >
     > 请务必保证该链接在业务机器可正常运行 wget。

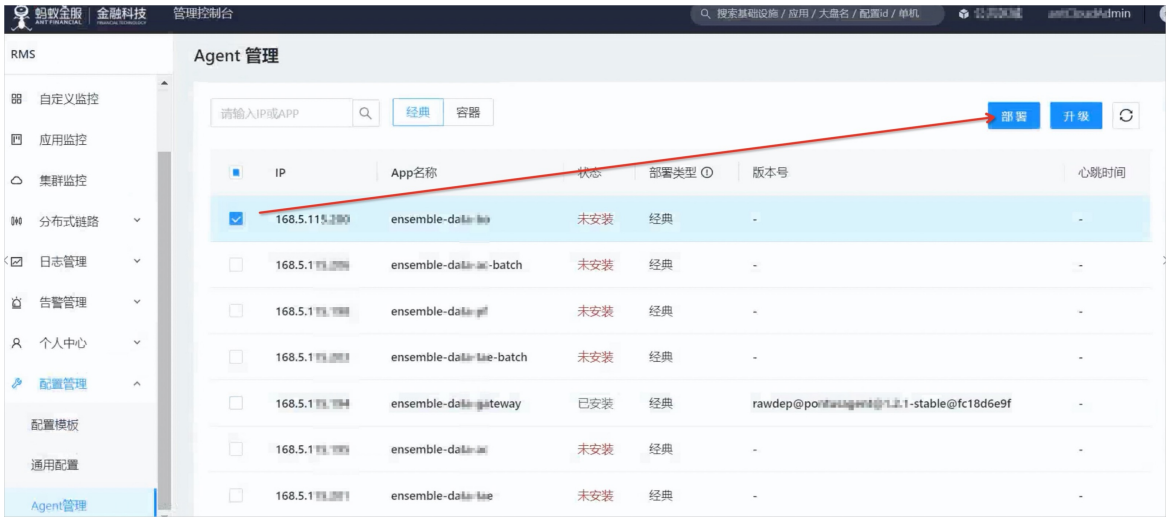
   - **版本号**：输入版本号，例如 **3.0.0.h**。关于版本号并无强制要求，方便运维即可。



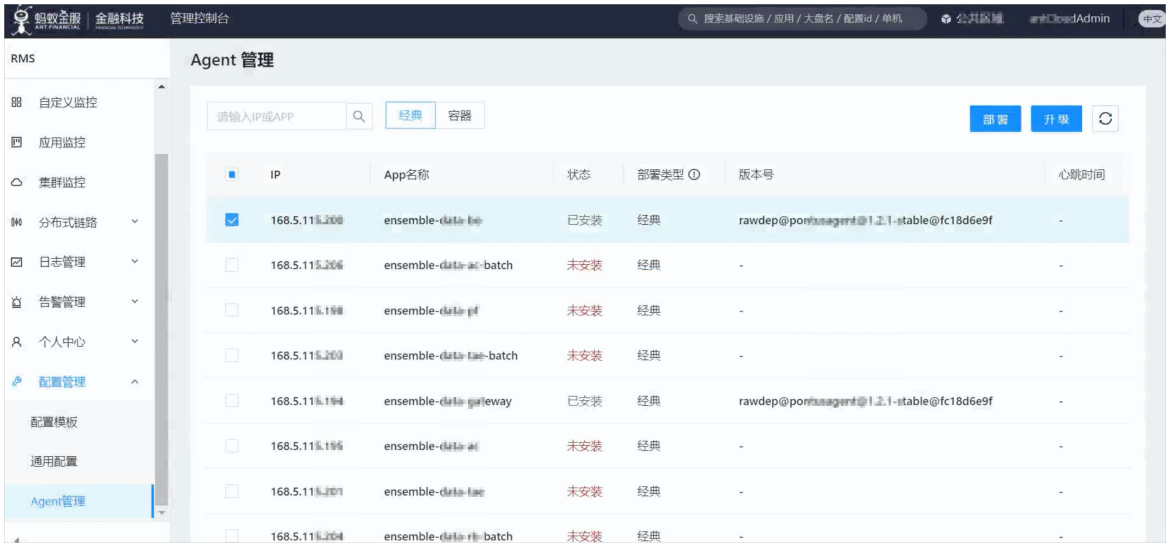
4. 在左侧导航栏上，单击 **配置管理** > **Agent 管理**。



5. 在 **Agent 管理** 页，选择一台机器进行灰度安装验证。



安装成功后，状态会由 **未安装** 变为 **已安装**，且心跳时间正常显示。



6. 验证完成后，勾选全部机器，单击 **部署** 等待安装完成即可。

4.双机房监控部署

4.1. 双机房无容灾部署

4.1.1. 概述

双机房无容灾部署模式，即仅部署一个机房的 BOS，另一个机房只需要在被监控的集群上安装 Agent 即可。

4.1.2. 版本信息

机房 A

产品	所属产品集	拓扑	规格	版本
CERESDBMR	3.21.X	按需选择 说明 请不要选择 **标准双机房**。	按需选择	1.2.12
CERESDBTRACE	3.21.X	按需选择 说明 请不要选择 **标准双机房**。	按需选择	1.1.8
PONTUS	3.21.X	按需选择	按需选择	3.12.0
BOS	3.21.X	按需选择	按需选择	3.21.X

机房 B

产品	所属产品集	拓扑	规格	版本
PONTUS	3.21.X	标准单机房部署-vessel	按需选择	3.12.0

4.1.3. 部署顺序

机房 A

按如下顺序进行部署：

1. CERESDBMR
2. CERESDBTRACE
3. PONTUS
4. BOS

机房 B

机房 B 只需部署 PONTUS 的一个组件 pontusvessel 即可。

4.1.4. 产品部署

机房 A 部署方式与单机房基本一致，具体请参见 单机房监控部署 中的 产品集部署。

机房 B 只需部署 PONTUS 的一个组件 pontusvessel 即可。

4.1.5. Agent 部署

BOS 双机房的 Agent 部署方式和单机房一致，请参见 单机房监控部署 中的 部署 Agent。

Agent 双机房链接

为了保证 Agent 的高可用，即让 Agent 同时连接到双机房的 registry 服务上。此时，Agent 需要注入双机房的 registry VIP，同时用 `,` 进行分割。具体为修改 YAML 中的 GaeaRegistryServers 的配置。

```
containers:
-name:pontusagent
image:'acs-reg.alipay.com/monitor-pontus/pontusagent:1.2.1.e'
env:
-name:EnableAgentRegistry
value:'false'
-name:GaeaRegistryServers
value: {{registry vip}}## 此处修改为 registry 的双机房 ip 列表
-name:FilterDaemonSet
value:'true'
```

4.2. 双机房容灾部署

4.2.1. 概述

双机房容灾部署，在两边的机房中都需要部署对应的 BOS 全量组件。

4.2.2. 版本信息

机房 A

产品	所属产品集	拓扑	规格	版本
CERESDBMR	3.21.X	按需选择 说明 请不要选择 **标准双机房**。	按需选择	1.2.12
CERESDBTRACE	3.21.X	按需选择 说明 请不要选择 **标准双机房**。	按需选择	1.1.8
PONTUS	3.21.X	按需选择	按需选择	3.12.0
BOS	3.21.X	按需选择	按需选择	3.21.X

机房 B

B 机房需要部署全量组件。

产品	所属产品集	拓扑	规格	版本
CERESDBMR	3.21.X	按需选择 说明 请不要选择 **标准双机房**。	按需选择	1.2.12

CERESDBTRACE	3.21.X	按需选择 说明 请不要选择 **标准双机房**。	按需选择	1.1.8
PONTUS	3.21.X	按需选择	按需选择	3.12.0
BOS	3.21.X	按需选择	按需选择	3.21.X

4.2.3. 部署顺序

机房 A

按如下顺序进行部署：

1. CERESDBMR
2. CERESDBTRACE
3. PONTUS
4. BOS

机房 B

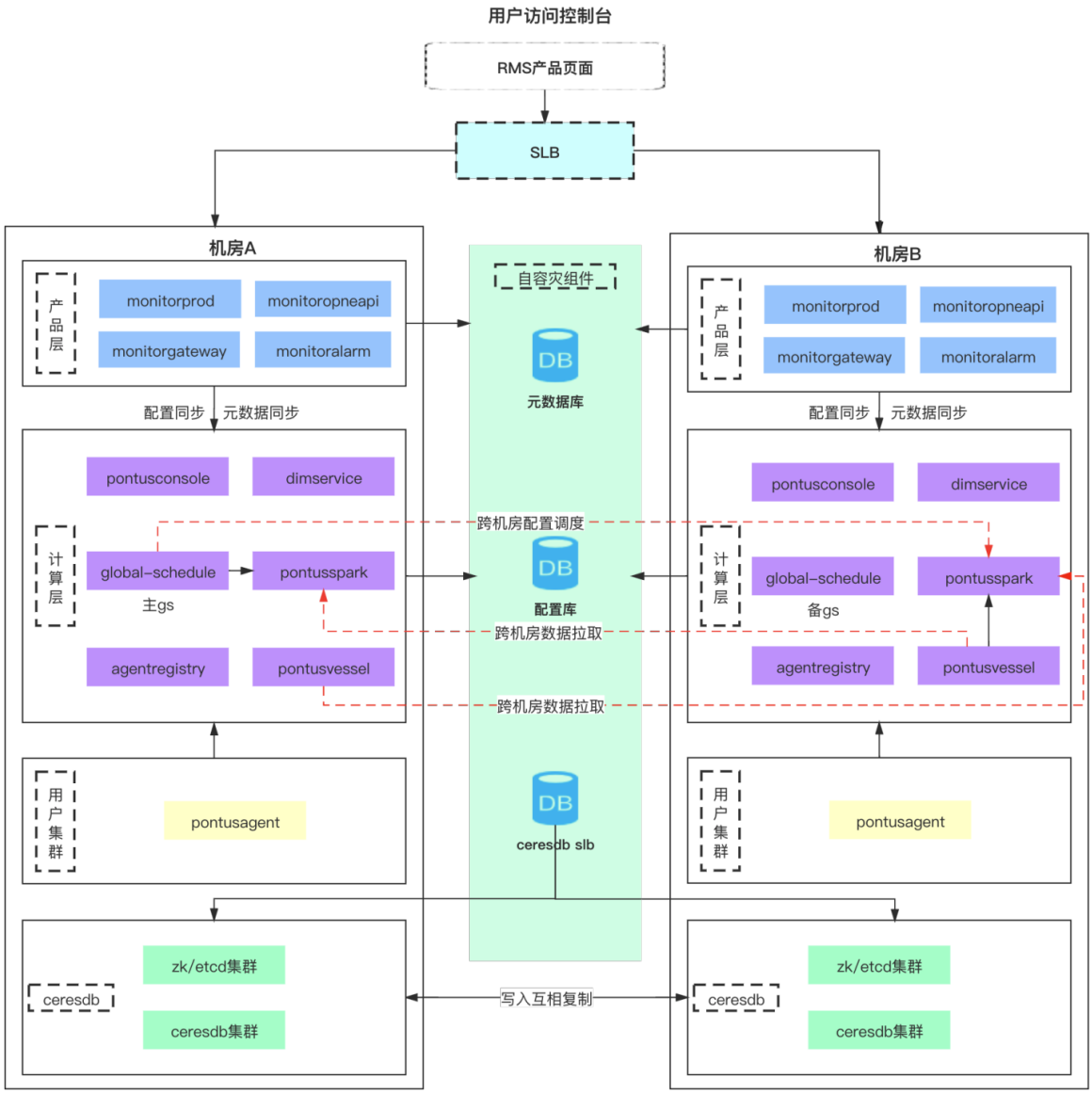
双机房容灾部署中机房 B 的部署顺序与机房 A 的相同，具体可参考 机房 A。

4.2.4. 产品部署

BOS 双机房的部署方式与单机房基本一致，以下内容需要重点关注。

容灾部署架构

双机房场景下，BOS 的所有组件需要双机房部署，BOS 所涉及到的所有 OB 或 MySQL 需要作共库处理。部署架构如下图所示：



双机房容灾参数说明

以下参数是双机房容灾时需要重点关注的项。

应用	Key	参数说明
ceresdbng	ENABLE_REDUNDANT	是否开启冗余读写（容灾）。 需更改为 true。开启之后，必须设置 **REDUNDANT_SLAVE_METRIC_ENDPOINT**。
ceresdbng	REDUNDANT_SLAVE_METRIC_ENDPOINT	指定另一个机房的 ceresdbng 的 VIP 地址，例如 **192.168.1.1:5000**。

ceresdbng	REDUNDANT_METRIC_SYNC_TYPE	双机房读写 sync type，默认值为 **Async**。
ceresdbtrace	CLUSTER_NAME	datanode agent 注册在 ZK 上的二级目录。 • 如果当前为主机房，则填写 **ceresdb_trace_mCluster**。 • 如果当前为备机房，则填写 **ceresdb_trace_sCluster**。
ceresdbtrace	ZK_PEERS	datanode_agent 和 trace proxy 注册的 ZK 地址。 若有容灾机房且容灾方案使用的是跨机房 ZK，则填写跨机房 ZK 集群的 IP 列表，例如 **127.0.0.1:218**、**127.0.0.2:2181**、**127.0.0.3:2181**。
ceresdbtrace	TRACE_ENABLE_FAILOVER	是否开启异步双写。配置为 **true**。
ceresdbtrace	TRACE_SLAVE_NODE_HOSTS	备机房中 ceresdbtrace 的三节点 host 列表。 • 若已经填写 ZK_PEERS，则直接确认即可。 • 若没有 ZK，需要填写该参数。
pontusvessel	DataCenter	idc name，双机房部署时，机房信息要和元数据保持一致。 关于 DataCenter 各机房参数的获取方式，请参见 多机房架构下元数据获取方案。 **注意** 该参数会影响双机房数据采集，必须正确配置。

注意

由于存储层的配置需要关注对端机房的 VIP 信息，因此，双机房发布完成之后，最好能两边重启下 ceresdbng 和 ceresdbtrace 服务。

多机房架构下元数据获取方案

【关键点】：

- 多机房部署场景下，BOS 共库。
- 通过 setup 的 idcNameTransformMapping，将云游部署出来的应用和容器的 idcName 强制转为 PaaS 维护的 idcName。然后，vessle 启动参数内配置的 idcName 也是 PaaS 维护的 idcName。

【获取 PaaS 内的 idcName】：

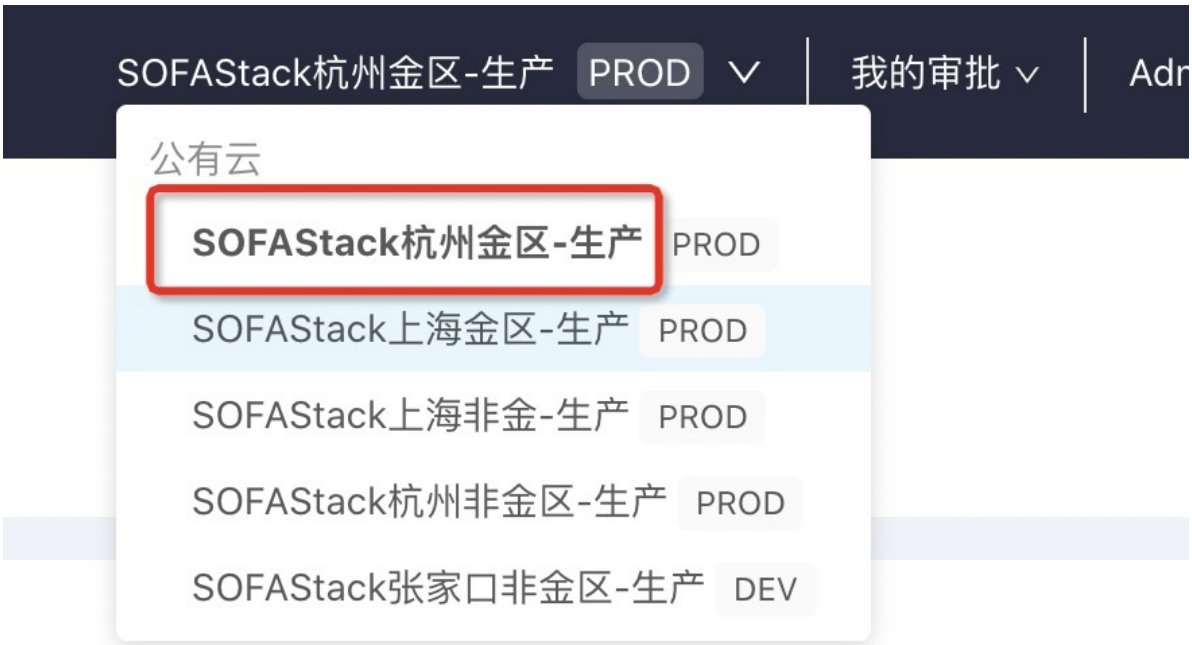
- 进入 metaservice 的容器（appplatform 内的 metaservice 或 basic 内的 metaservice）。
- 调用 `curl -v localhost:8341/privateapi/zones` 。
- 返回的 response 中的 identity 即为监控的 idcName。示例如下：

```
{"data":[{"displayName":"cn-hangzhou-a","iaasProviderId":"0000000002","id":"0000000001","identity":"cn-hangzhou-a","name":"hz","no":1,"regionId":"0000000001","state":"A","utcCreate":"2016-12-15T18:20:02+08:00","utcModified":"2016-12-15T18:20:02+08:00"}],"success":true}
```

【获取云游内的 idcName】：

- **非 LDC 的云游：**

  idcName 即云游 Local 导航头上的名称（见下图）。



- **LDC 的云游：**

  访问 `云游域名/api/ops/envs` ，从获取到的 response 内找到 zones 数组，zones 数组每一项的 name 即为 idcName。

```
            utcModified: "2021-01-12T21:58:13+0800",
          - zones: [
              - {
                  cloudId: "730420210105213612360300000",
                  displayName: "scnxs-a机房",
                  id: "73040000000020",
                  identity: "cn-neimeng-121e-amtest121001-a",
                  name: "scnxs-hgh-scnxa",
                  paascoreZoneName: "scnxs-hgh-scnxa",
                  regionId: "R_202101052142353277",
                  stack: "ALIYUN",
                  status: "ACTIVE",
                  utcCreate: "2021-01-12T21:58:09+0800",
                  utcModified: "2021-01-12T21:58:09+0800",
                  zoneId: "Z730420210106092848862908629"
                },
              - {
                  cloudId: "730420210105213612360300000",
                  displayName: "scnxs-b机房",
                  id: "73040000000021",
                  identity: "cn-neimeng-121e-amtest121001-a",
                  name: "scnxs-hgh-scnxb",
                  paascoreZoneName: "scnxs-hgh-scnxb",
                  regionId: "R_202101052142353277",
                  stack: "ALIYUN",
                  status: "ACTIVE",
                  utcCreate: "2021-01-12T21:58:09+0800",
                  utcModified: "2021-01-12T21:58:09+0800",
                  zoneId: "Z730420210106092911096800968"
                },
              - {
                  cloudId: "730420210105213612360300000",
                  displayName: "scnxs-c机房",
                  id: "73040000000022",
                  identity: "",
                  name: "scnxs-hgh-scnxc",
                  paascoreZoneName: "scnxs-hgh-scnxc",
                  regionId: "R_202101052142353277",
                  stack: "ANT_STACK",
                  status: "ACTIVE",
                  utcCreate: "2021-01-12T21:58:09+0800",
                  utcModified: "2021-01-12T21:58:09+0800",
                  zoneId: "Z730420210106092927039100391"
                }
            ]
        }
    ],
```


域名挂载

以下产品的 LB 需要挂载到同一个域名下（1 个域名 2 个 A 记录）。

- monitorprod LB
- monitorgateway LB

4.2.5. Agent 部署

双机房容灾部署中的 Agent 部署方式与双机房无容灾部署中的相同，具体可参考 Agent 部署。

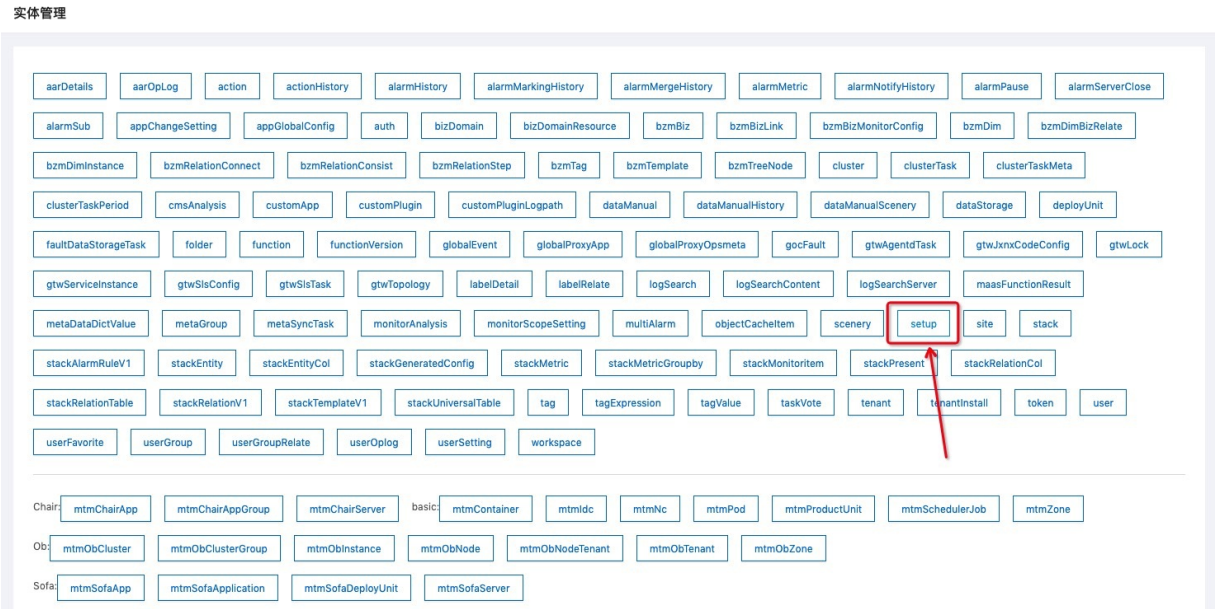
5.产品初始化

5.1. 元数据初始化

5.1.1. setup 配置

配置步骤

访问 `http://rms.{domain}/boss/manager` 页面。然后单击 **setup**，打开配置。



- 如果 setup 未配置，单击 **创建**，新增 setup 配置。
- 如果 setup 已配置（即，表内有一条记录），单击对应该记录的编辑。添加如下的配置：

```
{
"monitorDomain""http://rmsdomain",
"taskCMDBAksKube"true
}
```

注意事项

- 填入 setup 中的 json 需要先压缩为一行，然后再填入。
- 参数可能需要更改，具体如下表所示。

参数名	是否需要更改	参数说明
monitorDomain	可能需要更改	监控地址，替换为当前站点的信息，且确保可解析。
taskCMDBAksKube	可能需要更改	**是否开启云原生监控**。 如果部署了 AKS 或者 LHC，则配置为 true。否则，无需填写。

5.1.2. 原生监控初始化（可选）

如果需要容器集群监控，可以选择配置原生监控初始化。

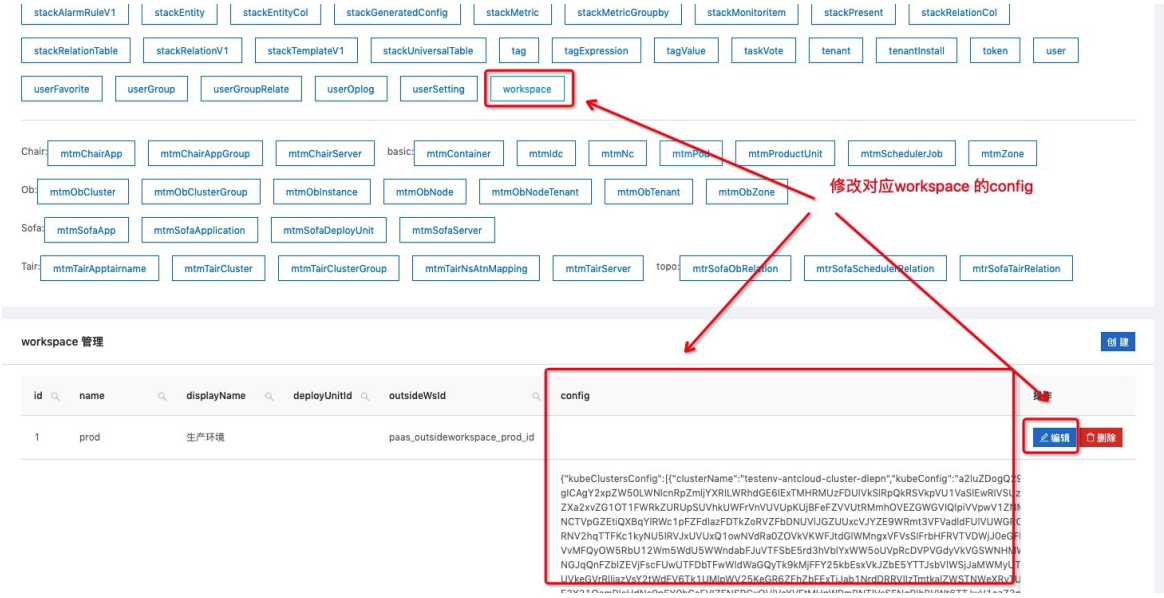
自动初始化

> **说明**
>
> 如果是存量站点、公有云站点，跳过本节内容，请勿作任何改动。

- **用户态：**
  - 若为 ACK 集群，在上文提到的 setup 的 json 值中新增 `"enableFetchUserKubeConfigWhenNotFound":true` 。
  - 若不是 ACK 集群（比如，物理机集群），请参考 手动初始化。
- **核心态：**
  - 若云游版本为 Yunyou Plus 版本（AntstackPlus 底座），则在上文提到的 setup 的 json 值中新增 `"enableFetchKernelKubeConfigWhenNotFound":true` 。
  - 若云游版本不是 Yunyou Plus 版本，请参考 手动初始化。

手动初始化

1. 首先需要从站点管理员处获取云原生配置所需要的三个必备信息：
   - 客户集群的名称（name）
   - 关联的工作空间（workspace）
   - K8s 集群的配置文件（kubeconfig）
2. 获取到 kubeconfig 后，对其执行 base64encode，例如 `cat ./admin.kubeconfig | base64 -w 0` 。
3. 打开 http://rms.{domain}/boss/manager 页面。然后单击 **workspace**。



4. 打开对应的获取的 workspace ，单击 **编辑**，修改其中的 config：
   - **clusterName**：替换为前面获取到三个必备信息中的集群 name。
   - **kubeConfig**：替换为前面的 base64encode 值。
5. 最后，配置 json 格式的 config：

```
{"kubeClustersConfig":[{"clusterName": "刚才第1步的clusterName","kubeConfig":"刚才第2步的kubeConfig"}]}
```

> **说明**
>
> 如果有多个 workspace 共享一个对客集群，只需配置其中一个 workspace 的 config 字段即可。

5.2. 告警初始化

5.2.1. 基本配置

在云游上找到一台 monitoropenapi 的机器，通过 bash 前往机器上执行：

```
curl--location--requestPOST'http://localhost:8080/putAuthorityEntity'\
--header'Content-Type: application/json'\
--header'Access-Key: e95a1139-2278-4a2b-91c2-51e9aa10'\
--header'Access-Id: f95326b0-2e48-4a29-bafe-8826d7e9'\
-d'{
"description": "antmonitor",
"name":"name",
"accessKey":"241d0d2c-7908-4cf6-83cf-61e042cc",
"accessId":"69907e7c-1732-429f-a009-8063b749"
}'
```

> 说明
>
> 上面的请求参数不需要做任何更改，直接复制调用即可。

5.2.2. 基于 Gotone 的告警配置（可选）

配置钉钉渠道

通过参数中心配置

如果站点输出了参数中心，则打开参数中心，找到 PS_gotone_params。创建钉钉渠道配置：

- **key**：DD_ROBOT
- **value**：{"webhook":"https://oapi.dingtalk.com/robot/send"}

单击 **DRM 推送**，使配置生效。

通过数据库配置

如果环境中没有输出参数中心，可以通过操作 Gotone 的数据库来配置。Gotone 的应用名为 acgotonecore。

```
insert into system_params(param_key, param_value)values("DD_ROBOT",'{"webhook":"https://oapi.dingtalk.com/robot/send"}');
```

5.3. 自监控配置导入

5.3.1. 概述

BOS 不仅负责监控业务应用和核心态应用，而且还监控自身服务的稳定性，因此在站点部署完成之后，还需要导入自监控配置。

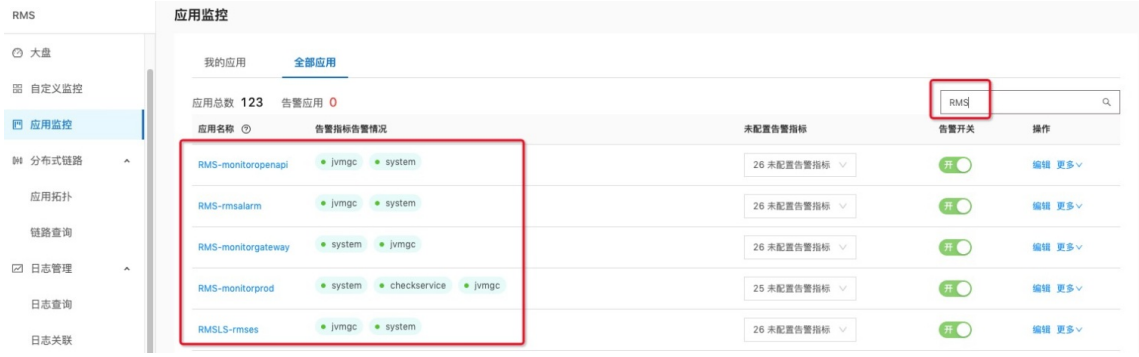
5.3.2. 前置条件

进行自监控配置导入前，需要满足以下条件：

- 元数据已完成初始化。

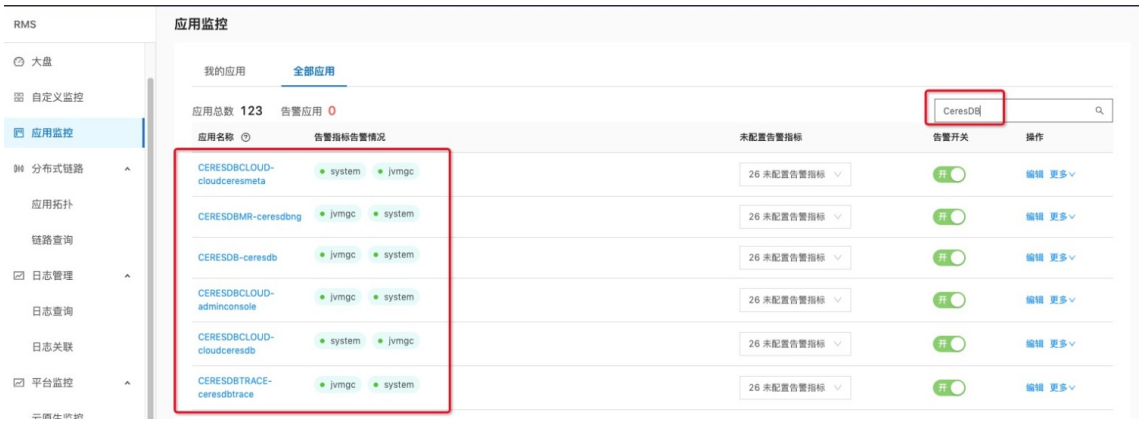
  在 BOS 核心态的应用列表内可以找到对应应用，具体步骤如下：

  i. 在搜索框内输入 **RMS** 可以找到对应应用，如下图所示。



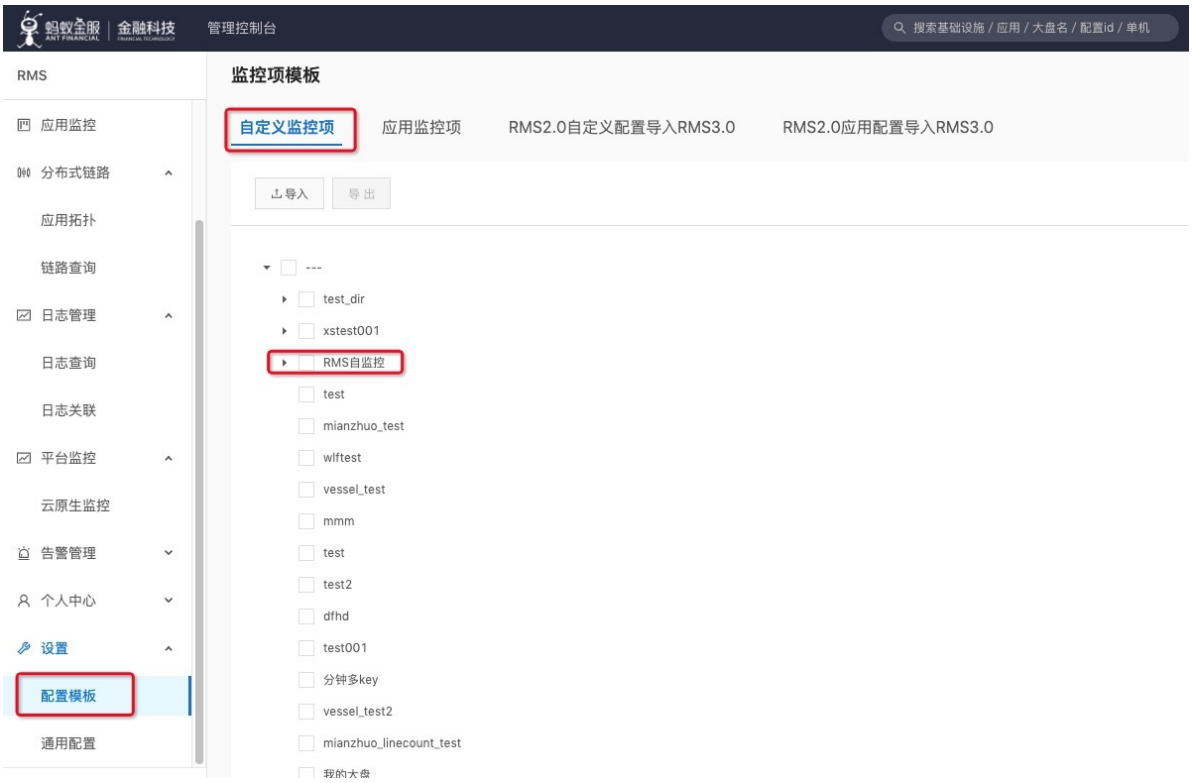
  ii. 在搜索框内输入 **PONTUS** 可以找到对应应用，如下图所示。

  iii. 在搜索框内输入 **CeresDB** 可以找到对应应用，如下图所示。



- 自监控数据尚未导入。

  如果 监控项模板 页有自监控选项（见下图），则不需要导入。



5.3.3. 准备工作

1. 下载 RMS 自监控配置 20210701.zip。
2. 将下载的压缩包解压到本地目录下，将获得 RMS-custom-monitor.json、RMS-application-monitor.json 两个文件。

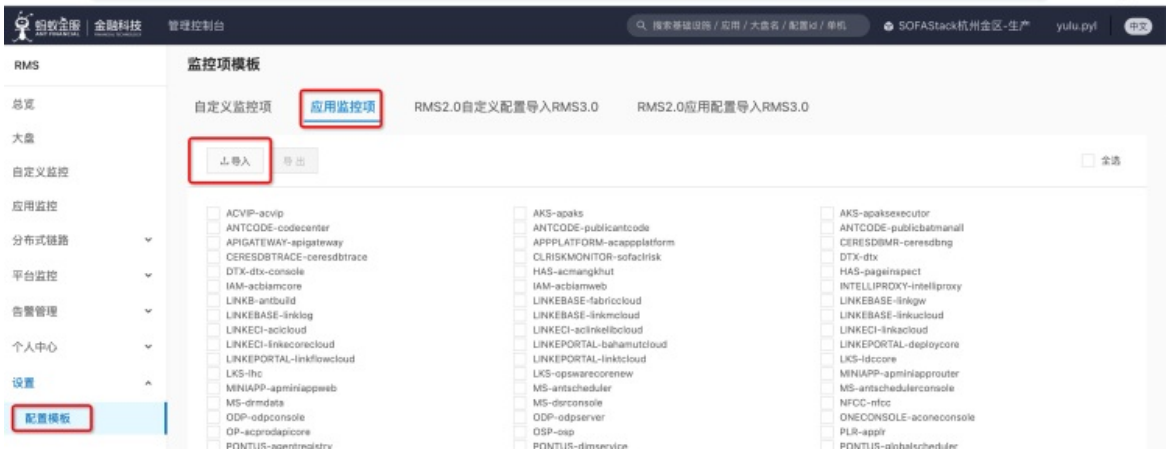
5.3.4. 配置导入

1. 登录控制台。
2. 选择核心态租户 PaaS **租户** 及核心态工作空间。
3. 单击 **设置** > **配置模板**，进入 监控项模板页。
4. 单击 **自定义监控项** > **导入**，然后导入 RMS-custom-monitor.json 文件。

> **说明**
>
> 文件获取方式请参考 准备工作。

5. 单击 **应用监控项** > **导入**，然后导入 RMS-application-monitor.json 文件。

> **说明**
>
> 文件获取方式请参考 准备工作。



6.产品升级

BOS 产品的升级为兼容升级，大致步骤如下：

1. 从云游 Global 导出解决方案。
2. 在 Local 中生成发布单。
3. 进行镜像升级。

7.产品验证

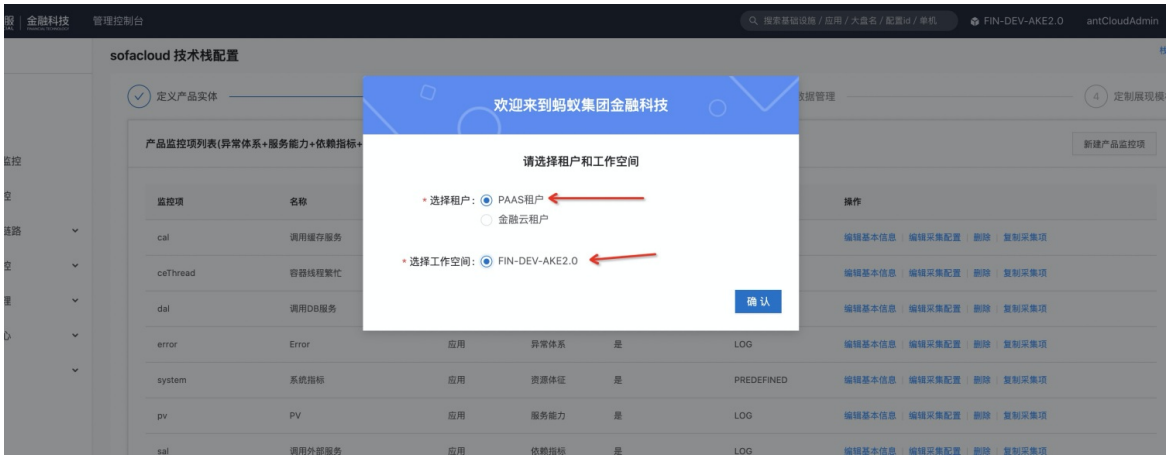
7.1. 核心态监控验证

7.1.1. 应用监控

应用监控从应用总览、系统指标以及 JVM 指标三个方面进行验证。

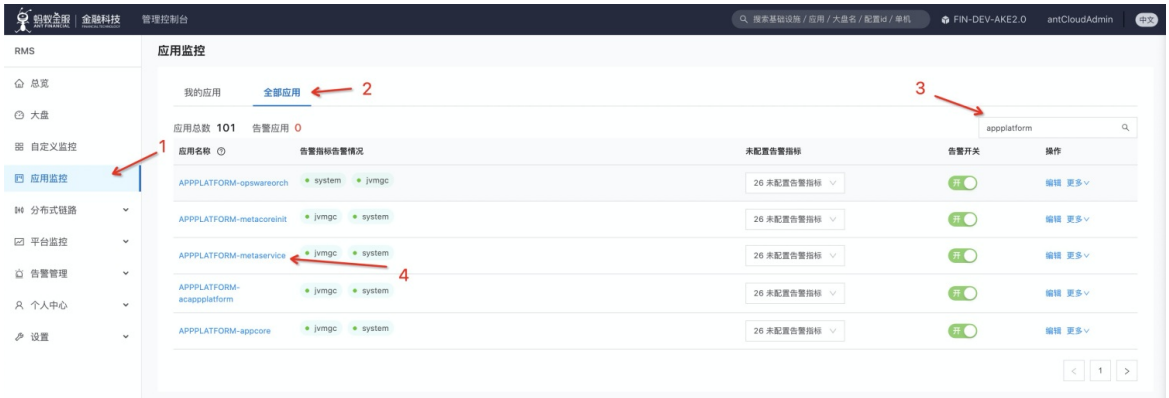
1. 登录控制台，输入核心态账号 antCloudAdmin。
2. 在顶部导航栏右上角选择 **PAAS 租户**，选择工作空间，如下图所示。

> 说明
>
> 关于如何选择工作空间，可前往云游 Local 进行查看。



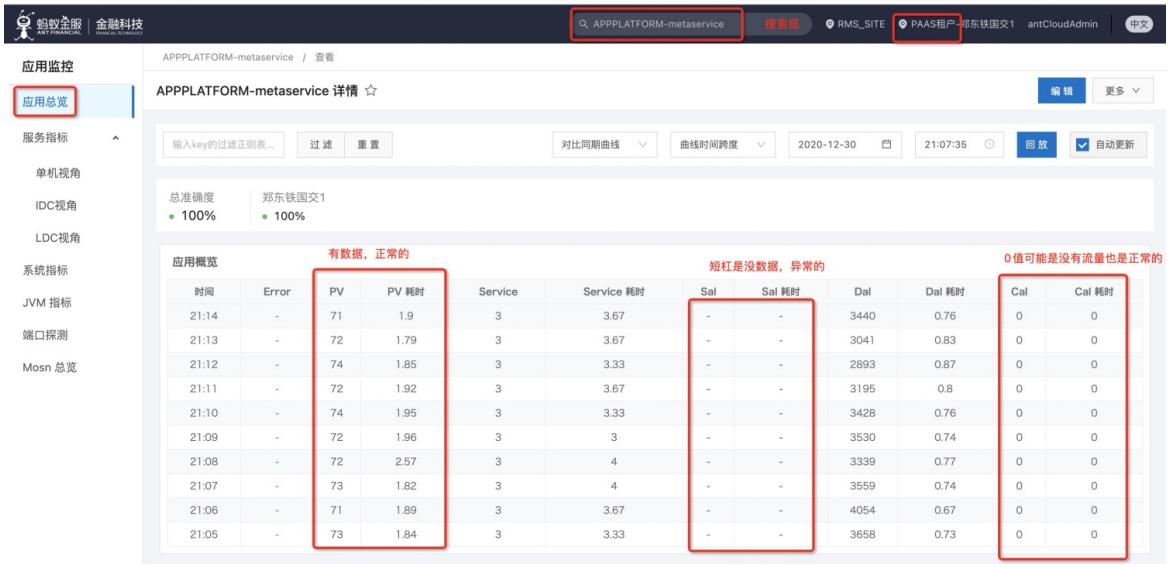
3. 在左侧导航栏上，单击 **应用监控** 进入应用监控页面。
4. 单击 **全部应用** 页签，选择 APPPLATFORM-metaservice 应用进行验证。

> 说明
>
> - 可通过右侧搜索框查找 APPPLATFORM-metaservice 应用。
> - 如果 APPPLATFORM-metaservice 应用不存在，可通过 aciamcore 应用进行验证。



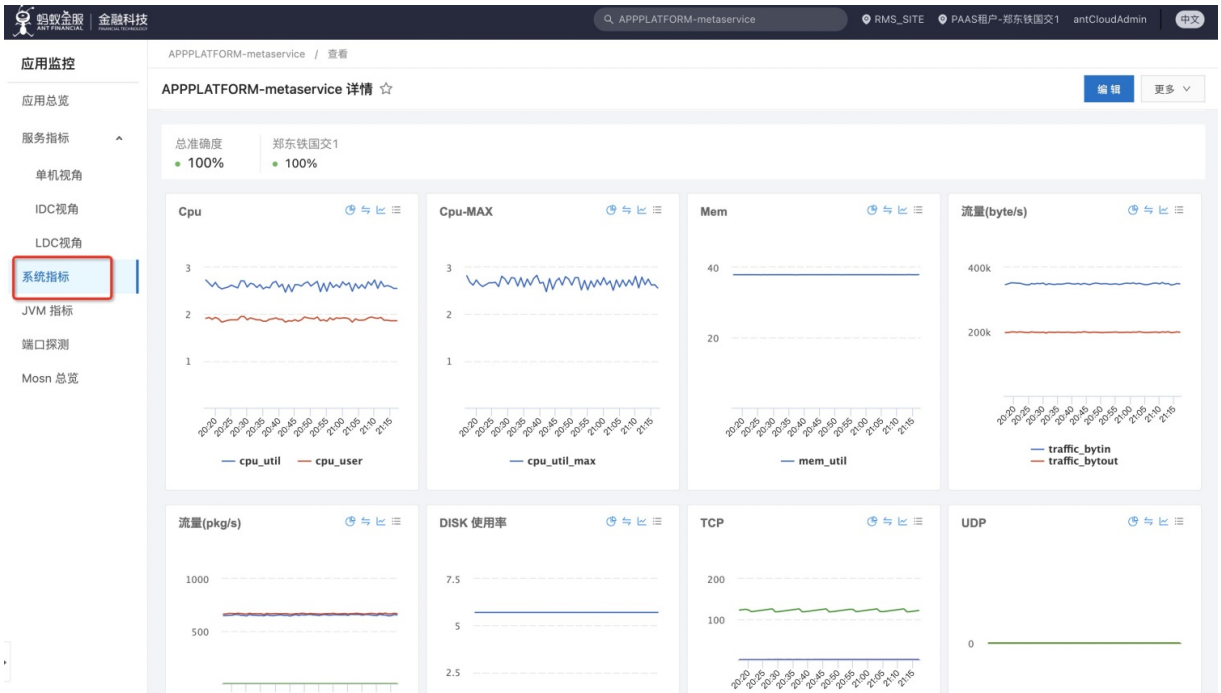
应用总览

1. 在左侧导航栏上，单击 **应用总览**，进入应用概览页。
2. 查看 PV 和 PV 耗时是否有数据。
   - 如果 PV 和 PV 耗时有数据，则说明功能正常。
   - 数据存在三种情况：非零、零以及“-”。“-”表示异常，数据缺失。



系统指标

在左侧导航栏上，单击 **系统指标**。CPU、memory、流量、disk 等指标数据正常，则表示系统指标功能正常。



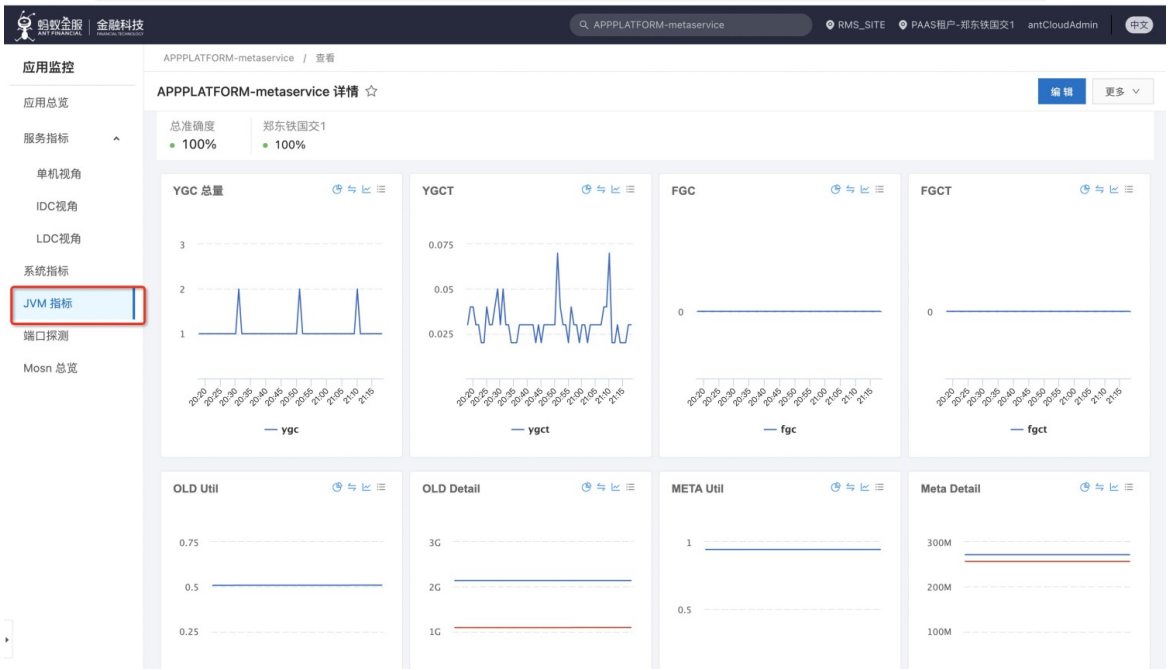
JVM 指标

1. 在左侧导航栏上，单击 **JVM 指标**。

2. 观察 YGC、FGC、Old Util、Meta Util 指标情况。

   如果有数据，则表示 JVM 指标正常。

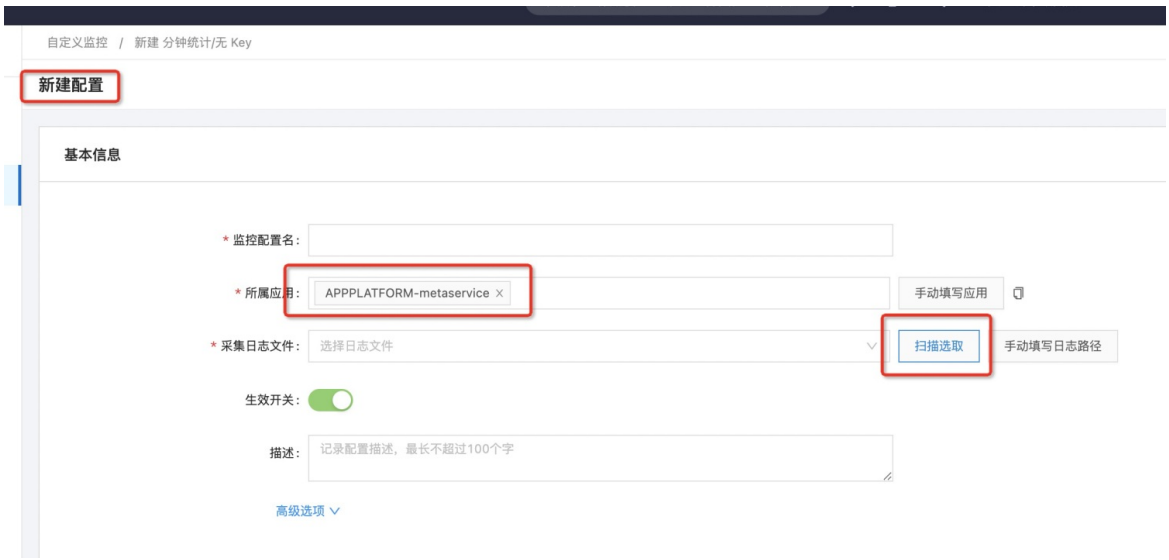
> **说明**
>
> - YGC：Young GenerationGarbage Collection，新生代垃圾收集。
> - FGC：Full Garbage Collection，全堆范围垃圾收集。
> - Old Util：年老代使用率。



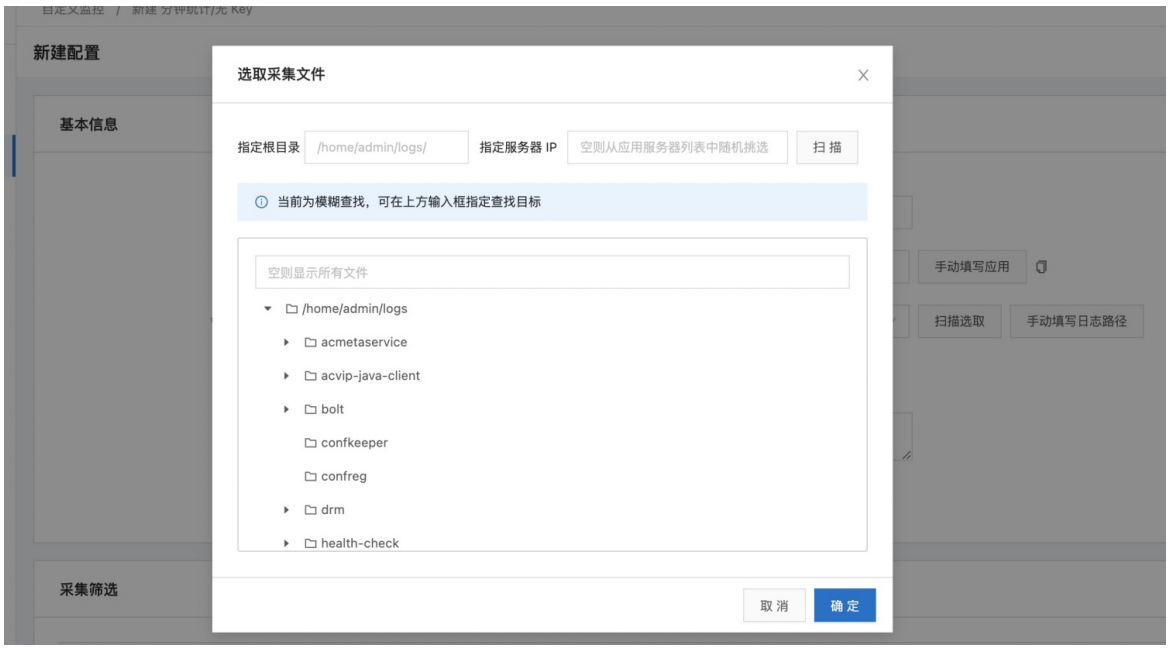
7.1.2. 自定义监控

扫描日志

1. 单击 **新建配置**。
2. 选择 **APPPLATFORM-metaservice**。
3. 单击 **扫描选取**。



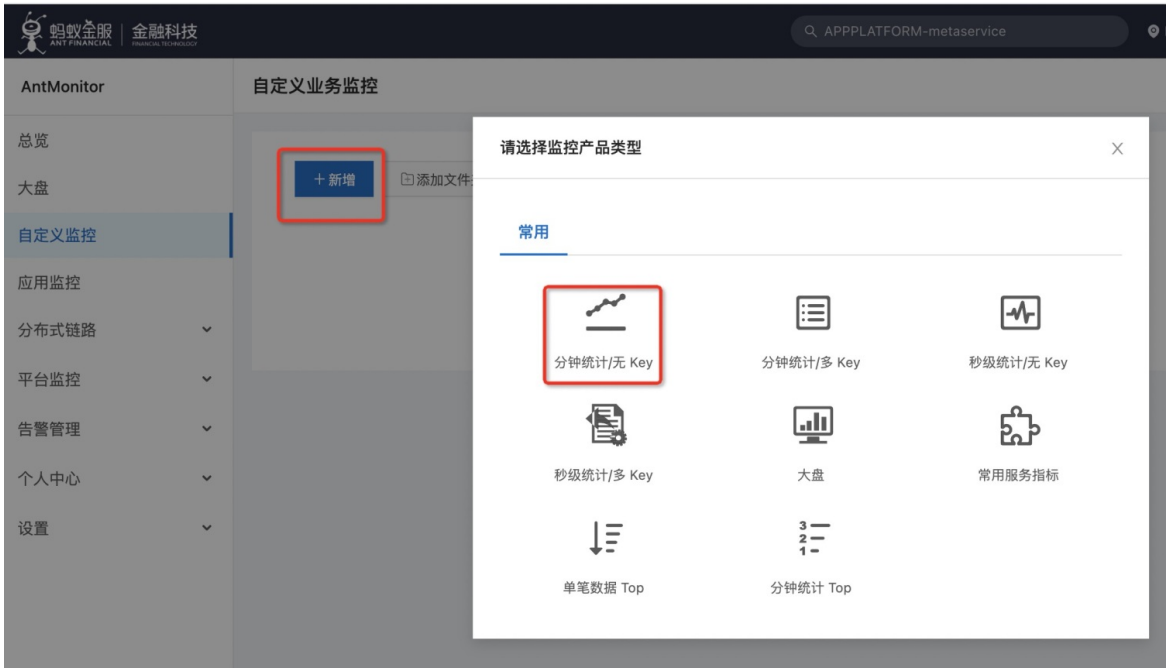
执行完上述步骤后，如果能看到日志 `/home/admin/logs` 的日志文件，则表示扫描日志功能正常。



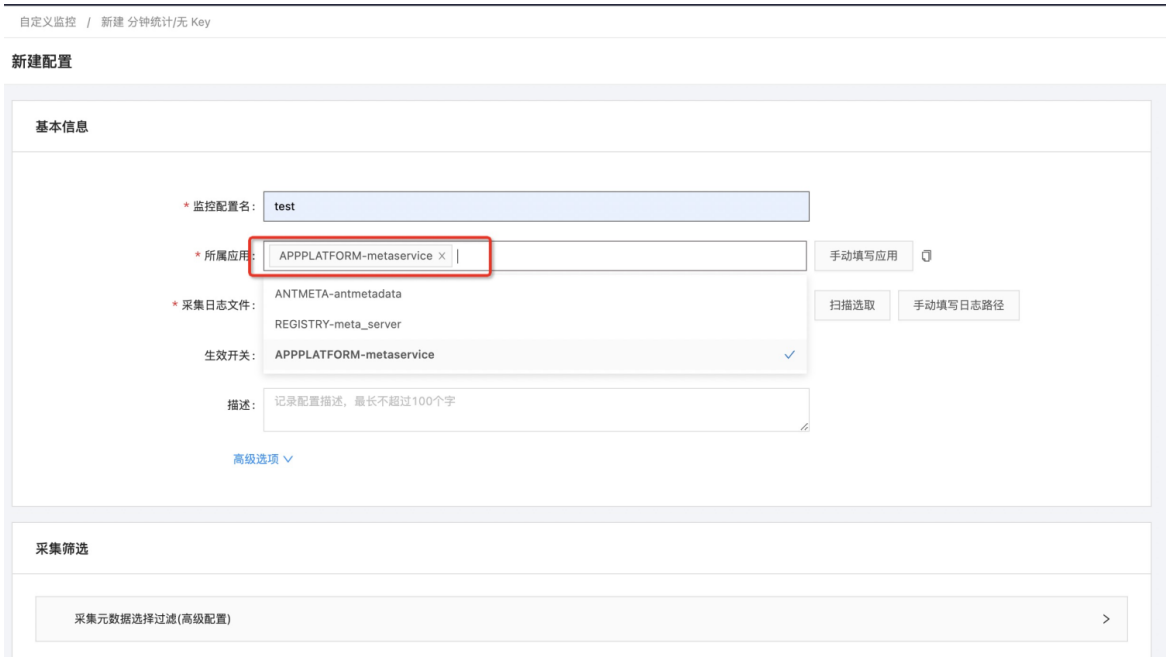
新增自定义监控配置

1. 单击左侧导航栏中的 **自定义监控**，进入 **自定义业务监控** 页面。
2. 单击 **新增**。

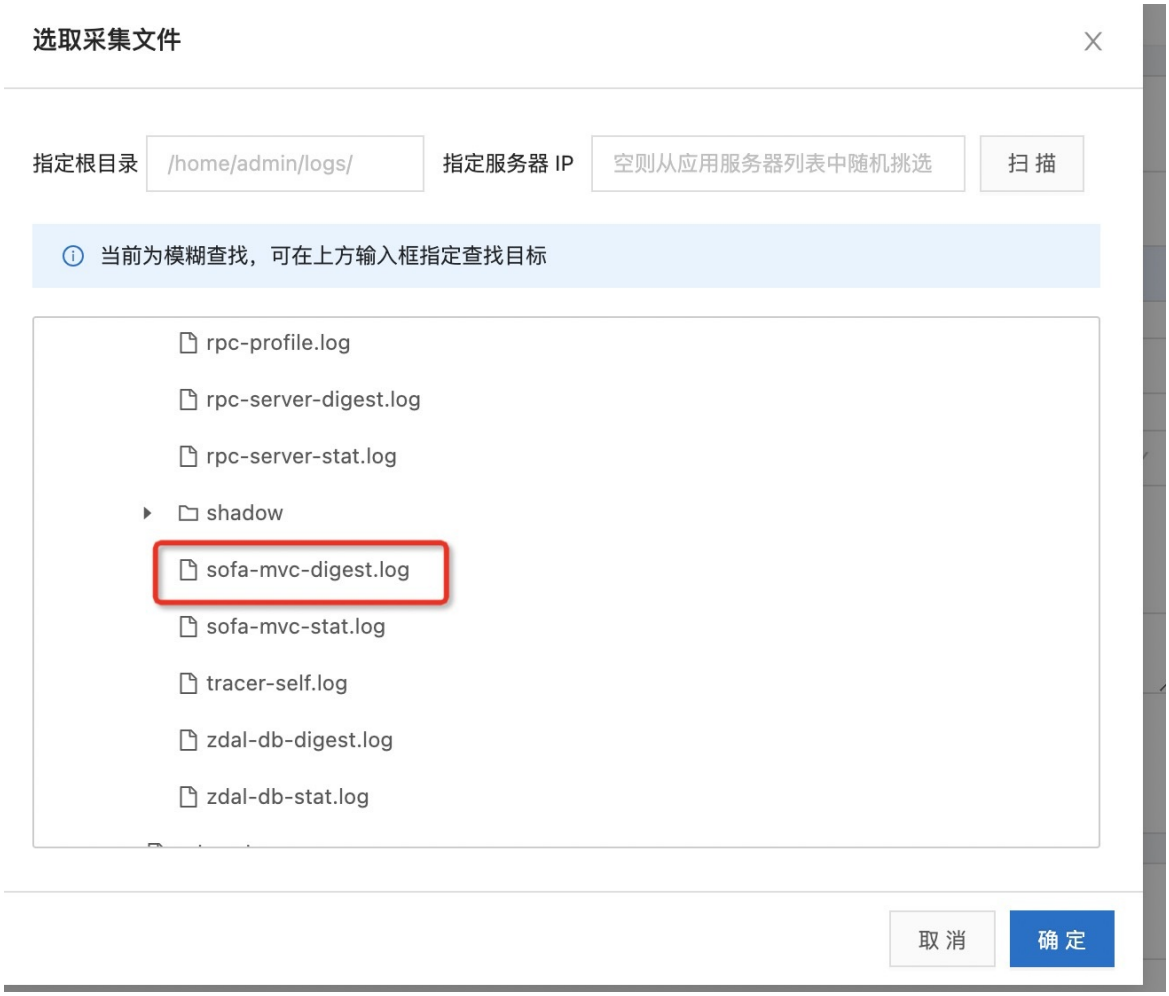
3. 选择 **分钟统计/无 key**。

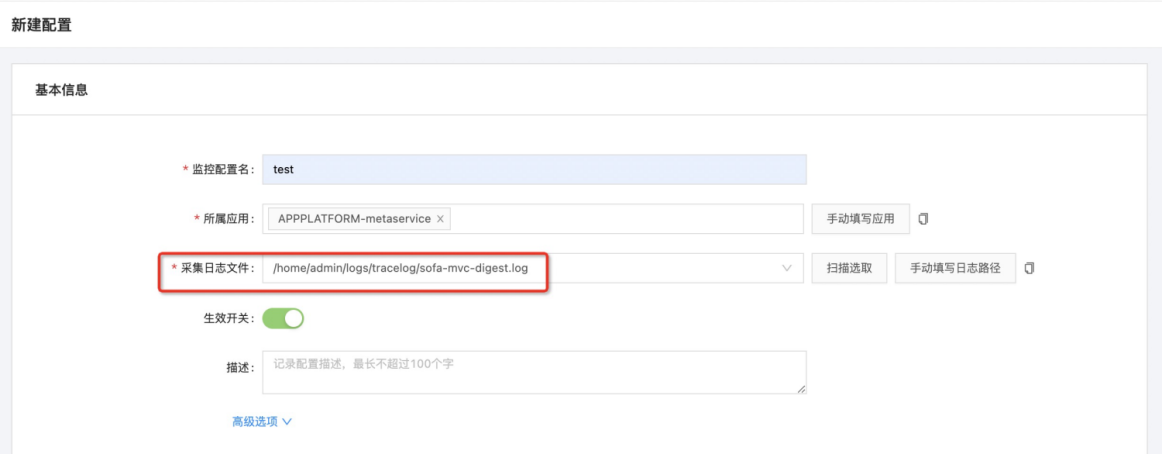


4. 选择所属应用。

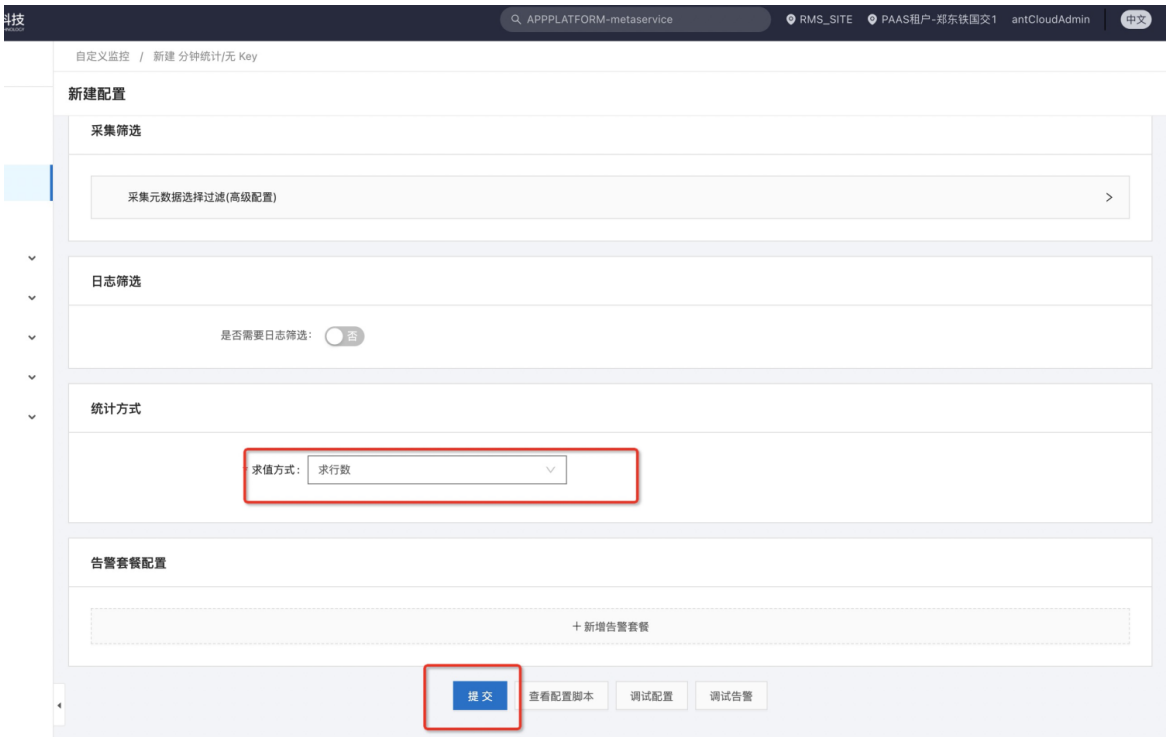


5. 选择日志文件 /home/admin/logs/tracelog/sofa-mvc-digest.log。

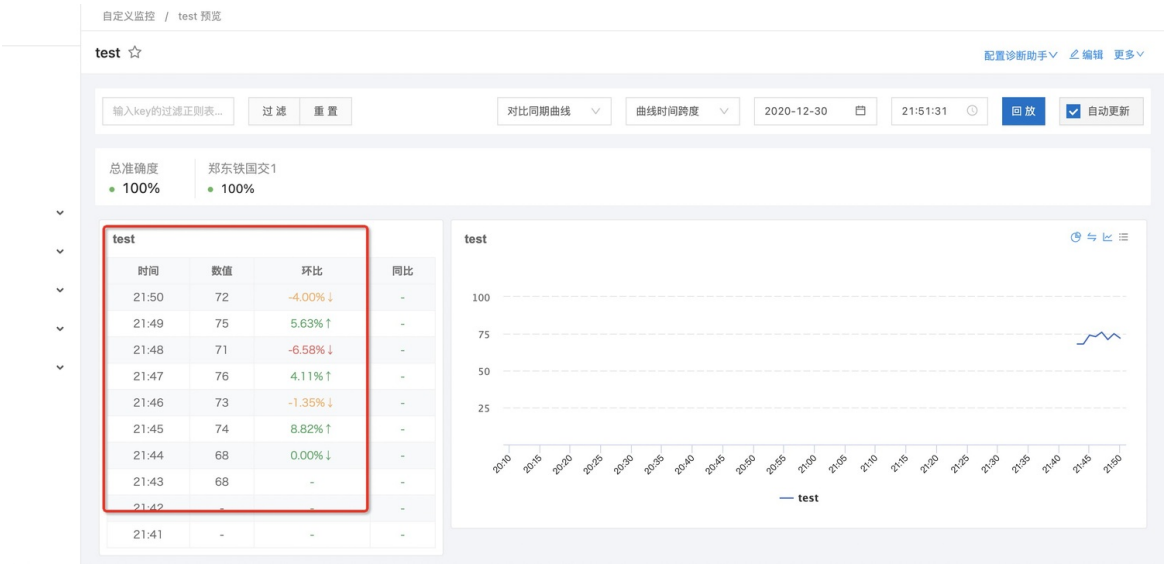




6. 单击 **提交**。

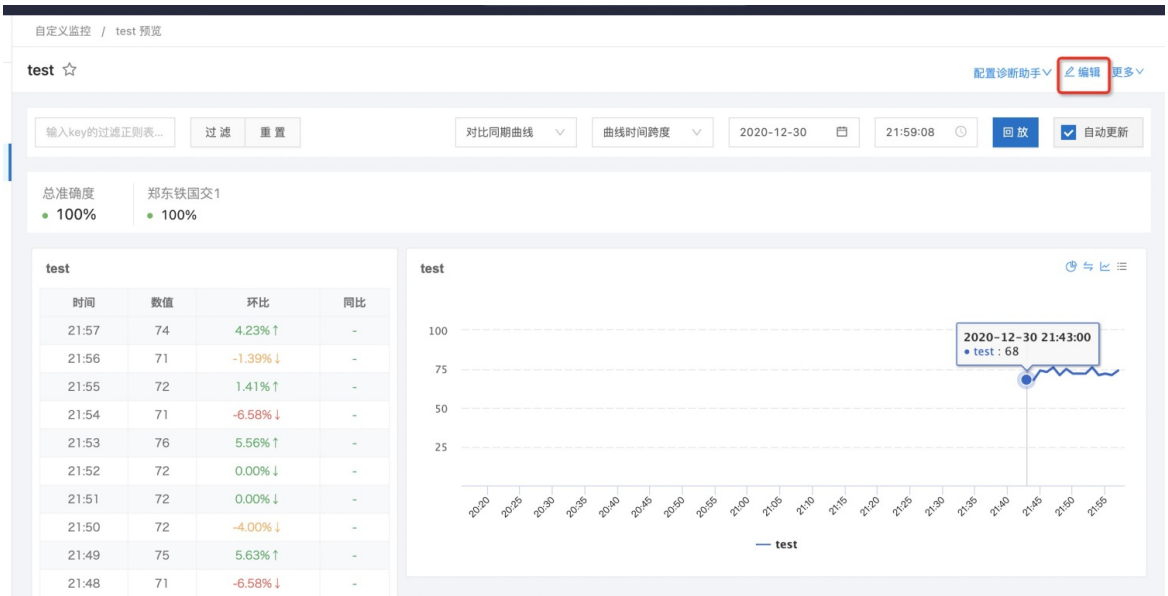


大约 5 分钟后观察数据，如果有数据，则表示自定义监控功能正常。

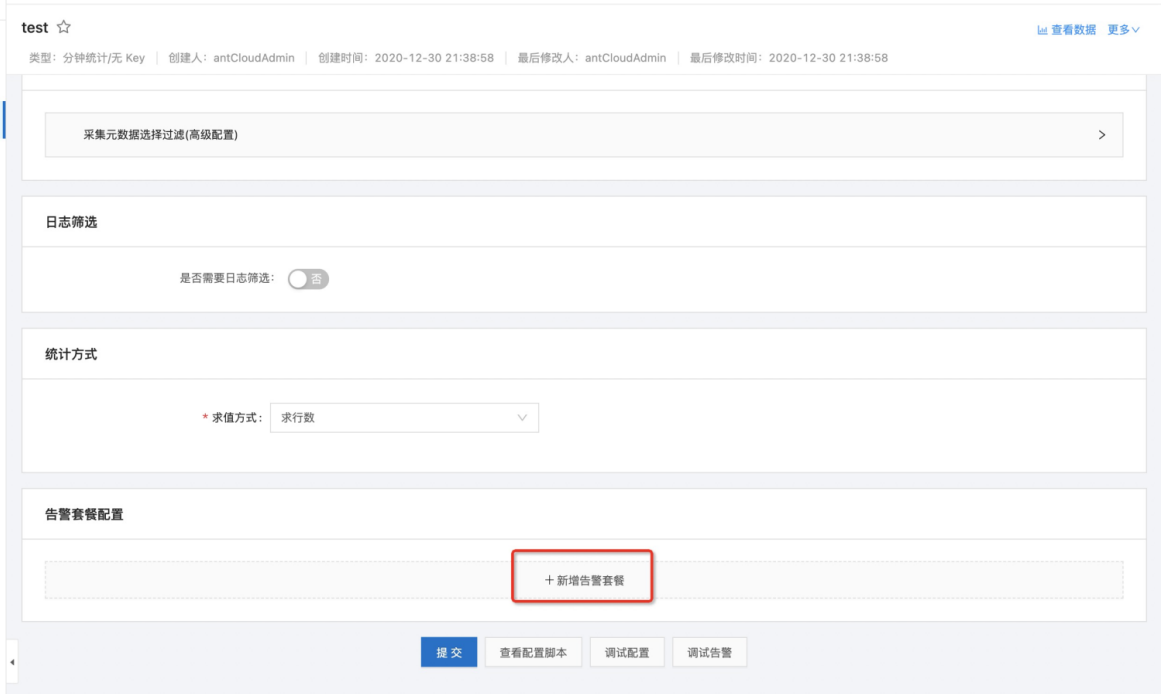


7.1.3. 告警计算验证

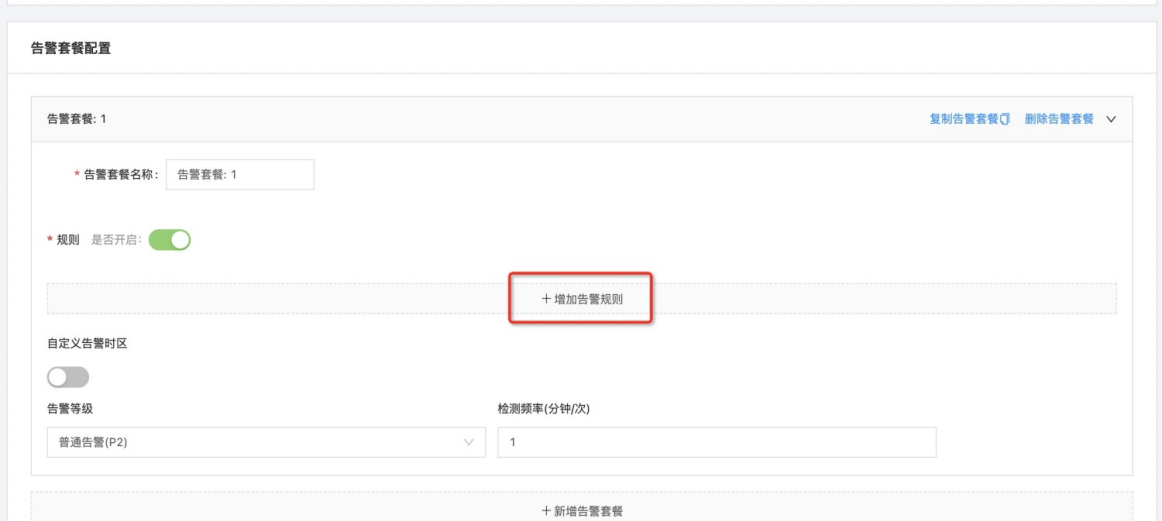
1. 基于上述步骤的配置，新增告警配置，单击右上角 **编辑**。



2. 单击 **+新增告警套餐** 新增告警套餐。



3. 单击 **+ 增加告警规则**。



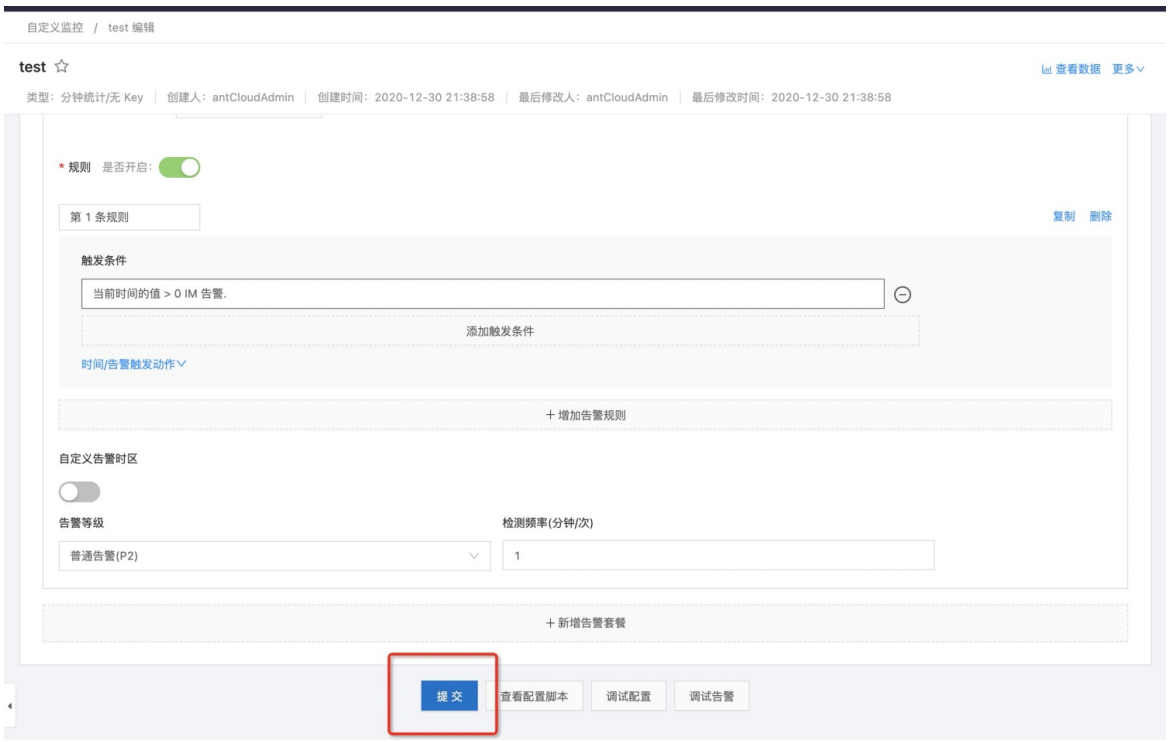
4. 单击 **添加触发条件**。

i. 打开 **告警触发条件** 页面后，选择 **取值方式**、**对比阈值方法**。

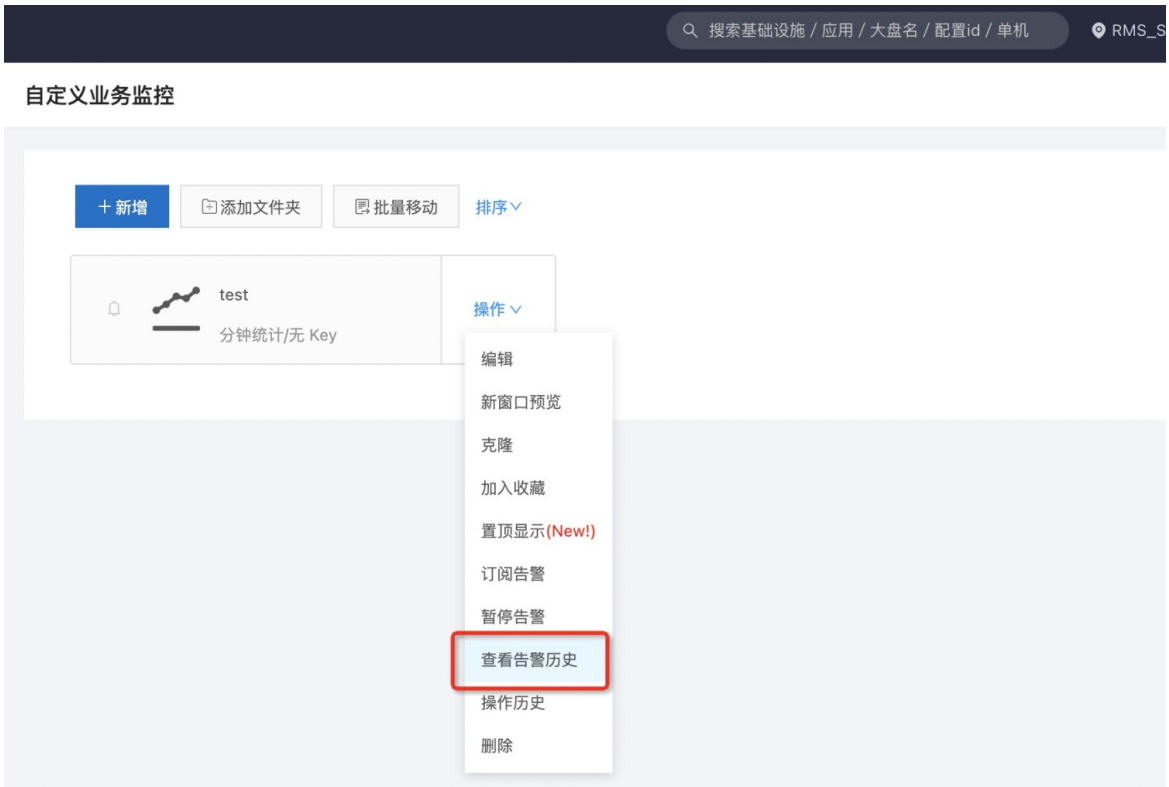
ii. 填写 **钉钉阈值**。

iii. 单击 **确定**。

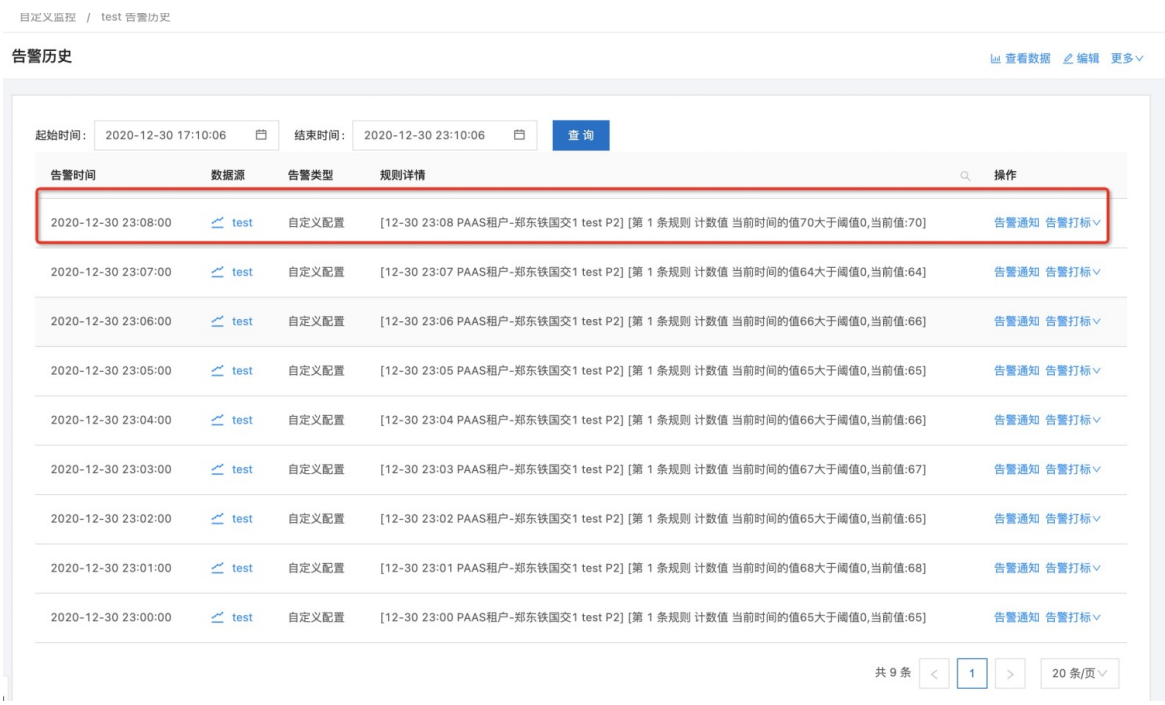
5. 单击 **提交**。



6. 提交后等待五分钟观察告警数据，单击 **查看告警历史**。



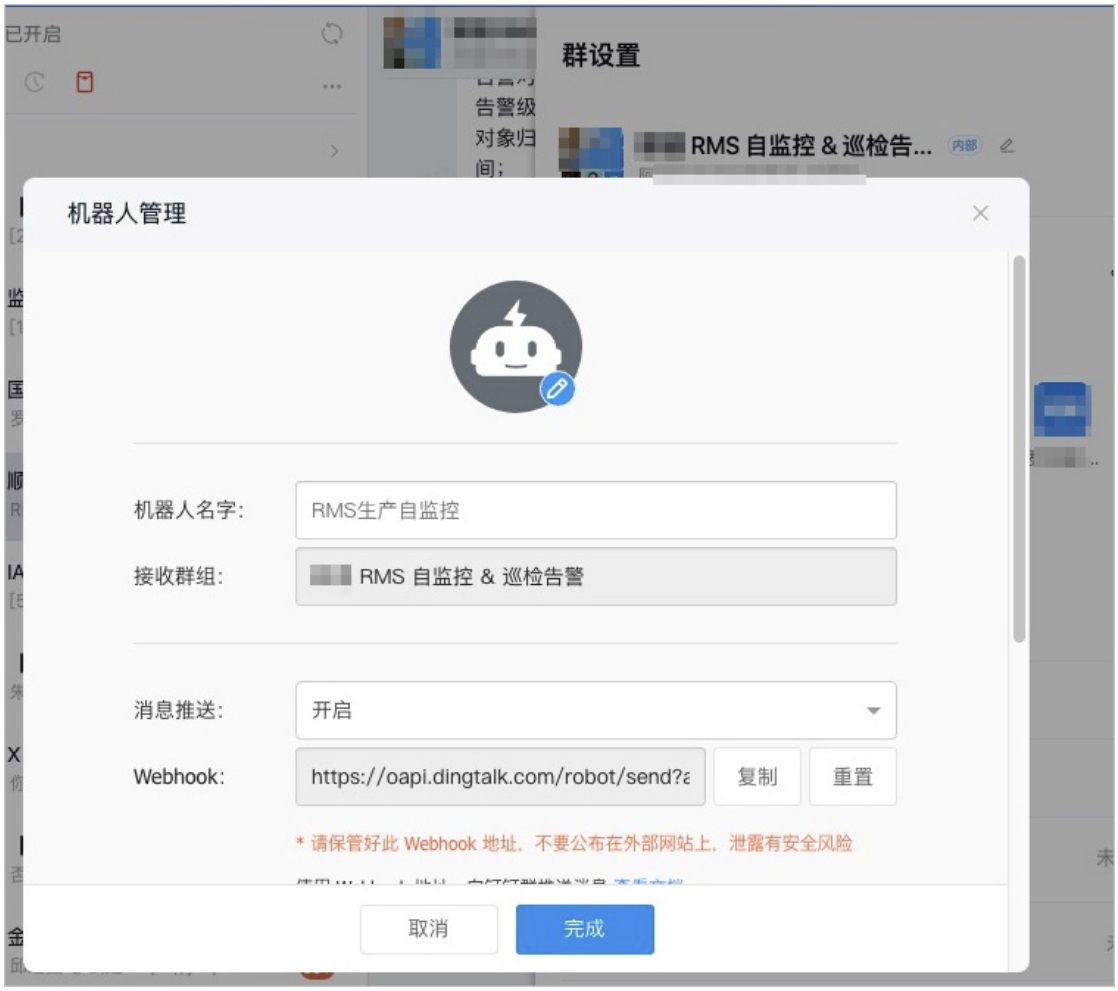
如果告警历史有数据，则表示告警功能正常。



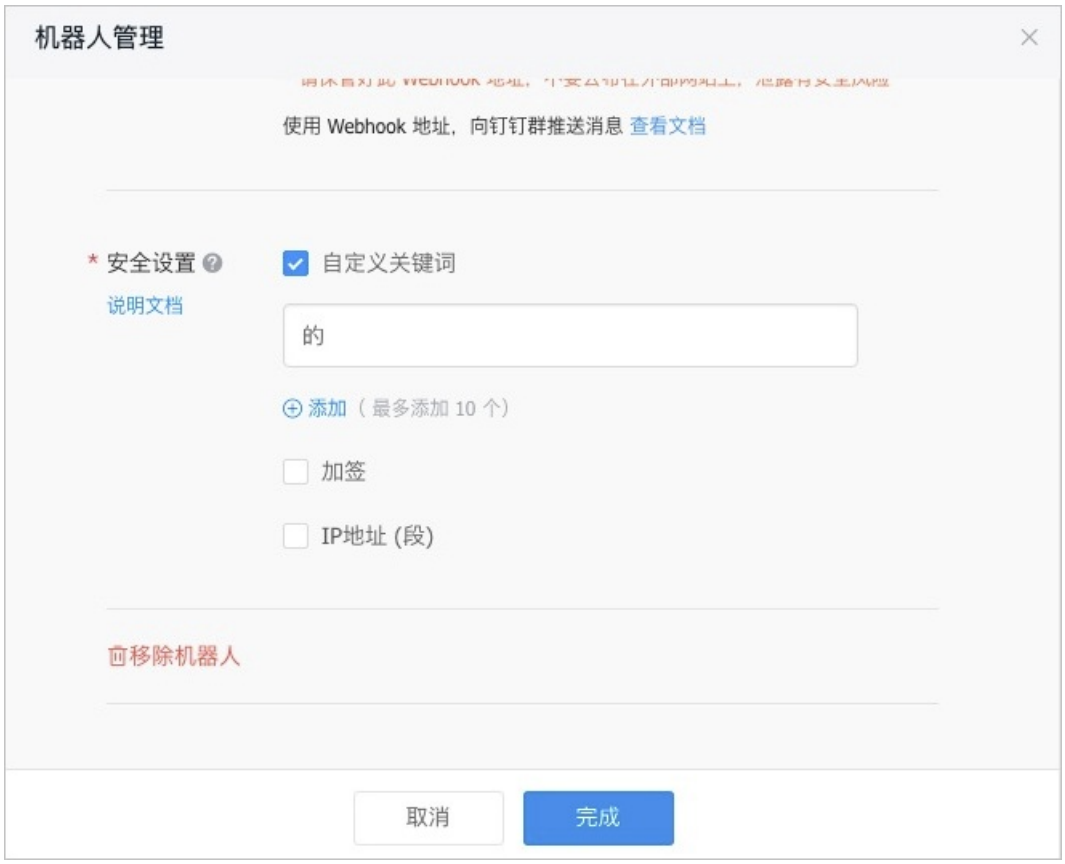
7.1.4. 告警推送验证

如果客户需要钉钉告警推送，则需要推送告警。本文介绍如何进行告警推送验证。

1. 获取自定义机器人 Webhook。
    i. 选择需要添加机器人的群聊，单击 **群设置** > **智能群助手**。


已开启
群设置
RMS 自监控 & 巡检告...
内部
机器人管理
机器人名字:
RMS生产自监控
接收群组:
RMS 自监控 & 巡检告警
消息推送:
开启
Webhook:
https://oapi.dingtalk.com/robot/send?a
复制
重置
* 请保管好此 Webhook 地址，不要公布在外部网站上，泄露有安全风险
取消
完成

ii. 在 **安全设置** 中，完成必要的关键词设置。本例中输入 `的` 关键词。

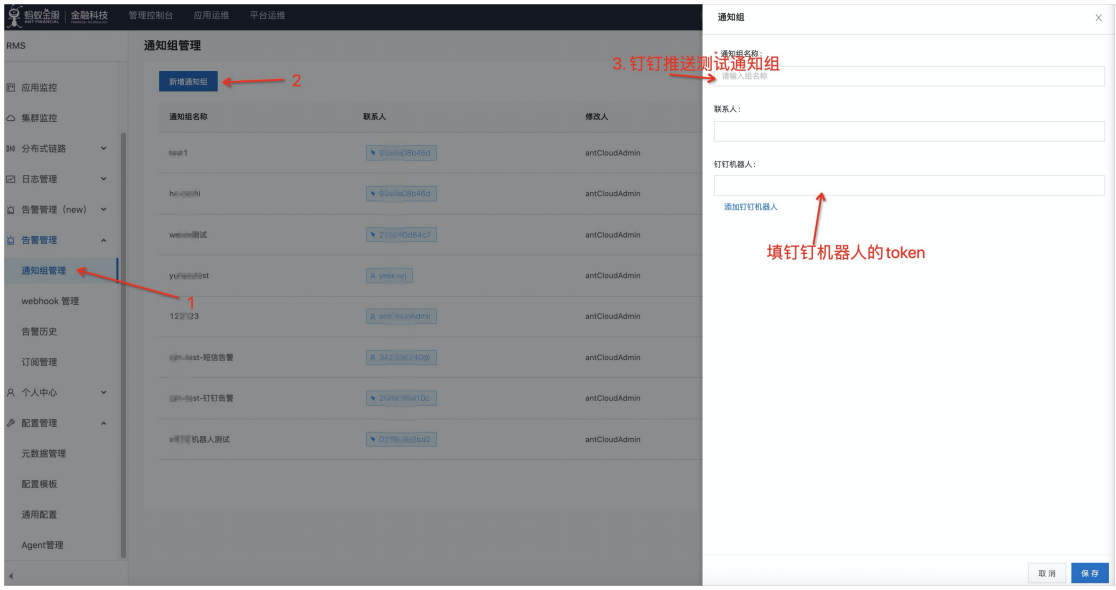


2. 告警配置。

   i. 登录实时监控控制台。

   ii. 在左侧导航栏上，单击 **告警管理 > 通知组管理**。

   iii. 在 **通知组管理** 页，单击 **新增通知组**，在弹框中完成相关配置（如下图所示）后，单击 **保存**。



   iv. 在左侧导航栏上，单击 **自定义监控**。

v. 单击所添加的监控产品右侧 **操作** > **订阅告警**。

vi. 在弹出的 **订阅告警** 页面上添加订阅告警信息。

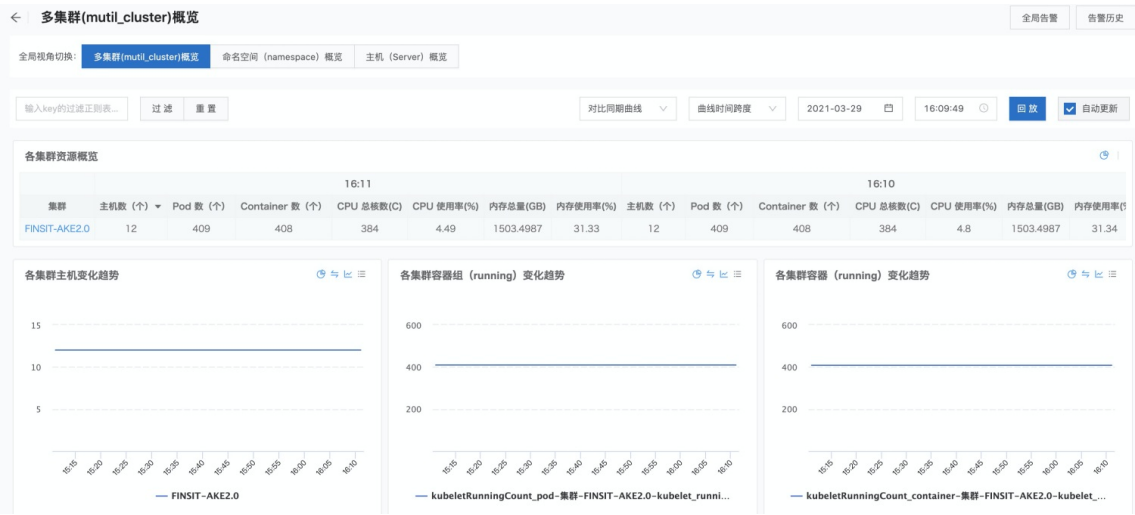
3. 告警推送验证。

当告警触发时，您将在设置接收告警的钉钉群中收到一则告警通知。

7.1.5. 云原生监控

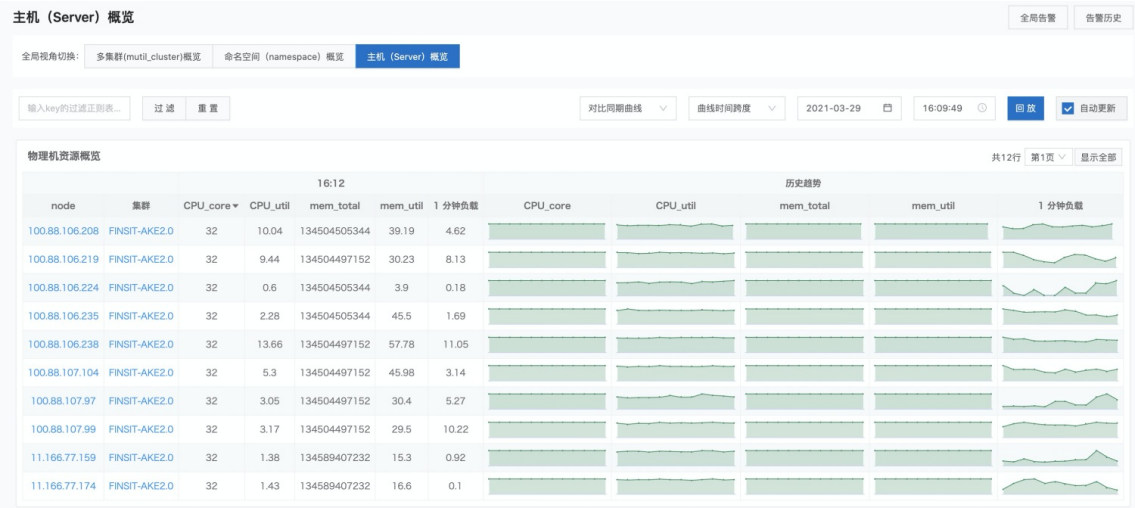
1. 在左侧导航栏上，单击 **云原生监控**。

2. 单击 **多集群（mutil_cluster）概览**、**命名空间（namespace）概览**、**主机（Server）概览** 页签，查看每个页面的数据是否正常。如果每个图表内的数据均非 0、非“-”，则表示功能正常。
   - 正常的多集群（mutil_cluster）概览示例：



- 正常的命名空间（namespace）概览示例：

- 正常的主机（Server）概览示例：



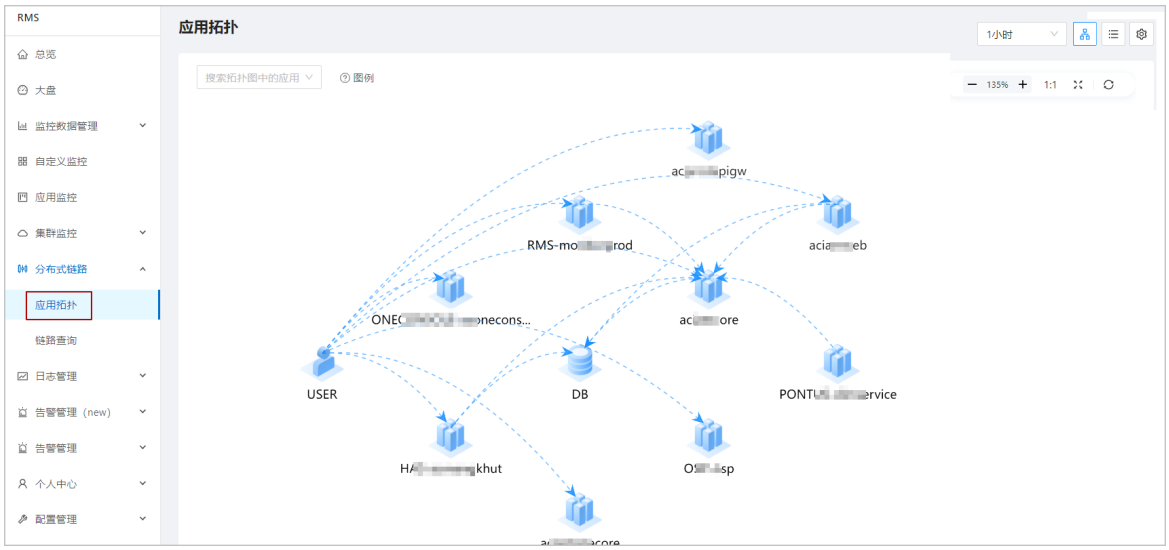
7.1.6. 分布式链路

7.1.6.1. 概述

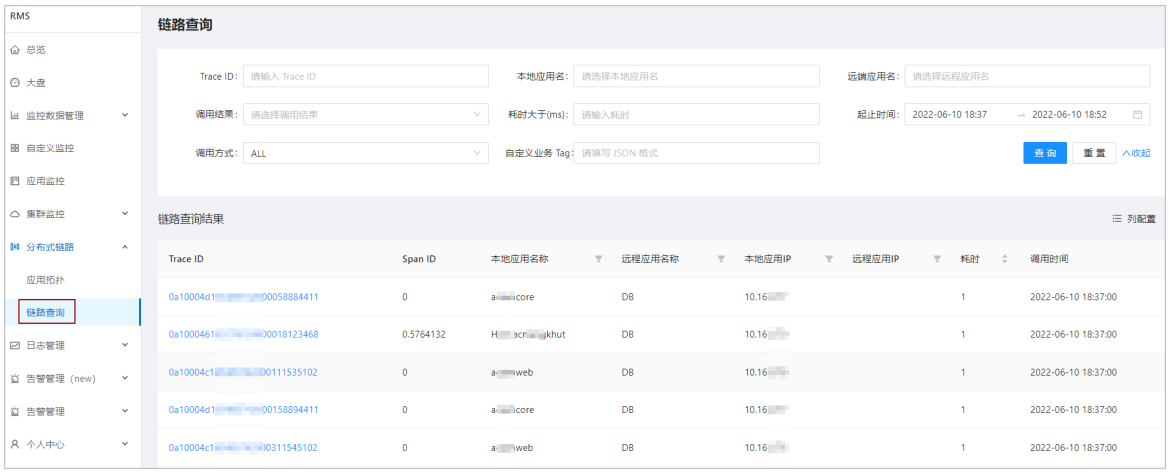
本部分将介绍链路追踪系统快速验证方法和 Mock 数据验证方法，并对部分常见问题及排查方法进行了简单介绍。如需获取更多故障排查信息，请参见 BOS 运维指南。

7.1.6.2. 快速验证法

1. 登录控制台，在左侧导航栏选择 **应用拓扑**，能够看到具体的拓扑图。

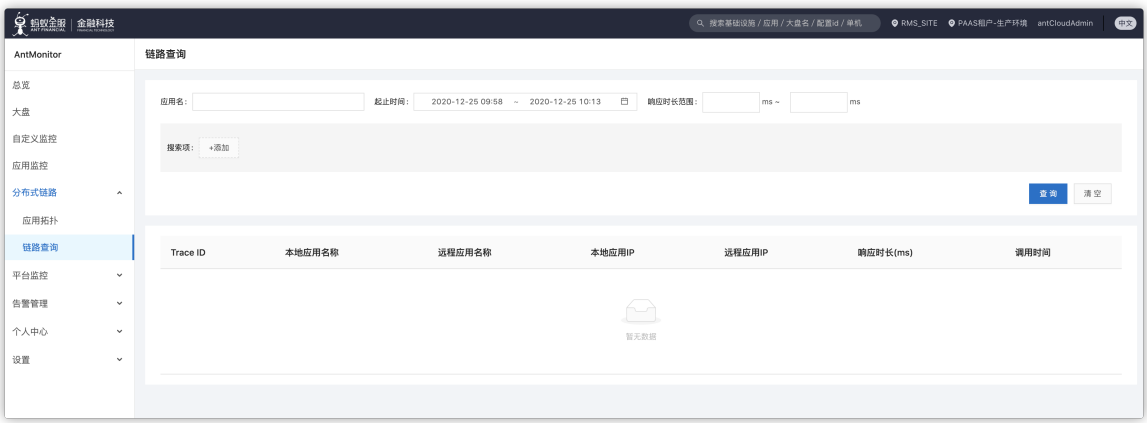


2. 在左侧导航栏选择 **链路查询**，能够看到列表内有数据。

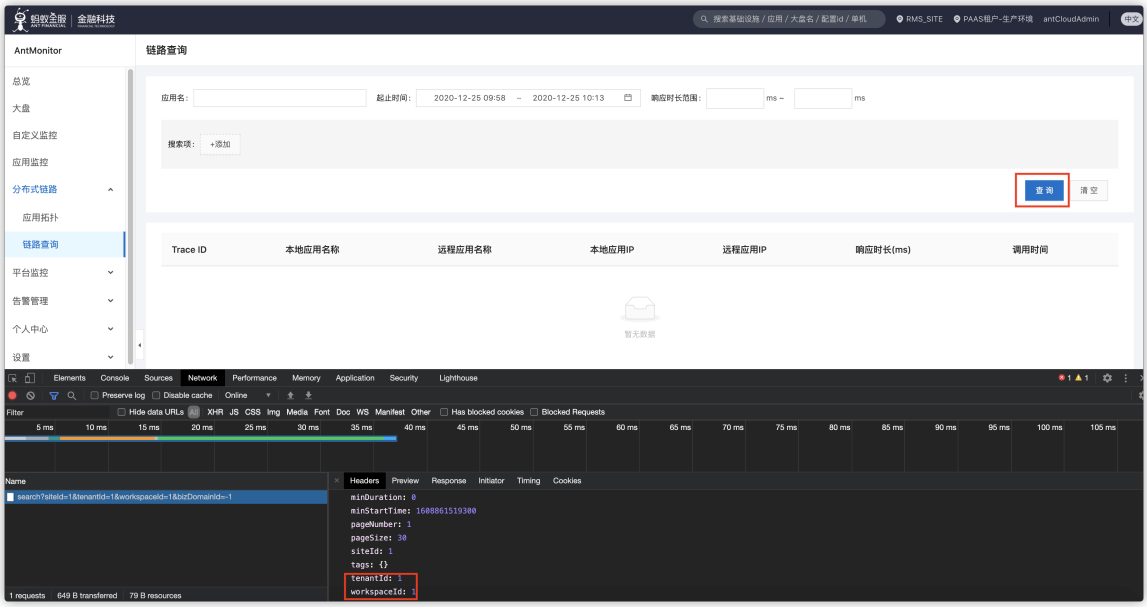


7.1.6.3. Mock 数据验证法

1. 首先选定一个用户空间（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和 workspaceId。



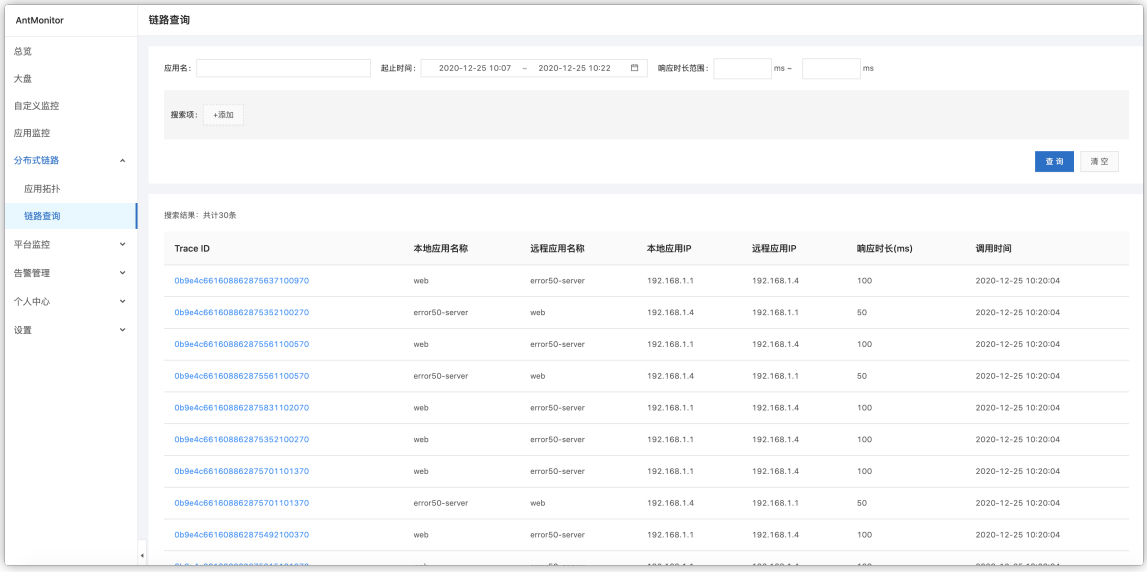
3. 触发 Mock 接口，留意 monitorgateway IP、tenantId 和 workspaceId。HTTP 响应码应为 200。

> **说明**
>
> 可通过部署工具（例如云游）来获取 monitorgateway IP。

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

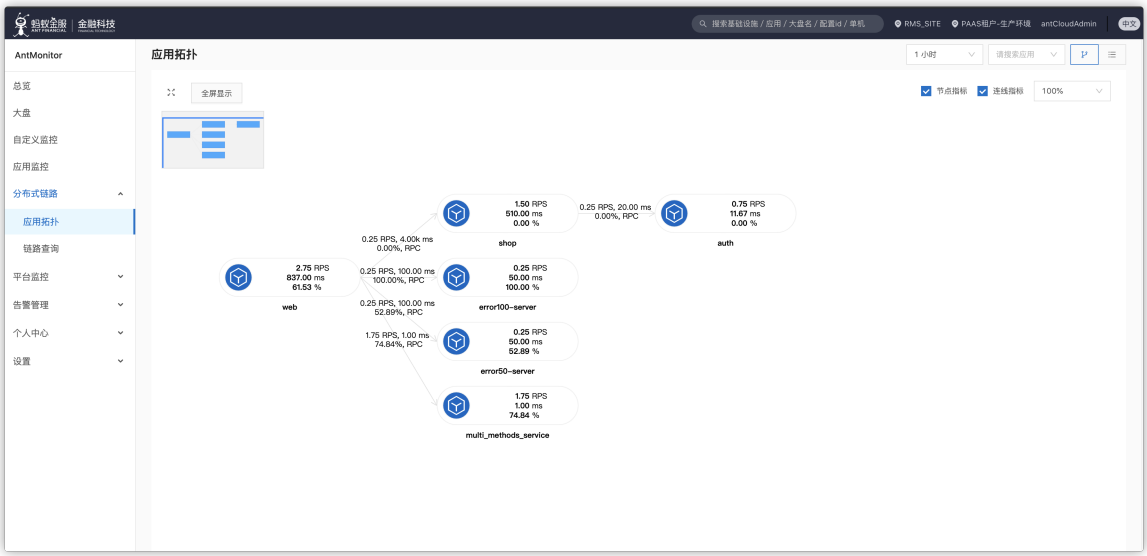
4. 刷新 BOS 链路查询页面，应该会有数据。

> **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。



7.1.6.4. 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】

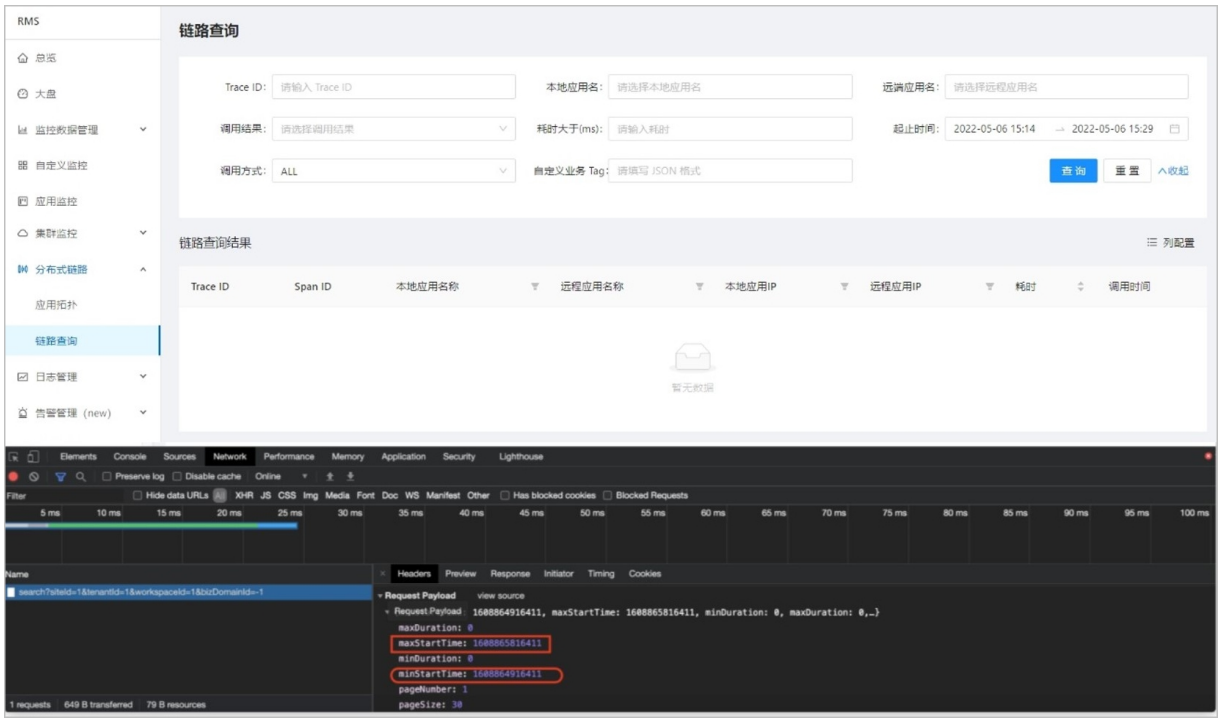
1. 检查 monitorgateway 的环境变量。

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

2. 检查 monitorprod 的环境变量。

```
sh-4.2# env | grep CERESDB_TRACE_URL
CERESDB_TRACE_URL=http://[redacted]: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
}'
```

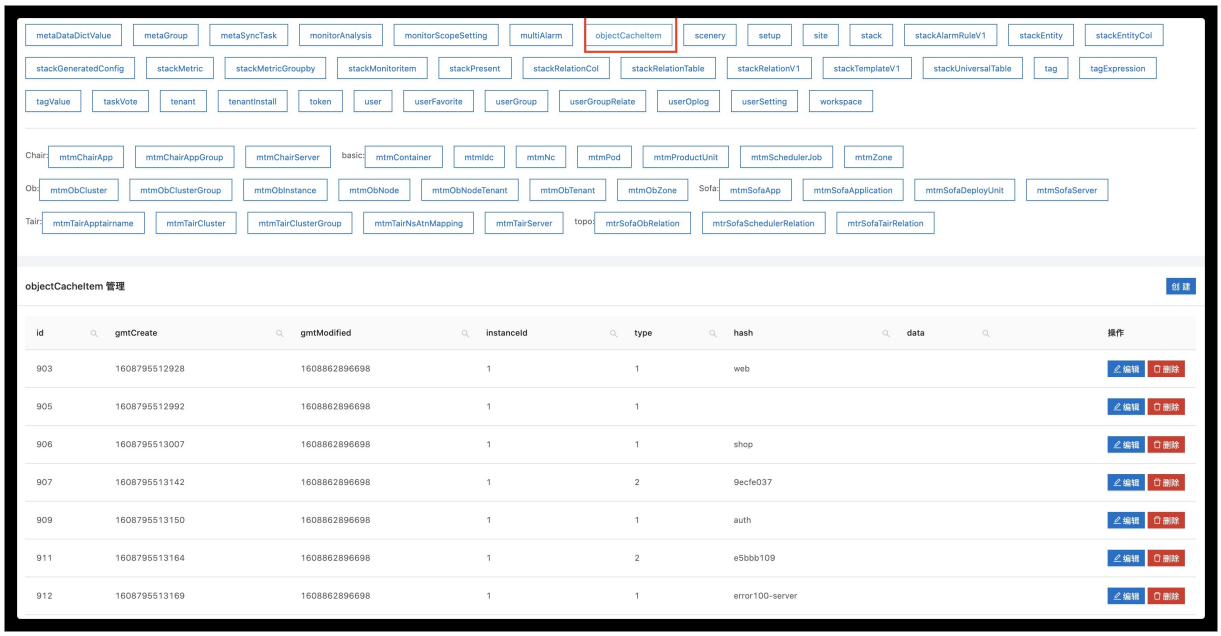
链路拓扑无数据

【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。



7.2. 用户态监控验证

用户态监控验证具体步骤与 核心态监控验证 步骤类似，区别点如下：

选择租户、工作空间及对应的应用时，需要选择用户的租户、用户的工作空间以及用户工作空间下已部署的应用。

8.附录：技术栈备份与恢复

以下操作主要用于技术栈的备份与恢复，属于高危操作，一般不需要。非特殊情况，请忽略本节。

技术栈备份

```
create table bak_stack like stack;
insert into bak_stack select * from stack;

create table bak_stack_entity like stack_entity;
insert into bak_stack_entity select * from stack_entity;

create table bak_stack_entity_col like stack_entity_col;
insert into bak_stack_entity_col select * from stack_entity_col;

create table bak_stack_metric like stack_metric;
insert into bak_stack_metric select * from stack_metric;

create table bak_stack_metric_groupby like stack_metric_groupby;
insert into bak_stack_metric_groupby select * from stack_metric_groupby;

create table bak_stack_monitoritem like stack_monitoritem;
insert into bak_stack_monitoritem select * from stack_monitoritem;

create table bak_stack_present like stack_present;
insert into bak_stack_present select * from stack_present;

create table bak_stack_relation like stack_relation;
insert into bak_stack_relation select * from stack_relation;

create table bak_stack_relation_col like stack_relation_col;
insert into bak_stack_relation_col select * from stack_relation_col;

create table bak_stack_relation_table like stack_relation_table;
insert into bak_stack_relation_table select * from stack_relation_table;

create table bak_stack_relation_v1 like stack_relation_v1;
insert into bak_stack_relation_v1 select * from stack_relation_v1;
```

技术栈清理

```
truncate table stack;
truncate table stack_entity;
truncate table stack_entity_col;
truncate table stack_metric;
truncate table stack_metric_groupby;
truncate table stack_monitoritem;
truncate table stack_present;
truncate table stack_relation;
truncate table stack_relation_col;
truncate table stack_relation_table;
truncate table stack_relation_v1;
```

技术栈备份数据清理

```
drop table bak_stack;
drop table bak_stack_entity ;
drop table bak_stack_entity_col ;
drop table bak_stack_generated_config ;
drop table bak_stack_metric ;
drop table bak_stack_metric_groupby ;
drop table bak_stack_monitoritem ;
drop table bak_stack_present ;
drop table bak_stack_relation ;
drop table bak_stack_relation_col ;
drop table bak_stack_relation_table ;
drop table bak_stack_relation_v1 ;
```

技术栈数据备份恢复

```
truncate table stack;
truncate table stack_alarm_config;
truncate table stack_alarm_rule;
truncate table stack_alarm_rule_v1;
truncate table stack_alarm_template;
truncate table stack_entity;
truncate table stack_entity_col;
truncate table stack_generated_config;
truncate table stack_metric;
truncate table stack_metric_groupby;
truncate table stack_monitoritem;
truncate table stack_present;
truncate table stack_relation;
truncate table stack_relation_col;
truncate table stack_relation_table;
truncate table stack_relation_v1;
truncate table stack_template;
truncate table stack_template_v1;
truncate table stack_universal_table;

insert into stack select * from bak_stack;
insert into stack_alarm_config select * from bak_stack_alarm_config;
insert into stack_alarm_rule select * from bak_stack_alarm_rule;
insert into stack_alarm_rule_v1 select * from bak_stack_alarm_rule_v1;
insert into stack_alarm_template select * from bak_stack_alarm_template;
insert into stack_entity select * from bak_stack_entity;
insert into stack_entity_col select * from bak_stack_entity_col;
insert into stack_generated_config select * from bak_stack_generated_config;
insert into stack_metric select * from bak_stack_metric;
insert into stack_metric_groupby select * from bak_stack_metric_groupby;
insert into stack_monitoritem select * from bak_stack_monitoritem;
insert into stack_present select * from bak_stack_present;
insert into stack_relation select * from bak_stack_relation;
insert into stack_relation_col select * from bak_stack_relation_col;
insert into stack_relation_table select * from bak_stack_relation_table;
insert into stack_relation_v1 select * from bak_stack_relation_v1;
insert into stack_template select * from bak_stack_template;
insert into stack_template_v1 select * from bak_stack_template_v1;
insert into stack_universal_table select * from bak_stack_universal_table;
```