SOFAStack

AntStack Plus

运维指南

产品版本：AntStack Plus 1.13.1

文档版本：20230705

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.概述

为加强对生产操作各个环节的质量控制及变更操作的规范管理，加强流程管控，明确必要责任，不断提升业务稳定性指标。

运维准备

进行运维工作前，需了解以下内容：

- 运维流程

  遵循客户的运维流程及规范制度。

- 运维平台
  - 云游：发布、运维、管理、控制系统。
  - Captain Plus：物理机、容器调度管控系统。
- 运维工具

  AKE 白屏化工具：容器引擎 AKE 云原生平台（Ant Financial Kubernetes Engine，简称 AKE3）管控系统。

运维注意事项

- 除了代码变更，任何物理上（服务器上的配置、脚本等）、逻辑上（数据库、服务器的用途、标识、容灾逻辑等）、业务上（压测、预案等）的改变都属于变更范围，任何变更即使是一行注释也要经过测试。
- 禁止在非变更窗口（封网期）执行变更，如有实际需求，请走紧急封网变更。
- 禁止未达到发布准入标准执行发布操作。
- 禁止在生产环境登录服务器进行测试操作。
- 禁止在业务高峰期进行任何变更。
- 禁止一切未经变更管理平台授权的变更操作。
- 禁止一切变更方案外的操作，必须严格按照变更方案执行。

2.物理机

2.1. 物理机端口

本文介绍物理机容器对应端口和进程。

AKE3 组件名称	端口号	进程名称
containerd	127.0.0.1:12029	/usr/bin/containerd
kubelet	10250、10255、127.0.0.1:10248	/usr/local/bin/kubelet
kube-proxy	127.0.0.1:10249	/usr/bin/kube-proxy
etcd	2379、2380	etcd
apiserver	6443	kube-apiserver
rama	19898	/rama/rama-webhook
	19899	/rama/rama-manager
	-	/rama/rama-daemon
yoda	-	yoda agent
	23000	yoda scheduler
scheduler	127.0.0.1:10251	kube-scheduler
controller-manager	127.0.0.1:10257	kube-controller-manager

2.2. 日常巡检

本文介绍物理机日常巡检项。

物理机运行状况巡检

1. 登录 ops1 机器，使用如下命令检查所有物理机是否正常运行。

```
ssh <username>@<物理机 IP> "uptime"
```

例如：

```
ssh root@100.**.**.254 "uptime"
21:03:19 up 111 days, 10:07,  1 user,  load average: 0.00, 0.02, 0.05
```

关注点：

- 检查所有物理机的启动时间，判断物理机是否有发生过宕机事件。
- 检查所有物理机的过去 1 分钟、5 分钟、15 分钟的系统平均负载情况。

2. 登录到物理机后台，使用如下命令观察 CPU、内存、磁盘的使用情况。
   - 查看 CPU 使用率：

     ```
     tsar --cpu
     ```

   - 查看内存使用率：

     ```
     tsar --mem
     ```

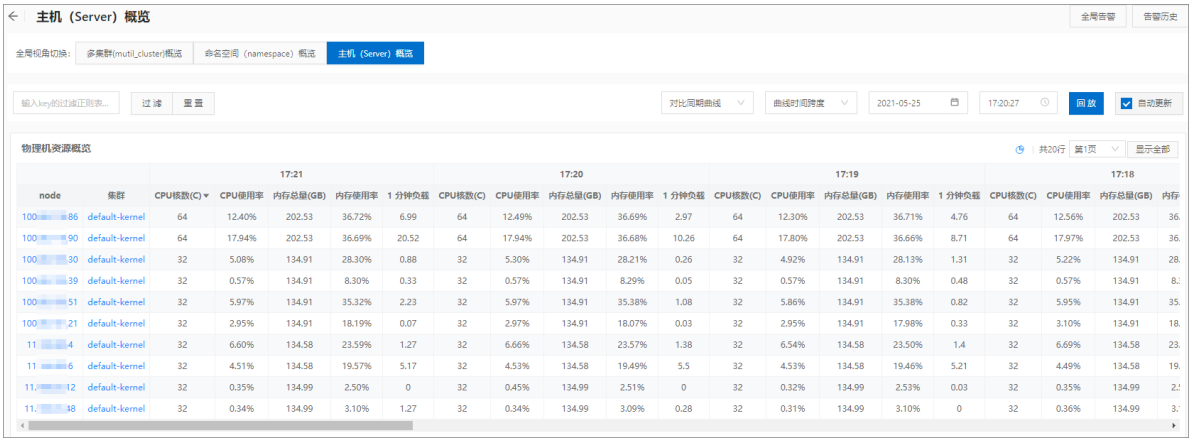
   - 查看磁盘使用率：

     ```
     tsar --partition
     ```

物理机基础监控巡检

1. 登录 RMS 控制台。
2. 单击 **集群监控**，然后单击 **主机（Server）概览**。

   您可以在物理机资源概览列表中查看 CPU、内存、磁盘等资源使用情况。



关注点：

- 告警情况：单击右上角 **告警历史** 查看是否有告警产生。
- CPU 使用率：不超过 90%。
- 内存使用率：不超过 90%。
- 磁盘使用率：不超过 90%。

2.3. 服务异常应急预案

2.3.1. 物理机宕机

本文介绍非 ops 物理机出现宕机时的紧急处理预案。

环境检查

1. 登录 Captain 控制台，选择 **集群管理 > 节点（Nodes）**。
2. 在 **节点（Nodes）**页面确认节点（物理机）状态。
3. 测试节点连通性。

```
ping <节点 IP>
```

4. 如果节点 IP 不可达（包括重启节点后，Captain 控制台上显示节点状态依旧为“故障”），按照如下实施步骤进行迁移。

实施步骤

1. 检查 PVC，确认 Pod 是否有关联的 PVC。

```
kubectl get pvc -n <namespace>
```

其中 NAME 是 PVC 的名称，VOLUME 一列显示的是 PVC 绑定的 PV 的名称。如果 Pod 有关联的 PVC，则需要删除相关的 PVC、PV；如果 PVC 没有绑定的 VOLUME 则不需要删除 PV。删除命令如下：

```
kubectl delete pvc -n <namespace> <name>
kubectl delete pv <volume>
```

> **警告**
>
> 删除 PV 的操作不可恢复，删除可能导致数据丢失，您需要事先和对应产品的负责人确认数据是否可以安全删除。

2. 检查 IPInstance，查看 Pod 关联的 IPInstances 资源。

```
kubectl get ipinstances -n asp-aks |grep <待迁移 Pod>
```

请确认 Pod 是否需要 IP 保持不变的能力，如果不需要，直接删除掉条 IPInstance 即可。

3. 删除待迁移 Pod。

删除故障 Pod 后，Pod 会自动迁移。

> **重要**
>
> - 应用数据均有多副本，数据丢失不影响应用。
> - 您必须保证其他节点有足够的资源来承载当前节点的应用。

- 删除 Pod

  命令如下：

  ```
  kubectl delete pod <待迁移 Pod>
  ```

- 删除批量 Node

a. 删除 machine。

命令如下：

```
kubectl delete machine <节点名称>
```

这一步会卡住，执行后直接按组合按键 ctrl+c 进行下一步即可。

b. 编辑 machine，把里面的 finalizer 都删除。

命令如下：

```
kubectl edit machine <节点名称>
```

c. 删除如下两行，然后输入 :wq 保存退出。

```
finalizers:
- finalizer.k8s.alipay.com/machine
```

d. 删除 node 对象。

命令如下：

```
kubectl delete node <节点名称>
```

e. 清理配置数据。

命令如下：

```
kubectl delete cm -n sigma-operator-machine-conditions <name>-conditions
kubectl delete cm -n sigma-operator-machine-conditions <name>-readiness-gates
```

f. 刷新 Captain 控制台的页面，节点已经从列表中消失了。

2.3.2. 物理机重启后管控状态显示为故障

问题描述

物理机重启后，Captain 控制台显示节点状态为 **故障**。

环境检查

1. 登录 Captain 控制台。
2. 在左侧导航栏选择 **集群管理** > **节点（Nodes）**，确认节点（物理机）状态。
3. 从一台正常运行节点的 shell 下使用 **ping** 命令测试节点连通性，确认网络可达。

```
ping <节点 IP>
```

4. 如果节点能 ping 通，但是Captain 控制台上显示节点状态为**故障**，请按照如下步骤恢复物理机状态。

实施步骤

1. 恢复物理机状态。

i. 在 Captain 控制台左侧导航栏选择 **集群管理** > **节点（Nodes）**

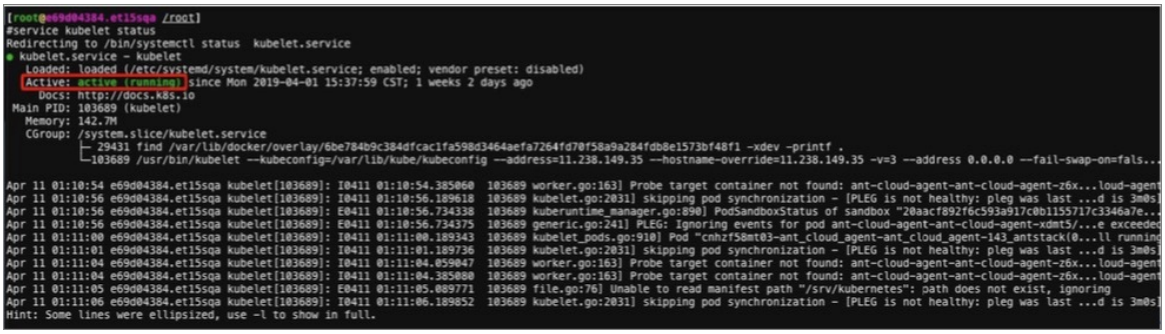
ii. 单击目标节点的 Shell，然后执行以下命令：

```
systemctl start kubelet
```

iii. 输入以下命令验证 kubelet 服务状态。

```
systemctl status kubelet
```

确认 kubelet 进程状态为 active。



如果 Kubelet 的状态不是 active，则使用如下命令查看 kubelet 启动日志：

```
journalctl -u kubelet
```

2. 恢复故障物理机上的容器。

Kubelet 启动完成后，容器会自动启动，无需人工干预。您可以在 Master 节点上通过以下命令确认主机是否恢复：

- 确认没有 Not Ready 的节点：

```
#kubectl get node
NAME              STATUS    ROLES     AGE     VERSION
100.**.**.241     Ready     <none>    110d    v1.20.0
100.**.**.242     Ready     <none>    110d    v1.20.0
100.**.**.243     Ready     <none>    110d    v1.20.0
100.**.**.244     Ready     <none>    110d    v1.20.0
100.**.**.177     Ready     <none>    88d     v1.20.0
```

- 查看是否有非 Ready 状态，且处于故障节点的 Pod：

```
#kubectl get pod -A -o wide | grep -vP '\s+([1-9]+[\d]*)\/\1\s+'
NAMESPACE                   NAME                                                    READ
Y    STATUS                         RESTARTS    AGE       IP                       NODE
NOMINATED NODE   READINESS GATES
dc-accountbook          dc-accountbook-bpaastest-0-jhk2r           0/1     Completed
0              2d18h     100.83.84.31     100.**.**.180   <none>          <none>
dc-acvip                        dc-acvip-acviptest-0-djv4w                          0/1
Completed                      0            46h          11.**.**.37         11.**.**.82
<none>          <none>
dc-aks                           dc-aks-akstest-0-ggnrm                             0/1
Completed                      0            8h            100.**.**.132     100.**.**.2
<none>          <none>
```

结果验证

在 Captain 控制台查看节点状态为 **健康**，调度状态为 **可被调度**。

补充说明

该问题由物理机重启后，管理进程 kubelet 未正常启动导致。容器是否会随物理机重启自启动取决于容器配置参数 `restart=always` 。

3.基础服务

3.1. NTP 服务可用性检查

计算机主机一般同多个时间服务器连接，利用统计学的算法过滤来自不同服务器的时间，以选择最佳的路径和来源来校正主机时间。即使主机在长时间无法与某一时间服务器相联系的情况下，NTP 服务依然有效运转。集群内部很多组件都是依赖时间戳来进行相关的逻辑操作，如果时间不一致会导致系统不可用。本文介绍如何检查 NTP 服务的可用性。

物理机或虚拟机场景通常使用集群的 ops1、ops2 作为集群内部的 NTP 服务器，上游使用用户的 NTP 服务器；阿里云场景使用 ECS 的 NTP 服务器，无需单独配置。

操作步骤

1. 登录 ops 机器。
2. 使用如下命令检查所有物理机的 NTP 服务：

```
ssh <username>@<物理机 IP> "ntpq -p"
```

示例：

```
ssh root@100.**.**.254 "ntpq -p"
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 LOCAL(0)        .LOCL.          10 l  66d   64    0    0.000    0.000   0.000
 10.**.**.230    .INIT.          16 u    -   64        0    0.000    0.000   0.000
 10.**.**.231    .INIT.          16 u    -   64        0    0.000    0.000   0.000
 dnslvs1.cm10.tb .INIT.          16 u    -   64        0    0.000    0.000   0.000
 ntp-l2-corp1.cm .INIT.          16 u    -   64        0    0.000    0.000   0.000
 10.**.**.29      .INIT.          16 u    -   64        0    0.000    0.000   0.000
 10.**.**.30      .INIT.          16 u    -   64        0    0.000    0.000   0.000
 10.**.**.43      .INIT.          16 u    -   64        0    0.000    0.000   0.000
 10.**.**.124     .INIT.          16 u    -   64        0    0.000    0.000   0.000
+time4.aliyun.co 10.**.**.7       2 u   70  256  216   23.456    0.114   0.754
-time5.aliyun.co 10.**.**.86      2 u  201  256  377   20.636    0.168   1.519
```

参数说明：

参数	说明
remote	响应这个请求的 NTP 服务器 的 IP 或 名称。显示时可能会携带以下特殊符号： ○ 星号（*）：响应查询的最精确 NTP 服务器。 ○ 加号（+）：响应查询请求的 NTP 服务器。 ○ 减号（-）：响应查询的不合格 NTP 服务器。 ○ 空格：没有响应查询的 NTP 服务器。

参数	说明
refid	NTP 服务器使用的更高一级服务器的名称。 `.INIT.` 表示连接不上上游的 NTP 服务器。
st	正在响应请求的 NTP 服务器级别。
when	上一次成功请求之后到当前间隔。单位：秒。
poll	本地和远程服务器同步的时间间隔。单位：秒。 初次同步 NTP 时，poll 值较小，服务器同步的频率大，能够尽快调整到正确的时间范围。之后 poll 值会逐渐增大，同步的频率也就会对应减小。
reach	用来测试是否能和服务器连接，是一个八进制值，每成功连接一次它的值就会添加。
delay	从本地机发送同步要求到 NTP 服务器的往返时间。
offset	主机通过 NTP 时钟同步与所同步时间源的时间偏移量。单位：毫秒。 offset 越接近 0，主机和 NTP 服务器的时间越接近。
jitter	统计了在特定个连续的连接数里 offset 的分布情况。这个数值的绝对值越小，主机的时间就越精确。

关注点

进行 NTP 服务可用性检查时，需关注以下参数：

- **remote**：应指向 ntp1 或 ntp2 时钟源，部分站点也可能直接指向到用户统一的 NTP 源。
- **refid**：NTP 服务器使用的更高一级服务器应为行方 NTP 源。
- **offset**：主机与 NTP 时钟源的偏移量应在 1s 内。

  若发现物理机与 NTP 服务的时差较大，可通过 `ntpdate` 命令手动进行时间同步。

```
ntpdate -u <NTP 服务器 IP>
```

注意事项

如果用一台虚拟机或物理机作为 NTP 服务器上游，缺少高可用。若上游 NTP 服务器宕机，会导致集群 etcd 节点时间不同步，容易造成 k8s 集群不可用。所以不推荐使用无高可用的 NTP 服务器作为上游。

使用 Chronyd 作为时间同步服务

如果您使用 Chronyd 作为时间同步服务，需使用 `chronyc sources -v` 命令查看与 NTP 时钟源的时钟偏移量。

3.2. DNS 服务可用性检查

DNS 服务主要用于中枢集群内部，蚂蚁核心产品（守夜人、RMS 等）访问提供 DNS 解析支持。客户可以通过打通 DNS 服务，在内网对这些服务进行访问。本文介绍如何对 DNS 服务进行巡检。

操作步骤

1. 检查云内所有的节点的 `/etc/resolv.conf` 。

   `/etc/resolv.conf` 应当指向 ADNS 的地址（如果有 VIP，应当是 VIP 地址，否则是物理机地址）。

2. 登录云内任意一台机器，执行如下命令：

   ```
   dig <domain-name>
   或者
   nslookup <domain-name>
   ```

   <domain-name> 可以是任意在 ADNS 上注册的域名。

关注点

解析结果返回 IP 地址则证明 DNS 服务正常。

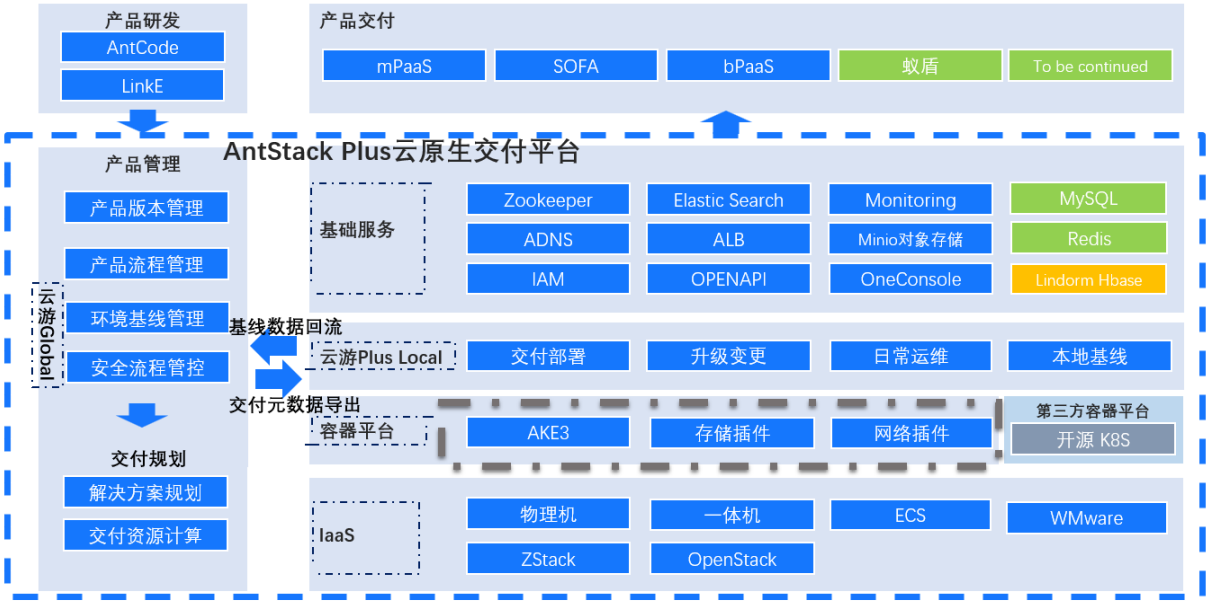
4.云游

4.1. 产品架构

云游采用星型结构，即以云游 Global 为中心，根据您的项目需求搭建多个云游 Local 环境。云游 Global 部署在蚂蚁集团内部，存储所有的产品元数据和环境元数据，以及根据这些元数据制作出的所有解决方案。云游 Global 和各云游 Local 环境之间通过解决方案进行沟通，不同的云游 Local 环境之间相互独立，互不影响。

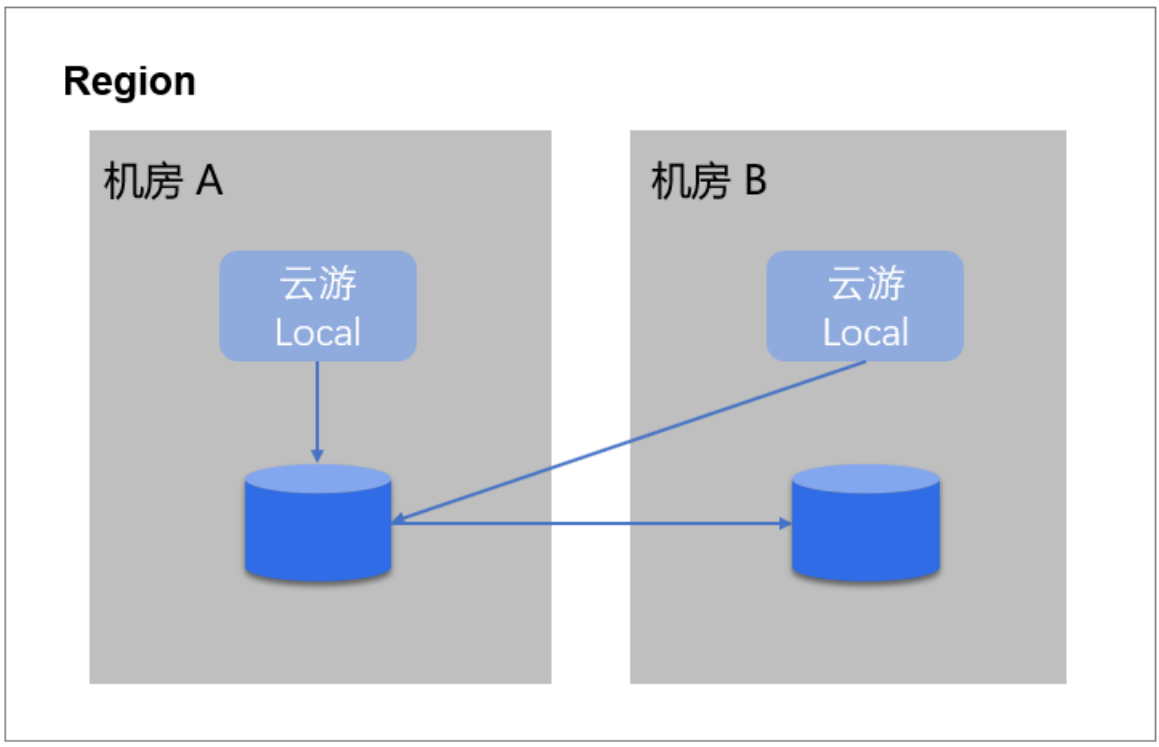
系统架构

云游负责发布和运维上层给客户的各种产品，例如 mPaaS、SOFA 等；下层通过 AKE 云原生平台（AKE3）屏蔽不同底座（如物理机、ECS、OpenStack 等）下的 IaaS 差异。云游内部通过流程编排、资源管理、配置变更等模块支撑对产品的各种运维操作。



部署架构

您可以在双机房部署 2 个云游 Local 的副本，从而保障机房内以及机房级容灾。底层数据库通过阿里云 RDS、OceanBase 或 XDB 方案实现数据主备同步。双机房的云游 Local 都连主库，只有在主库发生故障时，才会将连接切换到备库。



4.2. 系统日志

本文介绍云游的系统日志存储路径。

日志存储路径

日志名称	日志路径	日志说明
一般错误日志	/home/admin/logs/apyunqing/common-error.log	错误日志输出。
业务日志	/home/admin/logs/apyunqing/biz-resource.log	业务相关日志。
任务日志	/home/admin/logs/apyunqing/activity-default.log	任务调度流程相关。
任务事件日志	/home/admin/logs/apyunqing/activity-event.log	任务调度事件监听。

日志名称	日志路径	日志说明
参数渲染日志	/home/admin/logs/apyunqing/app-param.log	参数渲染相关。
SOFA 框架日志	/home/admin/logs/apyunqing/sofa-default.log	SOFA 框架相关日志。
编排日志	/home/admin/logs/apyunqing/orch-execute-digest.log	云游编排发布相关日志。
线程日志	/home/admin/logs/apyunqing/thread-monitor-digest.log	线程监控相关日志。

主要日志介绍

一般错误日志

日志路径：`/home/admin/logs/apyunqing/common-error.log`

日志说明：错误日志输出，一般用于定位异常问题。

- 示例 1：参数渲染失败

```
2021-06-22 00:08:29,757 [  - /// - ] ERROR render.LBVariableExpRender - [] Failed to render context, context: RenderTemplate[env=730420210201160340948850000;templateKey=com_alipay_confreg_url;template=#if (${res.MS.lb.confreg-lb-1.vip.exist})${res.MS.lb.confreg-lb-1.vip} #else @com_alipay_confreg_url@ #end;prodCode=APIGATEWAY;prodVersion=1.10.0;prodInstanceVersionId=null;appName=apigateway;index=null;deploymentUnitInstanceIdentity=null;]
java.lang.NullPointerException: null
```

主要原因：对应的资源没有创建、数据错误。

解决方案：查看依赖的资源是否创建成功。

- 示例 2：webshell 连接超时

```
2021-06-22 02:11:09,360 [c0a8050116243278038485724459 5 0 - /// - ] ERROR webshell.K8sWebshellController -
java.io.IOException: java.util.concurrent.ExecutionException: java.net.SocketException: Broken pipe (Write failed)
        at org.apache.tomcat.websocket.WsRemoteEndpointImplBase.startMessageBlock(WsRemoteEndpointImplBase.java:277) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        at org.apache.tomcat.websocket.WsSession.sendCloseMessage(WsSession.java:510) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        at org.apache.tomcat.websocket.WsSession.onClose(WsSession.java:464) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        at org.apache.tomcat.websocket.WsFrameBase.processDataControl(WsFrameBase.java:342) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        at org.apache.tomcat.websocket.WsFrameBase.processData(WsFrameBase.java:284) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        at org.apache.tomcat.websocket.WsFrameBase.processInputBuffer(WsFrameBase.java:13
```

```
0) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        at org.apache.tomcat.websocket.server.WsFrameServer.onDataAvailable(WsFrameServer
.java:56) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        at org.apache.tomcat.websocket.server.WsHttpUpgradeHandler$WsReadListener.onDataA
vailable(WsHttpUpgradeHandler.java:203) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        at org.apache.coyote.http11.upgrade.AbstractServletInputStream.onDataAvailable(Ab
stractServletInputStream.java:203) ~[?:?]
        at org.apache.coyote.http11.upgrade.AbstractProcessor.upgradeDispatch(AbstractPro
cessor.java:92) ~[?:?]
        at org.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process(AbstractP
rotocol.java:609) ~[?:?]
        at org.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.java:31
4) ~[?:?]
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142
) [?:1.8.0_131]
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617
) [?:1.8.0_131]
        at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java
:61) ~[?:?]
        at java.lang.Thread.run(Thread.java:748) [?:1.8.0_131]
Caused by: java.util.concurrent.ExecutionException: java.net.SocketException: Broken pipe
(Write failed)
        at org.apache.tomcat.websocket.FutureToSendHandler.get(FutureToSendHandler.java:1
02) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        at org.apache.tomcat.websocket.WsRemoteEndpointImplBase.startMessageBlock(WsRemot
eEndpointImplBase.java:272) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        ... 15 more
Caused by: java.net.SocketException: Broken pipe (Write failed)
        at java.net.SocketOutputStream.socketWrite0(Native Method) ~[?:1.8.0_131]
        at java.net.SocketOutputStream.socketWrite(SocketOutputStream.java:111) ~[?:1.8.0
_131]
        at java.net.SocketOutputStream.write(SocketOutputStream.java:155) ~[?:1.8.0_131]
        at org.apache.coyote.http11.upgrade.BioServletOutputStream.doWrite(BioServletOutp
utStream.java:38) ~[tomcat-embed-core-7.0.57.jar:7.0.57]
        at org.apache.coyote.http11.upgrade.AbstractServletOutputStream.writeInternal(Abs
tractServletOutputStream.java:153) ~[tomcat-embed-core-7.0.57.jar:7.0.57]
        at org.apache.coyote.http11.upgrade.AbstractServletOutputStream.write(AbstractSer
vletOutputStream.java:121) ~[tomcat-embed-core-7.0.57.jar:7.0.57]
        at org.apache.tomcat.websocket.server.WsRemoteEndpointImplServer.onWritePossible(
WsRemoteEndpointImplServer.java:95) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        at org.apache.tomcat.websocket.server.WsRemoteEndpointImplServer.doWrite(WsRemote
EndpointImplServer.java:82) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        at org.apache.tomcat.websocket.WsRemoteEndpointImplBase.writeMessagePart(WsRemote
EndpointImplBase.java:447) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        at org.apache.tomcat.websocket.WsRemoteEndpointImplBase.startMessage(WsRemoteEndp
ointImplBase.java:338) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        at org.apache.tomcat.websocket.WsRemoteEndpointImplBase.startMessageBlock(WsRemot
eEndpointImplBase.java:267) ~[tomcat-embed-websocket-7.0.57.jar:7.0.57]
        ... 15 more
```

主要原因：websocket 连接超时，没有数据。

解决方案：无需处理。

- 示例 3：机器磁盘已满

```
2021-06-22 02:13:15,098 [  - /// - ] ERROR v3.FacadeInterceptor - [] Invoke $Proxy128#que
ryAppDeployParamRenderDiff exception, params:[AppDeployParamRenderDiffRequest[env=7304202
10201160340948850000;branchOne=73042021060218151468274000 0;branchTwo=BASELINE;prodCode=PO
NTUS;prodVersion=3.5.0;prodInstanceVersionId=null;appName=agentregistry;operator=null;for
ce=false;],]
org.springframework.jdbc.UncategorizedSQLException:
### Error querying database.  Cause: java.sql.SQLException: Error writing file '/tmp/MY1g
3B8f' (Errcode: 28 - No space left on device)
### The error may exist in template-sql.xml
### The error may involve defaultParameterMap
### The error occurred while setting parameters
```

主要原因：物理机磁盘已满。

解决方案：清理磁盘空间，删除过期文件。

- 示例 4：业务逻辑异常

```
2021-06-22 03:00:02,863 [c0a805011624330802841314145 95 0 - /// - ] ERROR interceptor.Faca
deInterceptor - [c0a80501162433080284131414595] Invoke OpsPlanFacadeImpl#cancel exception
, params:[OpsPlanCancelRequest[opsPlanId=73040001526558;submitterId=mockId;submitterSourc
eUserId=mockId;submitterName=mockId;submitterLoginName=mockId;submitterWorkNo=null;submit
terNickN
ame=null;submitterRealName=null;force=false;scene=73045012;bizNo=9b982a69-4187-42dd-94ce-
1a49202256ee;fromApproval=false;],]
cn.com.antcloud.common.exception.inner.CommonInnerException: [SYSTEM, ERROR, ] - Possible
Exception Location: cn.com.antcloud.common.infrastructure.interceptor.CommonInterceptor.i
nvoke(CommonInterceptor.java:39), Reason: (Y, AC15073045012), 发布单取消失败, placeholder=
yunyoutest.cmd.ops.OpsPlanCancelCmd.ReleaseSingleCancelFailed, params=[] Cause: cn.com.an
tclou
d.common.exception.biz.BusinessException: [AC15173200414039(ERROR, BUSINESS, , )] - Possi
ble Exception Location: cn.com.antcloud.common.infrastructure.interceptor.CommonIntercept
or.invoke(CommonInterceptor.java:39), Reason: (Y, null), failed to cancel nodes Cause: cn
.com.antcloud.common.exception.biz.BusinessException: [AC15173200417039(ERROR, BUSINESS,
, )] - P
ossible Exception Location: cn.com.antcloud.common.infrastructure.interceptor.CommonInter
ceptor.invoke(CommonInterceptor.java:39), Reason: (Y, null), cancel failed to acquire loc
k, node 73040001526558 has no process instance binding, {0=cancel, 1=73040001526558}, , c
n.com.antcloud.common.exception.biz.BusinessException: [AC15173200417039(ERROR, BUSINESS,
, )] -
Possible Exception Location: cn.com.antcloud.common.infrastructure.interceptor.CommonInte
rceptor.invoke(CommonInterceptor.java:39), Reason: (Y, null), cancel failed to acquire lo
ck, node 73040001526558 has no process instance binding, {0=cancel, 1=73040001526558},
```

主要原因：业务逻辑异常。

解决方案：根据返回的异常信息进行处理。如果无法处理，请联系系统管理员。

业务日志

日志路径：`/home/admin/logs/apyunqing/biz-resource.log`

日志说明：业务日志输出，主要记录业务相关操作的核心日志，提供给开发人员定位问题。

任务日志

日志路径：`/home/admin/logs/apyunqing/activity-default.log`

日志说明：依赖的第三方任务调度框架日志，主要是提供给开发人员使用。

日志示例：

```
2021-06-21 14:01:22,323 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Starting
up the default async job executor [org.activiti.engine.impl.asyncexecutor.DefaultAsyncJobEx
ecutor].
2021-06-21 14:01:22,332 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Creating
thread pool queue of size 100
2021-06-21 14:01:22,332 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Creating
executor service with corePoolSize 20, maxPoolSize 30 and keepAliveTime 5000
2021-06-21 14:01:22,352 [  - /// - ] INFO  asyncexecutor.AcquireAsyncJobsDueRunnable - {} s
tarting to acquire async jobs due
2021-06-21 14:01:22,360 [  - /// - ] INFO  asyncexecutor.AcquireTimerJobsRunnable - {} star
ting to acquire async jobs due
2021-06-21 14:01:22,380 [  - /// - ] INFO  asyncexecutor.ResetExpiredJobsRunnable - {} star
ting to reset expired jobs
2021-06-21 14:16:52,526 [  - /// - ] INFO  impl.ProcessEngineImpl - ProcessEngine default c
reated
2021-06-21 14:16:52,532 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Starting
up the default async job executor [org.activiti.engine.impl.asyncexecutor.DefaultAsyncJobEx
ecutor].
2021-06-21 14:16:52,538 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Creating
thread pool queue of size 100
2021-06-21 14:16:52,539 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Creating
executor service with corePoolSize 20, maxPoolSize 30 and keepAliveTime 5000
2021-06-21 14:16:52,544 [  - /// - ] INFO  asyncexecutor.AcquireAsyncJobsDueRunnable - {} s
tarting to acquire async jobs due
2021-06-21 14:16:52,546 [  - /// - ] INFO  asyncexecutor.AcquireTimerJobsRunnable - {} star
ting to acquire async jobs due
2021-06-21 14:16:52,573 [  - /// - ] INFO  asyncexecutor.ResetExpiredJobsRunnable - {} star
ting to reset expired jobs
2021-06-21 14:53:24,484 [  - /// - ] INFO  impl.ProcessEngineImpl - ProcessEngine default c
reated
2021-06-21 14:53:24,490 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Starting
up the default async job executor [org.activiti.engine.impl.asyncexecutor.DefaultAsyncJobEx
ecutor].
2021-06-21 14:53:24,494 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Creating
thread pool queue of size 100
2021-06-21 14:53:24,494 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Creating
executor service with corePoolSize 20, maxPoolSize 30 and keepAliveTime 5000
2021-06-21 14:53:24,499 [  - /// - ] INFO  asyncexecutor.AcquireAsyncJobsDueRunnable - {} s
tarting to acquire async jobs due
2021-06-21 14:53:24,499 [  - /// - ] INFO  asyncexecutor.AcquireTimerJobsRunnable - {} star
ting to acquire async jobs due
2021-06-21 14:53:24,543 [  - /// - ] INFO  asyncexecutor.ResetExpiredJobsRunnable - {} star
ting to reset expired jobs
2021-06-21 15:06:23,588 [  - /// - ] INFO  deployer.BpmnDeployer - Process deployed: {id: _
ROOT73040000068308:1:320004, key: _ROOT73040000068308, name: null }
2021-06-21 15:13:09,050 [  - /// - ] INFO  deployer.BpmnDeployer - Process deployed: {id: _
ROOT73040000068463:1:320222, key: _ROOT73040000068463, name: null }
2021-06-21 15:21:07,614 [  - /// - ] INFO  deployer.BpmnDeployer - Process deployed: {id: _
ROOT73040000068619:1:320440, key: _ROOT73040000068619, name: null }
2021-06-21 15:35:52,197 [  - /// - ] INFO  deployer.BpmnDeployer - Process deployed: {id: _
ROOT73040000068775:1:320658, key:  ROOT73040000068775, name: null }
```

```
2021-06-21 15:45:33,031 [  - /// - ] INFO  deployer.BpmnDeployer - Process deployed: {id: _
ROOT73040000068930:1:320876, key: _ROOT73040000068930, name: null }
2021-06-21 15:51:28,319 [  - /// - ] INFO  deployer.BpmnDeployer - Process deployed: {id: _
ROOT73040000069085:1:321094, key: _ROOT73040000069085, name: null }
2021-06-21 16:07:56,155 [  - /// - ] INFO  impl.ProcessEngineImpl - ProcessEngine default c
reated
2021-06-21 16:07:56,159 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Starting
up the default async job executor [org.activiti.engine.impl.asyncexecutor.DefaultAsyncJobEx
ecutor].
2021-06-21 16:07:56,163 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Creating
thread pool queue of size 100
2021-06-21 16:07:56,163 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Creating
executor service with corePoolSize 20, maxPoolSize 30 and keepAliveTime 5000
2021-06-21 16:07:56,179 [  - /// - ] INFO  asyncexecutor.AcquireAsyncJobsDueRunnable - {} s
tarting to acquire async jobs due
2021-06-21 16:07:56,207 [  - /// - ] INFO  asyncexecutor.AcquireTimerJobsRunnable - {} star
ting to acquire async jobs due
2021-06-21 16:07:56,216 [  - /// - ] INFO  asyncexecutor.ResetExpiredJobsRunnable - {} star
ting to reset expired jobs
2021-06-21 16:13:19,847 [  - /// - ] INFO  deployer.BpmnDeployer - Process deployed: {id: _
ROOT73040000069240:1:322509, key: _ROOT73040000069240, name: null }
2021-06-21 18:07:05,764 [  - /// - ] INFO  impl.ProcessEngineImpl - ProcessEngine default c
reated
2021-06-21 18:07:05,773 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Starting
up the default async job executor [org.activiti.engine.impl.asyncexecutor.DefaultAsyncJobEx
ecutor].
2021-06-21 18:07:05,777 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Creating
thread pool queue of size 100
2021-06-21 18:07:05,777 [  - /// - ] INFO  asyncexecutor.DefaultAsyncJobExecutor - Creating
executor service with corePoolSize 20, maxPoolSize 30 and keepAliveTime 5000
2021-06-21 18:07:05,785 [  - /// - ] INFO  asyncexecutor.AcquireAsyncJobsDueRunnable - {} s
tarting to acquire async jobs due
2021-06-21 18:07:05,789 [  - /// - ] INFO  asyncexecutor.AcquireTimerJobsRunnable - {} star
ting to acquire async jobs due
2021-06-21 18:07:05,831 [  - /// - ] INFO  asyncexecutor.ResetExpiredJobsRunnable - {} star
ting to reset expired jobs
2021-06-21 20:18:11,157 [  - /// - ] INFO  deployer.BpmnDeployer - Process deployed: {id: _
ROOT73040000069395:1:325009, key: _ROOT73040000069395, name: null }
```

任务事件日志

日志路径：`/home/admin/logs/apyunqing/activity-event.log`

日志说明：依赖的第三方任务调度框架日志，主要是提供给开发人员使用。

日志示例：

```
2021-06-21 14:01:22,591 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:22 CST 2021 process: 312505 execution: 310453 entity: 315002#JobEntityImpl  cre
ated
2021-06-21 14:01:22,591 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:22 CST 2021 process: 312505 execution: 310453 entity: 315002#JobEntityImpl  ini
tialized
2021-06-21 14:01:32,618 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
```

```
un 21 14:01:32 CST 2021 process: 312505 execution: 310453 entity: 315002#JobEntityImpl  del
eted
2021-06-21 14:01:32,651 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:32 CST 2021 process: 312505 execution: 310453 job: 315002  succeed
2021-06-21 14:01:34,007 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:34 CST 2021 process: 312505 execution 315003 of activity null  created
2021-06-21 14:01:34,011 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:34 CST 2021 process: 312505 execution 315003 of activity null  initialized
2021-06-21 14:01:34,125 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:34 CST 2021 process: 312505 execution 310453 of activity _73040000067478  delet
ed
2021-06-21 14:01:34,143 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:34 CST 2021 process: 312505 activity: _73040000067478  started
2021-06-21 14:01:34,151 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:34 CST 2021 process: 312505 execution 315005 of activity null  created
2021-06-21 14:01:34,151 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:34 CST 2021 process: 312505 execution 315005 of activity null  initialized
2021-06-21 14:01:34,151 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:34 CST 2021 process: 312505 activity: _73040000067531  started
2021-06-21 14:01:34,160 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:34 CST 2021 process: 312505 activity: _73040000067531  completed
2021-06-21 14:01:34,170 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:34 CST 2021 process: 312505 activity: _73040000067531_p  started
2021-06-21 14:01:34,171 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:34 CST 2021 process: 312505 activity: _73040000067531_p  completed
2021-06-21 14:01:35,129 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 activity: _73040000067532  started
2021-06-21 14:01:35,152 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 activity: _73040000067532  completed
2021-06-21 14:01:35,153 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution 315009 of activity null  created
2021-06-21 14:01:35,153 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution 315009 of activity null  initialized
2021-06-21 14:01:35,192 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution 315005 of activity _73040000067479  delet
ed
2021-06-21 14:01:35,194 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution: 315009 entity: 315010#JobEntityImpl  cre
ated
2021-06-21 14:01:35,194 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution: 315009 entity: 315010#JobEntityImpl  ini
tialized
2021-06-21 14:01:35,402 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution: 315009 entity: 315010#JobEntityImpl  del
eted
2021-06-21 14:01:35,407 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution: 315009 job: 315010  succeed
2021-06-21 14:01:35,451 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution 315011 of activity null  created
2021-06-21 14:01:35,452 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution 315011 of activity null  initialized
2021-06-21 14:01:35,546 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution 315009 of activity _73040000067479  delet
ed
2021-06-21 14:01:35,546 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
```

```
un 21 14:01:35 CST 2021 process: 312505 activity: _73040000067479  started
2021-06-21 14:01:35,546 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution 315013 of activity null  created
2021-06-21 14:01:35,546 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution 315013 of activity null  initialized
2021-06-21 14:01:35,547 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 activity: _73040000067535  started
2021-06-21 14:01:35,547 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 activity: _73040000067535  completed
2021-06-21 14:01:35,547 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 activity: _73040000067535_p  started
2021-06-21 14:01:35,547 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 activity: _73040000067535_p  completed
2021-06-21 14:01:35,656 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution: 315013 entity: 315016#JobEntityImpl  cre
ated
2021-06-21 14:01:35,656 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution: 315013 entity: 315016#JobEntityImpl  ini
tialized
2021-06-21 14:01:35,866 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution: 315013 entity: 315016#JobEntityImpl  del
eted
2021-06-21 14:01:35,874 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 execution: 315013 job: 315016  succeed
2021-06-21 14:01:35,876 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:35 CST 2021 process: 312505 activity: _73040000067480  started
2021-06-21 14:01:36,148 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 activity: _73040000067469_er  error received
2021-06-21 14:01:36,256 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 activity: _73040000067480  cancelled
2021-06-21 14:01:36,258 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 execution 315013 of activity _73040000067480  delet
ed
2021-06-21 14:01:36,293 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 activity: _73040000067479  cancelled
2021-06-21 14:01:36,305 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 execution 315011 of activity _73040000067479  delet
ed
2021-06-21 14:01:36,364 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 activity: _73040000067478  cancelled
2021-06-21 14:01:36,364 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 execution 315003 of activity _73040000067478  delet
ed
2021-06-21 14:01:36,393 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 activity: _73040000067471  cancelled
2021-06-21 14:01:36,394 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 execution 312526 of activity _73040000067471  delet
ed
2021-06-21 14:01:36,419 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 activity: _73040000067470  cancelled
2021-06-21 14:01:36,419 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 execution 312518 of activity _73040000067470  delet
ed
2021-06-21 14:01:36,453 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
```

```
un 21 14:01:36 CST 2021 process: 312505 activity: _73040000067469  cancelled
2021-06-21 14:01:36,453 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 execution 312510 of activity _73040000067469  delet
ed
2021-06-21 14:01:36,459 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 activity: _73040000067469_er  completed
2021-06-21 14:01:36,459 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 activity: _73040000067512  started
2021-06-21 14:01:36,464 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 activity: _73040000067512  completed
2021-06-21 14:01:36,515 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 execution 312512 of activity _73040000067512  delet
ed
2021-06-21 14:01:36,606 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 execution 312505 of activity null  deleted
2021-06-21 14:01:36,606 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [] Mon J
un 21 14:01:36 CST 2021 process: 312505 completed with root execution: 312505
2021-06-21 15:06:23,774 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [6458961
f1624259174743824397463] Mon Jun 21 15:06:23 CST 2021 process: 320005 execution 320005 of a
ctivity null  created
2021-06-21 15:06:23,775 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [6458961
f1624259174743824397463] Mon Jun 21 15:06:23 CST 2021 process: 320005 execution 320005 of a
ctivity null  initialized
2021-06-21 15:06:23,775 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [6458961
f1624259174743824397463] Mon Jun 21 15:06:23 CST 2021 process: 320005 execution 320006 of a
ctivity null  created
2021-06-21 15:06:23,775 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [6458961
f1624259174743824397463] Mon Jun 21 15:06:23 CST 2021 process: 320005 execution 320006 of a
ctivity null  initialized
2021-06-21 15:06:23,793 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [6458961
f1624259174743824397463] Mon Jun 21 15:06:23 CST 2021 process: 320005 started with root exe
cution: 320006
2021-06-21 15:06:23,822 [  - /// - ] INFO  activiti.DefaultActivitiEventListener - [6458961
f1624259174743824397463] Mon Jun 21 15:06:23 CST 2021 process: 320005 activity: _7304000006
8352  started
```

日志清理

支持日志自动清理，定时任务自动清理。清理脚本路径为：`/home/admin/bin/zclean.sh`。

```
crontab -l -u admin
* * * * *  /bin/bash /home/admin/bin/zclean.sh >> /home/admin/logs/zclean.log 2>&1
```

4.3. 服务巡检

云游页面检查

登录云游 Local，检查各页签功能。

关键页面：

- 交付升级 Tab 页。

- 产品基线 Tab 页。
- 解决方案详情页。
- 产品基线-产品详情页。

关注点：

各页面可正常访问，无报错。

apyunqing 容器资源使用情况检查

在 apyunqing 宿主机上执行如下命令：

```
docker exec -it <id> /bin/bash
tsar --cpu
tsar --mem
tsar --partition
netstat -an |grep 81
```

关注点：

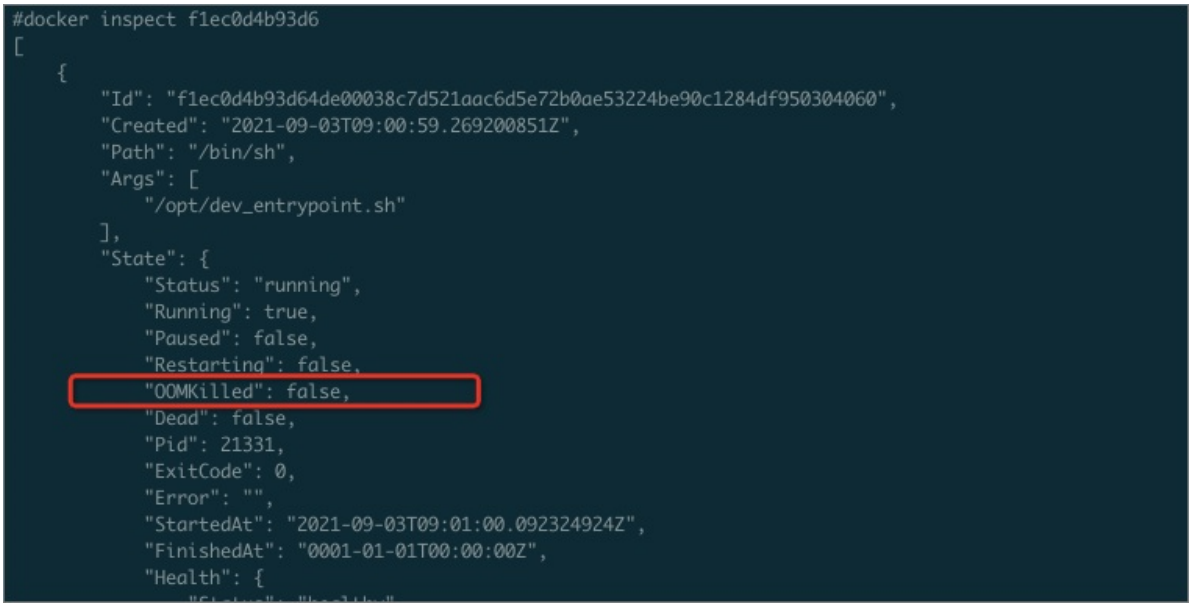
关注点	说明
tsar CPU 使用情况	CPU 使用率不超过 90%。
tsar 内存使用率	内存使用率不超过 90%。
tsar 磁盘使用率	磁盘使用率不超过 90%
端口监控	81 端口正常监听。

排查建议

若服务出现问题，云游不可用时，请从以下几个方面进行排查。

- 查看 `common-error.log` 日志，根据具体的 Error 日志进行处理。详情请参见 一般错误日志。
- 使用 `ps -ef|grep java` 命令查看业务进程是否存在，若不存在，需重启进程。您可以通过 `docker restart <云游容器 ID>` 命令重启云游。

- 您可以通过 `docker inspect` 命令查看 Java 进程是不是因为 OOM 被关闭。



4.4. 常见运维场景

4.4.1. 常见问题

本文介绍云游的常见运维场景。

常见问题排查思路

1. 查看云游前端暴露的任务日志，一般可定位到大部分问题。例如参数渲染错误、数据库、负载规划配置异常等。排除因为误操作导致的失败。
2. 如果任务日志信息解决不了，查看云游的 `common-error.log` 日志，按照日志的提示处理。
3. 如果还是解决不了，通过 Captain 平台查看相关 Pod 的状态、Event 日志、容器启动参数是否和预期一致等。
4. 如果 Pod 状态没有问题，说明底层的集群等相关 Controller 组件可能出现了异常，查看这些组件是否正常工作。

   您需要联系底层集群和 Controller 相关运维人员，查看这些组件是否存在异常。组件恢复之后，重试云游发布单。

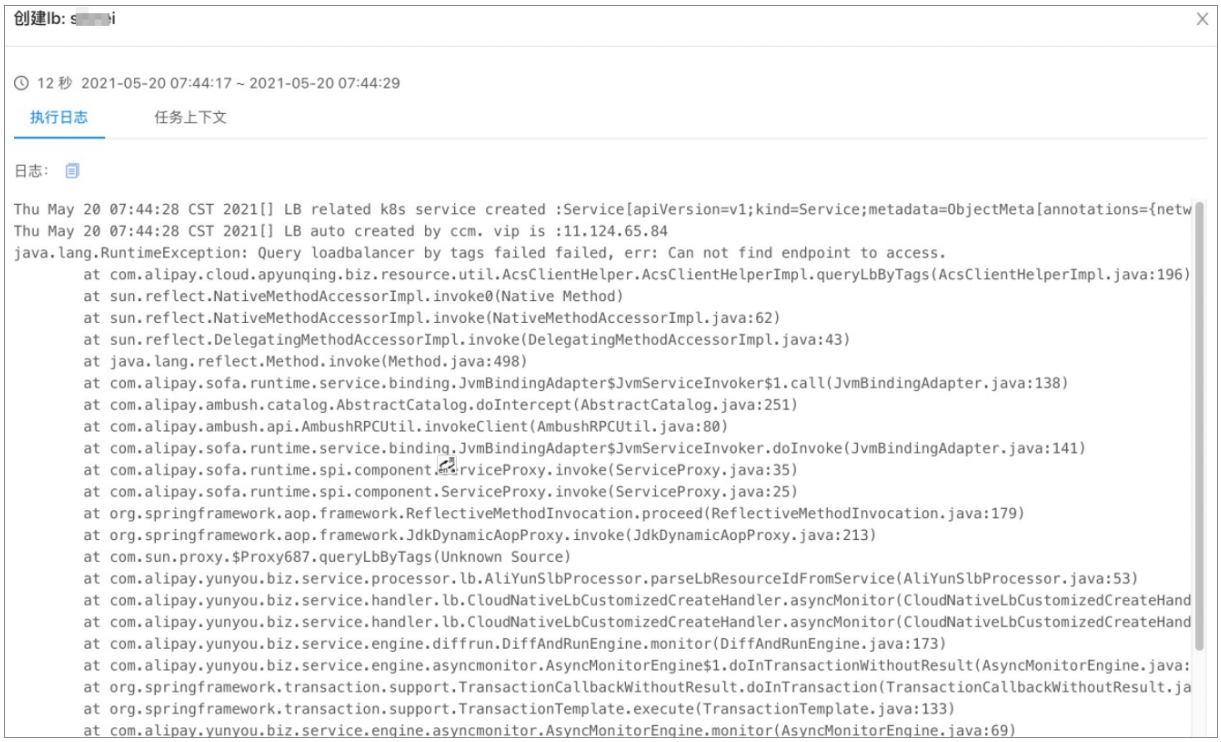
主要问题分类：

1. 用户误操作导致的配置错误、发布操作失败。
2. 集群、节点、核心组件异常。

创建 LB 资源时卡单

问题现象：

创建 LB 资源时出现卡单，查看日志内容如下图所示：



问题原因：

云游配置错误，导致无法成功调用阿里云 API。

解决方案：

根据日志反馈，检查云游的相关配置，并修正错误配置。

配置地址为：`http://{云游域名}:81/envs/{envId}/system-settings`，配置项如下：

- `LB_REGION_KEY`
- `aliyun_vpc_id`
- `aliyun_access_key`
- `aliyun_access_key_secret`（加密后的）
- `aliyun_slb_endpoint`
- `aliyun_slb_host`
- `aliyun_cloud_instance_type`
- `aliyun_cloud_uid`

这些参数是从阿里云平台获取，请自行检查。

删除已创建的 Sidecar 相关任务时卡单

问题现象：

产品容器加 Sidecar 容器一共两个，登陆集群发现 Pod 存在 3 个容器。

问题原因：

存量 Sidecar 未清除。

解决方案：

手动清除多余资源，操作步骤如下：

1. 查看集群内 Sidecar 资源。

```
kubectl get sidecarsets.apps.kruise.io
```

查看发现集群内存在多个 Sidecar 资源，如图所示：

```
[junbo@B-54LWMD6R-2048 kubeconfig % k get sidecarsets.apps.kruise.io
NAME                                    MATCHED   UPDATED   READY   AGE
asp-mpaas-mcube-2410-mcube              2         2         2       19h
asp-mpaas-mdsweb-1320-mdsweb            2         2         2       18h
asp-mpaas-mgs-13412-mpaasgw             0         0         0       59d
asp-mpaas-mps-1192-mcometgw             0         0         0       58d
asp-mpaas-mps-1192-yunpushcore          0         0         0       58d
asp-mpaas-mss-1110-msync2               2         2         0       10m
asp-mpaas-mss-193-msync2                2         2         0       58d
asp-mproxy-122-mproxy                   2         2         1       58d
asp-sidecartest2-100-chmappcenter       4         4         2       62d
asp-yunyoutest2-1110-promethus2         0         0         0       61d
sidecartest-100-apptest                 0         0         0       63d
yunyoutest-mpaas-mas-9170-mpaas-mdap    0         0         0       10d
yunyoutest-sidecartest2-100-chmappcenter 0        0         0       62d
```

2. 删除需要被清理的 Sidecar 资源。

```
kubectl delete sidecarsets.apps.kruise.io asp-mpaas-mss-193-msync2
```

3. 删除 Pod，使其被 stateful 重新启动。

```
kubectl delete po -n asp-mpaas-mssasp-mpaas-mss-msync2-0 asp-mpaas-mss-msync2-1
```

4. 查看 Pod 状态。

```
kubectl get po -n asp-mpaas-mss
```

Pod 状态已经正常，如图所示：

```
NAME                      READY   STATUS    RESTARTS   AGE
asp-mpaas-mss-msync2-0    2/2     Running   0          12m
asp-mpaas-mss-msync2-1    2/2     Running   0          12m
```

云游 Local 控制台发布产品报错

问题现象：

云游 Local 控制台发布产品时出现“**451 业务异常**”的报错。

问题原因：

您之前删除过产品，集群中还有有残留项 Pod。

解决方案：

1. 手动删除残留项 Pod。

   操作步骤参见上文 删除已创建的 Sidecar 相关任务时卡单。

2. 重新发布产品。

删除 Pod、Ns 时处于 Terminating 状态

问题现象：

用户因为当前部署的服务状态异常，卸载重发。重新发布时，发布单创建失败。

问题原因：

Ns 卸载失败时处于 Terminating 状态，导致云游 Local 重新发布产品时默认为灰度发布。云游 Local 误认为部署单元等信息存在（实际上不存在），查询时导致 NPE。

解决方案：

强制删除 Terminating 状态的 Pod。操作步骤参见上文 删除已创建的 Sidecar 相关任务时卡单。

测试应用一次性任务发布失败

问题现象：

执行自动化测试时失败。

问题原因：

- 用户的测试应用输出的日志没有满足云游 Local 的需求。 输出的日志没有满足以下格式：

```
"[YuNY0u&tESt&RESuLt]:";
业务单元测试日志==》JSON 对象看查看云游 Local 代码===》云启标准输出
"[YuNY0u&tESt&END]";
```

- containerd 底座日志的问题。

  目前，云游 Local 查看 containerd 日志的方式强依赖 dockerClient，而 containerd 底座不支持。您可以使用 `kubectl` 命令查看日志，以一次性任务 ospaccoreinit 为例：

```
[root@master001 /root]
#kubectl -n dc-osp get pod
NAME                                READY   STATUS      RESTARTS   AGE
dc-osp-osp-0                        1/1     Running     0          5d7h
dc-osp-osp-1                        1/1     Running     0          5d7h
dc-osp-ospaccoreinit-0-qg2z5        0/1     Completed   0          5d7h

[root@master001 /root]
#kubectl -n dc-osp logs -f dc-osp-ospaccoreinit-0-qg2z5
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/accoreinit-0.0.1.jar!/BOOT-INF/lib/logback-classic-1.2.3.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/accoreinit-0.0.1.jar!/BOOT-INF/lib/slf4j-log4j12-1.7.29.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [ch.qos.logback.classic.util.ContextSelectorStaticBinder]

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.1.RELEASE)

2021-09-10 10:10:13.757  INFO 6 --- [           main] c.c.a.a.a.AccoreinitApplication          : Starting AccoreinitApplication v0.0.1 on dc-osp-ospaccoreinit-
0-qg2z5 with PID 6 (/opt/accoreinit-0.0.1.jar started by root in /opt)
2021-09-10 10:10:13.760  INFO 6 --- [           main] c.c.a.a.a.AccoreinitApplication          : No active profile set, falling back to default profiles: defau
lt
Sofa-Middleware-Log SLF4J : Actual binding is of type [ com.alipay.sofa.rest Logback ]
2021-09-10 10:10:14.971  INFO 6 --- [           main] com.alipay.sofa.common.log               : Sofa-Middleware-Log SLF4J : Actual binding is of type [ com.al
ipay.sofa.rest Logback ]
log4j:WARN No appenders could be found for logger (org.jboss.resteasy.plugins.providers.DocumentProvider).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
2021-09-10 10:10:16.857  INFO 6 --- [           main] c.c.a.a.a.config.SofaRestConfig          : init sofa rest config complete!
2021-09-10 10:10:16.858  INFO 6 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean 'sofaRestConfig' of type [cn.com.antcloud.accoreinit.acco
reinit.config.SofaRestConfig$$EnhancerBySpringCGLIB$$348242f9] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for a
uto-proxying)
2021-09-10 10:10:17.076  INFO 6 --- [           main] c.c.a.a.accoreinit.config.EnvInfo        : ========== Env read start ==========
2021-09-10 10:10:17.076  INFO 6 --- [           main] c.c.a.a.accoreinit.config.EnvInfo        : Env line: HOOK.changeUrl=/envs/73040000000006/ops-plans/730400
06280944
2021-09-10 10:10:17.077  INFO 6 --- [           main] c.c.a.a.accoreinit.config.EnvInfo        : Env line: POD_NAMESPACE=dc-osp
2021-09-10 10:10:17.077  INFO 6 --- [           main] c.c.a.a.accoreinit.config.EnvInfo        : Env line: timezone=Asia/Shanghai
2021-09-10 10:10:17.077  INFO 6 --- [           main] c.c.a.a.accoreinit.config.EnvInfo        : Env line: HOSTNAME=dc-osp-ospaccoreinit-0-qg2z5
2021-09-10 10:10:17.077  INFO 6 --- [           main] c.c.a.a.accoreinit.config.EnvInfo        : Env line: DC_LB_LB_OSP_OSP_INTERNAL_LB_35AEE26B_1575_469A_8_SV
C_PORT_80_TCP_PROTO=tcp
```

- 镜像异常。

  当镜像有问题时，容器日志格式如下：

```
standard_init_linux.go:211: exec user process caused "exec format error"
```

出现这种问题时，您需要将镜像改成对应的二进制。

解决方案：

按照上面的三个问题逐一排查。

容器创建成功，但页面上找不到

问题原因：

- Pod 没有被调度。
- Pod 已被调度，但没有合适的节点，导致调度失败。
- Pod 已被成功调度，但 kubelet 拉起 Pod 时，创建 sandbox 失败。例如网络插件（CNI）出现问题，或者 IP 不足等情况。

解决步骤：

1. 先检查 AKE 系统组件是否都运行正常（尤其是四大件）。
2. 到 Captain 工作负载中的 Pod 列表页搜索该 Pod 并查看状态。
3. 如果状态不是运行中，那么进入详情页查看 event，一般会有具体的问题信息，请根据信息来解决问题。例如无可调度节点，需要检查节点资源是否足够；无可分配 IP，需要进行 IP 资源扩充等。

容器无法升级或变配

问题现象：升级或变配时出现“Pod current status is XXX, not allow RequestUpgrade”报错。

问题原因：

- 上一次的容器升级或变配还在进行中，云游在 apiproxy 的操作结束前重复发起了请求。
- 由于节点资源不足或 Pod 本身的各类异常问题导致升级或变配失败。

解决步骤：

1. 如果上一次升级或变配操作未完成，先等待其完成，然后重试。
2. 如果还是有问题，可以通过 Captain 工作负载中的 Pod 列表找到相应容器，然后单击 **编辑 YAML**，查看 status 字段下是否有错误信息，并根据具体错误信息进行排查解决。

4.4.2. 修改数据库密码

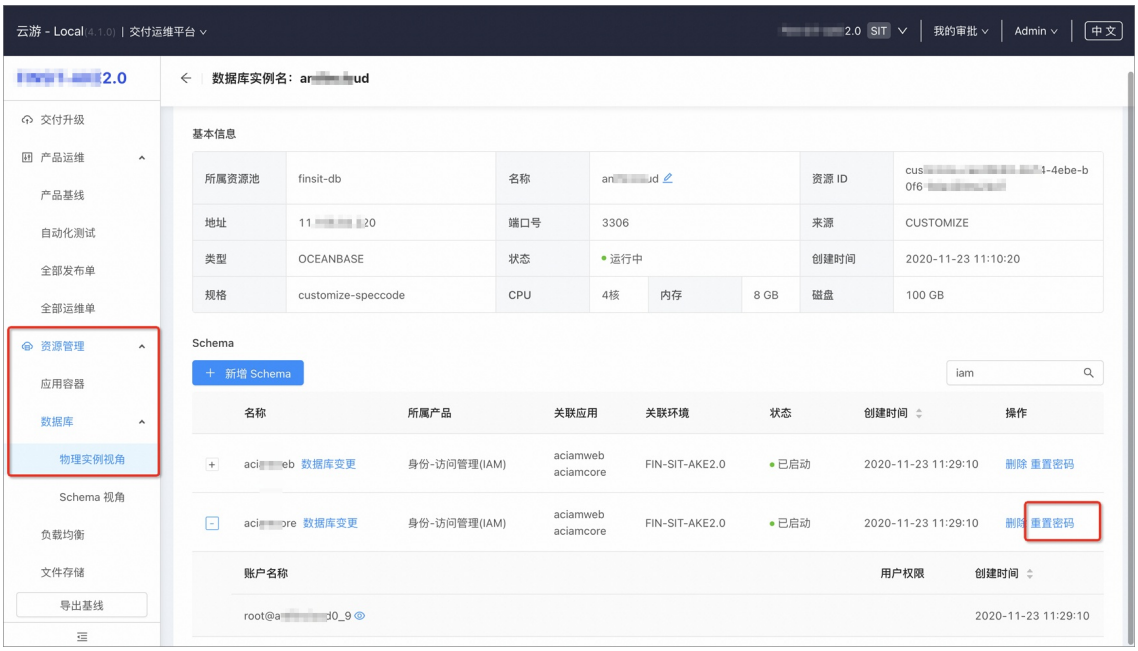
为了您的数据安全，建议您定期修改所有应用的数据库账号密码，并在云游上修改对应产品的数据库账号密码。密码修改后，您需要重发应用。

操作步骤

若您使用的是云游 Plus，请按照如下步骤修改：

1. 在 OB 侧更新对应租户的账号和密码。
2. 在云游 Local 中修改产品对应的 Schema 密码。
   - 云游 Plus 环境
     a. 登录云游 Local。
     b. 在左侧导航栏，选择 **资源管理 > 数据库 > 物理实例视角**。
     c. 单击目标数据库实例名称。

d. 在 **Schema** 区域单击目标 Schema 右侧的 **重置密码**。



e. 在 **重置密码** 对话框输入新的数据库密码，然后单击 **确定**。

- 老版本云游环境

a. 加密数据库的新密码。

> **重要**
>
> 您的电脑必须安装 Java。

a. 下载 AESUtil.class.tar。

b. 将文件解压到任意目录。

c. 通过命令进入工具解压目录，然后执行以下命令：

```
java AESUtil [encrypt|decrypt] [password]
```

- 加密示例

```
java AESUtil encrypt Aliyun@_123
encrypted string: dm91uiJM3kZwsRBacSRndA==
```

- 解密示例

```
# java AESUtil decrypt dm91uiJM3kZwsRBacSRndA==
decrypted string: Aliyun@_123
```

b. 使用得到的密文更新云游的数据库中 `baseline_rdb_user` 表中 `real_user_name` 对应的 password 字段。

```
update baseline_rdb_user set password='dm91uiJM3kZwsRBacSRndA==' where real_user_name
='apmonitorcore@obpaas_e8b2b06d_ID_52';
```

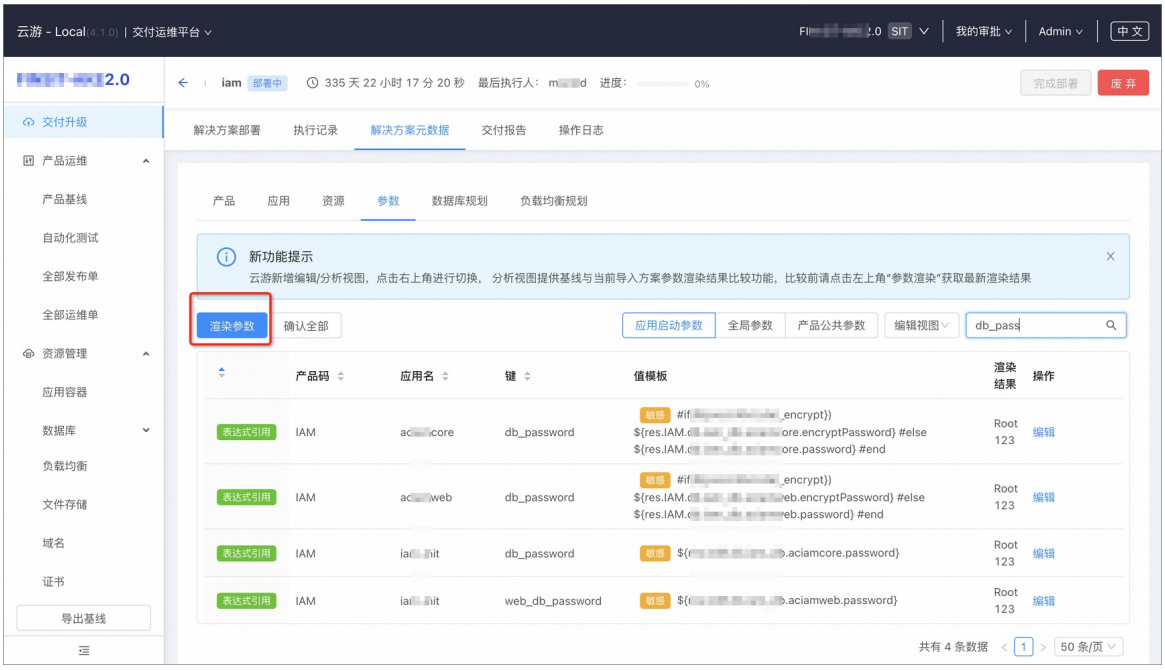
3. 重新渲染解决方案。

i. 在云游 Local 控制台单击 **交付升级**。

ii. 单击目标应用名称。

iii. 单击 **解决元数据**，然后单击 **参数**。

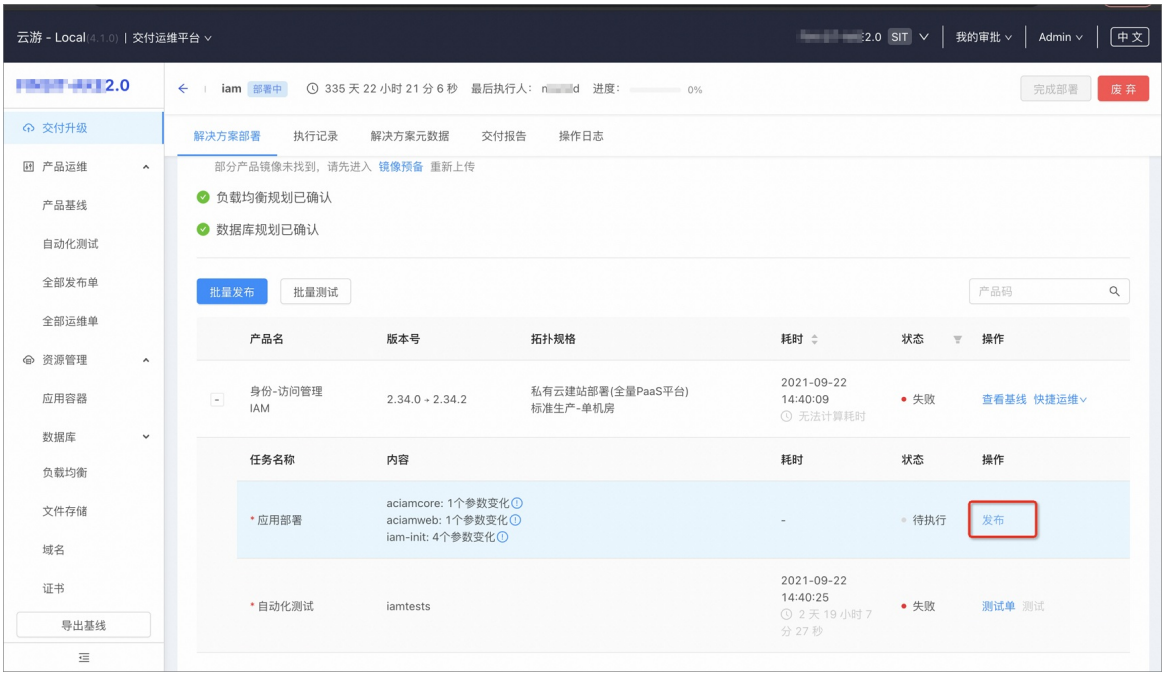
iv. 单击 **渲染参数**，然后单击 **确定**。



4. 重新发布应用。

i. 在应用详情页单击 **解决方案部署**。

ii. 单击 **应用部署** 右侧的 **发布**，并按照提示完成发布。



4.5. 云游接入 IAM 审计功能

> **说明**
>
> 云游 Local 1.10.0 及以上版本支持接入 IAM 审计功能。本文以云游 Local 1.10.2 版本为例，请根据实际情况修改版本。

准备工作

- 下载 apyunqingplus 镜像。

```
docker run --net=host -d -v /home/yunyou/staticfiles:/home/staticfiles acs-reg.alipay.com
/acloud/apyunqingplus:EI62154649_20211122_20211122161132_cdd9ca69
```

- 下载 apyunqingsetup 镜像。

```
docker run --net=host -d -v /home/yunyou/staticfiles:/home/staticfiles acs-reg.alipay.com
/acloud/apyunqingsetup:master_20211022091743_dcb7629d
```

- 备份 apyunqing 的数据库。
- 导入云游 Plus 1.10.2 的镜像到镜像仓库。

```
docker push acs-reg.alipay.com/acloud/registry:1.1
```

升级云游

1. 登录云游 Local 服务器。
2. 进入 `apyunqing-compose` 目录。

```
 cd /home/yunyou/staticfiles/apyunqing-compose
```

3. 修改 `apyunqing-compose` 目录下的 `config.json` 文件。

   在 `config.json` 中添加以下环境变量：

```
"ANTCLOUD_SOFA_iam_customer_id=0000000001",
"ANTCLOUD_SOFA_audit_address=iamcore.jr.alipay.net",
```

   `audit_address` 对应地址为可用的 aciamcore 地址。

4. 升级云游 Plus 镜像。

```
python yunqing-setup.py y --image <apyunqingplus 镜像>
```

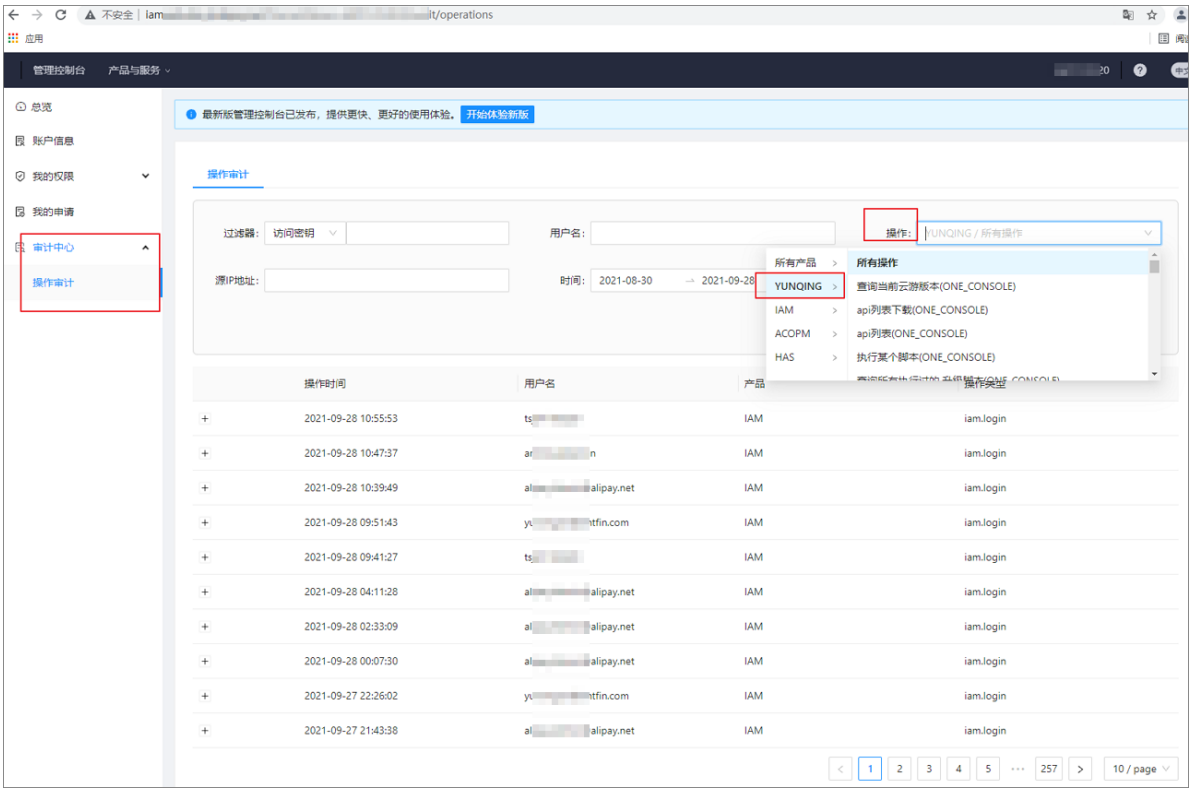
导入审计元数据

1. 下载 SQL 脚本。
   - operation_type.sql
   - operation_type_onex_extra.sql
2. 在对应环境的数据库 aciamcore 中执行步骤 1 下载的 SQL 脚本来导入已经准备好的审计元数据。

查看审计结果

1. 登录 IAM 控制台。
2. 在左侧导航栏选择 **审计中心** > **操作审计**。

3. 在 **操作** 区域选择 **YUNQING**。



附录：回滚方案

1. 登录云游 Local 服务器。
2. 使用之前镜像对云游 Local 进行回滚。

```
cd /home/yunyou/staticfiles/apyunqing-compose
## 回滚云游 Plus 镜像
python yunqing-setup.py y --image <apyunqingplus 上一版本镜像>
```

3. 还原数据库。

```
mysql -uapyunqing -pAntcloud2018 -h127.0.0.1 apyunqing < apyunqing.dump --force
```

4.6. 服务异常应急预案

云游依赖的数据库不可用

云游一般使用外部提供的数据库，需要数据库提供方检查数据库服务。从云游检查数据库链接的方法如下：

1. 进入云游容器。

```
docker exec -it apyunqing bash
```

2. 链接数据库。

```
mysql -u $ANTCLOUD_SOFA_yunqing_db_username \
-h $ANTCLOUD_SOFA_yunqing_db_url \
-P $ANTCLOUD_SOFA_yunqing_db_port \
-p$ANTCLOUD_SOFA_yunqing_db_password
```

云游服务不可用（请求 504、OOM 等异常情况）

您可以通过以下命令重启云游服务：

```
docker restart apyunqing
```

单机房部署下的单机宕机

单机房部署时，同一机房下，通常会在 osp1、osp2 部署 2 个实例共用一个数据库，单击宕机无任何影响，您仅需重启宕机的容器即可。

多机房容灾架构下的单机房宕机

双机房场景下，通常主备机房各部署一台云游。当主机房发生故障时，您可以将数据库连接到备机房。当机房恢复后，容器启动即可恢复。

5.容器底座

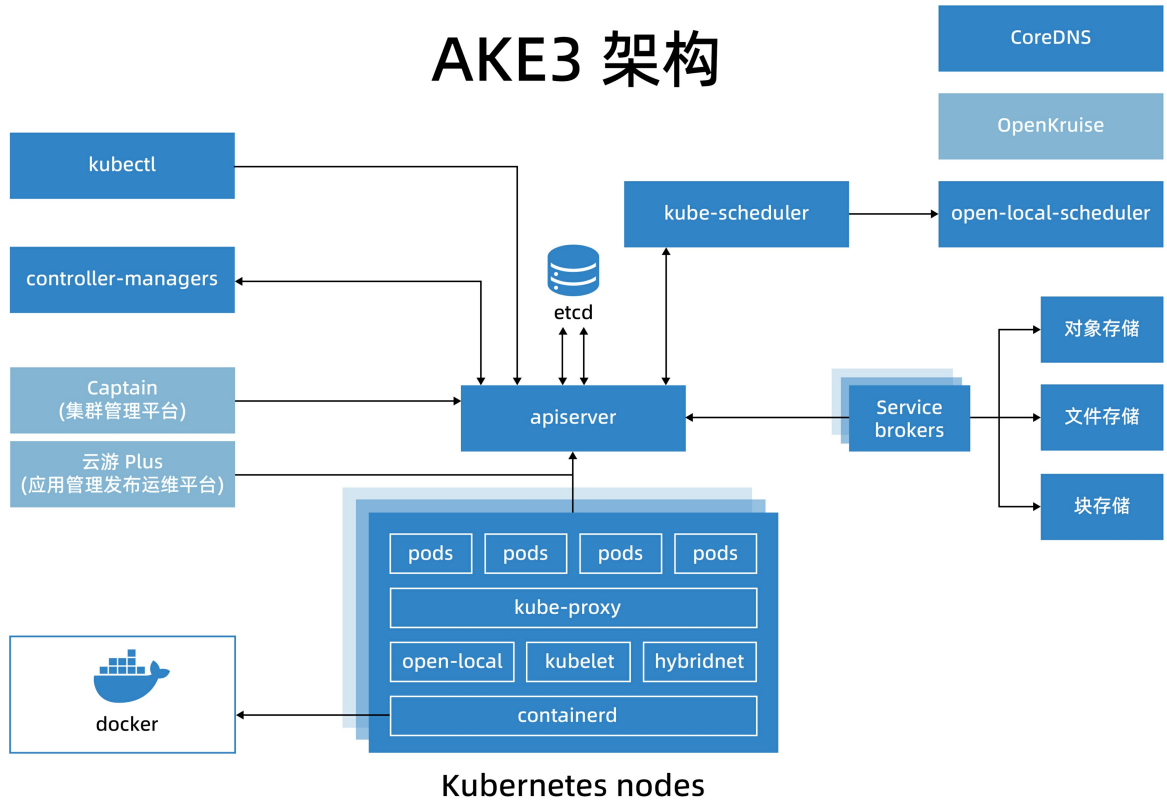
5.1. 产品架构

云原生容器底座是一个基于 Kubernetes 和 Containerd 的应用管理平台，用于部署蚂蚁各产品组件，主要由容器引擎 AKE 云原生平台（AKE3）和产品控制台 Captain Plus 构成。本文介绍容器底座的产品架构。

技术架构

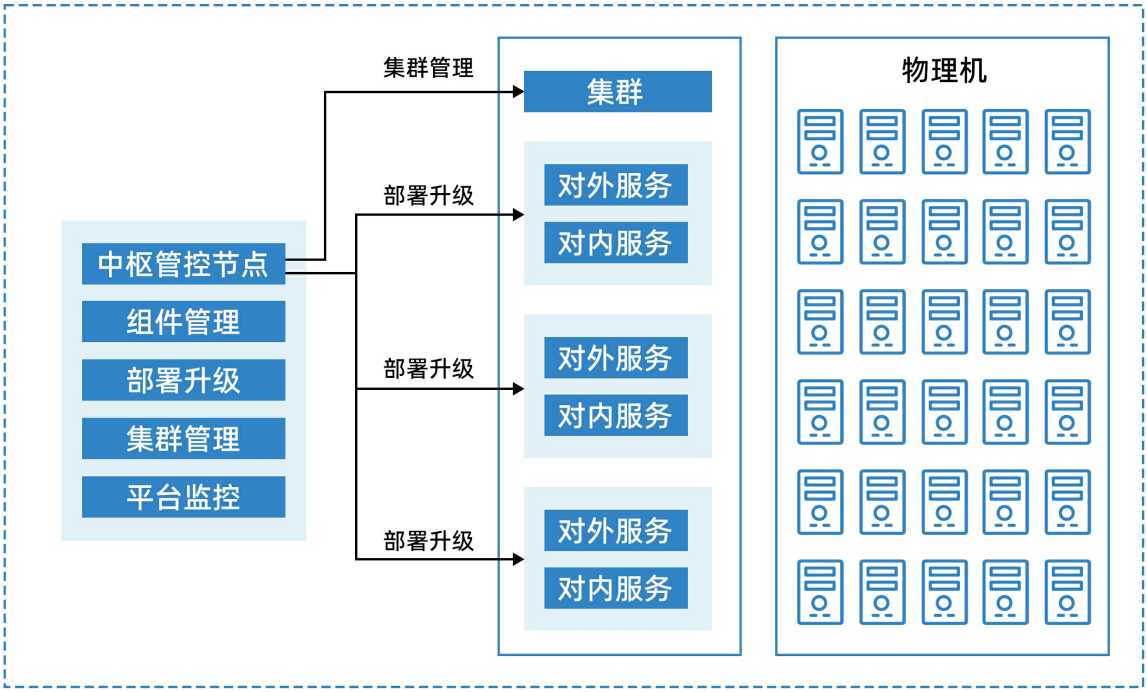
云原生容器底座中的 AKE3 构建在开源的 Kubernetes 和 Containerd 技术之上。通过蚂蚁自研的组件，如网络、调度、存储、日志等，对开源产品进行了适配和增强，更适合开箱即用和追求稳定的用户场景。

Captain Plus 通过打通用户中心和镜像中心，提供了可视化操作界面，方便对 AKE3 集群进行管理升级，也可以进行工作负载的发布运维，同时集成的应用商店可以一键部署已定义和自定义组件。



部署架构

云原生容器底座支持多种部署架构。在 AntStack Plus 产品输出时，IaaS 层一般为物理机，因此 AKE3 提供的是一个集群，通过为各节点服务器打标，完成产品部署。由云游完成所有 IaaS、PaaS 产品的部署和升级。



5.2. AKE 集群备份机制

本文介绍 AKE 集群的备份机制。

备份 etcd

前置条件

etcd 正常运行。

操作步骤

1. 登录任意一台 etcd 节点。
2. 备份 etcd 至指定目录。
   - 有 docker 的集群

     执行以下脚本，将 etcd 数据备份至指定目录。其中 `ETCD_BACKUP_DIR` 可根据需要自行替换。

```
ETCD_ID=$(sudo docker ps | grep k8s_etcd_etcd | awk '{print $1}')

#变量，etcd 3.3 ENDPOINTS 备份只要一个就可以。
ENDPOINTS="https://127.0.0.1:2379"
CACERT="/etc/kubernetes/pki/etcd/ca.crt"
CERT="/etc/kubernetes/pki/etcd/server.crt"
KEY="/etc/kubernetes/pki/etcd/server.key"
ETCDCTL_API=3
ETCD_BACKUP_DIR=/home/t4/etcd/etcd_bk

docker cp ${ETCD_ID}:/usr/local/bin/etcdctl /usr/local/bin/etcdctl && chmod +x /usr/loc
al/bin/etcdctl
docker exec ${ETCD_ID} cat /etc/kubernetes/pki/etcd/..data/server.key > ${KEY}
docker exec ${ETCD_ID} cat /etc/kubernetes/pki/etcd/..data/server.crt > ${CERT}
docker exec ${ETCD_ID} cat /etc/kubernetes/pki/etcd/..data/ca.crt > ${CACERT}

#检测备份目录。
mkdir -p ${ETCD_BACKUP_DIR}

# 执行备份命令。
/usr/local/bin/etcdctl --cacert=${CACERT} --cert=${CERT} --key=${KEY}  --endpoints=${EN
DPOINTS} snapshot save ${ETCD_BACKUP_DIR}/etcd_v3_$(date +"%Y-%m-%d").db
```

以上脚本可通过 crontab 定时执行。

- 没有 docker 的集群

a. 安装 etcdctl。

```
#!/bin/bash
ETCD_VER=v3.4.13
ETCD_DIR=etcd-download
DOWNLOAD_URL=https://github.com/coreos/etcd/releases/download

# 下载 etcdctl。
cd /root
wget ${DOWNLOAD_URL}/${ETCD_VER}/etcd-${ETCD_VER}-linux-amd64.tar.gz
tar -xzvf etcd-${ETCD_VER}-linux-amd64.tar.gz

# 安装 etcdctl。
cd etcd-${ETCD_VER}-linux-amd64
cp etcdctl /usr/local/bin/
```

b. 进行 etcd 集群健康检查。

```
#变量，etcd 3.4.13 ENDPOINTS 备份只要一个就可以。
MASTER1IP="100.**.**.139"   //Master1 的实际 IP 地址。
MASTER2IP="100.**.**.141"   //Master2 的实际 IP 地址。
MASTER3IP="100.**.**.142"   //Master3 的实际 IP 地址。
CACERT="/etc/kubernetes/pki/etcd/ca.crt"
CERT="/etc/kubernetes/pki/etcd/server.crt"
KEY="/etc/kubernetes/pki/etcd/server.key"
ETCDCTL_API=3

#检查集群状态。
/usr/local/bin/etcdctl --endpoints=https://${MASTER1IP}:2379 \
--endpoints=https://${MASTER2IP}:2379 \
--endpoints=https://${MASTER3IP}:2379 \
--cacert=${CACERT} \
--key=${KEY} \
--cert=${CERT} \
endpoint health
```

c. 生成备份脚本 `/opt/etcd_back/etcd-backup.sh` 。

脚本内容如下：

```
#! /bin/bash

#变量，etcd 3.4.13 ENDPOINTS 备份只要一个就可以。
ENDPOINTS="https://127.0.0.1:2379"
CACERT="/etc/kubernetes/pki/etcd/ca.crt"
CERT="/etc/kubernetes/pki/etcd/server.crt"
KEY="/etc/kubernetes/pki/etcd/server.key"
ETCDCTL_API=3
ETCD_BACKUP_DIR=/home/t4/etcd/etcd_bk

#检测备份目录。
[ ! -d $ETCD_BACKUP_DIR ] && mkdir -p ${ETCD_BACKUP_DIR}

# 执行备份命令。
/usr/local/bin/etcdctl --endpoints=${ENDPOINTS} \
--cacert=${CACERT} \
--key=${KEY}  \
--cert=${CERT} \
snapshot save ${ETCD_BACKUP_DIR}/etcd_v3_$(date +"%Y-%m-%d").db

# 清理 7 天前的备份文件
find /home/t4/etcd/etcd_bk/ -mtime +7 -name "*.db" -exec rm -rf {} \;
```

d. 执行定时备份。

```
chmod +x /opt/etcd_back/etcd-backup.sh
# 加入 crontab 指令。
crontab -e
# 新增一行指令，每天凌晨2点执行。
0 2 * * * sh /opt/etcd_back/etcd-backup.sh
```

还原 etcd

前置条件

- 确保待还原的 etcd 节点上已经上传了备份文件以及 etcdctl 工具。
- 由于 etcd 备份还原存在数据丢失的情况，需在确认该风险的情况下执行还原操作。

操作步骤

1. 将 YAML 文件移到 `/root/` 目录下。三台机器都需要操作。

```
mv /etc/kubernetes/manifests/etcd.yaml /root/etcd.yaml
```

2. 找到之前的 etcd 的容器。

```
# 有 docker 的情况。
docker ps -a | grep etcd

# 无 docker 的情况，如 Antstack Plus 环境。
ctr -n k8s.io c ls | grep etcd
```

   容器会自动删除，请确保 etcd 容器被删除再执行后面步骤。

3. 删除 data 目录。

```
rm -rf /home/t4/etcd/data
```

4. 通过以下命令以此还原 etcd 数据。

   其中 `ETCD_ENDPOINTS` 、 `ETCD_BACKUP_FILE` 以及 etcd 节点的 IP 和端口需要根据实际情况做修改。

```
ETCD_BACKUP_FILE=/home/t4/etcd/etcd_bk/etcd_v3_2020-11-07.db

INITIAL_CLUSTER
#第一台机器
ETCD_ENDPOINTS="https://192.**.**.99:2379"  #ake1 的 IP。
ETCDCTL_API=3 etcdctl snapshot restore--endpoints=${ETCD_ENDPOINTS} --data-dir ${ETCD_BAC
KUP_FILE}\
--name kube-etcd1 \
--initial-cluster kube-etcd1=https://192.**.**.99:2380,kube-etcd2=https://192.**.**.98:23
80,kube-etcd3=https://192.**.**.97:2380\
--initial-cluster-token ant-k8s \
--initial-advertise-peer-urlshttps://192.**.**.99:2380

#第二台机器
ETCD_ENDPOINTS="https://192.**.**.98:2379"  #ake2 的 IP。
ETCDCTL_API=3 etcdctl snapshot restore--endpoints=${ETCD_ENDPOINTS} --data-dir ${ETCD_BAC
KUP_FILE}\
--name kube-etcd2 \
--initial-clusterkube-etcd1=https://192.**.**.99:2380,kube-etcd2=https://192.**.**.98:238
0,kube-etcd3=https://192.**.**.97:2380\
--initial-cluster-token ant-k8s \
--initial-advertise-peer-urlshttps://192.**.**.98:2380

#第三台机器
ETCD_ENDPOINTS="https://192.**.**.97:2379"  #ake3 的 IP。
ETCDCTL_API=3 etcdctl snapshot restore--endpoints=${ETCD_ENDPOINTS} --data-dir ${ETCD_BAC
KUP_FILE}\
--name kube-etcd3 \
--initial-cluster kube-etcd1=https://192.**.**.99:2380,kube-etcd2=https://192.**.**.98:23
80,kube-etcd3=https://192.**.**.97:2380\
--initial-cluster-token ant-k8s \
--initial-advertise-peer-urlshttps://192.**.**.97:2380
```

5. 将 YAML 文件移回 `etc` 目录下。

```
mv /root/etcd.yaml /etc/kubernetes/manifests/
```

5.3. 集群证书备份

AKE3 集群搭建完成后，需要备份 master 节点上的证书。操作命令如下：

```
mkdir /home/t4/master_bk
tar -zcvf /home/t4/master_bk/ake3-master-$(ifconfig eth0 | grep 'inet addr:' | awk '{print
$2}' | cut -c 6-)-$(date +%Y-%m-%d_%H:%M:%S_%Z).tar.gz /etc/kubernetes /var/lib/kubelet
```

5.4. 预警和监控

5.4.1. AKE3 监控项说明

云原生容器底座中的 AKE 云原生平台（AKE3）构建在开源的 Kubernetes 和 Containerd 技术之上。通过蚂蚁自研的组件，如网络、调度、存储、日志等，对开源产品进行了适配和增强，更适合开箱即用和追求稳定的用户场景。本文介绍 AKE3 的监控项。

AKE3 的监控项如下表所示：

应用	指标	告警阈值	告警间隔（分钟）	说明	告警处理方式
containerd	cpu_usage	>90%	3	containerd 的 CPU 使用率监控。	使用 `top` 命令查看物理机资源使用率。
containerd	mem_usage	>90%	3	containerd 的内存使用率监控。	使用 `top` 命令查看物理机资源使用率。
containerd	disk_usage	>90%	3	containerd 的磁盘使用率监控。	使用 `top` 命令查看物理机资源使用率。
containerd	containerd.sock/2376	不通	3	dockerd 的端口监控。 监控暂时不支持 sock 检查。	使用 `systemctl status containerd` 命令查看 containerd 状态。
kubelet	cpu_usage	>90%	3	kubelet 的 CPU 使用率监控。	使用 `top` 命令查看物理机资源使用率。
kubelet	mem_usage	>90%	3	kubelet 的内存使用率监控。	使用 `top` 命令查看物理机资源使用率。
kubelet	disk_usage	>90%	3	kubelet 的磁盘使用率监控。	使用 `top` 命令查看物理机资源使用率。
kubelet	"4194"	不通	3	kubelet 的端口监控。	使用 `systemctl status kubelet` 命令查看 kubelet 状态。
kube-proxy	cpu_usage	>90%	3	kube-proxy 的 CPU 使用率监控。	使用 `top` 命令查看物理机资源使用率。
kube-proxy	mem_usage	>90%	3	kube-proxy 的内存使用率监控。	使用 `top` 命令查看物理机资源使用率。

kube-proxy	disk_usage	>90%	3	kube-proxy 的磁盘使用率监控。	使用 `top` 命令查看物理机资源使用率。
kube-proxy	"10256"	不通	3	kube-proxy 的端口监控。	使用 `kubectl -n kube-system get daemonset kube-proxy-daemonset` 和 `kubectl -n kube-system get pod -o wide -l k8s-app=kube-proxy` 命令查看 kube-proxy 的状态。
ETCD	cpu_usage	>90%	3	ETCD 的 CPU 使用率监控。	使用 `kubectl top` 命令查看容器资源使用率。
ETCD	mem_usage	>90%	3	ETCD 的内存使用率监控。	使用 `kubectl top` 命令查看容器资源使用率
ETCD	"2379"	不通	3	ETCD 的端口监控。	使用 `systemctl restart kubelet` 命令重启服务。

apiserver	cpu_usage	>90%	3	apiserver 的 CPU 使用率监控。	在各 master 节点执行 `top -p `pgrep kube-apiserver`` 命令查看容器资源使用率。
					使用如下命令修改 CPU 字段：
					`cp /etc/kubernetes/manifests/kube- apiserver.yaml /tmp/`
					`vi /tmp/kube-apiserver.yaml #修改 CPU 字段。`
					`cp -f /tmp/kube-apiserver.yaml /etc/kubernetes/manifests/kube- apiserver.yaml #保存修改结果。`

apiserver	mem_usage	>90%	3	apiserver 的内存使用率监控。	在各 master 节点执行 `top -p `pgrep kube-apiserver`` 命令查看容器资源使用率。 使用如下命令修改 MEM 字段： `cp /etc/kubernetes/manifests/kube- apiserver.yaml /tmp/` `vi /tmp/kube-apiserver.yaml #` **修改** `memory` **的值。** `cp -f /tmp/kube-apiserver.yaml /etc/kubernetes/manifests/kube- apiserver.yaml #` **保存修改结果。**
apiserver	"443"	不通	3	apiserver 的端口监控。	使用 `systemctl restart kubelet` 命令重启服务。
scheduler	cpu_usage	>90%	3	scheduler 的 CPU 使用率监控。	在各 master 节点执行 `top -p `pgrep kube-scheduler`` 命令查看容器资源使用率。 使用如下命令修改 CPU 字段： `cp /etc/kubernetes/manifests/kube-scheduler.yaml /tmp/` `vi /tmp/kube-scheduler.yaml #` **修改** `CPU` **字段。** `cp -f /tmp/kube-scheduler.yaml /etc/kubernetes/manifests/kube-scheduler.yaml #`**保存修改结果。**

scheduler	mem_usage	>90%	3	scheduler 的内存使用率监控。	在各 master 节点执行 `top -p `pgrep kube-scheduler`` 命令查看容器资源使用率。
					使用如下命令修改 MEM 字段：
					`cp /etc/kubernetes/manifests/kube-scheduler.yaml /tmp/`
					`vi /tmp/kube-scheduler.yaml # 修改 memory 的值。`
					`cp -f /tmp/kube-scheduler.yaml /etc/kubernetes/manifests/kube-scheduler.yaml #保存修改结果。`

controllermanager	cpu_usage	>90%	3	controllermanager 的 CPU 使用率监控。	在各 master 节点执行 `top -p `pgrep kube-controller`` 命令查看容器资源使用率。
					使用如下命令修改 CPU 字段：
					`cp /etc/kubernetes/manifests/kube- controller-manager.yaml /tmp/`
					`vi /tmp/kube- controller-manager.yaml # 修改 CPU 字段。`
					`cp -f /tmp/kube-controller-manager.yaml /etc/kubernetes/manifests/kube- controller-manager.yaml #保存修改结果。`

controllermanager	mem_usage	>90%	3	controllermanager 的内存使用率监控。	在各 master 节点执行 `top -p \`pgrep kube-controller\`` 命令查看容器资源使用率。 使用如下命令修改 MEM 字段： `cp /etc/kubernetes/manifests/kube- controller-manager.yaml /tmp/` `vi /tmp/kube- controller-manager.yaml #` **修改** `memory` **的值。** `cp -f /tmp/kube-controller-manager.yaml /etc/kubernetes/manifests/kube- controller-manager.yaml #`**保存修改结果。**
apiserver	“apiserver.failure”	error 关键字 count() > 1	3	无	无
controller-manager	“controller-manager.failure”	error 关键字 count() > 1	3	无	无
scheduler	“scheduler.failure”	error 关键字 count() > 1	3	无	无

5.4.2. 查看监控报警

本文介绍如何查看 AKE 云原生平台（AKE3）相关系统组件有无异常告警以及您需要关注的监控项说明。

操作步骤

1. 登录 RMS 控制台。
2. 在左侧导航栏选择 **告警管理** > **告警历史**。
3. 在 **告警历史** 页面右上角选择需要查看的日期后单击 **确定**。

   页面将显示指定时段的数据源、告警类型、规则详情等信息。

关注监控项

您需要关注的监控项内容如下：

应用	分类	指标(关键字)	告警阈值
ake-sigma-proxyake-apiserverake-controller-managerake-etcdake-scheduler	基础监控-CPU 使用率监控	cpu_util	>90%
	基础监控-内存使用率监控	mem_util	>90%
	基础监控-磁盘使用率监控	disk_util	>90%
ake-sigma-proxy	基础监控-端口监控	18080	不通
ake-kubelet	基础监控-端口监控	10250、10255	不通
ake-apiserver	基础监控-端口监控	6443	不通
ake-controller-manager	基础监控-端口监控	10252（仅元集群）	不通
ake-etcd	基础监控-端口监控	2379	不通
ake-cni-service	基础监控-端口监控	8849	不通
ake-dockerd	基础监控-端口监控	4243	不通

5.5. 系统日志

本文介绍容器底座相关系统日志路径。

日志目录

- 系统组件

组件名称	日志目录
containerd	/home/t4/containerd/logs
kubelet	/home/t4/sigma-slave/logs

组件名称	日志目录
kube-apiserver	/home/t4/sigma-master/logs/apiserver
kube-controller-manager	/home/t4/sigma-master/logs/controller-manager
kube-scheduler	/home/t4/sigma-master/logs/scheduler

- 容器化部署组件
  - etcd
  - rama-daemon、rama-webhook、rama-manager
  - minio

  容器化部署组件可以通过以下方式查看日志：
  - 通过 k8s 命令查看日志

    以查看 etcd 日志为例，命令如下：

    ```
    kubectl get pod -o wide -n kube-system | grep etcd
    kubectl logs --tail=100 -f etcd-11.**.**.73 -n kube-system
    ```
  - 在 Pod 所在的节点上查看

    命令如下：

    ```
    kubectl get pod -n kube-system -o wide |grep rama-daemon |grep 11.**.**.73
    ```

日志清理

支持日志自动清理，定时任务自动清理。清理脚本路径为：`/home/admin/bin/zclean.sh`。

```
crontab -l -u admin
* * * * *  /bin/bash /home/admin/bin/zclean.sh >> /home/admin/logs/zclean.log 2>&1
```

5.6. 服务巡检

5.6.1. 集群信息概览

操作步骤

1. 登录云 Captain 控制台。
2. 单击 **集群管理**，然后单击目标元集群。
3. 单击 **概览**。

关注点

查看群节点、容器组（Pod）的健康情况，以及集群的 CPU、内存、磁盘等基础资源使用情况。

5.6.2. Deployment 部署组件巡检

操作步骤

1. 登录云 Captain 控制台。
2. 单击 **集群管理**，然后单击目标元集群。
3. 选择 **工作负载 > 部署（Deployments）**。
4. 在 **部署（Deployments）**页面选择命名空间为 **kube-system**。

关注点

- 若 Deployment 状态为运行中，且 Pod 实际数等于期望数，则表示元集群组件运行正常。
- 单击指定 Deployment 名称查看 Deployment 详情，确认 Deployment 对应 Pod 运行情况、Deployment 事件和 Yaml。

5.6.3. DaemonSet 部署组件巡检

操作步骤

1. 登录云 Captain 控制台。
2. 单击 **集群管理**，然后单击目标元集群。
3. 选择 **工作负载 > 守护进程集（Daemonsets）**。
4. 在 **守护进程集（Daemonsets）**页面选择命名空间为 **kube-system**。

关注点

- 若 Deployment 状态为运行中，且 Pod 实际数等于期望数，则表示元集群组件运行正常。
- 单击指定 Deployment 名称查看 DaemonSet 对应 Pod 运行情况、DaemonSet 事件和 Yaml。DaemonSet 部署组件每个节点都会运行一个 Pod，若有运行异常的 Pod 则表示运行不正常。

5.6.4. StaticPod 和 Pod 部署组件巡检

操作步骤

1. 登录 Captain 控制台。
2. 单击 **集群管理**，然后单击目标元集群。
3. 选择 **工作负载 > 容器组（Pods）**。
4. 在 **容器组（Pods）**页面选择命名空间为 **kube-system**。

关注点

- 通过搜索框搜索带巡检的组件（如 kube-apiserver），如果 Pod 状态为 Running，则表示正常。
- 单击指定 Pod 查看 Pod 对应容器运行情况、Pod 事件等。Pod 中的所有容器状态为 Running 则表示正常，异常原因可通过 Pod 事件排查。

5.6.5. Node 部署组件巡检

操作步骤

1. 登录云 Captain 控制台。
2. 单击 **集群管理**，然后单击目标元集群。
3. 选择 **集群管理 > 节点（Nodes）**。

关注点

- 查看集群内所有节点的状态、CPU 分配率、内存分配率、磁盘分配率是否正常。
- 单击指定节点名称查看节点的 CPU 分配率、内存分配率、容器组分配率等是否有异常。

5.6.6. AKE 证书有效期巡检

执行以下命令，若发现有自签证书有效期临近当前日前（30 天 < X < 60 天），需要重新签证书：

```
for i in `ls -R /etc/kubernetes/pki/*.crt /etc/kubernetes/pki/etcd/*.crt`; do echo $i ; not
after=`openssl x509 -in $i -noout -dates | grep notAfter | awk -F"=" '{print $2}'` ; TZ=CST
date '+%Y-%m-%d %H:%M:%S' -d "$notafter" ; done
```

```
[root@master003 /etc/kubernetes/pki]
#for i in `ls -R /etc/kubernetes/pki/*.crt /etc/kubernetes/pki/etcd/*.crt`; do echo $i ; notafter=`openssl x509 -in $i -noout -dates | grep notAfter | awk -F"=
" '{print $2}'` ; TZ=CST date '+%Y-%m-%d %H:%M:%S' -d "$notafter" ; done
/etc/kubernetes/pki/admin.crt
2031-05-18 03:45:48
/etc/kubernetes/pki/apiserver.crt
2031-05-18 03:45:46
/etc/kubernetes/pki/apiserver-etcd-client.crt
2031-05-18 03:45:47
/etc/kubernetes/pki/apiserver-kubelet-client.crt
2031-05-18 03:45:47
/etc/kubernetes/pki/ca.crt
2031-05-18 03:45:40
/etc/kubernetes/pki/etcd/ca.crt
2031-05-18 03:45:40
/etc/kubernetes/pki/etcd/healthcheck-client.crt
2031-05-18 03:45:47
/etc/kubernetes/pki/etcd/peer.crt
2031-05-18 03:45:47
/etc/kubernetes/pki/etcd/server.crt
2031-05-18 03:45:46
/etc/kubernetes/pki/front-proxy-ca.crt
2031-05-18 03:45:40
/etc/kubernetes/pki/front-proxy-client.crt
2031-05-18 03:45:47
/etc/kubernetes/pki/kube-controller-manager.crt
2031-05-18 03:45:50
/etc/kubernetes/pki/kubelet.crt
2022-05-20 02:48:46
/etc/kubernetes/pki/kube-scheduler.crt
2031-05-18 03:45:50

[root@master003 /etc/kubernetes/pki]
#
```

5.7. 常见运维场景

5.7.1. Captain 运维常见问题

常见问题排查思路

- 检查节点的 `/etc/resolv.conf` 是否符合预期。
- 检查节点的 `/etc/hosts` 是否符合预期。
- 检查节点的 selinux 和防火墙配置是否符合预期。

  命令如下：

```
# getenforce
Disabled

#systemctl status firewalld
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: di
sabled)
   Active: inactive (dead)
     Docs: man:firewalld(1)
```

- 出现节点网络不通或容器无法使用 telnet 命令连接服务的情况，可以先尝试重启节点的 rama-daemonset 插件。
- 集群出现无法调度或容器无法启动的情况，可以查看如下日志：
  - kubelet

    ```
    /home/t4/sigma-slave/logs/KUBELET.INFO
    ```

  - containerd

    ```
    /home/t4/containerd/logs
    ```

删除节点时一直显示执行中

问题现象：

查看删除节点的运维单发现某个节点的状态一直是执行中，无法完成。

解决方案：

1. 在 Captain 控制台的左侧导航栏单击 **集群管理**，然后单击目标元集群。
2. 选择 **工作负载 > 节点（Nodes）**。
3. 选择目标节点右侧的 **操作 > 修改标签**。
4. 单击 **批量增加/覆写**，然后添加如下标签：

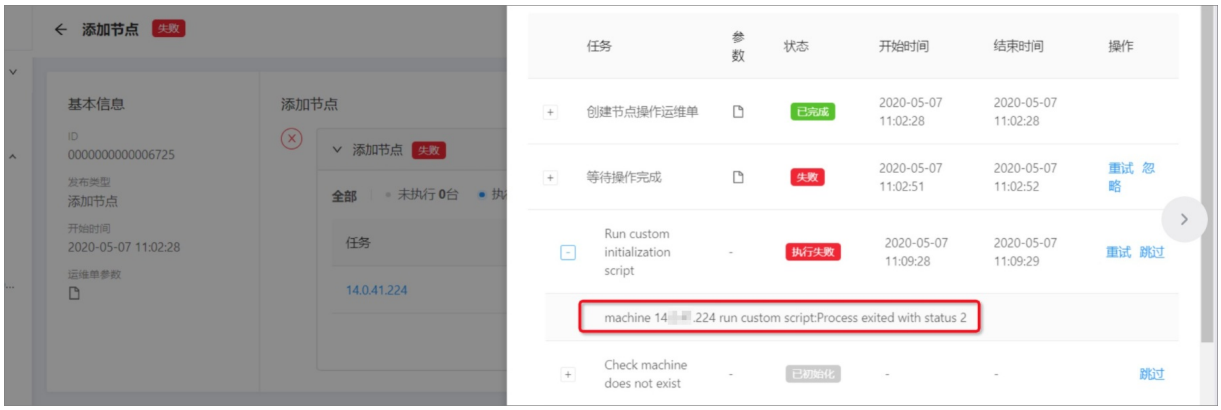
   ```
   machine.sigma.alipay.com/no-daemonset: ""
   ```

5. 重新执行删除操作。

添加节点执行失败

问题现象：

添加节点时显示如下状态：



问题原因：

用户自定义的脚本有问题导致。

解决方式：

使用正确的脚本。

Captain 控制台和云游 Local 控制台上容器信息不一致

问题现象：

云游 Local 控制台上可以看到容器，但是 Captain 控制台上看不到对应容器。

问题原因：

去物理机上使用 `docker ps -a` 命令发现容器并不存在，说明 Captain 控制台没有问题，是云游 Local 上残留的异常数据。

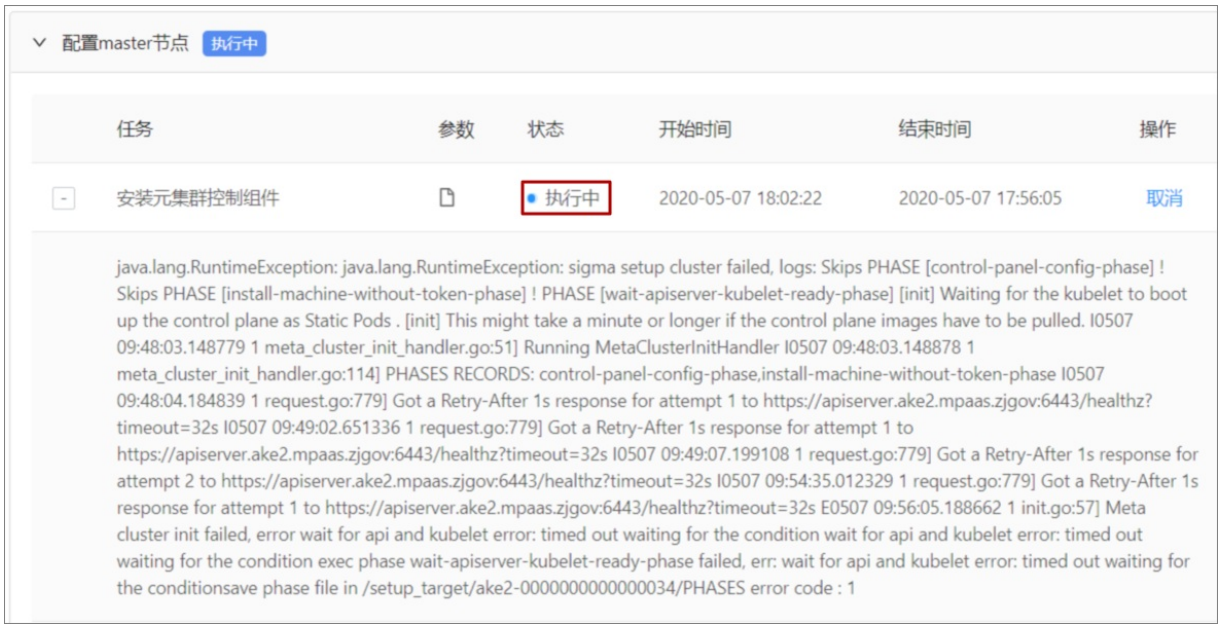
解决方案：

在云游 Local 上删除这部分异常数据。

元集群组件无法启动

问题现象：

安装元集群控制组件一直显示执行中，无法完成，如下图所示：



问题原因：

节点的资源配置太小（4C8G），导致 etcd 启动失败，从而导致 apiserver 无法启动。

解决方案：

增加节点资源配置。

添加节点时检查 /home/t4 失败

问题原因：

因为节点没有单独挂载 `/home/t4` 盘导致。如果默认不挂载，则需要修改集群版本配置来关闭校验。

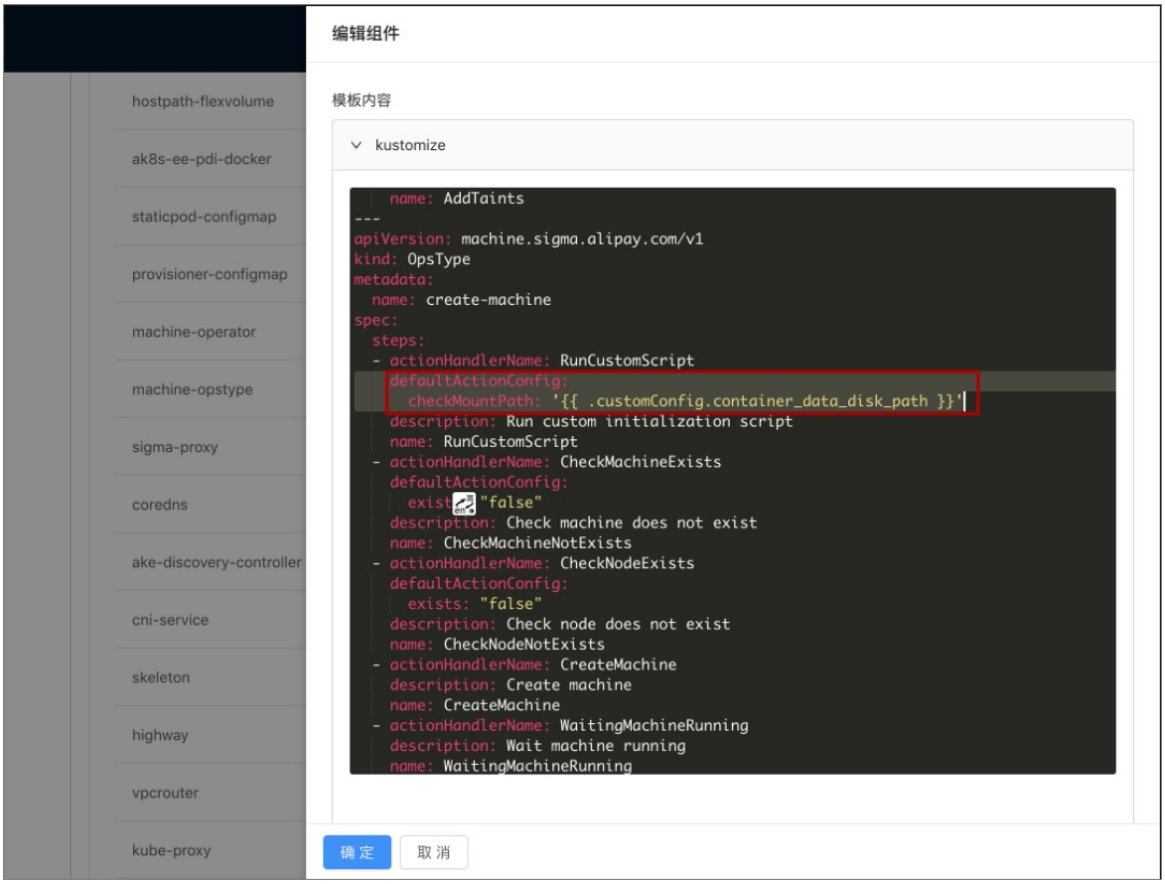
解决方案：

1. 在 Captain 控制台的左侧导航栏单击 **版本**。
2. 单击目标集群当前使用的版本。
3. 单击 **根据此版本创建新版本**，并配置以下参数：

参数	说明
新版本名称	填写一个和原来版本不一样的版本名称。
新版本（包名 master）	填写一个和 **新版本名称** 不一样的名称，例如加一个后缀。
新版本（包名 machine）	填写一个和 **新版本**（包名 master）一样的名称。

4. 单击 **确定**。
5. 编辑 machine-opstype 组件。
   i. 单击名为 machine-opstype 组件名称右侧的 **编辑**。

ii. 找到 create-machine 的 OpsType，将下图选中的两行删除（注意不能留有空行）。



iii. 单击 **确定**。

6. 将原来的集群升级到这个新创建的版本即可。

如何修改集群 DNS 解析服务器

问题原因：

创建集群的时候填错了 DNS 解析服务器的参数，或者因其他特殊情况需要在不铲除集群的情况下修改该参数。

解决方案：

1. 登录云 Captain 控制台。
2. 单击 **集群管理**，然后单击目标元集群。
3. 选择 **配置管理 > 配置项（Configmaps）**。
4. 选择命名空间为 **kube-system**。
5. 单击名为 **cluster-advance-config** 的配置项。
6. 单击 **编辑**，然后将 `dnsResolveIp` 的值修改为新的域名服务器地址。
7. 编辑 **sigma-slave-service** 配置项，将 `--cluster-dns` 的值修改为新的域名服务器地址。
8. 到集群的每一台节点上，修改 **/etc/systemd/system/kubelet.service** 这个文件的内容，将 `--cluster-dns` 的值修改为新的域名服务器地址。
9. 修改完后保存配置，并重启节点上的 kubelet。

命令如下：

```
systemctl restart kubelet
```

后续新建的 Pod，就会使用正确的 DNS 解析服务器 IP 了。

节点无法删除，一直显示删除中或等待中

问题原因：

实际的物理机或 ECS 已经出现问题无法使用（或者已经不存在），导致 machine-operator 尝试去卸载节点上软件时失败。

解决方案：

按如下步骤使用 kubectl 连上集群强制删除节点：

1. 删除 machine。

   命令如下：

   ```
   kubectl delete machine <节点名称>
   ```

   这一步会卡住，执行后直接按组合按键 ctrl+c 进行下一步即可。

2. 编辑 machine，把里面的 finalizer 都删除。

   命令如下：

   ```
   kubectl edit machine <节点名称>
   ```

   删除如下两行，然后输入 `:wq` 保存退出。

   ```
   finalizers:
   - finalizer.k8s.alipay.com/machine
   ```

4. 删除 node 对象。

   命令如下：

   ```
   kubectl delete node <节点名称>
   ```

5. 清理配置数据。

   命令如下：

   ```
   kubectl delete cm -n sigma-operator-machine-conditions <name>-conditions
   kubectl delete cm -n sigma-operator-machine-conditions <name>-readiness-gates
   ```

6. 刷新下 Captain 控制台的页面，节点已经从列表中消失了。

节点相关运维操作报 ssh test connectivity failed

问题原因：

在进行一些节点相关运维操作时，部署在集群内的 machine-operator 会通过 ssh 去连接要操作的节点，这种情况是 ssh 不通导致，通常是网络不通或 ssh key 问题。

解决方案：

1. 先排查网络问题。

   到 AKE3 集群的 master 节点上尝试下 ssh 目标节点，看是否可以成功连接。

- 如果可以正常连接，则进行步骤 2。
- 如果不能正常连接，则检查网络是否出现故障。

2. 检查 ssh key 是否有问题。

   machine-operator 使用的 ssh key 存储在 AKE 集群的 kube-system 命名空间下的一个名为 sigma-machine-external-client 的 Secret 里面（这个 Secret 是根据交付同学在创建集群的时候填写的 ssh key 生成的）。

   i. 在 Captain 控制台的目标集群的管理页面，选择 **配置管理 > 保密字典（Secrets）**。

   ii. 选择命名空间为 **kube-system**。

   iii. 单击名为 **sigma-machine-external-client** 的保密字典。

   其中 SshKey 后面双引号里的内容就是 machine-operator 使用的 ssh key。

   iv. 将 SshKey 内容拷到一个文件里。

   拷贝时注意将所有的 \n 替换为换行符。

   v. 使用这个文件作为 ssh key 尝试登录目标节点。

   如果连接不上，说明目标节点没有对这个 key 做免密，需要在目标节点上对这个 key 做免密操作，并且检查现有的免密操作脚本等是否有问题。这种通常是因为不标准的运维操作导致免密配置错误，需及时修改正确。

访问 Captain 控制台页面出现 502 错误

问题原因：

Captain 控制台启动失败导致此问题。

解决方案：

1. 重启 Captain 容器。

   如果没有解决，则进行步骤 2。

2. 查看 Captain 的错误日志。

   查看 `logs/akeconsole/common-error.log` 和 `logs/sofa-runtime/common-error.log`，如果发现有形如 `Failed Join to cluster, address=xxx` 的错误，说明是 cafeflow 有异常数据导致启动失败（通常是 Captain 所在机器异常重启或者网络有问题导致），需要清理 Captain 数据库中 cafeflow 的脏数据。

3. 连接 Captain 数据库，执行下面的 SQL 语句清除异常数据：

   ```
   TRUNCATE TABLE cafe_flow_cluster_node;
   ```

   > **重要**
   >
   > 清理前请先备份数据库，防止数据丢失。

4. 依次重启所有 Captain 容器。

   重启一台后，等 5 分钟再重启另一台，不要同时重启。

集群证书更换之后，Captain 连不上集群

问题原因：

Captain 会在其数据库中存储用于连接集群的证书，如果集群自身的证书发生变化，Captain 数据库里的证书也会失效，需要更新。

解决方案：

1. 获取取集群目前可用的 admin kubeconfig。

   由于集群连接的相关证书都存储在 kubeconfig 中，因此首先要获取集群目前可用的 admin kubeconfig：

   - 对于元集群来说，该文件为 master 节点上的 `/etc/kubernetes/kubeconfig/admin.kubeconfig` 文件。
   - 对于 KOK 集群来说，该文件为其所在元集群上名为 KOK 集群名的命名空间下的 `admin.kubeconfig` 这个 Secret 的内容。

2. 获取 admin kubeconfig 后，将其中的内容拆解为如下几个变量：

   - kube_config：值为 `admin.kubeconfig` 本身的全部内容。
   - ca_cert：值为 `admin.kubeconfig` 内 `certificate-authority-data` 的值 base64 解码之后的内容。
   - admin_cert：值为 `admin.kubeconfig` 内 `client-certificate-data` 的值 base64 解码之后的内容。
   - admin_key：值为 `admin.kubeconfig` 内 `client-key-data` 的值 base64 解码之后的内容。

3. 获取要更新证书的集群的 cluster_id。

   您可以通过查询 Captain 数据库的 cluster 表找到其 id。

4. 执行如下 SQL 语句更新 Captain 数据库里的证书。

```
update cluster_cert set cert = '<ca_cert>' where cluster_id = '<cluster_id>' and type = '
root';
update cluster_cert set cert = '<admin_cert>' where cluster_id = '<cluster_id>' and type
= 'admin';
update cluster_cert set cakey = '<admin_key>' where cluster_id = '<cluster_id>' and type
= 'admin';
update cluster_cert set kube_config = '<kube_config>' where cluster_id = '<cluster_id>' a
nd type = 'admin';
```

> **重要**
>
> 请将变量替换为前面步骤 2 得到的实际的值。

5. 重启 Captain。

Grafana 如何对外暴露服务

Grafana 对外暴露服务时有以下两种方式：

- NodePort 方式

  i. 创建 Service。

```
apiVersion: v1
kind: Service
metadata:
  name: grafana-node-service
  labels:
    name: grafana-node-service
spec:
  type: NodePort      #这里代表是 NodePort 的类型。
  ports:
  - port: 3000        #这里的端口和 clusterIP 对应（即 IP:8080），供内部访问。
    targetPort: 3000  #端口一定要和 container 暴露出来的端口对应。
    protocol: TCP
    nodePort: 31111   # 所有的节点都会开放此端口，端口号大于 30000。此端口供外部调用。
  selector:
    app.kubernetes.io/instance: grafana #这里一定要选择容器的标签。
    app.kubernetes.io/name: grafana
```

ii. 部署 Service。

```
kubectl apply -f service.yaml -n kube-plugin
```

iii. 访问服务。

访问集群任意一个节点地址。访问格式为 `节点 IP:节点端口` 。节点端口可自定义，但是需要符合规范。不要和其他的 Service 的节点端口冲突。

- PodIP方式

如果用户的容器网络和用户内部网络打通，可以直接使用 PodIP 来进行访问。访问方式为直接使用 `PodIP:3000` 访问。

5.7.2. Containerd 运维常见问题

支持多个私有镜像中心

1. 修改配置跳过 HTTPS 拉取镜像命令。

在 Containerd 的配置文件 `/etc/containerd/config.toml` 添加如下内容：

```
[plugins."io.containerd.grpc.v1.cri".registry.mirrors."your_registry.com"]endpoint = ["http://your_registry.com"]
```

2. 重启 Containerd。

```
systemctl daemon-reload
systemctl restart containerd
```

3. 跳过 HTTPS 拉取镜像。

```
ctr images pull --plain-http=true acs-reg.alipay.com/acloud/rama-daemon:v0.1.0
```

5.7.3. 安装集群网络诊断工具

在本地镜像中心导入 `acs-reg.alipay.com/acloud/netshoot:latest` 镜像，命令如下：

```
kubectl apply -f netshoot.yaml
```

`netshoot.yaml` 内容如下：

```
# 安装工具
apiVersion: v1
kind: Pod
metadata:
  name: netshoot
  namespace: kube-system
spec:
  containers:
  - name: netshoot
    image: acs-reg.alipay.com/acloud/netshoot:latest
    resources:
      limits:
        cpu: "0.5"
        memory: "200Mi"
      requests:
    command: ["sleep"]
    args: ["infinity"]
```

5.8. 服务异常应急预案

5.8.1. 容器异常应急预案

问题描述

容器无法访问，主库磁盘容量满导致宕机或者访问异常，且从库与主库主从复制异常导致主从切换失败，无法提供服务。

实施步骤

1. 登录任意实例容器主库，查看磁盘容量。

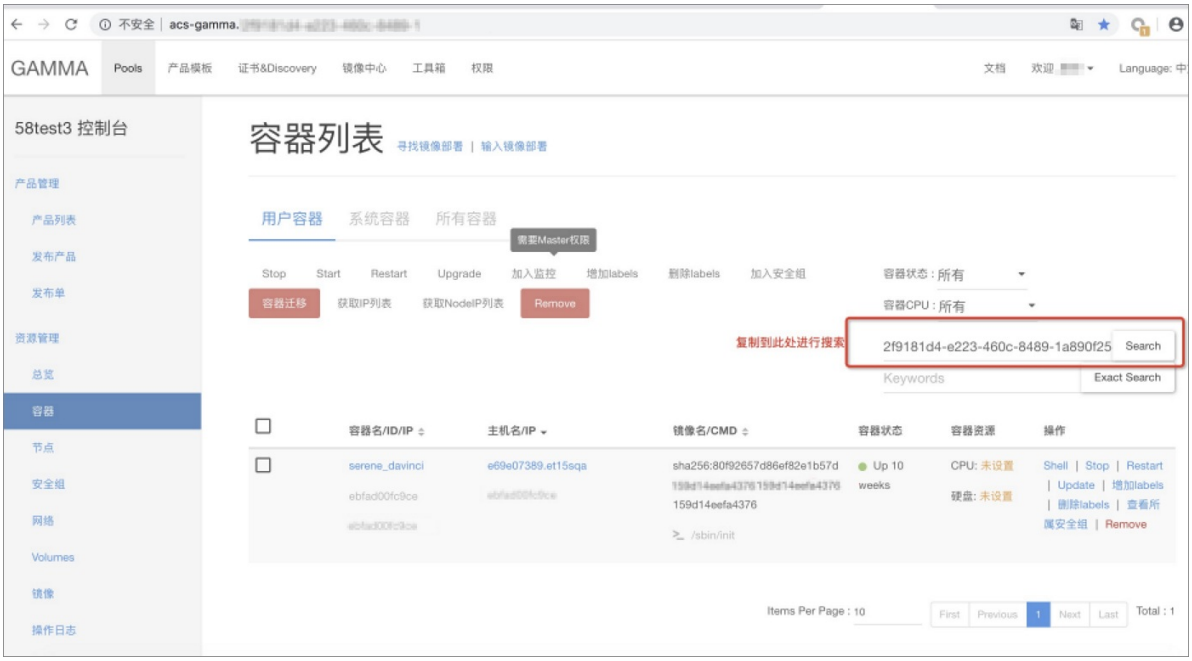
```
shell>su mysql
shell>df -h
Filesystem      Size  Used Avail Use% Mounted on
/                40G  3.9G   37G  10% /
/dev/v01d        14T  4.6T  8.6T  35% /u01/my3306  #数据盘
/dev/v02d       3.5T  3.5T  673G  100% /u02/my3306  #日志盘
shm              64M   12K   64M   1% /dev/shm
```

   查看发现日志盘磁盘满，使用量为 100%。

2. 登录物理机，获取日志使用量最大的实例。

```
shell>cd /u02
shell>du -sh *|grep G |sort -rn
624G    2f9181d4-e223-460c-8489-1a890f25ca75
605G    8a8671ac-0524-43e7-80ab-96104a3121f1
102G    144174cd-3382-4010-82e2-e9ebad50ee9d
99G     513e5c37-76ce-4a4b-8560-e03514732b27
42G     7c22d153-6f21-4e1b-a7bc-e1ed2560193f
42G     6361d900-7893-4393-9ad7-b11113d44bc0
```

3. 根据文件名找到对应的实例容器。



4. 登录容器，查看日志情况。

```
shell>su mysql
shell>/u02/my3306/log
shell>ls |grep mysql-bin |wc -l  #统计 binlog 日志数据量。
2471
shell>ls |grep relay |wc -l      #统计 relaylog 日志数据量，若大于 1，需关注主从复制情况。
1172
```

5. 检查主从复制情况，确认可删除的 binlog 日志。

```
Slave_IO_Running: Yes     #IO 线程为 Yes 表示正常；为 No 表示异常。
Slave_SQL_Running: Yes     #SQL 线程为 Yes 表示正常；为 No 表示异常。
Slave_SQL_Running_State: Slave has read all relay log; waiting for the sla
ve I/O thread to update it         #状态信息。
shell>tbsql slave |grep Seconds_Behind_Master
Seconds_Behind_Master: 0       #主从延迟为 0，表示正常。
shell>su mysql
shell>dbasql
shell>tbsql slave |grep Running
*************************** 1. row ***************************
Slave_IO_State: Waiting for master to send event
Master_Host: 11.165.245.24
Master_User: slave
```

```
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.036625
Read_Master_Log_Pos: 26254
Relay_Log_File: relaylog.073251
Relay_Log_Pos: 279
Relay_Master_Log_File: mysql-bin.036625  re
Slave_IO_Running: Yes    #IO 线程为 Yes 表示正常；为 No 表示异常。
Slave_SQL_Running: Yes     #SQL 线程为 Yes 表示正常；为 No 表示异常。
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 26254
Relay_Log_Space: 484
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0     #主从延迟为 0 表示正常；为 null 表示主从复制中断。
Master_SSL_Verify_Server_Cert: No
Last_IO_Errno: 0
Last_IO_Error:
Last_SQL_Errno: 0
Last_SQL_Error:
Replicate_Ignore_Server_Ids:
Master_Server_Id: 195426802
Master_UUID: c9475f86-13d9-11e9-b149-7a420ba5f518
Master_Info_File: mysql.slave_master_info
SQL_Delay: 0
SQL_Remaining_Delay: NULL
Slave_SQL_Running_State: Slave has read all relay log; waiting for the sla
ve I/O thread to update it
Master_Retry_Count: 86400
Master_Bind:
Last_IO_Error_Timestamp:
Last_SQL_Error_Timestamp:
Master_SSL_Crl:
Master_SSL_Crlpath:
Retrieved_Gtid_Set:
Executed_Gtid_Set:
Auto_Position: 0
```

5.8.2. AKE3 集群使用 Captain 执行脚本解决日志爆满问题

适用场景

本文适用于 AKE3 1.0.0 ~ 1.4.1 的集群，AKE3 2.0.0 及以上集群会在安装节点时自动生成日志清理脚本。

问题描述

目前，AntStack Plus 的 `/home/t4/sigma-slave /home/t4/sigma-master` 日志未设置自动清理，可能导致日志文件占满存储空间。

解决方案

您可以按照如下步骤通过 Captain 执行脚本清理日志：

1. 安装 run-script opstypes。

   > **说明**
   >
   > 已安装则忽略该步骤。您可以通过以下命令确认是否已安装过 run-script opstypes。
   >
   > ```
   > kubectl  get opstypes.machine.sigma.alipay.com run-script
   > ```

   i. 将以下代码保存为 `run-script.yaml` 。

   ```
   apiVersion: machine.sigma.alipay.com/v1
   kind: OpsType
   metadata:
     name: run-script
   spec:
     steps:
     - actionHandlerName: RunCustomScript
       description: Run custom initialization script
       name: RunCustomScript
   ```

   ii. 执行以下命令：

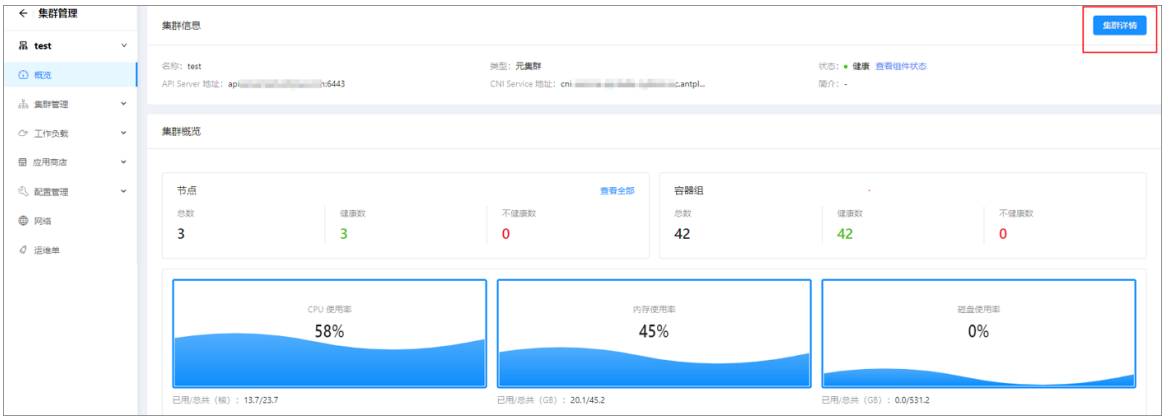
   ```
   kubectl apply -f run-script.yaml
   ```

2. 安装脚本。

   i. 登录 Captain 控制台。

   ii. 单击 **集群管理**，然后单击目标集群。

iii. 单击 **概览**，然后单击 **集群详情**。



iv. 在 **集群详情** 页的 **初始化脚本** 文本框，输入以下脚本。


```
#!/bin/bash
masterCronFilePath=/etc/cron.d/sigma-master-clean-log
masterLogcleanScript=/etc/kubernetes/sigma-master-clean-expire-logs.sh
masterDir="/home/t4/sigma-master/logs"

slaveCronFilePath=/etc/cron.d/sigma-slave-clean-log
slaveLogcleanScript=/etc/kubernetes/sigma-slave-clean-expire-logs.sh
slaveDir="/home/t4/sigma-slave/logs"

#master-------------
if [ -d "$masterDir" ]; then
cat >$masterLogcleanScript <<'EOF'
#!/bin/bash

if [[ $# -lt 3 ]]; then
        echo "need target_dir log_level remain_count"
        exit -1
fi

target_dir="/home/t4/sigma-master/logs"
master_component=$1
log_level=$2
remain_count=$3
:
if (($remain_count} <= 0)); then
        echo "remain_count need greater than 0!"
        exit -2
fi
```

```
# judge if target_dir exists
if ! [[ -d ${target_dir}/${master_component} ]]; then
        echo "dir ${target_dir}/${master_component} not exists!"
        exit -3
fi

# filter files under target_dir
to_sort_files=()
todo_count=0

exclude_files=()
exclude_count=0

target_files=$(ls ${target_dir}/${master_component}|awk '{print i$0}' i=${target_dir}/$
{master_component}'/')

condition="^${target_dir}/${master_component}/kube-${master_component}\.[^\.]+\.[^\.]+\
.log\.${log_level}\.[^\.]+\.[^\.]+$"
for f in ${target_files[@]}
do
        if ! [[ -f ${f} ]]; then
                continue
        elif [[ -L ${f} ]]; then
                link_file=`readlink ${f}`
                if [[ "${link_file}" =~ ^[^/]+ ]]; then
                        link_file=${target_dir}/${master_component}'/'${link_file}
                fi
                if [[ "${link_file}" =~ ${condition} ]]; then
                        exclude_files[exclude_count]=${link_file}
                        ((exclude_count+=1))
                fi
                continue
        elif ! [[ "${f}" =~ ${condition} ]]; then
                continue
        fi

        to_sort_files[todo_count]=${f}
        ((todo_count+=1))
done

# sort
sorted_files=$(for f in ${to_sort_files[@]}; do echo $f; done | sort -t "." -k 6 -r)

to_del_files=()
todel_count=0
filtered_count=0

for f in ${sorted_files[@]}
do
        if [[ "${exclude_files[@]}" = *"${f}"* ]]; then
                continue
        elif [[ ${filtered_count} -lt ${remain_count} ]]; then
                ((filtered_count+=1))
```

```
                continue
        fi

        to_del_files[todel_count]=${f}
        ((todel_count+=1))
done

# print and clean
for f in ${to_del_files[@]}
do
        ionice -c2 -n7 rm -f "${f}"
done
EOF
chmod +x $masterLogcleanScript

cat >$masterCronFilePath <<EOF
# every 5 minutes
*/5 * * * * root /usr/bin/sh $masterLogcleanScript apiserver ERROR 2 >/dev/null 2>&1 0>
&1
*/5 * * * * root /usr/bin/sh $masterLogcleanScript apiserver INFO 5 >/dev/null 2>&1 0>&
1
*/5 * * * * root /usr/bin/sh $masterLogcleanScript apiserver WARNING 2 >/dev/null 2>&1
0>&1
*/5 * * * * root /usr/bin/sh $masterLogcleanScript apiserver FATAL 2 >/dev/null 2>&1 0>
&1

# every 5 minutes
*/5 * * * * root /usr/bin/sh $masterLogcleanScript controller-manager ERROR 2 >/dev/nul
l 2>&1 0>&1
*/5 * * * * root /usr/bin/sh $masterLogcleanScript controller-manager INFO 5 >/dev/null
2>&1 0>&1
*/5 * * * * root /usr/bin/sh $masterLogcleanScript controller-manager WARNING 2 >/dev/n
ull 2>&1 0>&1
*/5 * * * * root /usr/bin/sh $masterLogcleanScript controller-manager FATAL 2 >/dev/nul
l 2>&1 0>&1

# every 5 minutes
*/5 * * * * root /usr/bin/sh $masterLogcleanScript scheduler ERROR 2 >/dev/null 2>&1 0>
&1
*/5 * * * * root /usr/bin/sh $masterLogcleanScript scheduler INFO 5 >/dev/null 2>&1 0>&
1
*/5 * * * * root /usr/bin/sh $masterLogcleanScript scheduler WARNING 2 >/dev/null 2>&1
0>&1
*/5 * * * * root /usr/bin/sh $masterLogcleanScript scheduler FATAL 2 >/dev/null 2>&1 0>
&1
EOF

fi

#slave----------------------
if [ -d "$slaveDir" ]; then

cat >$slaveLogcleanScript <<'EOF'
#!/bin/bash
```

```
if [[ $# -lt 3 ]]; then
        echo "need target_dir log_level remain_count"
        exit -1
fi

target_dir=$1
log_level=$2
remain_count=$3
:
if ((${remain_count} <= 0)); then
        echo "remain_count need greater than 0!"
        exit -2
fi

# judge if target_dir exists
if ! [[ -d ${target_dir} ]]; then
        echo "dir $target_dir not exists!"
        exit -3
fi

# filter files under target_dir
to_sort_files=()
todo_count=0

exclude_files=()
exclude_count=0

target_files=$(ls ${target_dir}|awk '{print i$0}' i=${target_dir}'/')

condition="^${target_dir}/kubelet\.[^\.]+\.[^\.]+\.log\.${log_level}\.[^\.]+\.[^\.]+$"
for f in ${target_files[@]}
do
        if ! [[ -f ${f} ]]; then
                continue
        elif [[ -L ${f} ]]; then
                link_file=`readlink ${f}`
                if [[ "${link_file}" =~ ^[^/]+ ]]; then
                        link_file=${target_dir}'/'${link_file}
                fi
                if [[ "${link_file}" =~ ${condition} ]]; then
                        exclude_files[exclude_count]=${link_file}
                        ((exclude_count+=1))
                fi
                continue
        elif ! [[ "${f}" =~ ${condition} ]]; then
                continue
        fi

        to_sort_files[todo_count]=${f}
        ((todo_count+=1))
done

# sort
sorted_files=$(for f in ${to_sort_files[@]}; do echo $f; done | sort -t " " -k 6 -r)
```

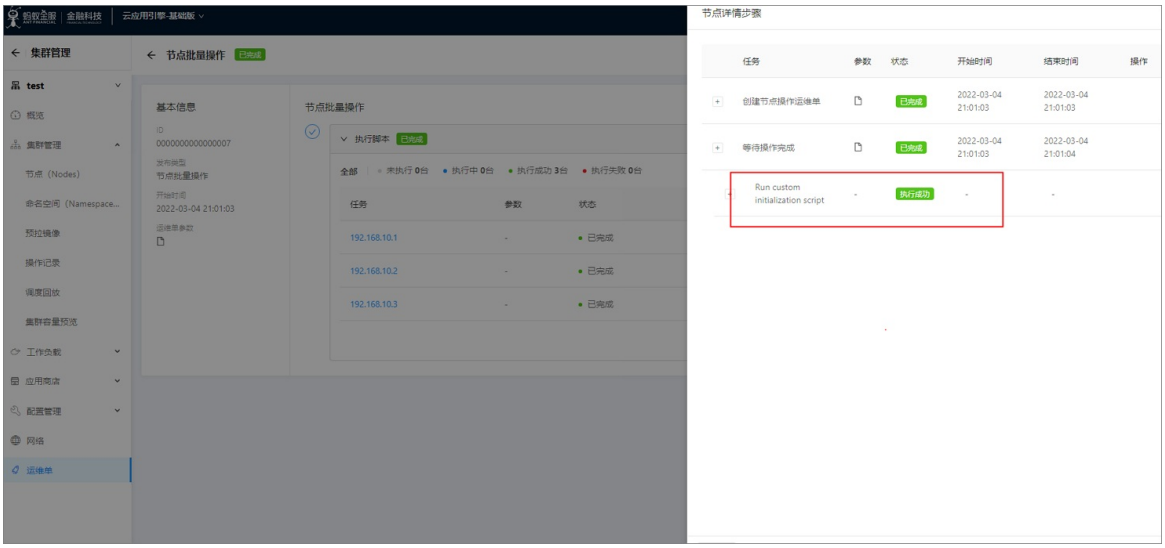
```
sorted_files=$(for f in ${to_sort_files[@]}; do echo $f; done | sort -t "." -k 6 -r)

to_del_files=()
todel_count=0
filtered_count=0

for f in ${sorted_files[@]}
do
        if [[ "${exclude_files[@]}" = *"${f}"* ]]; then
                continue
        elif [[ ${filtered_count} -lt ${remain_count} ]]; then
                ((filtered_count+=1))
                continue
        fi

        to_del_files[todel_count]=${f}
        ((todel_count+=1))
done

# print and clean
for f in ${to_del_files[@]}
do
        ionice -c2 -n7 rm -f "${f}"
done
EOF
chmod +x $slaveLogcleanScript

cat >$slaveCronFilePath <<EOF
# every 5 minutes
*/5 * * * * root /usr/bin/sh $slaveLogcleanScript $slaveDir ERROR 2 >/dev/null 2>&1 0>&
1
*/5 * * * * root /usr/bin/sh $slaveLogcleanScript $slaveDir INFO 5 >/dev/null 2>&1 0>&1
*/5 * * * * root /usr/bin/sh $slaveLogcleanScript $slaveDir WARNING 2 >/dev/null 2>&1 0
>&1
*/5 * * * * root /usr/bin/sh $slaveLogcleanScript $slaveDir FATAL 2 >/dev/null 2>&1 0>&
1
EOF
fi
```

重要

- 脚本必须 Linux 格式（lf），不能为 Windows 格式（crlf）。
- 如果您已经填写自己的脚本，建议您先对脚本进行备份。

3. 执行自定义脚本。
   i. 在 Captain 控制台左侧导航栏，选择 **集群管理 > 节点（Nodes）**。
   ii. 在节点列表选中目标节点后，选择 **批量操作 > 执行自定义脚本**。

iii. 单击 **提交**，然后等待运维单完成。



4. 查看结果。
   - 登录 Master 和 slave 节点，确认 Master 节点有 master 和 slave 文件；slave 节点只有 slave 文件。

```
[root@ake-worker003 ~]# ll /etc/kubernetes/
total 32
drwxr-xr-x 2 root root 4096 Dec 30 15:37 kubeconfig
-rwxr-xr-x 1 root root 6474 Jan 24 13:40 logclean.sh
drwxr-xr-x 2 root root 4096 Jan 18 16:05 manifests
drwxr-xr-x 3 root root 4096 Dec 30 15:37 pki
-rw-r--r-- 1 root root  385 Jan 18 16:25 scheduler-policy-config.json
-rwxr-xr-x 1 root root 2055 Mar  4 21:01 sigma-master-clean-expire-logs.sh
-rwxr-xr-x 1 root root 1870 Mar  4 21:01 sigma-slave-clean-expire-logs.sh
[root@ake-worker003 ~]# ll /etc/cron.d
total 20
-rw-r--r--. 1 root root  128 Aug 25  2021 0hourly
-rw-r--r--. 1 root root  108 Oct 14 16:08 raid-check
-rw-r--r--  1 root root 1511 Mar  4 21:01 sigma-master-clean-log
-rw-r--r--  1 root root  551 Mar  4 21:01 sigma-slave-clean-log
-rw-------. 1 root root  235 Nov 30 10:37 sysstat
[root@ake-worker003 ~]#
```

   - 5 分钟后看节点磁盘的占用是否已减少。

5.8.3. 容器解析 DNS 异常

问题描述

容器内置 DNS 无法正常解析域名时会导致内部组件通信异常。

环境检查

在 Captain 控制台，通过 WebShell 登录容器，查看容器 DNS 配置。命令如下：

```
cat /etc/resolve.conf
```

如果输出结果第一行 nameserver 返回值为 127.0.0.11，则代表容器使用的内置 DNS。出现域名解析失败时，可认为内置 DNS 失败。

实施步骤

登录云应用引擎基础版，通过 WebShell 登录容器宿主机，重启 docker 进程。命令如下：

```
service docker restart
```

结果验证

1. 通过 WebShell 登录容器宿主机，检查 docker 进程状态。命令如下：

   ```
   service docker status
   ```

2. 从相同网络的其他容器尝试解析原问题容器的域名，确保可以正常解析。命令如下：

   ```
   dig <问题容器的域名>
   ```

3. 登录原问题容器，可以正常解析其他上下游组件域名。命令如下：

   ```
   dig <上下游组件的域名地址>
   ```

补充说明

对于不需要使用内置 DNS 的容器，应该去掉内置的 DNS。通过是否需要通过容器名来访问同一个容器网络的其他容器来判断是否需要内置 DNS。如果是，则需要内置 DNS；反之，则不需要内置 DNS。

根据如下操作步骤去掉内置 DNS 方案：

1. 停止容器。
2. 修改或增加 label。

   ```
   com.alipay.acs.container.server_type=DOCKER_VM
   ```

3. 启动容器。

   容器的 `/etc/resolv.conf` 会被同步为宿主机的 `/etc/resolv.conf` 。

5.8.4. 扩容节点失败

问题描述

扩容节点失败，查看详情步骤中，创建节点运维单成功，但在等待操作完成时，其中某一步执行失败。重试后仍然失败，页面上看不到明确的报错信息。

实施步骤

1. 找到元集群和业务集群的的 kubeconfig 文件。
2. 查询对应 machineops 和 machine 的状态。

   命令如下：

   ```
   kubectl --kubeconfig <kubeconfig> get machineops <machineops_name>
   kubectl --kubeconfig <kubeconfig> get machine <machine_name>
   ```

3. 查看节点扩容失败的详细报错信息。

   命令如下：

   ```
   kubectl --kubeconfig <admin.kubeconfig> -n <kok_cluster_namespace> logs <sigma-machine-operator-pod-name>
   ```

4. 修复节点的配置问题后，重试扩容节点任务。

结果验证

重试成功，集群详情中也能看到添加的节点。

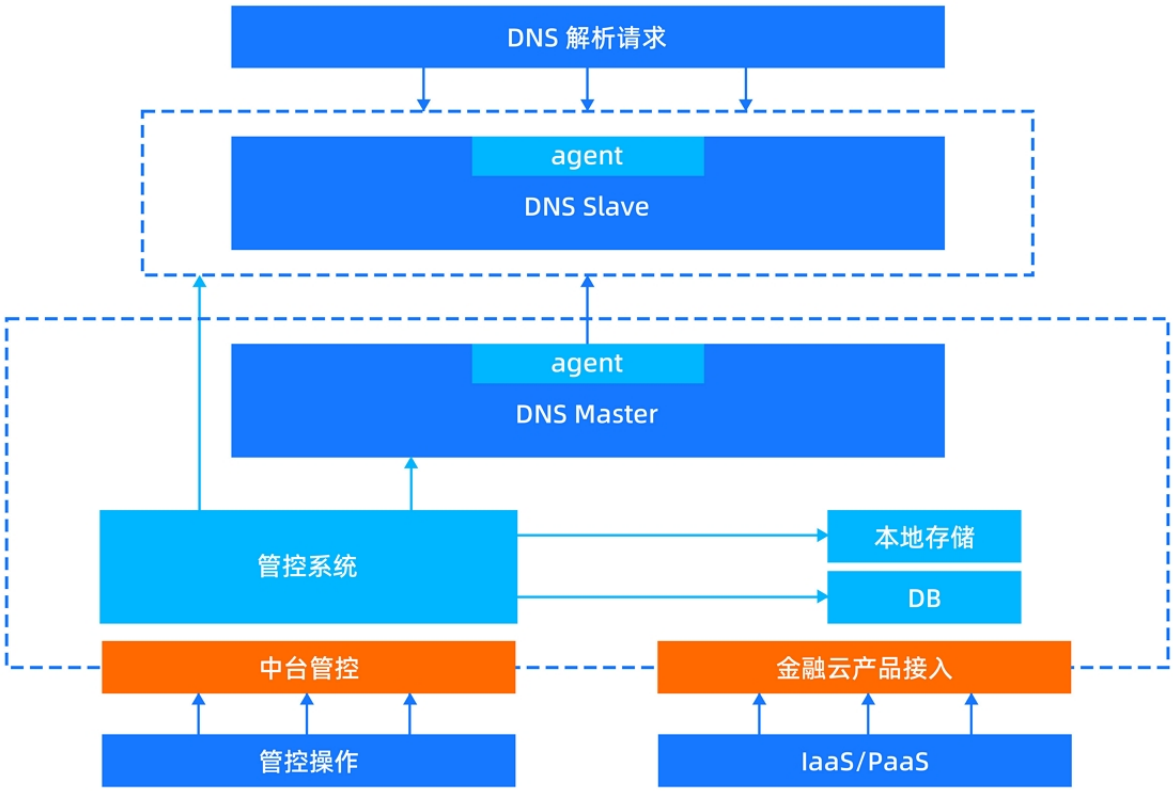
6.AntStack DNS（ADNS）

6.1. 产品架构

本文介绍 AntStack DNS 的产品架构。

逻辑架构

管控系统管理 DNS 集群（包括 DNS Master 和 DNS Slave）对外提供 DNS 变配和管理服务。DNS 的变配会写入 DNS Master，并异步落库。同时 DNS Slave 会根据区域传输协议从 DNS Master 同步数据，对外提供解析服务。根据不同底座，DNS Master 和 Slave 通过 VRRP 或四层负载均衡提供高可用接入能力。逻辑架构如下：



DNS Master

多个 Master 中，只有一个 active，其余的 Master 为 standby。在物理机模式，VIP 会绑定在 active 的 Master 上。active Master 定时做以下动作：

- 将和 DB 中的 zone 相关的数据同步至本地。
- 将 active Master 的 DNS 缓存同步至 DB。
- 将 DB 中的数据与 standby Master 的 DNS 配置进行同步。
- 将 active Master 的 DNS 缓存与 Slave 的 DNS 通过区域传输协议同步数据。

DNS Slave

Slave 提供 RPC 接口，由 DNS Master 调用接口同步 DNS 数据。

Master 和 Slave 主要功能是同步数据以及生成 zone 配置文件，DNS解析服务由 bind9 提供。

- 通过 DNS API 添加 RR 记录时，会通过 bind9 的服务端口动态更新 RR，再异步写入 DB。
- 通过 DNS API 添加 zone 时，则是先同步写 DB，再调用各个 DNS 的 RPC 接口添加 zone。

在数据同步的过程中，当 DB 中的 zone 与本地的 zone 不一致时（例如 DB 中有 zone A，本地没有 zone A），以 DB 中的 zone 作为数据基准。当 DB 中的某个 zone 的 RR 记录与本地的不一致，则以本地的 RR 记录作为数据基准。

内部调用链路

Master 通过 HTTP 接口提供 DNS 数据的增删改查、主备切换以及节点加入与移除等功能。

- DNS 的增删改查：Master 会调用 Slave 的 RPC 接口动态更新 DNS 数据。
- 主备切换：更新 DB 中的数据，同步 standby Master 的数据。
- 节点加入与移除：更新 DB 中的数据，调用所有 Master 的 sync-ha-state 接口。

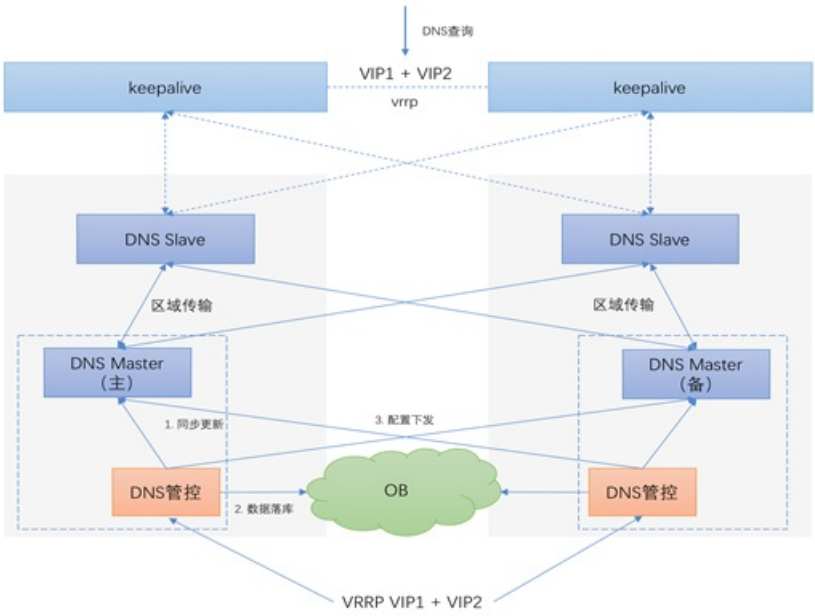
产品依赖

ADNS 与其他产品没有依赖，也没有部署依赖。

部署架构

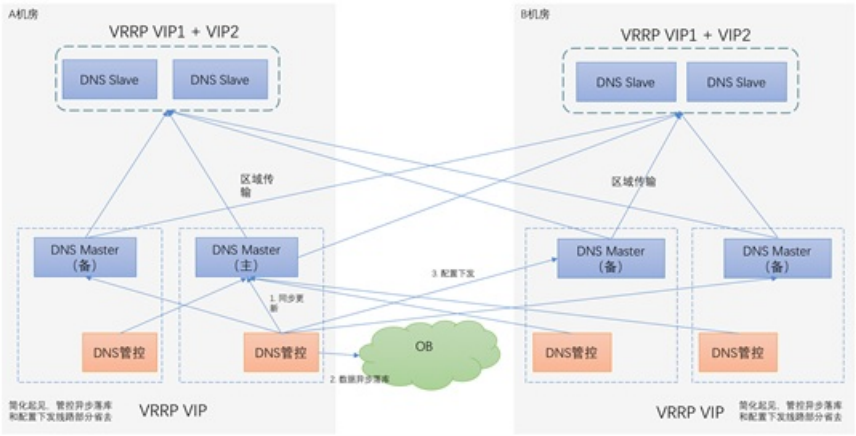
- 单机房部署

  单机房部署架构如下：



- 双机房部署

双机房部署架构如下：



6.2. 预警和监控

6.2.1. 监控项说明

本文介绍 ADNS 的监控项。

ADNS 需要关注的监控项如下表所示：

应用	分类	指标	告警阈值	采集间隔（分钟）	告警间隔（分钟）	说明	告警处理方式
ADNS 解析服务	DNS 解析时延	Cost	>200ms	1	3	客户端做 DNS 解析需要的时间。	• 如果是偶发告警，可以忽略。 • 如果告警持续时间较长，您需要确认是网络延时影响，还是 DNS 解析本身有问题。
ADNS 解析服务	DNS 解析成功数	Success	<10	1	1	客户端做 DNS 解析的成功数。一般在 30~40 之间。	重启容器或联系运维人员。

6.3. 系统日志

本文介绍 AntStack DNS 相关系统日志路径。

日志说明

所在容器	日志名称	日志路径	日志说明
master	dnsmng 管控进程日志	/home/admin/dnsmng/log/adns_biz.log	记录了管控进程的业务日志，包括 zone、DNS 记录增删改、数据同步等相关日志。
master	dnsmng API 调用日志	/home/admin/dnsmng/log/adns_api.log	记录了 API 调用记录，包括调用 URI、method、返回码以及调用方 IP。
master/slave	keepalived 日志	supervisorctl tail keepalived stderr	keepalived 日志。
master/slave	dnsagent 日志	/home/admin/dnsagent/log/adns_biz.log	dnsagent 日志，日志中的“failed to update timeout dispatch job: DB unavailable”可以忽略，agent 不连接 DB。

日志示例

dnsmng 管控进程日志：

```
time="2021-08-02T13:04:56+08:00" level=error msg="failed to update master list on DNS nodes
: failed to update master list on DNS nodes:
192.**.**.59:53: dial tcp 192.**.**.59:15353: connect: connection refused; 192.**.**.23:54:
dial tcp 192.**.**.23:15454:
connect: connection refused; 192.**.**.23:53: dial tcp 192.**.**.23:15353: connect: connect
ion refused" service=DNSClusterHAService
time="2021-08-03T22:00:46+08:00" level=warning msg="DNS service(port 54) on node 192.**.**.
23 is found to be unavailable for 4m0s: read udp 192.**.**.59:58323->192.**.**.23:53: read:
connection refused" service=DNSHealthChecker
time="2021-08-03T22:00:46+08:00" level=warning msg="DNS service(port 53) on node 192.**.**.
23 is found to be unavailable for 5m0s: read udp 192.**.**.59:58323->192.**.**.23:53: read:
connection refused" service=DNSHealthChecker
time="2021-08-03T22:00:46+08:00" level=warning msg="DNS agent/named process on node 192.**.
**.23 is found to be down for 4m0s: dial tcp 192.**.**.23:15353: connect: connection refuse
d" service=DNSHealthChecker
time="2021-08-03T22:00:46+08:00" level=warning msg="DNS agent/named process on node 192.**.
**.23 is found to be down for 5m0s: dial tcp 192.**.**.23:15353: connect: connection refuse
d" service=DNSHealthChecker
time="2021-08-03T22:00:46+08:00" level=error msg="failed to check zone conf drift between D
B and DNS: failed to fix zone conf drift with node 192.**.**.23:53: failed to connect dnsag
ent on server 192.**.**.23:15353: dial tcp 192.**.**.23:15353: connect: connection refused"
service=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=error msg="failed to fetch zone 10.in-addr.arpa.'s R
Rs from slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: connection refused" service=
DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=info msg="skip slave 192.**.**.23, continue to next
slave server" service=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=error msg="failed to fetch zone 172.in-addr.arpa.'s
```

```
RRs from slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: connection refused" service
=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=info msg="skip slave 192.**.**.23, continue to next
slave server" service=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=error msg="failed to fetch zone alibaba-inc.com.'s R
Rs from slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: connection refused" service=
DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=info msg="skip slave 192.**.**.23, continue to next
slave server" service=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=error msg="failed to fetch zone aliclone.net.'s RRs
from slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: connection refused" service=DNS
DataChecker
time="2021-08-03T22:01:02+08:00" level=info msg="skip slave 192.**.**.23, continue to next
slave server" service=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=error msg="failed to fetch zone aliyun-inc.com.'s RR
s from slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: connection refused" service=D
NSDataChecker
time="2021-08-03T22:01:02+08:00" level=info msg="skip slave 192.**.**.23, continue to next
slave server" service=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=error msg="failed to fetch zone aliyun.com.'s RRs fr
om slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: connection refused" service=DNSDa
taChecker
time="2021-08-03T22:01:02+08:00" level=info msg="skip slave 192.**.**.23, continue to next
slave server" service=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=error msg="failed to fetch zone aliyuncs.com.'s RRs
from slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: connection refused" service=DNS
DataChecker
time="2021-08-03T22:01:02+08:00" level=info msg="skip slave 192.**.**.23, continue to next
slave server" service=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=error msg="failed to fetch zone cloud.aliyun-inc.com
.'s RRs from slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: connection refused" ser
vice=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=info msg="skip slave 192.**.**.23, continue to next
slave server" service=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=error msg="failed to fetch zone ngis.net.'s RRs from
slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: connection refused" service=DNSDataC
hecker
time="2021-08-03T22:01:02+08:00" level=info msg="skip slave 192.**.**.23, continue to next
slave server" service=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=error msg="failed to fetch zone tbcdn.cn.'s RRs from
slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: connection refused" service=DNSDataC
hecker
time="2021-08-03T22:01:02+08:00" level=info msg="skip slave 192.**.**.23, continue to next
slave server" service=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=error msg="failed to fetch zone tbsite.net.'s RRs fr
om slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: connection refused" service=DNSDa
taChecker
time="2021-08-03T22:01:02+08:00" level=info msg="skip slave 192.**.**.23, continue to next
slave server" service=DNSDataChecker
time="2021-08-03T22:01:02+08:00" level=error msg="failed to fetch zone yum.tbsite.net.'s RR
s from slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: connection refused" service=D
NSDataChecker
time="2021-08-03T22:01:02+08:00" level=info msg="skip slave 192.**.**.23, continue to next
slave server" service=DNSDataChecker
```

```
time="2021-08-03T22:01:02+08:00" level=error msg="failed to fetch zone oceanbase.com.'s RRs
from slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: connection refused" service=DNS
DataChecker
time="2021-08-03T22:01:02+08:00" level=info msg="skip slave 192.**.**.23, continue to next
slave server" service=DNSDataChecker
time="2021-08-03T22:01:46+08:00" level=warning msg="DNS service(port 53) on node 192.**.**.
23 is found to be unavailable for 7m0s: read udp 192.**.**.59:371
89->192.**.**.23:53: read: connection refused" service=DNSHealthChecker
time="2021-08-03T22:01:46+08:00" level=error msg="failed to check zone conf drift between D
B and DNS: failed to fix zone conf drift with node 192.**.**.23:53
: failed to connect dnsagent on server 192.**.**.23:15353: dial tcp 192.**.**.23:15353: con
nect: connection refused" service=DNSDataChecker
time="2021-08-03T22:01:46+08:00" level=warning msg="DNS agent/named process on node 192.**.
**.23 is found to be down for 6m0s: dial tcp 192.**.**.23:15353:
connect: connection refused" service=DNSHealthChecker
time="2021-08-03T22:01:46+08:00" level=warning msg="DNS agent/named process on node 192.**.
**.23 is found to be down for 7m0s: dial tcp 192.**.**.23:15353:
connect: connection refused" service=DNSHealthChecker
time="2021-08-03T22:02:02+08:00" level=error msg="failed to fetch zone 10.in-addr.arpa.'s R
Rs from slave 192.**.**.23: dial tcp 192.**.**.23:53: connect: co
nnection refused" service=DNSDataChecker
```

以上日志表示：192.**.**.23 节点 slave 连接失败，数据同步失败，您需要检查该节点上的 dnsslave 状态。

日志清理

ADNS 默认配置了 logrotate，不需要手动清理日志。ADNS 的日志会以周为单位，生成日志文件，最多保存 4 个日志文件。当创建第 5 个日志文件时，第一个日志文件会被清理。

```
[root@h07b13156 ~]# cat /etc/logrotate.conf
# see "man logrotate" for details
# rotate log files weekly
weekly

# keep 4 weeks worth of backlogs
rotate 4

# create new (empty) log files after rotating old ones
create

# use date as a suffix of the rotated file
dateext

# uncomment this if you want your log files compressed
#compress

# RPM packages drop log rotation information into this directory
include /etc/logrotate.d

# no packages own wtmp and btmp -- we'll rotate them here
/var/log/wtmp {
    monthly
    create 0664 root utmp
        minsize 1M
    rotate 1
}

/var/log/btmp {
    missingok
    monthly
    create 0600 root utmp
    rotate 1
}

# system-specific logs may be also be configured here.
[root@h07b13156 ~]# ls /etc/logrotate.d/
adns_api.logrotate  metrics.logrotate   syslog              tsar.logrotate
adns_biz.logrotate  named               tsar                yum
[root@h07b13156 ~]# ls /etc/logrotate.d/
adns_api.logrotate  adns_biz.logrotate  metrics.logrotate  named  syslog  tsar  tsar.logrotate  yum
```

6.4. 服务巡检

高可用和数据一致性巡检

AntStack DNS 的服务巡检主要是保障 DNS 的高可用和数据一致性。巡检内容如下：

- DNS master 和 slave 端口巡检

  包括 slave（解析服务）53 端口的探测、agent 端口的探测。如果端口探测失败，会输出 error 日志到 `/home/admin/dns{mng,agent}/logs` 。

- DNS 数据一致性检测

  包括 DNS zone、DNS 记录的巡检。ADNS 会巡检 DNS master 和 DB、DNS master 和 DNS slave 的数据。如果发现不同，会以 DNS master（主）的数据为准，进行自动补充。

深度巡检

1. 通过 `nslookup {domain_name} {adns_slave_vip}` 命令确认 ADNS 是否正常提供服务。
2. 确认 ADNS slave 是否正常工作。
   i. 在 DNS slave节点中执行 `supervisorctl status` 命令，确认 bind、keepalived、dnsagent 服务是否正常。
   ii. 在 slave 容器中执行 `cat /home/admin/dnsagent/log/adns_biz.log` 命令查看日志是否有错误信息。

3. 通过 `curl {master_vip}:80/ha/hastate` 确认是否能正常返回。
4. 确认 ADNS master 是否正常工作。
   i. 在 DNS master节点中执行 `supervisorctl status` 命令，确认 bind、keepalived、dnsmng 服务是否正常。
   ii. 在 master 容器中执行 `cat /home/admin/dnsmng/log/adns_biz.log` 命令查看日志是否有报错信息。
5. DNS CR 检查。
   i. 执行 `kubectl get zone` 命令， 查看各个 zone 的状态是否为 `Created` 。
   ii. 执行 `kubectl get dnsendpoint -A` 命令， 查看各个记录的 `PHASE` 是否为 `Succeed` 。

6.5. 常见运维场景

6.5.1. 常见问题排查思路

Master容器内关键进程 bind、dnsagent、dnsmng、keepalived。dnsmng 就是管控进程，提供 API 接口。keepalived 在物理机环境下使用，提供高可用。

当 DNS Master 不可用时或故障时，从以下几个方面排查：

- 进入 master 容器，通过 `supervisorctl status` 命令查看各个进程是否正常运行。
- 物理机环境下，如果 keepalived 发生故障，按以下步骤排查：
  i. 在 master 容器中通过 env 查看环境变量，主要关注：
     - `KEEPALIVED_VIRTUAL_IPS` 是否为期望 vip。
     - `USE_VRRP` 是否为 yes。
     - `LISTEN_ON_VIRTUAL_IPS` 是否为 yes，并查看 `/etc/keepalived/keepalived.conf` 配置网卡名是否正确、 `virtual_ipaddress` 是否为期望的 vip。检查环境变量中是否有空格。
  ii. 使用 `supervisorctl tail keepalived stderr` 命令查看 keepalived 日志，找到 keepalived 启动失败的原因。如果日志提示已有 keepalived 启动中，则需要执行如下命令：

```
mv /etc/supervisord/conf.d/keepalived.conf /etc/supervisord/
supervisorctl reload
mv /etc/supervisord/keepalived.conf /etc/supervisord/conf.d/
supervisorctl reload
```

  iii. dnsmng 启动故障。在容器中查看 `cat /home/admin/dnsmng/log/adns_biz.log` 查看日志。
  iv. 增删 zone 或 dnsrr 记录失败，通过 `curl {master_vip}:80/ha/hastate` 看是否能正常返回。如果是404，先确定一下使用的 IP 是否正确。如果是其他错误，进入容器查看 `/home/admin/dnsmng/log/adns_biz.log` ，判断是否为数据库无法连接。
  v. dnsagent启动故障，在容器中查看 `/home/admin/dnsagent/log/adns_biz.log` ，其中“failed to update timeout dispatch job: DB unavailable”错误可以忽略，查看是否有其他错误信息。
- Slave 容器内 bind、dnsagent、keepalived 等关键进程的排查思路与 master 类似。

6.5.2. 设置 allow-recursion 网段

ADNS 默认允许所有用户发起递归查询请求，您可以通过设置 `allow-recursion` 参数，允许部分用户发起递归查询请求。

配置步骤

1. 登录 ADNS 容器。

   ```
   docker exec -it <dns container name> bash
   ```

   `dns container name` 包含 master DNS 以及 slave DNS。

2. 执行 `vim /var/named/chroot/etc/named.conf.options` 命令修改 `allow-recursion` 参数。

   例如修改配置为 `allow-recursion { 11.**.0.0/16; 100.**.**.26; }`，则网段内所有 IP 允许发起递归查询。如下图所示：

   ```
   options {
       directory "/var/named";
       dump-file "/var/named/data/cache_dump.db";
       statistics-file "/var/named/data/named_stats.txt";
       memstatistics-file "/var/named/data/named_mem_stats.txt";
       allow-query { localhost; 0.0.0.0/0; };
       allow-recursion { 11.[redacted].0.0/16;100.[redacted].26; };
       empty-zones-enable no;
       max-journal-size 100m;
       request-ixfr yes;
       serial-query-rate 50;
       transfers-in 30;
       transfers-per-ns 30;
       allow-notify { any; };
       allow-transfer { any; };
       try-tcp-refresh no;
       rrset-order { order cyclic; };
       notify no;
       auth-nxdomain no;
       multi-master yes;
       forwarders { 10.101.0.1; };
       forward only;
   };
   ~
   ```

3. 重启服务。

   ```
   supervisorctl restart bind
   ```

6.5.3. DNS 容器死机

问题描述

DNS 容器死机，无法提供 DNS 解析服务。

操作步骤

1. 登录容器对应的 ops 物理机。
2. 通过以下命令行重启容器。

```
docker restart <容器名称>
```

6.5.4. DNS 进程异常退出

问题描述

DNS 容器进程异常关闭。

解决方案

1. 登录容器对应的 ops 物理机。
2. 进入容器。

   命令如下：

   ```
   docker exec -it
   ```

3. 重启相关进程。

   命令如下：

   ```
   supervisorctl restart <目标进程>
   ```

   相关进程如下：

   - DNSmng 进程：dnsmng
   - DNSagent：dnsagent
   - DNS 解析服务：bind
   - keepalived：keepalived

6.6. 服务异常应急预案

6.6.1. VIP 未绑定

问题描述

域名解析失败，发现是因为 VIP 不通。

环境检查

1. 先确定部署环境，通常只有物理机环境需要通过 keepalived 绑定 VIP。
2. 在 ops 节点执行 `ip -a` 命令，确定是 Master 的 VIP 未绑定还是 Slave 的 VIP 未绑定。

实施步骤

1. 在未绑定 VIP 的容器中执行 `supervisorctl status` 命令，查看以下服务是否为 running 状态：

   bind、dnsagent、dnsmng、keepalived。Slave 没有 dnsmng 服务。如果某个服务状态异常，请使用 `supervisorctl restart <service name>` 命令重启该服务，例如 `supervisorctl restart bind` 。
2. 在未绑定 VIP 的容器中查看 `/etc/keepalived/keepalived.conf` 文件，确认以下信息。
   - 确认网卡名是否正确。

- 确认所有 Master 或者所有 Slave 的网卡 IP 是否在同一网段
- 确认 `virtual_ipaddress` 、 `virtual_router_id` 是否一致。

若以上信息有问题，请修改配置，然后执行 `supervisorctl restart keepalived` 命令。

结果验证

1. 在 ops 节点执行 `ip -a` 命令，确定 Master 的 VIP 绑定成功。
2. ping 测 VIP 可以通。

6.6.2. 解析域名失败

如果遇到域名解析失败的问题，请确认以下问题：

ADNS/CoreDNS 问题

- 通过 `nslookup {domain_name} {adns_slave_vip}` 命令确认是否为 ADNS 的解析问题。
  - 是 ADNS 解析问题
    - dnsslave 53 端口不通

      确认 adns slave 是否正常工作：

      a. 在 ADNS slave 节点中执行 `supervisorctl status` 命令，查看 bind 和 dnsagent 服务是否正常。

      b. 若 bind 和 dnsagent 服务状态为 `stopped` ，则执行 `supervisorctl start bind` 和 `supervisorctl start dnsagent` 命令重启服务。
    - 域名解析显示 nodomain
      - 通过 ADNS API 查询 DNS 记录，确认记录是否添加。若未添加，则手动添加记录。
      - 若域名为产品规划域名，则找到相关集群分别确认 ADNS Operator 、ADNS CR 状态。若不为产品规划域名，则通过 ADNS API 查询域名是否注册。若未注册，则手动添加记录。
  - 非 ADNS 解析问题

    排查 CoreDNS 转发配置是否正确：

    a. 执行 `kubectl -n kube-system get cm coredns -oyaml` 命令查看配置的转发规则是 `/etc/resolv.conf` 还是 ADNS slave SLB IP。

    b. 若为转发规则 `/etc/resolv.conf` ，则登录至 CoreDNS 节点，查看节点的 `/etc/resolv.conf` 中的 `nameserver` 是否为 ADNS slave SLB IP。

    c. 若 `nameserver` 不为 ADNS slave SLB IP，则修改为 ADNS slave SLB IP。

ADNS Operator、CR 状态确认

- 使用 `kubectl get zone` 命令查看 zone 状态是否为 `Created` 。若不为 `Created` ，则执行 `kubectl get zone {zoneName} -oyaml` 命令查看 error 详细信息。

  例如，错误是 ADNS master SLB IP 接口调用失败，则进行 ADNS master 状态确认。
- 使用 `kubectl -n {产品ns} get dnsrecord` 和 `kubectl -n {产品ns} get dnsendpoint` 命令查看产品域名注册资源状态是否为 `Succeed` 。若不为 `Succeed` ，则执行 `kubeclt -n {adns ns} logs {hesitamanager}` 命令查看 Operator 日志（只有一个 Operator master 在工作，只看 Operator master 的日志）。查看报错是否为 ADNS master SLB IP 接口调用失败，若是则进行 ADNS master 状态确认。

ADNS master 状态确认

- 执行 `curl {adns master vip}:80/ha/hastate` 获取 active master。如果提示 unavailable，则表明数据库连接失败，请检查数据库是否正常。
- 进入 dnsmaster 容器，执行 `supervisorctl status` 查看 dnsmng、bind、dnsagent 是否运行。若否，则执行 `supervisorctl start dnsmng` 、 `supervisorctl start bind` 、 `supervisorctl start dnsagent` 命令重启服务。
- 进入 dnsmaster active master 容器，查看日志 `/home/admin/dnsmng/log/adns_biz.log` ，检查报错信息是否有连接其他 dnsmaster、dnsslave 53、54、15353、15454 端口失败信息。如果有，则恢复对应容器即可。
- 如果是主备机房解析结果不一致，添加完 DNS 记录后，等待 1~2 分钟，再次尝试解析。

6.6.3. 云游域名注册失败

版本确认

- 确认 ADNS 版本为 1.15.0 及以上版本，ADNS 1.15.0 之后的版本才支持云游域名注册。
- 云游版本为 1.9.0 及以上版本。

排查步骤

1. 检查 ADNS operator 状态与参数是否正常。
   i. 检查 ADNS operator 的 DNSMasterIP 是否为 ADNS 的地址。
   ii. 使用 `curl {adns ip}:80/ha/hastate` 命令查看 ADNS 是否正常工作
   iii. 使用 `kubectl -n {adns operator namespace} get pods` 命令获取 ADNS operator 的 Pod。
2. 查看域名注册失败详细信息。

   使用 `kubectl get zone` 命令查看 zone 的状态是否为 `Created` 。如果不是，则使用 `kubectl get zone {zone_name} -oyaml` 命令查看具体报错。

   - 如果报错信息显示访问的 ADNS IP 与预期不符合，则检查 ADNS operator 部署时的 DNSMasterIP 参数是否填写正确。若填写错误，则需要修改后重新发布 ADNS operator。
   - 如果 zone 的报错信息显示资源冲突，则检查 ADNS master 和 slave 状态。进入容器，执行 `supervisorctl status` 命令查看进程状态；在 master 容器查看日志 `/home/admin/dnsmng/log/adns_biz.log` ，确认是否有错误日志。
     - 如果 ADNS 状态不正确，请参见 常见问题排查思路。
     - 执行 `curl -XPOST {adns_master_vip}:80/zone/listzone?querysource=dns` 命令查看相关 zone 是否存在，如果不存在，联系售后技术支持处理。
   - 如果提示 webhook 访问失败，则执行如下命令：

     ```
     kubectl delete mutatingwebhookconfigurations hesita-mutating-webhook
     kubectl delete validatingwebhookconfigurations hesita-validating-webhook
     ```

     然后重启 ADNS webhook pod。
   - 如果是 listzoneset 失败，则 curl 一下提示的地址。
     - 确认 IP 是否为 ADNS IP。

       如果 ADNS 没有使用 VIP 且与 Docker registry 同一个节点，需要让 ADNS 使用 VIP。

- 如果是 `zone_set` 表不存在，需按照步骤升级 ADNS 到 1.15.0 及以上版本。

- 如果是 setzoneset 失败：

  a. 使用 `kubectl get zone` 命令查看 zone 是否创建成功，以及状态。

  b. 如果状态不是 `Created` ，使用 `kubectl get zone -oyaml` 命令查看相关错误。检查是否为 ADNS operator 使用的 ADNS IP 错误。

6.6.4. 域名无法解析

排查步骤

1. 通过 `nslookup {domain_name} {adns_slave_vip}` 命令确认 ADNS 是否正常提供服务。
2. 确认 ADNS slave 是否正常工作，在 DNS slave节点中执行 `supervisorctl status` ，确认 bind 和 dnsagent 服务是否正常。
3. 通过 API 调用 zone 接口，查询对应的 zone 是否在 ADNS 中。如果不在，无法解析是正常的。

```
curl -X POST -H 'Content-Type: application/json' \
  http://endpoint:80/zone/listZone
```

4. 通过 API 调用 zone 接口，查询对应的域名记录是否在 ADNS 中，如果不在，无法解析是正常的。

```
curl -H 'Content-Type: application/json' \
    -d '{
    "name": "aliyuncs.com"  // name 的值需替换为对应的 zone 名称。
}' \
    http://endpoint:80/zone/queryrrs
```

6.6.5. 域名解析慢

排查步骤

1. 通过 `nslookup {domain_name} {adns_slave_vip}` 命令确认 ADNS 服务是否正常。
2. 查看 ADNS 容器的 CPU、内存使用量，确认是否负载过高。如果达到 limit 上限，需要调整容器 CPU、内存的 limit。

   在 OSP 主机上执行 `docker stats --all --no-stream | grep dns` 命令，筛选找到 dnsmaster 和 dnsslave 的容器。

```
[root@ops001 /root]
#docker stats --all --no-stream | grep dns
115aa7abe572        newadnsmaster       1.44%               311.9MiB / 4GiB     7.61%               0B / 0B             0B / 161MB          0
affd6fa7ffe2        newadnsslave        14.43%              444MiB / 4GiB       10.84%              0B / 0B             0B / 185MB          0
98ab040202cc        dnsslave1           0.00%               0B / 0B             0.00%               0B / 0B             0B / 0B             0
14b667a26180        dnsmaster1          0.00%               0B / 0B             0.00%               0B / 0B             0B / 0B             0
```

6.6.6. 容灾切换

单机房部署下的单机宕机

同一机房下，主备两台 Master 节点组成一个 cluster。ADNS通常通常部署在 osp1、osp2 两个节点上，需要保证至少一个 ADNS 节点正常工作才不会影响。

当出现单机宕机时，您需要进行主备切换，以恢复 DNS 解析。

主备切换：如果主节点（osp1）宕机，将 ADNS Master 切换为 ops2 的 IP 即可。

```
fun_chgactive()
{
  curActive=`curl -s -H 'Content-Type: application/json' http://${B_VIP}/ha/hastate |awk -F ',' '{ print $2 }' |awk -F
'"' '{print$4}'`

  newActive=$B_M_IP1

  curl -s -H 'Content-Type: application/json' -d '{"newActive":"'$newActive'","curActive":"'$curActive'"}' http://$
{B_VIP}:80/ha/chgactive
}
```

恢复：将 ADNS master 切换至主节点（ osp1 的 IP）。

多机房容灾架构下的单机房宕机

主备机房的 Master 节点组成一个 cluster。当出现单机房宕机时，可切换到备机房以恢复 DNS 解析。

主备切换：

1. 切换备机房的 DB。
2. 将 ADNS Master 切换为备机房 ops1 的 IP，ADNS API 地址切换到备机房的 VIP。

```
fun_chgactive()
{
  curActive=`curl -s -H 'Content-Type: application/json' http://${B_VIP}/ha/hastate |awk -F ',' '{ print $2 }' |awk -F
'"' '{print$4}'`

  newActive=$B_M_IP1

  curl -s -H 'Content-Type: application/json' -d '{"newActive":"'$newActive'","curActive":"'$curActive'"}' http://$
{B_VIP}:80/ha/chgactive
}

fun_chgdnsapi()
{
  curl -H 'Content-Type: application/json' \
      -d '{
      "new": {
         "type": "A",
          "name": "dnsapi1.tbsite.net.",
          "value": "'$B_VIP'",
          "ttl": 60
      },
      "old": {
          "type": "A",
          "name": "dnsapi1.tbsite.net.",
          "value": "'$A_VIP'"
      },
      "zone": "tbsite.net"
  }' \
    http://$B_VIP:80/rr/chgrr
}

fun_chgactive

#echo"\n"
sleep 5

fun_chgdnsapi
```

恢复：

1. 将主机房数据库及 AntDNS 恢复，并完成回切。
2. 将 ADNS Master 切换至主机房 osp1 的 IP，ADNS API 地址切换到主机房的 VIP。

7.身份访问 IAM

7.1. 监控项说明

本文介绍身份访问 IAM 的监控项。

您需要关注的监控项如下表所示：

应用	分类	指标	告警阈值
Iamcoreiamweb	基础监控-CPU 使用率监控	cpu_usage	>90%
	基础监控-内存使用率监控	mem_usage	>90%
	基础监控-磁盘使用率监控	disk_usage	>90%
iamweb	基础监控-端口监控	2022、8341、12200	不通
iamcore	基础监控-端口监控	2022、80、12200	不通

7.2. 系统日志

日志名称	日志路径	日志说明
通用业务日志	/home/admin/logs/aciamcore/app-default.log	可以查询到所有的 facade 和 OpenAPI 的调用记录。
通用错误日志	/home/admin/logs/aciamcore/common-error.log	可以查询到所有的报错日志。
阿里云 API 通信日志	/home/admin/logs/aciamcore/ram-invoke.log	可以查询到调用阿里云接口的一些明细信息。
	/home/admin/logs/tracelog/*.log	-
	/home/admin/logs/tracelog/zdal-db-stat.log	记录数据库操作统计、耗时。

常用框架日志 日志名称	日志路径	日志说明
	/home/admin/logs/tracelog/zdal-db-digest.log	ZDAL 的摘要日志，只会打印耗时超过一定阈值的执行成功的 SQL 以及所有执行失败的 SQL。
	/home/admin/logs/zdal/*.log	-

7.3. 服务巡检

7.3.1. 系统组件监控检查

基础监控检查

登录 RMS 监控系统，查看 IAM 相关系统组件有无异常告警，需要关注的监控项内容如下表所示：

应用	分类	指标	阈值
aciamcore	CPU 使用率监控	cpu_usage	>90%
	内存使用率监控	mem_usage	>90%
	磁盘使用率监控	disk_usage	>90%
	端口监控	2022, 8341, 12200	不通
aciamweb	CPU 使用率监控	cpu_usage	>90%
	内存使用率监控	mem_usage	>90%
	磁盘使用率监控	disk_usage	>90%
	端口监控	2022, 80, 12200	不通

自定义业务检查

登陆到 IAM 组件应用的容器，执行 `curl localhost:9500/checkService | grep false` 命令。返回的结果如果没有 `false` 则说明正常。

7.3.2. 业务功能检查

1. 使用 superAdmin 账号登录 IAM 控制台。

   IAM 控制台的域名一般为：`http://iamweb.<domain_name>` 。

2. 验证控制台功能是否正常。

   进行诸如创建账号、使用新账号登录、删除刚创建的账号等操作。

7.4. 服务异常应急预案

IAM 架构上分为 aciamweb 和 aciamcore，iamweb 宕机不影响业务功能，只影响 IAM 页面访问。

单机宕机

影响：同一机房下，由于是多个 IAM 副本部署（目前是一个中心 2 台），aciamcore 任意一台宕机，对于业务功能没有影响。

恢复：重启 IAM 实例（Pod）即可恢复。

多机房容灾架构依赖

- Ant DNS：IAM 通过 Ant DNS 提供的域名访问数据库，容灾时，需将数据库域名切换至备机房的数据库。依赖 Ant DNS 的容灾能力。
- OB/MySQL：IAM 主备机房共库，依赖 OB/MySQL 的容灾能力。

单机房宕机

影响：无影响，主机房故障时可切换到备机房。

切换：

1. 确保备机房 DB 及 Ant DNS 已切换完成。
2. 切换 IAM、OP 的公网域名（只有公网域名才会注册在 Ant DNS 中）至备机房。

恢复：

1. 确保主机房数据库及 Ant DNS 已恢复，并回切完成。
2. 切换 IAM、OP 的公网域名（只有公网域名才会注册在 Ant DNS 中）至主机房 。

8.OpenAPI（OP）

8.1. 系统日志

日志名称	日志路径	日志说明
通用业务日志	/home/admin/logs/acprodapigw/app-default.log	您可以根据 requestId 在该日志中找到一个请求的完整生命周期的日志。
通用错误日志	/home/admin/logs/acprodapigw/common-error.log	可以查询到所有的报错日志。
网关摘要日志	home/admin/logs/acprodapigw/gateway-digest.log	可以查询网关摘要日志。 使用如下方式查看 API 提供方执行方法时，抛出的异常： `cat /home/admin/logs/acprodapigw/gateway-digest.log |grep GW_UNKNOWN_ERROR`

8.2. 服务巡检

基础监控检查

登录 RMS 监控系统，查看 OpenAPI（OP）相关系统组件有无异常告警，需要关注的监控项内容如下表所示：

应用	分类	指标	阈值
acprodapigw	CPU 使用率监控	cpu_usage	>90%
	内存使用率监控	mem_usage	>90%
	磁盘使用率监控	disk_usage	>90%
	端口监控	2022, 80, 12200	不通

自定义业务检查

登录到 IAM 组件应用的容器，执行 `curl localhost:9500/checkService | grep false` 命令，返回的结果如果没有 `false` 则说明正常。

8.3. 常见运维场景

8.3.1. 结果码

OpenAPI 调用常见结果码如下：

结果码	说明
OK	调用成功。
UNKNOW_ERROR	调用失败，下游服务未知原因错误，请排查下游服务。
GW_UNKNOWN_ERROR	调用失败，网关未知原因错误，请联系售后技术支持。
INVALID_SIGNATURE	签名错误，请检查 SDK 版本和 AK 是否正确、SDK 或者签名过程中使用的 secret 是否正确。 一般不会出现此类错误。
INVALID_ACCESS_KEY	AK 错误，请检查 AK 是否正确，是否出现跨云 AK 错配问题。
MISSING_PARAMETER	请求参数缺失，其中： • `req_msg_id` 必须在 32 到 64 个字节之间。 • `access_key` 不能为空。 • `sign_tye` 必须为 HmacSHA1。 • `req_time` 必须满足 ISO 8601 标准。
INVALID_PARAMETER	参数值无效。
API_NOT_EXIST	接口不存在或指定版本的接口不存在。通常是对应的接口在网关上还没有配置，元数据未正确录入。您可以向 API 实现方管理员确认。
RPC_ERROR	核心网关 TR 服务调用失败，请检查 TR 服务是否正常发布，检查配置中心是否连接正确。

结果码	说明
OVER_RATE_LIMIT	调用查出网关限流配额。
TIMEOUT	调用超时，检查下游服务是否正常
INVALID_AUTH_TOKEN	三方授权调用时，auth token 非法，请检查三方授权流程，并联系售后技术支持。
INVALID_AUTH_SCOPE	三方授权调用时，auth token 正确，但是授权范围非法，请检查三方授权流程，并联系售后技术支持。
ACCESS_DENIED	操作员未开通调用权限，租户管理员需要为对应的操作员配置调用接口权限。操作步骤请参见 IAM 使用指南的《账号管理》章节。 如果无法解决，请联系售后技术支持。联系时，请务必提供请求具体信息，方便排查。信息包括：访问 API 名称、ReqMsgId、访问网关地址。
METHOD_NOT_FOUND	网关正确配置 API 信息，但是路由到下游服务时，下游服务可能由于版本问题，没有找到对应的接口实现。请联系具体产品负责人。
WRITE_LIST_DENIED	未开通白名单，请联系售后技术支持开通。
INVALID_PROVIDER_RESPONSE_SIGNATURE	产品端返回的数据签名失败，请检查产品服务端签名的时候所用的 secret 是否正确。网关的返回提示中通常包含有部分脱敏的 secret 信息。
WRONG_PROVIDER_AK	产品端 AK 信息不正确，集群配置的 AK 找不到对应的信息。通常是用错了环境，或者配置了错误的 AK。 目前 AK 是通过 Spec 中的 clusterDeployment 设置，您需要与产品方确认。
INVALID_PRODUCT_INSTANCE_ID	合约校验失败，产品实例 ID 不存在。
PRODUCT_INSTANCE_NOT_ACTIVE	合约校验失败，产品实例不处于激活状态。可能是合约已经过期，请联系售后技术支持。

结果码	说明
BAD_PROVIDER_RESPONSE	下游服务返回错误，请查看返回值中的 message。如有问题，请联系对应产品负责人。 message 中记录的下游具体错误原因，例如： • 签名值非法 • 返回格式错误 • 返回值为空 • Http status code 非 200
NO_ROUTE_INFO	产品端的 productInstanceId 不正确，或产品端未配置，请和产品提供者确认。
API_UNAVAILABLE	找不到下游服务，请和下游服务提供者确认。
DEFAULT_SERVICE_NOT_EXIST	找不到默认路由，一般是非核心组件才会出现。请确认是否传入路由信息，比如 product_instance_id 或 region_name。
PROVIDER_SERVICE_TIMEOUT	下游服务超时，请联系产品提供者。
PROVIDER_SERVICE_CONNECTION_REFUSED	下游服务拒绝链接，请联系产品提供者。

8.3.2. 错误码

OpenAPI 调用常见错误码如下：

错误码	错误码描述	处理建议

错误码	错误码描述	处理建议
INVALID_SIGNATURE	签名验证失败	签名验证流程如下： 1. OP Client SDK 把入参通过 AccessKey Secret 生成一个签名，与 AK 等参数传给 OP。 2. OP 通过传入的 AK 去 IAM 拿对应的 AccessKey Secret，然后再生成个签名。 3. OP 将生成的签名与 Client 传进来的签名进行比对。 出现这个报错的原因是生成的签名与 Client 传进来的签名比对不一致，请求被拒绝了。您需要确保 Client 设置的 AccessKey Secret 和 OP 通过 AK 从 IAM 拿到的 AccessKey Secret 是匹配的。
CHANNEL_API_NOT_EXIST	POP API 管道信息未录入，可能原因： 1. OP 版本低于 0.28.0。 2. 元数据未同步到 accoreinit。	检查 API 是否发布成功。步骤请参见 附录：API 发布查询。
REGION_ACCESS_KEY_NOT_EXIST	Region 路由的 AccessKey 不存在。	OP 相关元数据未录入，您需要重新录入。
API_NOT_EXIST	API 不存在	进入 OP 容器，然后登录数据库获取 API 信息。 例如 API 名称为 `antcloud.iam.session.token.verify`，命令如下： `select status, api_suite_id, provider_id from api_meta where name='antcloud.iam.session.token.verify';`

错误码	错误码描述	处理建议

CLUSTER_ACCESS_KEY_NOT_EXIST	OP 相关元数据未录入。问题原因如下：
	• accoreinit 中没录入。
	• accoreinit 没执行成功，需要重执行。
		进入 OP 容器，登录数据库查询 OP 的数据库中是否存在该 API 的元数据 。操作示例如下：
		集群路由信息如下：
		• host：`kafkaconsole.allsite.alipay.net`
		• productInstanceId：`MiddleWareCluster-SOFAKAFKA`
		• OP 平台产品码：`SOFAKAFKA`
		查询命令为：
		`select id,host from runtime_api_group where runtime_provider_id=(select id from runtime_provider where name = 'MiddleWareCluster-SOFAKAFKA');`
		`select id from runtime_provider where name = 'SOFAKAFKA'`

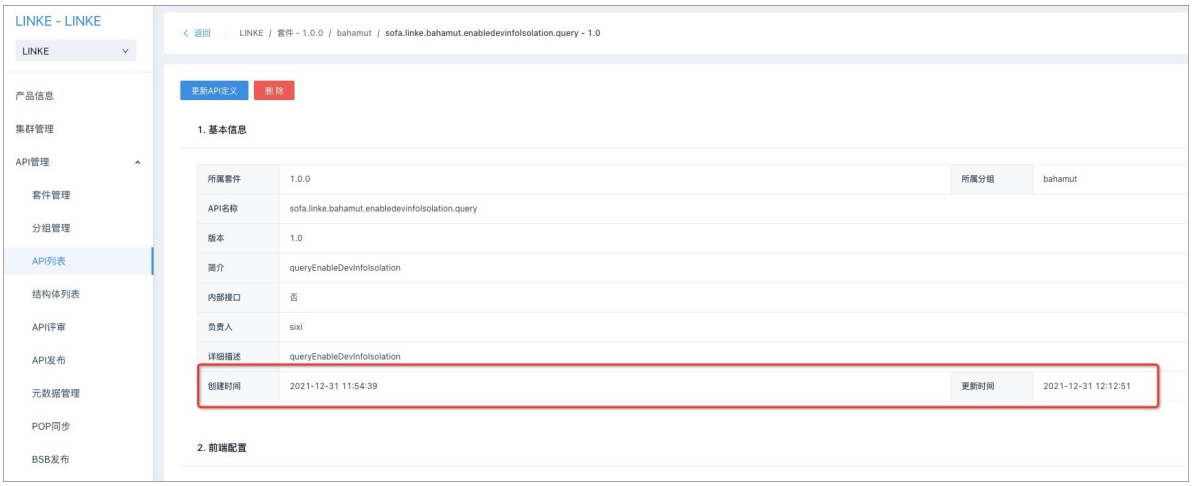
附录：API 发布查询

以下示例以查询 LinkE 的 API 为例，其他产品请到相应的控制台查看。查询方式如下：

控制台查询

1. 登录 LinkE 控制台。
2. 在左侧导航栏选择 **API 管理** > **API 列表**。
3. 查看目标 API 详情。

例如下图示例中，API 更新时间为 2021-12-31。



最后一次发布时间为 2021-12-15。

可以看出 API 在12-31更新后未发布，需要先提交 API 评审，然后提交 API 发布， 最后 bsb 发布。

数据库查询

您也可以通过数据库查询 API 信息，示例如下：

```
SELECT * FROM `channel_api` where `inner_api_name` ='sofa.linke.bahamut.enabledevinfolsolation.query';
```

8.3.3. 常见问题处理流程

OpenAPI 调用出错后，您可以从调用方获取结果码，然后参照 结果码 说明进行处理。若无法解决，请按照以下流程处理。

排查流程

1. 获取 Request ID。
2. 获取报错摘要。

   opgw 容器日志路径为 `/home/admin/logs/acprodapigw` ，命令格式如下：

   ```
   grep "<request id>" gateway-digest.log*
   ```

   示例如下：

```
grep "72fd4d895b37******6eb8dcd860" gateway-digest.log*
```

结果示例如下：

```
gateway-digest.log:2022-05-19 00:15:26,548 [/// - ] INFO  GATEWAY-DIGEST - 72fd4d895b37**
****6eb8dcd860;antcloud.gotone.ddrobot.notify;1.0;GOTONE;LTAI4F******xWvqDNaMy3K;WEWCXUCN
;0116659020;CALL_REQUEST_ERROR;47;39;false;NA;8
```

您需要重点关注如下信息：

- API 接口：antcloud.gotone.ddrobot.notify
- API 产品：GOTONE
- 返回码：CALL_REQUEST_ERROR
- 下游业务处理耗时： 39

此例中，错误码是 CALL_REQUEST_ERROR，在 OP 结果码列表未搜寻到。根据下游业务处理耗时为 39 判断请求已经到下游应用，所以需要找下游应用提供者排查问题。

完整的日志说明如下：

日志字段	字段含义
72fd4d895b374787987646eb8dcd8600	Request ID
antcloud.gotone.ddrobot.notify	API 接口
1.0	API 版本
GOTONE	API 产品码 **说明** 在 OPM 中录入的产品码，不一定与应用实际产品码匹配。例如应用 LinkE，在 OPM 中录入的 API 产品码为：LINKE、LINKEANTCODE、LINKEDEPLOYCORE、LINKEFABRIC、LINKELINKFLOW。
LTAI4F******xWvqDNaMy3K	AccessKey ID
WEWCXUCN	对应的租户
0116659020	用户 ID

CALL_REQUEST_ERROR	错误码 更多信息，请参见 错误码。
47	API 调用耗时
39	下游业务处理耗时
false	调用是否成功
NA	auth_token
8	OP 自身处理耗时

8.3.4. OP 调用后端 TR 服务失败，出现 Cannot find RPC service 报错

问题分析

出现此问题的根本原因是环境未部署服务注册中心。

解决方案

您可以通过以下两种方式解决：

- 在中枢集群中部署 SOFARegistry。
- 按以下步骤订库：
  i. 获取 API 名称。
     本文以 RMS 的 `antcloud.monitor.metrics.query` 为例。
  ii. 进入 OP 容器，通过环境变量获取 DB 的登录信息，然后登录 DB。
  iii. 选择 Schema。
     本文选择 `use prodapicore` 。
  iv. 获取相应 `api_meta` 的 ID。

```
select id from api_meta where name='antcloud.monitor.metrics.query';
return: 123
```

  v. 获取 API TR 调用的 `dispatch_context` 。

```
select dispatch_context from api_version where api_meta_id='123';
return: {"skipAuth":true,"timeout":10000,"trUniqueId":"antcloud.monitor"}
```

  vi. 获取 RMS motinor 容器 IP 以及 RPC 端口号。

本文涉及的 IP 和端口号为：`127.2.3.1:12200`。

vii. 将 API provider 的 `ip:port` 插入 `dispatch_context`。

```
{"skipAuth":true,"testUrl":"127.2.3.1:12200","timeout":10000,"trUniqueId":"antcloud.monitor"}
```

viii. 将带有后端地址的 `dispatch_context` 更新至数据库。

```
update api_version set dispatch_context='{"skipAuth":true,"testUrl":"127.2.3.1:12200","timeout":10000,"trUniqueId":"antcloud.monitor"}' where api_meta_id='123';
```

9.掉电恢复与验证

9.1. DNS 服务恢复与验证

服务恢复

正常情况下，DNS 服务随 ake-master 管控物理机（如 ops1、ops2、ops3）的启动而自动恢复。

服务验证

1. 登录 ops1 物理机。
2. 对一个已经注册的域名执行 **dig** 命令。

   命令如下：

   ```
   dig dnsapi1.tbsite.net
   ```

   如果返回结果可以解析到 IP 地址，则说明 DNS 服务正常。

```
[root@ops1-1 /root]
#dig dnsapi1.tbsite.net

; <<>> DiG 9.9.4-RedHat-9.9.4-29.1.alios7 <<>> dnsapi1.tbsite.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2872
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;dnsapi1.tbsite.net.            IN      A

;; ANSWER SECTION:
dnsapi1.tbsite.net.     600     IN      A       192.168.12.3

;; AUTHORITY SECTION:
tbsite.net.             60      IN      NS      ns2.tbsite.net.
tbsite.net.             60      IN      NS      ns1.tbsite.net.

;; ADDITIONAL SECTION:
ns1.tbsite.net.         60      IN      A       192.168.12.1
```

9.2. NTP 服务恢复及验证

服务恢复

正常情况下，NTP 服务随管控物理机的启动而自动恢复。

服务验证

1. 登录各 ops 物理机，执行以下命令：

```
ntpq -np
```

2. 在返回结果中查看 offset 值。
   - 若 offset 值小于 30（毫秒），则说明 NTP 误差值在允许范围内。

```
[root@ops1-1 /root]
#pgm -f all.txt  "ntpq -np"
[1] 20:54:48 [SUCCESS] 192.168.12.1
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 192.168.12.1    .INIT.          16 u    -   64    0    0.000    0.000   0.000
 192.168.12.2    192.168.12.1     6 u    2   16  377    0.130    0.005   0.007

[2] 20:54:49 [SUCCESS] 192.168.12.2
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*192.168.12.1    127.0.0.1        5 u    4   16  376    0.128   -0.006   0.004
 192.168.12.2    .STEP.          16 u    -   64    0    0.000    0.000   0.000

[3] 20:54:49 [SUCCESS] 192.168.12.32
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 127.127.1.0     .LOCL.          10 l   8d   64    0    0.000    0.000   0.000
*192.168.12.1    127.0.0.1        5 u   65   64  377    0.110    0.134   0.018
+192.168.12.2    192.168.12.1     6 u   53   64  377    0.110    0.125   0.009

[4] 20:54:49 [SUCCESS] 192.168.12.47
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 127.127.1.0     .LOCL.          10 l    -   64    0    0.000    0.000   0.000
```

   - 若 offset 值超过 30 毫秒，则需要在出现问题的 ops 物理机上执行以下命令，与 NTP 服务器进行时间同步。

```
ntpdate -u <ntp_server_ip>
```

   时间同步后，再按照步骤 1 检查 offset 值是否已恢复正常。

9.3. YUM 服务恢复及验证

服务重启

正常情况下，YUM 服务随 2 台管控物理机（如 ops1、ops2）的启动而自动恢复。

服务验证

1. 登录任意一台物理节点。
2. 执行 YUM 列举命令。

```
yum list
```

如果能显示 YUM 列表，则说明 YUM 服务恢复正常。

```
[root@ops1-1 /root]
#yum list|tail -20
Repodata is over 2 weeks old. Install yum-cron? Or run: yum makecache fast
yum-plugin-show-leaves.noarch   1.1.31-34.1.alios7        alios.7u2.base.x86_64
yum-plugin-tmprepo.noarch       1.1.31-34.1.alios7        alios.7u2.base.x86_64
yum-plugin-tsflags.noarch       1.1.31-34.1.alios7        alios.7u2.base.x86_64
yum-plugin-upgrade-helper.noarch
                                1.1.31-34.1.alios7        alios.7u2.base.x86_64
yum-plugin-verify.noarch        1.1.31-34.1.alios7        alios.7u2.base.x86_64
yum-plugin-versionlock.noarch   1.1.31-34.1.alios7        alios.7u2.base.x86_64
yum-rhn-plugin.noarch           2.0.1-5.1.alios7          alios.7u2.base.x86_64
yum-updateonboot.noarch         1.1.31-34.1.alios7        alios.7u2.base.x86_64
zeromq.x86_64                   4.1.4-5.alios7            ops.7.x86_64
zeromq-devel.x86_64             4.1.4-5.alios7            ops.7.x86_64
zlib-devel.x86_64               1.2.7-16.2.alios7         alios.7u2.base.x86_64
zlib-static.x86_64              1.2.7-16.2.alios7         alios.7u2.base.x86_64
zookeeper-client.x86_64         3.4.6-1.alios7            ops.7.x86_64
zookeeper-client-devel.x86_64   3.4.6-1.alios7            ops.7.x86_64
zsh.x86_64                      5.0.2-14.1.alios7         alios.7u2.base.x86_64
zsh-html.x86_64                 5.0.2-14.1.alios7         alios.7u2.base.x86_64
zziplib.x86_64                  0.13.62-5.1.alios7        alios.7u2.base.x86_64
zziplib-devel.x86_64            0.13.62-5.1.alios7        alios.7u2.base.x86_64
zziplib-utils.x86_64            0.13.62-5.1.alios7        alios.7u2.base.x86_64
```

9.4. AKE3 恢复及验证

服务恢复

正常情况下，AKE 云原生平台（AKE3）容器都配置了 `restart=always` 参数，会随着物理机的 docker 进程启动而自启动。

服务验证

1. 登录 AKE-master 容器运行的物理机（如 ops1）。
2. 获取 kube 容器状态。

   命令如下：

   ```
   kubectl get pod -n kube-system
   ```

除 network-controller 容器处于 ImagePullBackOff 状态外，其他容器都应处于 Running 状态。

```
[root@ops1-1 /root]
#kubectl get pod -n kube-system
NAME                                READY     STATUS    RESTARTS   AGE
ak8s-ee-pdi-docker-6p5ph            1/1       Running   0          35d
ak8s-ee-pdi-docker-6tqls            1/1       Running   0          31d
ak8s-ee-pdi-docker-7mwp4            1/1       Running   0          34d
ak8s-ee-pdi-docker-7nclq            1/1       Running   0          34d
ak8s-ee-pdi-docker-7w2th            1/1       Running   0          35d
ak8s-ee-pdi-docker-8g7vv            1/1       Running   0          21d
ak8s-ee-pdi-docker-9j74s            1/1       Running   0          31d
ak8s-ee-pdi-docker-f9h5h            1/1       Running   0          35d
ak8s-ee-pdi-docker-grshb            1/1       Running   0          35d
ak8s-ee-pdi-docker-pzcjt            1/1       Running   0          31d
ak8s-ee-pdi-docker-sh459            1/1       Running   0          21d
ak8s-ee-pdi-docker-thsqr            1/1       Running   0          34d
ak8s-ee-pdi-docker-wffxp            1/1       Running   0          21d
ake-sigma-proxy-97b87dc45-dj7dd     1/1       Running   0          34d
ake-sigma-proxy-97b87dc45-vnhvb     1/1       Running   0          35d
ake-sigma-proxy-97b87dc45-wd8z9     1/1       Running   0          34d
```

注意事项

- 通常情况下不要同时对 3 台 AKE-master 做掉电恢复，若发现业务容器未正常启动，可以手动启动。
- 如果要集群全部断电然后重启，请优先启动 master 节点，正常后，再启动计算节点，则可自动拉起。
- 若发现部分服务器节点处于宕机状态，恢复 DNS 服务后重启物理机即可。

9.5. 云游 Local 服务恢复及验证

服务恢复

正常情况下，云游 Local 服务会随着 2 台 ops 管控物理机的恢复而自动恢复。

服务验证

1. 登录各 ops 物理机。
2. 执行以下命令，确认 MySQL 服务处于运行状态。

```
systemctl status mysql
```

3. 执行以下命令，确认 apyunqing 容器处于运行状态。

```
docker ps |grep apyunqing
```

命令输出形如 `Up 1 days` 则表示 apyunqing 容器处于运行状态。

4. 访问云游 Local 控制台，确认服务已正常启动。

9.6. IAM 恢复与验证

服务重启

1. 登录云游 Local 控制台。

2. 选择 **产品运维 > 产品基线**。
3. 单击 **身份-访问管理/RAM**。
4. 单击 **aciamcore** 卡片的 **容器**，并重启所有容器。

> 说明
>
> aciamcore 依赖 antvip，若反复重启无效，请检查中间件。

5. 以相同方式重启 **aciamweb** 的所有容器。

服务验证

登录 IAM 页面，确认可以正常使用。

9.7. 单机房集群断电恢复

前提条件

为确保容器可以在电力恢复后自动启动，需要节点启用 docker/containerd 和 kubelet 开机自启动。

恢复步骤

1. 机房电力恢复后先登陆容器底座，确认底座集群 master 是否恢复，集群依赖的负载均衡、数据库是否恢复。
2. 检查 ops1、ops2 上的 ADNS 是否启动，如果没有恢复要手动启动。

   启动命令如下：

   ```
   docker start <container id>
   ```

   Captain 和 apyunqing 也是一样的恢复方法，但是可以在业务恢复后再最后启动，节约业务抢救时间。
3. 确认底座集群 master 恢复后，使用如下命令检查集群的状态是否已经恢复。

   ```
   kubectl get node
   ```
4. 集群节点如果都恢复为 Ready 状态后，查看集群上的 Pod 是否都为 Running 状态。

   ```
   kubectl get pod --all-namespaces -owide
   ```

   如果有 Pod 不为 Running 状态，需要人工干预。先查看 Pod 的状态和 describe 下事件描述信息，找到原因后再处理。
5. 登陆 IAM 控制台确认可以通过身份认证，再登陆各个产品的控制台确保可以正常使用，最后再检查业务应用的状态。