蚂蚁集团

研发效能平台 LinkE
运维指南

产品版本：AntStack Plus 1.13.1

文档版本：20230710

蚂蚁集团
ANT GROUP

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

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

商标声明

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

免责声明

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

通用约定

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

目录

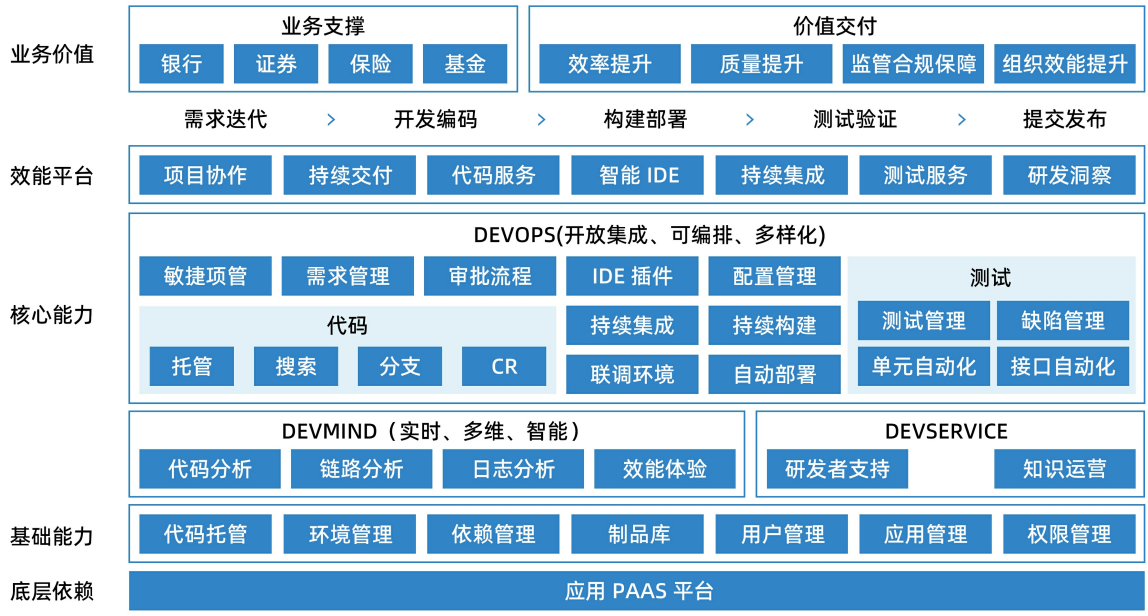
1.运维概述 06
1.1. 功能架构 06
1.2. 系统架构 06
1.3. 运维架构 12
1.4. 代码服务架构 12
1.5. 技术支持 16
2.运维准备 17
2.1. 运维界面 17
2.2. 运维说明 17
2.2.1. 有持久化存储的应用 17
3.高危操作 26
3.1. 运维操作的危险级别定义 26
3.2. 运维操作和文件 26
4.日常运维 28
4.1. 监控 28
4.2. 巡检 30
4.2.1. 组件巡检 30
4.2.2. 守夜人 HAS 巡检配置指南 37
4.3. 日志 44
4.4. 升级/重启 48
4.4.1. 有持久化存储的应用 48
4.4.2. 普通应用 48
4.4.3. 其他应用 50
4.5. 扩容 50
4.5.1. 有持久化存储的应用 50
4.5.2. 其他应用 52

5.应急维护 53
5.1. 流程定义 53
5.2. 异常处理 53
5.3. 主备切换 54
5.4. 数据备份 55
5.5. 网络异常 56
5.6. 数据容灾 56
5.7. 容器宕机 58
5.8. 断电恢复 59
6.附录：基础术语 60
7.LinkE 自带数据库运维 61
7.1. MongoDB 集群 61
7.2. Redis 主从运维架构 68
8.ANTCODE 代码仓库运维 73
8.1. ANTCODE 代码仓库高可用说明书 73
8.2. 代码仓库存储节点宕机的处理方式 78

1.运维概述

1.1. 功能架构

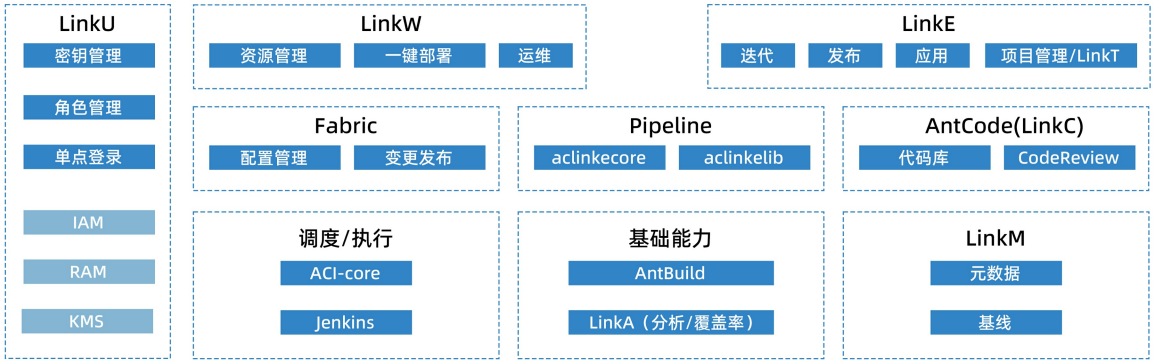
研发效能平台 LinkE 封装了持续交付、智能 IDE、代码托管、配置变更、流程中心、项目协作、测试服务、洞察分析等众多性能优异的功能模块，兼具开放可扩展、灵活可配置等属性，快速适配客户不同业务场景，覆盖软件研发的各个阶段。



- 应用架构设计采用模块化、松耦合设计原则，系统各个功能采用模块化集成，实现新的业务模块能够增加新的业务功能，保证系统各功能模块相互之间的有效集成。
- 采用开放式系统架构和统一技术平台，开发和部署方案不依赖于操作系统、中间件和数据库，具备开放性、通用性、标准性和安全性的特点，满足稳定、灵活、可扩展的要求，提升开发效率和质量。
- 具备统一的系统日志管理，可以收集、分类、检索和分析日志文件和内容，便于问题定位和运维管理。
- 系统具备完备的容错能力，能够保持高效、可靠的运行能力，正确监控、预警、捕获、分析和处理各类异常或故障，具备适用的系统服务启停机制，具备合理的冲正或重发机制，保证账务数据的一致性。
- 提供稳定高效的批量数据处理机制，支持联机交易处理与批量处理的有效分离，保证批量业务处理过程中联机业务的正常执行。

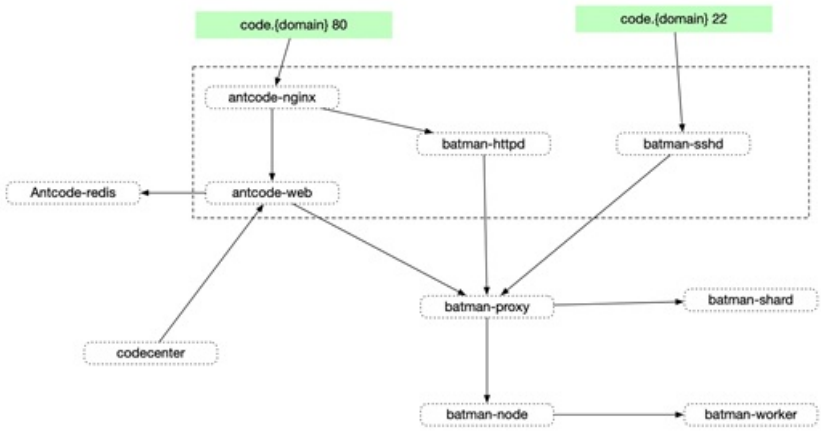
1.2. 系统架构

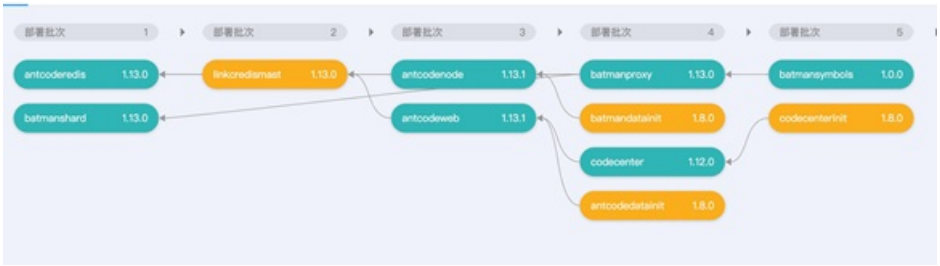
研发效能平台 LinkE 的应用架构设计采用模块化、松耦合设计原则，系统各个功能采用模块化集成，实现新的业务模块能够增加新的业务功能，实现系统各功能模块相互之间的有效集成。



AntCode

作为 LinkE 产品集下的代码托管服务，主要提供代码托管能力，前置依赖 LinkEBase（与 LinkE 系列共享登录态）。主要给 LinkE 工作台，AntBuild 等提供代码服务能力。





图二黄色部分为一次性任务，蓝色节点为实际部署节点，与图一的关系为一对多，具体如下：

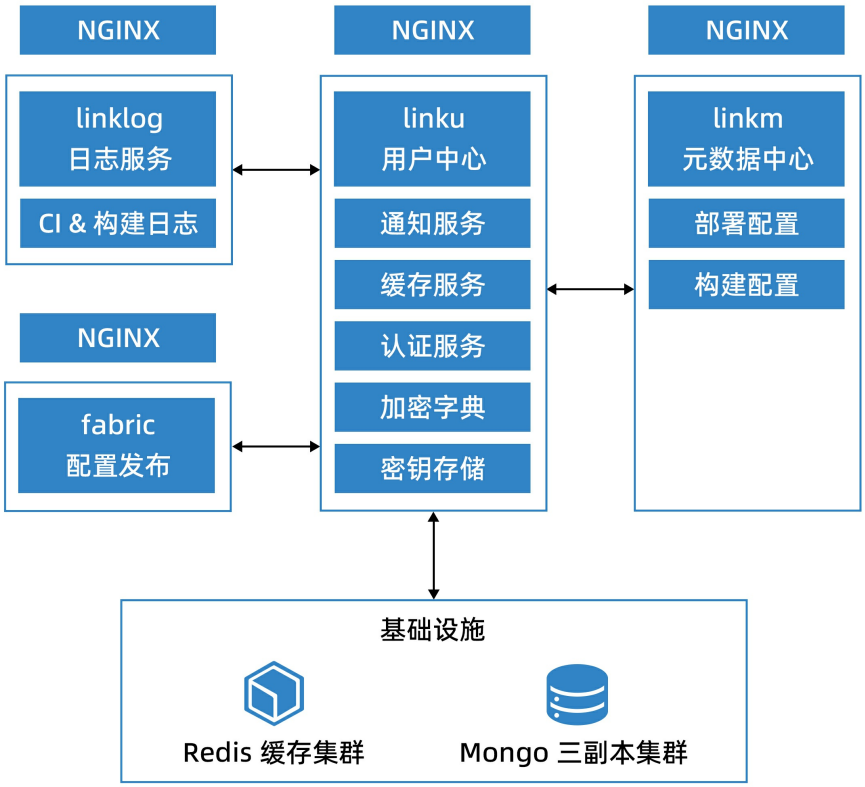
- AntCoderedis：Redis 服务，主要为 AntCodeweb 服务提供业务数据缓存。
- AntCodeweb：实际部署聚合了图一的 antcode-nginx，antcode-web，batman-httpd，batman-sshd 具体如下：
  - **AntCode-nginx**：使用 tengine 的 proxy 的功能，用来分发 API 和 Clone 的 HTTP 协议请求到 antcodeweb，antcodehttpd。

- AntCode-web：Git 的业务数据，包括用户、群组、仓库等信息并且承担了codereview、搜索、代码操作的权限功能，强依赖的 DB（MySQL/OB），Redis 等基础设施。强依赖 LinkU 做用户认证。
- Batman-httpd：提供了 Git Clone 的 HTTP 协议的实现。
- Batman-sshd：提供了 Git Clone 的 SSH 协议的实现。

- AntCode-proxy：基于 gRpc 的网关服务实现，向外统一暴露的 Git 数据，根据租户、命名空间以及读写来分发请求到不同 AntcodeNode 实例，以及处理 AntcodeNode 的数据聚合操作。
- AntCode-shard：提供的 Git 数据的存储元数据，通过 gRpc 以及 HTTP 来暴露服务，强依赖数据库。
- Batman-node：通过 gRpc 暴露对 Git 数据的存储操作的接口。
- Codecenter：对 antcodeweb 的部分接口进行封装后供 LinkE 工作台调用；维护 LinkE 的发布基线数据，强依赖数据库。

LINKBASE

LINKBASE 组件为整个 LinkE 的产品集提供了基础的服务，包括了元数据、配置、日志、用户管理等。主要的依赖是在用户鉴权模块，会依赖金融云网关与鉴权服务（OP 与 IAM）。



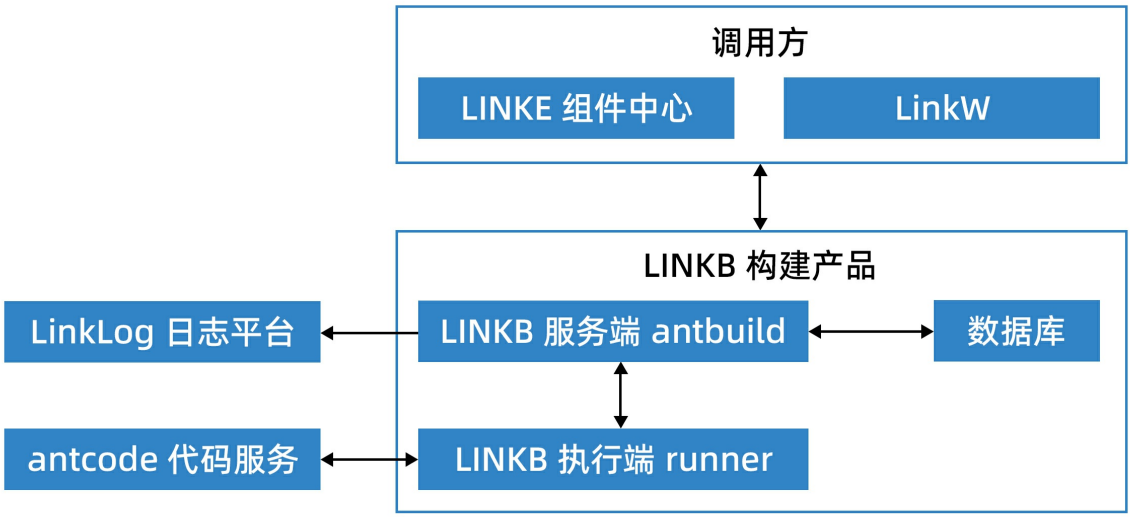
LINKBASE 包括如下几个应用模块：

- linklog（日志存储与查询服务）
- MongoDB（用于非关系型数据存储）
- redis（缓存存储）
- linku（用户鉴权服务）

- fabric（配置管理）
- linkm（元数据管理）

LinkB

作为 LinkE 产品集下的构建组件，是 LinkE 和 环境中心 LinkW 的下游产品，目前本身不直接对外提供构建接口和页面，构建任务的执行来自 Antcode 代码服务平台 Clone 代码；构建的日志对接到 Linklog 日志平台，进行日志的展示和分析。

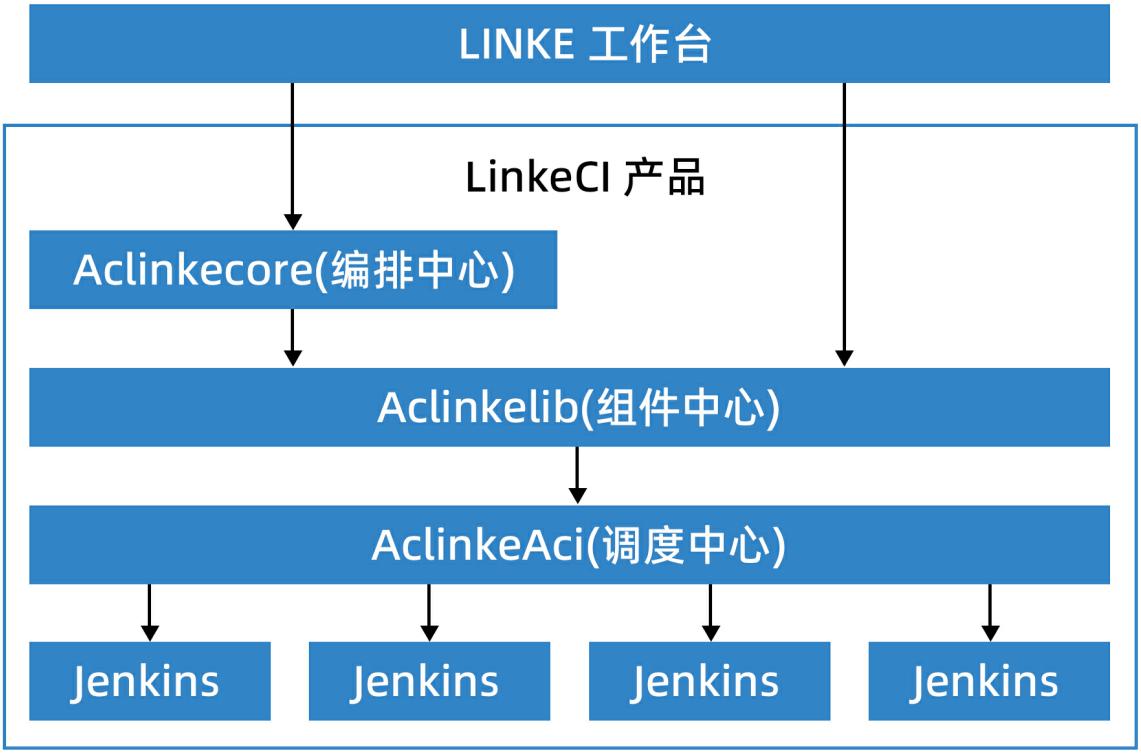


如上图所示，LinkB 主要包括服务端、服务端数据库和执行端 Runner，各模块主要的功能如下：

- LinkB 服务端 antbuild：对外提供构建任务的创建和查询接口、构建任务生成、构建任务状态跟踪、构建日志收集并推送到 LonkLog 日志平台。
- 数据库：对 antbuild 构建的状态、配置等数据进行持久化。
- 执行端 Runner： 构建任务的执行端，该模块是具体构建任务的执行节点，用于消费服务端生成的构建任务，并将执行的结果反馈给服务端，Runner 消费服务端任务采用拉模式消费任务。

LinkCI

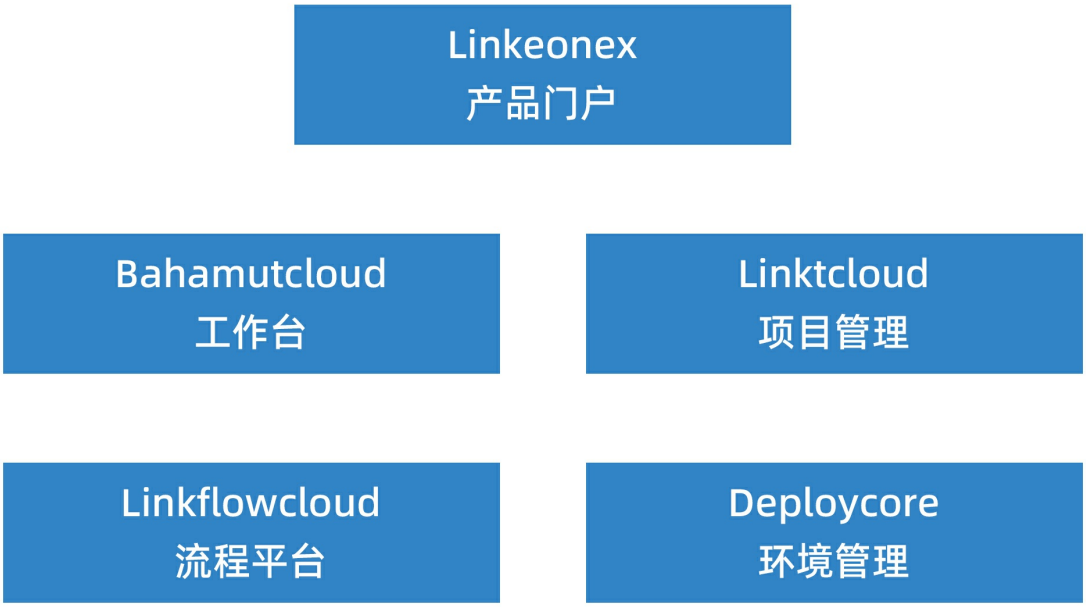
集组件编排、执行于一体的产品组件，能够接入不同的组件通过编排组件生成不同流水线去执行，例如：单元测试、集成测试等。



各组件介绍如下：

- Aclinkecore：用于存储和执行组件的编排。
- Aclinkelib：用于对接组件和执行组件。
- Aclinkeaci：用于模板转换和调度单测和集测。
- Jenkins：用于执行转换后的模板。

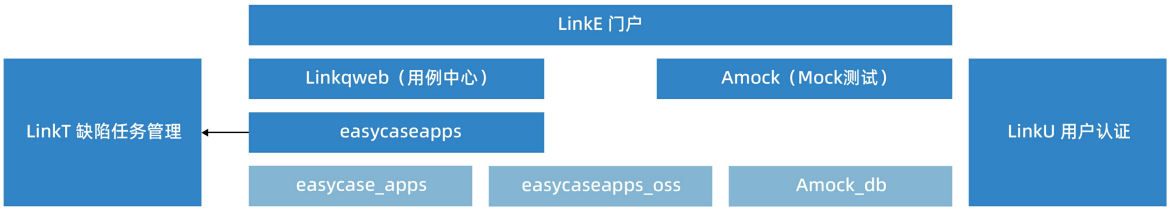
LINKEPORTAL



各组件介绍如下：

- Linkeonex：提供页面访问。
- Bahamutcloud：持续交付工作台。
- LinkTCloud：项目管理平台。
- LinkflowCloud：流程平台。
- Deploycore：环境管理。

LINKQ



- Linkqweb：用例管理的 web 层、实现用户认证、web 页面渲染，操作参数检查、数据映射，文件上传 OSS 读写操作等。系统本身无 DB 链接。强依赖 LinkU 做用户认证。
- Easycaseapps：用例操作是实际服务层、实现用例操作落库、报告生成等操作，依赖 LinkT 做缺陷关联。

> **说明**
>
> Easycaseapps 与 LinkT 的关联属于弱依赖，LinkT 不会导致主流程不可操作。

- Amock：HTTP 协议 Mock 平台，通过配置 API/SPI 的报文及校验规则模拟 HTTP 服务端行为。除了依赖 LinkU 进行用户认证外对其他产品的系统无依赖。

1.3. 运维架构

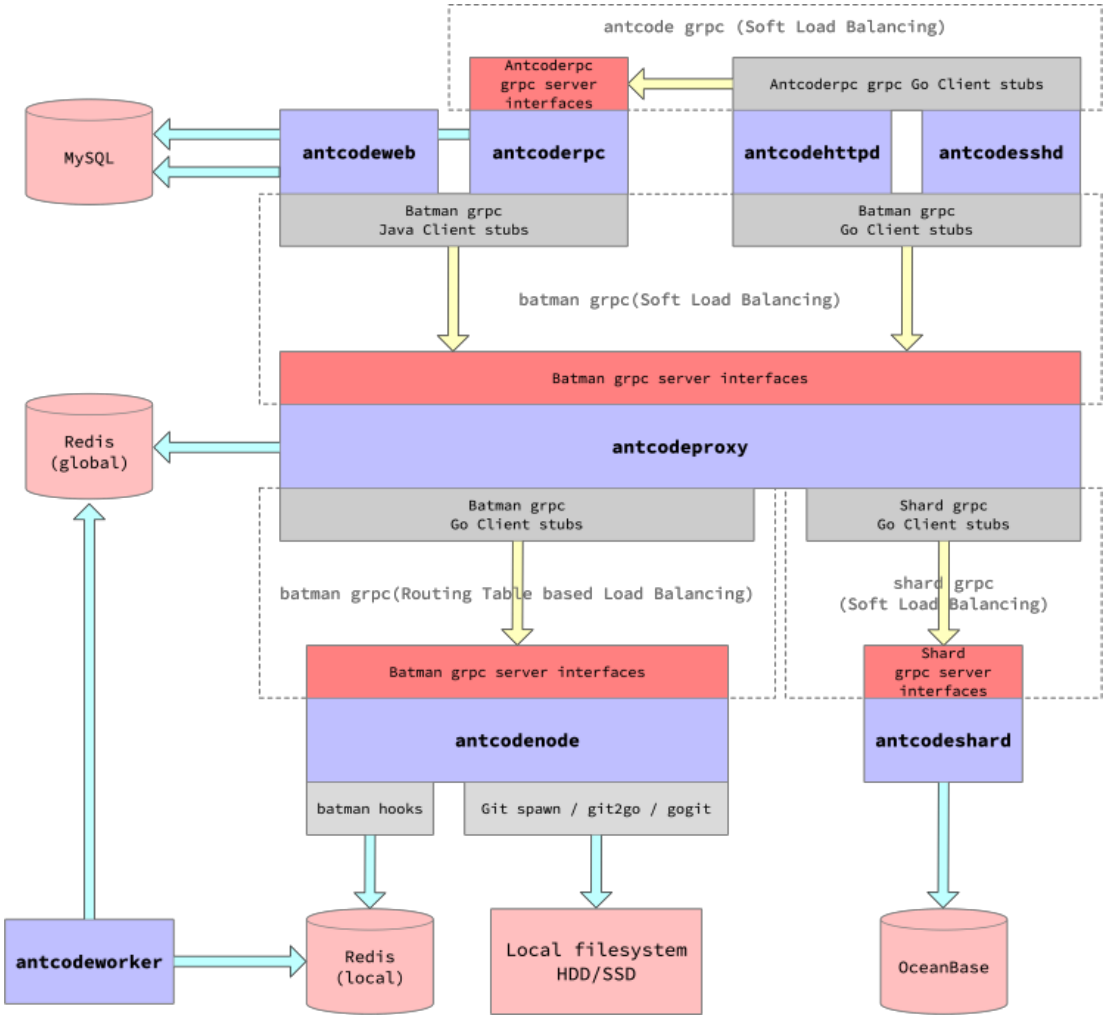
告警与巡检均通过查询 RMS 来操作，主要观察 CPU、内存和磁盘占用量，如 CPU、内存异常均需要重启容器，磁盘占用量过高需要清空 `/home/admin/logs` 下的日志文件。

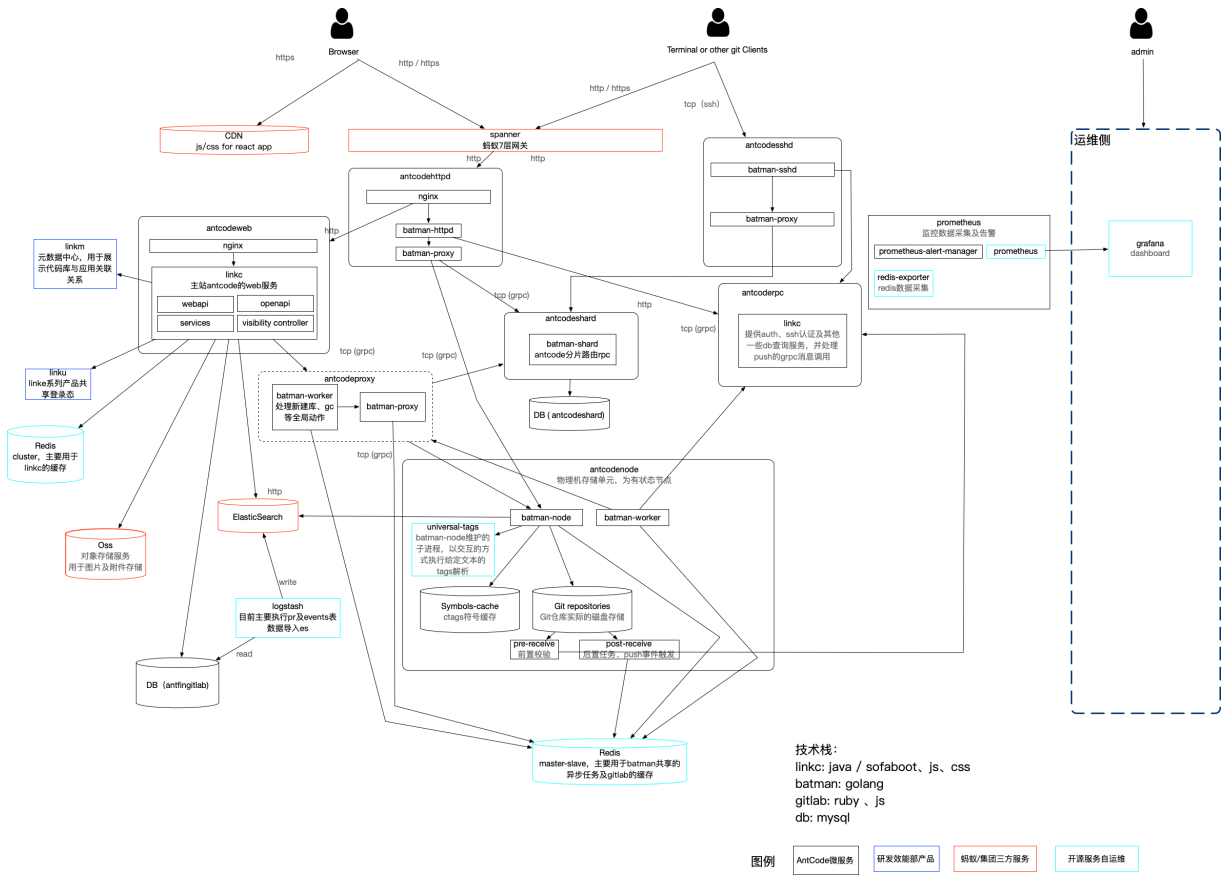
运维大类	描述	运维工具
日常运维	人工巡检和监控	RMS
停机维护	• 检查、确认容器、组件状态 • 停止启动容器、组件 • 升级组件	云游和命令行
故障处理	处理异常故障	云游和命令行

1.4. 代码服务架构

微服务单元

架构图


antcode grpc (Soft Load Balancing)
Antcoderpc grpc server interfaces
Antcoderpc grpc Go Client stubs
MySQL
antcodeweb
antcoderpc
antcodehttpd
antcodesshd
Batman grpc Java Client stubs
Batman grpc Go Client stubs
batman grpc(Soft Load Balancing)
Batman grpc server interfaces
Redis (global)
antcodeproxy
Batman grpc Go Client stubs
Shard grpc Go Client stubs
batman grpc(Routing Table based Load Balancing)
shard grpc (Soft Load Balancing)
Batman grpc server interfaces
Shard grpc server interfaces
antcodenode
antcodeshard
batman hooks
Git spawn / git2go / gogit
antcodeworker
Redis (local)
Local filesystem HDD/SSD
OceanBase



Antcodehttpd

- 代码服务 HTTP 流量的主入口和分发器，由 Golang 编写的 batman 二进制启动运行，除负责 HTTP 协议的代码 Pull/Push 链路外，还通过 tengine 负责反向代理 API 和 WebAPI 的流量到 Antcodeweb。
- Antcodehttpd 向上依赖 antcoderpc，以实现用户登录校验、Pull/Push 权限校验等能力，向下依赖 Antcodeproxy，向用户提供 Git 数据流。

Antcodesshd

- 主要承载代码服务 SSH Pull/Push 链路，由 Golang 编写的 batman 二进制启动运行。
- Antcodehttpd 向上依赖 antcoderpc，以实现用户登录校验、Pull/Push 权限校验等能力，向下依赖 Antcodeproxy，向用户提供 Git 数据流。

Antcodeproxy

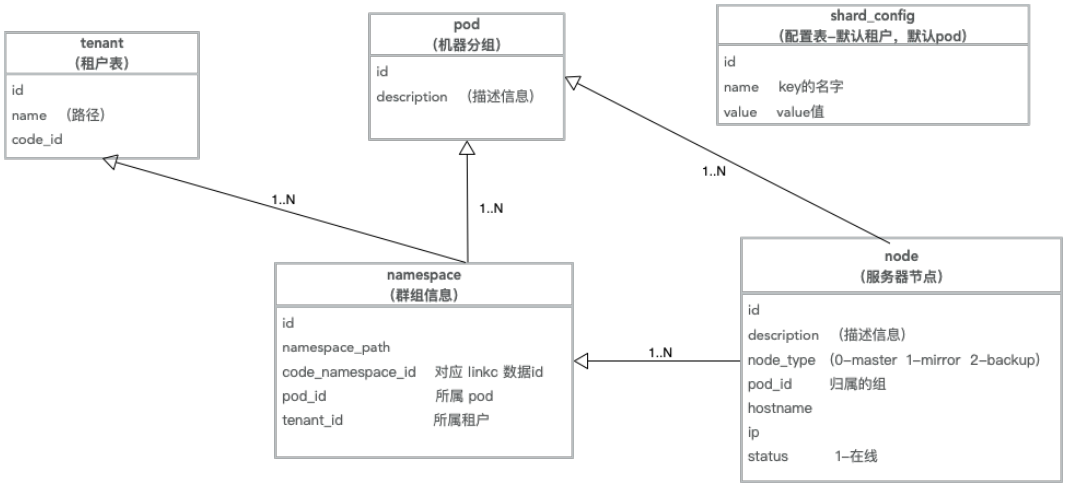
- 根据请求的群组和仓库信息，查询 Antcodeshard 获取底层分片的路由并对请求进行转发。
- 提供一些集群操作的特殊 API。
- 承担代码高亮等服务。
- Antcodeproxy 向上依赖 Antcodeshard 提供路由能力，向下依赖 Antcodenode 实现 Git 元数据相关请求的转发，另外，仓库 GC/删除/transfer 等耗时操作，会生成对应的任务存入 redis 任务队列，实现异步消费。

Antcodeshard

- 由 Golang 编写的 batman 二进制启动运行，访问 batmanshard 数据库读写分片数据，分片单位 pod 表

- 提供群组/仓库新建等 HTTP API，实现路由表写入数据库（OB）；提供仓库/群组路由查询能力的 grpc 服务。

数据结构



Antcodenode

- Git 元数据相关所有 grpc 服务的最终实现，保存了实际的 Git 代码版本库数据，由 Golang 编写的 batman 二进制启动运行。随 Antcodenode 部署的还有 batman-hooks，hooks 提供了代码提交二次校验、仓库同步、代码提交事件生成等定制能力，为 Git 协议实现的标准钩子。
- 另外，Antcodenode 还是提供了代码搜索、代码统计等增值服务的底层对接。

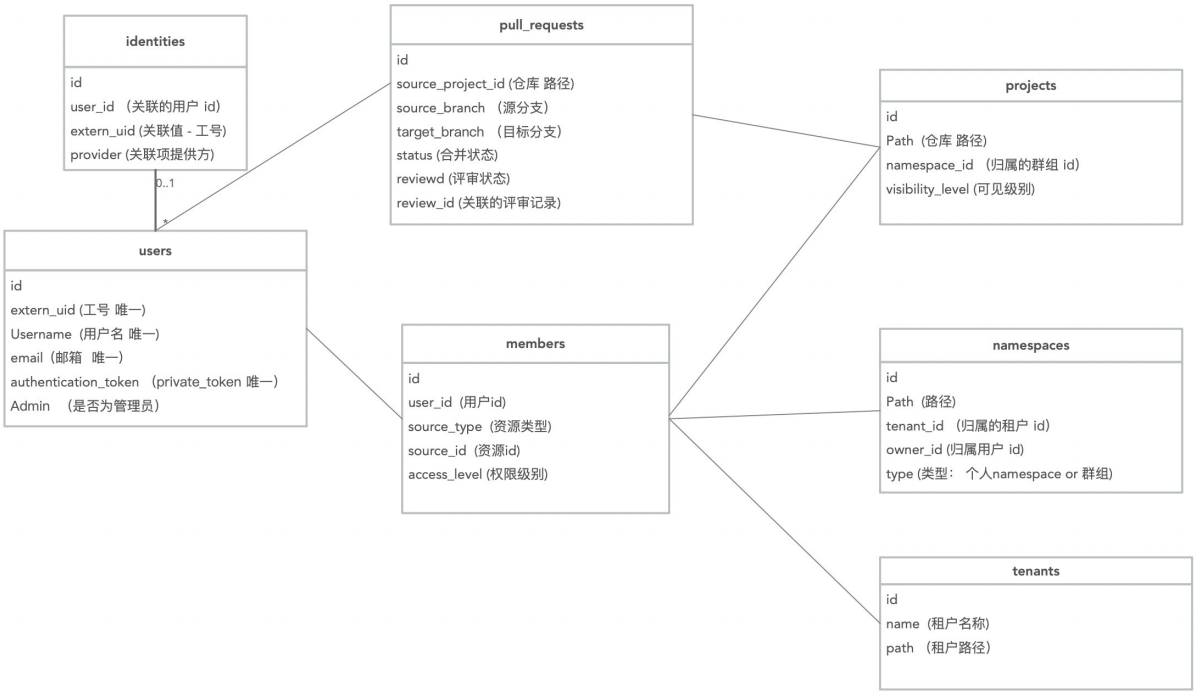
Antcodeworker

- 依赖 redis 任务队列，提供各种事件的消费能力，消费内容为 batmanproxy（antcodeproxy）和 batmanhook 写入的事件，完成仓库创建删除、代码提交、仓库同步、webhook 等事件消费。

Aantcodeweb

- 提供代码服务整个 web 服务的应用，基于 SOFABoot 开发，提供页面访问所需的 Web API 和 Gitlab 规范的 API，所有的用户、仓库、群组、成员、权限、事件等产品使用数据，调用 batman 的各种服务，同时提供 gprc 服务给 batman 完成鉴权等操作。

核心数据结构



1.5. 技术支持

服务热线

若有需要，可随时拨打热线电话：400-995-1887

提交工单

若您有额外的产品功能需求或非紧急的技术咨询，可通过提交工单联系蚂蚁集团技术支持团队。

工单提交地址：https://user.cloud.alipay.com/#/tickets/submit。

2.运维准备

2.1. 运维界面

研发效能平台未提供专属的运维后台，需提前准备云游账号，所有运维操作均在云游上操作，如应用上下线、重启等，其他应用内运维需要通过云游或基础平台进行容器或者物理节点内部操作。

数据库操作

- **MySQL 数据库**：依赖底座提供，需要进入数据库管理后台操作，如 OceanBase 控制台或者 DMS 控制台。
- **Mongo 数据库**：LinkE 依赖 Mongo 数据库，对于未使用底座创建 Mongo 的情况则由 LinkE 自建 Mongo 集群，需要进入 Mongo 容器中使用 mongo 进入命令行操作。

  > **说明**
  >
  > 建议安装 robo3t 的免费 Mongo 客户端进行相关操作，以便于应用相关的数据运维。

- **Redis 数据库**：LinkE 依赖 Redis 数据库，对于未使用底座创建 Redis 的情况则由 LinkE 自建 Redis 集群，目前暂无 Redis 操作的运维需要。

2.2. 运维说明

2.2.1. 有持久化存储的应用

LINKEBASE-linkemongo

linkemongo 是 LinkE 根据 Docker 官方提供的 mongo 3.6-replica 镜像改造，基于云游部署的副本集 Mongo 集群，用来支撑 LinkE 依赖的 Mongo 数据库服务。

启动方式为 `/root/entrypoint.sh`，SLB 通过后端 27018 端口的 Rinetd 转发流量到主节点容器。

```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 04:42 ?        00:00:01 bash /opt/entrypoint.sh
root        18     1  0 04:42 ?        00:00:00 /usr/sbin/sshd
root        24     1  0 04:42 ?        00:00:00 /usr/sbin/cron
root        49    18  0 04:43 ?        00:00:00 sshd: agent [priv]
agent       51    49  0 04:43 ?        00:00:00 sshd: agent@notty
root        70     1  2 04:43 ?        00:17:30 mongod --dbpath /data/db --keyFile /etc/keyFile --replSet rs_linke --logpath /data/mongo.log --bind_ip 0.0.0.0 --fork
root      2500     1  0 04:44 ?        00:00:09 rinetd -c /data/rinetd.conf
root      2540    18  0 04:45 ?        00:00:00 sshd: agent [priv]
agent     2543  2540  0 04:45 ?        00:00:00 sshd: agent@notty
root     31403     1  0 14:31 ?        00:00:00 sleep 20
root     31404     0  0 14:31 ?        00:00:00 /bin/bash -i
root     31409 31404  0 14:31 ?        00:00:00 ps -ef
root@LINKE-2c4g-10:/# cat /data/rinetd.conf
0.0.0.0 27018 14.0.42.154 27017
root@LINKE-2c4g-10:/# mongo --port 27018 -u linke -p $MONOGO_INITDB_ROOT_PASSWORD --authenticationDatabase admin
MongoDB shell version v3.6.5
connecting to: mongodb://127.0.0.1:27018/
MongoDB server version: 3.6.5
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
Server has startup warnings:
2021-10-14T04:34:57.363+0800 I STORAGE  [initandlisten]
2021-10-14T04:34:57.363+0800 I STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine
2021-10-14T04:34:57.363+0800 I STORAGE  [initandlisten] **          See http://dochub.mongodb.org/core/prodnotes-filesystem
2021-10-14T04:35:20.016+0800 I CONTROL  [initandlisten] ** WARNING: You are running this process as the root user, which is not recommended.
2021-10-14T04:35:20.016+0800 I CONTROL  [initandlisten]
2021-10-14T04:35:20.017+0800 I CONTROL  [initandlisten]
2021-10-14T04:35:20.017+0800 I CONTROL  [initandlisten] ** WARNING: soft rlimits too low. rlimits set to 14924 processes, 655350 files. Number of processes should be at
number of files.
rs_linke:PRIMARY> show dbs;
aclinkeaci_main  0.008GB
aclinkelib_main  0.282GB
admin            0.000GB
bahamut          0.056GB
config           0.000GB
linka            2.439GB
linka_task       0.001GB
local            2.395GB
rs_linke:PRIMARY>
```

Mongo 终端下执行 `rs.status()` 可查看所有主副本节点情况：

```
rs_linke:PRIMARY> rs.status()
{
        "set" : "rs_linke",
        "date" : ISODate("2021-10-14T06:35:03.731Z"),
        "myState" : 1,
        "term" : NumberLong(16),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1634193303, 2),
                        "t" : NumberLong(16)
                },
                "readConcernMajorityOpTime" : {
                        "ts" : Timestamp(1634193303, 2),
                        "t" : NumberLong(16)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1634193303, 2),
                        "t" : NumberLong(16)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1634193303, 2),
                        "t" : NumberLong(16)
                }
        },
        "members" : [
                {
                        "_id" : 0,
                        "name" : "14.0.[illegible]7",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 35447,
                        "optime" : {
                                "ts" : Timestamp(1634193303, 2),
                                "t" : NumberLong(16)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1634193303, 2),
                                "t" : NumberLong(16)
                        },
                        "optimeDate" : ISODate("2021-10-14T06:35:03Z"),
                        "optimeDurableDate" : ISODate("2021-10-14T06:35:03Z"),
                        "lastHeartbeat" : ISODate("2021-10-14T06:35:03.360Z"),
                        "lastHeartbeatRecv" : ISODate("2021-10-14T06:35:03.662Z"),
                        "pingMs" : NumberLong(0),
                        "syncingTo" : "14.0.[illegible]",
                        "configVersion" : 3
                },
                {
                        "_id" : 2,
                        "name" : "14.[illegible]7",
```

应用部署

默认采用标准的三节点（1 primary + 2 secondary）副本部署，如需要可支持 扩容。

数据持久化

由于 Mongo 数据库服务提供的是数据服务，为有状态应用，需要做数据持久化，目前基于云游的持久化方式有两种：

- 宿主机 hostPath 方式挂载 Volume，路径为 `/home/t4/data/mongo` ，容器内路径为 `/data` 。

> **说明**
>
> `/home/t4` 通常为宿主机节点默认增加的数据盘挂载点，因此可以确保 Mongo 的持久化数据落在数据盘上。

- k8 pvc 方式挂载的云盘，单独为每个 Pod 挂载一个数据盘（一般为 vdc），容器内路径为 `/data` 。

> 说明
>
> 没有 pvc 且宿主机没有单独挂载 `/home/t4` 的数据盘，需要和底座交付运维同学确认。

容器启动

- linkemongo 启动时会用到云游渲染的 container hosts，根据 hosts 的值匹配当前容器和检测集群，首次启动时所有容器均自动进行副本集和账号的初始化（使用的副本集为 `rs_linke` ，账号为环境变量的值 `$MONGO_INITDB_ROOT_USERNAME/$MONOGO_INITDB_ROOT_PASSWORD` ），并默认选择第一台容器（即 linkemongo-0）作为 primary，待另两台容器的 mongo 进程 ready 之后添加到副本集中。
- Mongo 集群的连接必须通过 primary 节点，由于负载均衡会转发到所有 Pod，因此在 Mongo 容器内部有一层 rinetd 转发将流量转发到 primary 节点。为了兼容部分环境里可能没有 LB 的情况，需要保持容器和负载均衡有相同端口可以访问，因此内部转发通过 27018 到 primary 节点的 27017 端口，负载均衡同时监听 27018 > 27018 和 27017 > 27018 端口。

LINKEBASE-linkuredis & ANTCODE-antcoderedis

启动方式为 `/root/entrypoint.sh` ，SLB 通过后端 6378 端口的 rinetd 转发流量到主容器。

LINKEBASE 2.38 及以后的版本和 ANTCODE 1.32 及以后的版本，Redis 升级为主从+哨兵架构，一主两从通过哨兵监控和切换主从节点，且持久化数据到 `/data/redis` 。

通过 `redis-cli -a $REDIS_PASSWORD` 进入 Redis 命令行终端后输入 `info replication` 即可查看主从节点及副本信息：

```
[root@LINKE-1c2g-7 /]
#ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 04:34 ?        00:00:01 bash /root/entrypoint.sh
root        13     1  0 04:34 ?        00:00:26 redis-server 0.0.0.0:7788
root        92     1  0 04:34 ?        00:01:25 redis-server 0.0.0.0:6379
root       113     1  0 04:34 ?        00:00:55 redis-sentinel *:26379 [sentinel]
root       298     1  0 04:37 ?        00:00:17 rinetd -c /data/rinetd.conf
root     22910     0  0 12:04 ?        00:00:00 /bin/bash -i
root     23109     1  0 12:06 ?        00:00:00 sleep 30
root     23127 22910  0 12:06 ?        00:00:00 ps -ef

[root@LINKE-1c2g-7 /]
#cat /data/rinetd.conf
0.0.0.0 6378 14.0.42.136 6379

[root@LINKE-1c2g-7 /]
#redis-cli -a $REDIS_PASSWORD
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
127.0.0.1:6379> info replication
# Replication
role:master
connected_slaves:2
slave0:ip=14.0.42.132,port=6379,state=online,offset=734756391,lag=1
slave1:ip=14.0.42.131,port=6379,state=online,offset=734749088,lag=1
master_failover_state:no-failover
master_replid:ac40ac2a7ac9a6a8b42aa8c4d206693b6082a59c
master_replid2:82522bb4055460cc58dfca37fa3bef053ea9c6ad
master_repl_offset:734768552
second_repl_offset:7364
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:733719977
repl_backlog_histlen:1048576
127.0.0.1:6379>
```

在 26379 端口进入到哨兵的控制台，输入 `sentinel masters` 可查看监控的主从集群情况。

```
[root@LINKE-1c2g-7 /]
#redis-cli -p 26379
127.0.0.1:26379> sentinel masters
1)  1) "name"
    2) "mymaster_14.0.42.132"
    3) "ip"
    4) "14.0.42.136"
    5) "port"
    6) "6379"
    7) "runid"
    8) "532146aedee1f8204b747a5090a695c46a61dc52"
    9) "flags"
   10) "master"
   11) "link-pending-commands"
   12) "0"
   13) "link-refcount"
   14) "1"
   15) "last-ping-sent"
   16) "0"
   17) "last-ok-ping-reply"
   18) "714"
   19) "last-ping-reply"
   20) "714"
   21) "down-after-milliseconds"
   22) "30000"
   23) "info-refresh"
   24) "6931"
   25) "role-reported"
   26) "master"
   27) "role-reported-time"
   28) "31459247"
   29) "config-epoch"
   30) "1"
   31) "num-slaves"
   32) "2"
   33) "num-other-sentinels"
   34) "1"
   35) "quorum"
   36) "1"
   37) "failover-timeout"
   38) "180000"
```

应用部署

新版本默认采用标准的主从（1 master + 2 slave）架构部署，哨兵服务在主从节点容器内部启动，如需要可支持扩容。

数据持久化

由于 redis 服务提供的是数据服务，为有状态应用，需要做数据持久化，目前基于云游的持久化方式有两种：

- 宿主机以 hostPath 方式挂载 `Volume /home/t4/` 下的路径，容器内路径为 `/data/redis` 。

> 说明
>
> `/home/t4` 一般为宿主机节点默认增加的数据盘挂载点，因此可以确保 Mongo 的持久化数据落在数据盘上。

- k8 pvc 方式，以此种方式挂载的云盘，会单独为每个 pod 挂载一个数据盘（一般为 vdc），容器内路径为 `/data/redis` 。

> 说明
>
> 没有 pvc 且宿主机没有单独挂载 `/home/t4` 的数据盘，需要和底座交付运维同学确认。

容器启动

- Redis 启动时会用到云游渲染的 container hosts，根据 hosts 的值匹配当前容器和检测集群，首次启动时所有容器均自动进行副本集和账号的初始化（密码为环境变量 `$REDIS_PASSWORD` ），并默认选择第一台容器作为 master，另两台用 slave 的配置文件启动。
- Mongo 集群的连接目前必须通过 master 节点，负载均衡会转发到所有 Pod，因此在 mongo 容器内部有一层 rinetd 转发将流量转发到 master 节点，为了兼容部分环境里可能没有 LB 的情况，需要保持容器和负载均衡有相同端口可以访问，因此内部转发通过 6378 到 master 节点的 6379 端口，负载均衡同时监听 6379->6378 和 6378->6378 端口。

LINKEBASE-linkenexus

linkenexus 是 LinkE 根据 sonatype 官方提供的 Nexus3 镜像改造，基于云游部署的 Nexus 服务，提供 Maven 等仓库服务。

应用部署

默认通过云游部署双容器，一主一备，日常只读写主节点，备节点做数据备份（可以提供读取功能，不能提供写入功能）。

数据持久化

由于 Nexus 提供的是数据服务，为有状态应用，需要做数据持久化，目前基于云游的持久化方式有两种：

- 宿主机以 hostPath 方式挂载 Volume，路径为 `/home/t4/data/mongo` ，容器内路径为 `/nexus-data` 。

> 说明
>
> `/home/t4` 一般为宿主机节点默认增加的数据盘挂载点，因此可以确保 Mongo 的持久化数据落在数据盘上。

- k8 pvc 方式，以此种方式挂载的云盘，单独为每个 Pod 挂载一个数据盘（一般为 vdc），容器内路径为 `/nexus-data` 。

> 说明
>
> 没有 pvc 且宿主机没有单独挂载 `/home/t4` 的数据盘，需要和底座交付运维同学确认。

容器启动

- linkenexus 启动时会用到云游渲染的 container hosts，根据 hosts 的值匹配当前容器和检测主备节点，首次启动时所有容器由 Nexus 的启动脚本自动进行初始化，第一台为主第二台为备，主节点通过 rsync 向备节点实时同步数据，初始化启动之后初始密码在 `/nexus-data/admin.password` 下。
- 负载均衡是转发到所有 Pod 的，因此在 Nexus 容器内部有一层 rinetd 转发将流量转发到主节点，为了兼容部分环境里可能没有 LB 的情况，需要保持容器和负载均衡有相同端口可以访问，因此内部转发通过 8080 到主节点的 8081 端口，负载均衡同时监听 8080 > 8080 和 8081 > 8081 端口，8080 端口支持读写，8081 只能读取（可以在 settings 中进行限制，在服务中不能限制写入操作）。

ANTCODE-antcodenode

antcodenode 为 LinkE 提供的代码托管能力底层服务，提供 Git 操作等核心代码服务操作。

进程为 Golang 编写的 batman 二进制程序，并通过 Supervisor 管理启动。

```
[root@y15016lac-antcodenode-1 /home/admin]
#ps -ef
UID          PID   PPID  C STIME TTY          TIME CMD
root           1      0  0 Oct13 ?        00:00:00 /bin/bash /home/admin/bin/start.sh
root          49      1  0 Oct13 ?        00:00:07 bash /home/admin/bin/redis-start.sh
root         199      1  0 Oct13 ?        00:03:31 redis-server 0.0.0.0:6379
root         219      1  0 Oct13 ?        00:03:21 redis-sentinel *:26379 [sentinel]
root         392      1  0 Oct13 ?        00:00:24 /usr/bin/python /usr/bin/supervisord -c /etc/supervisord.conf
root         395    392  0 Oct13 ?        00:00:00 /usr/sbin/sshd -D
git          396    392  0 Oct13 ?        00:00:21 /usr/local/bin/batman node --config_file /etc/batman/batman.yaml
admin        397    392  0 Oct13 ?        00:00:56 /usr/local/bin/batman worker --config_file /etc/batman/batman.yaml
root         398    392  0 Oct13 ?        00:00:16 python /home/admin/bin/node_helper.py
git          416    396  0 Oct13 ?        00:00:00 universal-ctags --_interactive=default --fields=* --languages=Basic,C,C#,C++,Clojure,Cobol,CSS,CUD
git          419    396  0 Oct13 ?        00:00:00 universal-ctags --_interactive=default --fields=* --languages=Basic,C,C#,C++,Clojure,Cobol,CSS,CUD
root         643    395  0 Oct13 ?        00:00:00 sshd: agent [priv]
agent        646    643  0 Oct13 ?        00:00:00 sshd: agent@notty
agent        851      1  0 Oct13 ?        00:00:51 /opt/taobao/java/bin/java -cp /home/agent/xflush/xflush-agenth/system/*:/home/agent/xflush/xflush-
root       18984     49  0 15:24 ?        00:00:00 sleep 30
root       18987      0  0 15:24 ?        00:00:00 /bin/sh -i
root       18993  18987  2 15:24 ?        00:00:00 bash
root       19028  18993  0 15:24 ?        00:00:00 ps -ef

[root@y15016lac-antcodenode-1 /home/admin]
#supervisorctl
batman-node                      RUNNING   pid 396, uptime 1 day, 5:20:35
batman-worker                    RUNNING   pid 397, uptime 1 day, 5:20:35
node-register                    RUNNING   pid 398, uptime 1 day, 5:20:35
sshd                             RUNNING   pid 395, uptime 1 day, 5:20:35
supervisor>
```

> **说明**
>
> 1.32 版本之后采用了内置 Redis（也可以使用外部 Redis，通过容器启动参数直接配置即可，支持相互切换），内置 Redis 同 linkuredis 或 antcoderedis 一样，是采用主从 + 哨兵架构，主从关系与 batman node 的主备关系一致，哨兵服务为三个 Redis 集群为一组哨兵，可用 redis-cli 终端查看详情信息。

应用部署

默认通过云游部署双容器，一主一备，日常只读写主节点，备节点做备份，日常不接收客户端流量。

数据持久化

由于 antcodenode 提供的是代码托管服务，为有状态应用，需要做数据持久化，目前基于云游的持久化方式有两种：

- 宿主机 hostPath 方式挂载 Volume，路径：`/home/t4/data/repo`，容器内路径为 `/home/git/repo`。

> **说明**
>
> `/home/t4` 一般为宿主机节点默认增加的数据盘挂载点，因此可以确保 Mongo 的持久化数据落在数据盘上。

- k8 pvc 方式挂载的云盘，单独为每个 Pod 挂载一个数据盘（一般为 vdc），容器内路径为 `/home/git/repo` 。

> **说明**
>
> 没有 pvc 且宿主机没有单独挂载 `/home/t4` 的数据盘，需要和底座交付运维同学确认。

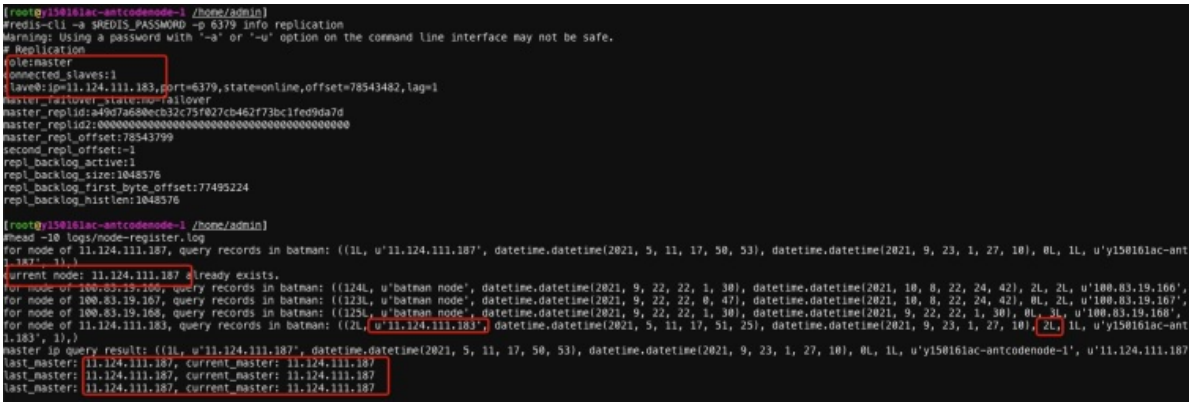
容器启动

- antcodenode 启动时会用到云游渲染的 container hosts，根据 hosts 的值匹配当前容器和检测主备节点，第一台为主第二台为备，主节点通过 `git push --mirror` 向备节点异步同步数据。

> **说明**
>
> 多于两台容器时，默认奇数序号为主，偶数序号为备，支持手动更改或切换。

- 由于依赖 Redis，启动内置 Redis 时会按照与 batman node 一致的主备关系来配置主从节点，且三组主备集群为一组哨兵，节点多时也会按照此逻辑关系进行自动分组。
  - 启动参数 **SHARE_REDIS** 表示是否共享外部 Redis。
    - 为 true 时，node、worker 连接的 REDIS_HOST 地址为外部 redis slb。
    - 为 false 时，说明要使用内置 Redis 作为 node、worker 和 batman 的 redis 服务。
  - 启动参数 **REDIS_INSIDE** 表示是否启动容器内置 Redis。

> **说明**
>
> $SHARE_REDIS=true 时，即使启动内置 Redis 也将不作为 node、worker 的 redis 服务，仍然使用外部 Redis，启动内置 Redis 的情况仅作为哨兵服务需要。



- 节点自动注册，由一个 node-register 的 supervisor 进程完成节点的自动注册和主备监控与切换。

数据同步

主节点通过 `git --git-dir <git repo path> push --mirror git@<备节点ip>` 向备节点异步同步数据，因此需要开启 sshd 进程，同时在 Git `home` 目录下默认写入了 ssh 公私钥对和 `authorized_keys`，用于 Git 用户执行同步操作时登录验证。

```
[root@iZi0e05yjt3ypki4l89dnwZ /home/admin]
#ll /home/git/.ssh/
total 12
-rw-r--r-- 1 git git  394 Oct 14 06:30 authorized_keys
-rw------- 1 git git 1679 Oct 14 06:30 id_rsa
-rw-r--r-- 1 git git  394 Oct 14 06:30 id_rsa.pub

[root@iZi0e05yjt3ypki4l89dnwZ /home/admin]
#cat /home/git/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDtrCUwCMxTKkmSC4Nh3Y6qAZ8nWwpt2Fe14yg5FX3g66hmyWZDlFT8KdsXzM7ZLqqQFHi5gLXq0dFE198v6UX0tp
IW81RlZ4gZ9oXAdZdmn8TxoTW+YnPZyUwo/iMaoj6HpfqaotM3BkuBgyM4+mnD86gAjZwS8+HHU3HHzZ3rGypuHCQVYLNeRY3v5Tz3lof7wEWpIsruABwl4/5pB+8X

[root@iZi0e05yjt3ypki4l89dnwZ /home/admin]
#cat /home/git/.ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDtrCUwCMxTKkmSC4Nh3Y6qAZ8nWwpt2Fe14yg5FX3g66hmyWZDlFT8KdsXzM7ZLqqQFHi5gLXq0dFE198v6UX0tp
IW81RlZ4gZ9oXAdZdmn8TxoTW+YnPZyUwo/iMaoj6HpfqaotM3BkuBgyM4+mnD86gAjZwS8+HHU3HHzZ3rGypuHCQVYLNeRY3v5Tz3lof7wEWpIsruABwl4/5pB+8X

[root@iZi0e05yjt3ypki4l89dnwZ /home/admin]
#
```

在 `/home/git/repo` 目录下校验文件大小即可确认是否数据同步成功，也可以通过 `logs/batman-node.log` 日志搜索 `push --mirror` 的记录。

3.高危操作

3.1. 运维操作的危险级别定义

所有运维操作按业务场景和操作的危险程度被分为三级：G1、G2、G3。

各级别定义如下：

G1：L1 | L2 可以根据文档完成操作，不需要做变更申请，该操作按文档执行完全安全，该操作不会对业务产生影响。

G2：L1 | L2 驻场人员可以根据文档指引，需要做变更申请，向产品侧咨询确认方可执行操作。该操作不会对业务产生影响。

G3：L1 | L2 驻场人员可以根据文档指引，需要做变更申请，向产品侧以及客户侧咨询确认方可执行操作。该操作有可能对业务产生影响。

3.2. 运维操作和文件

高危操作

高危操作即 定义 中提到的 G3 级别的运维操作，进行该类操作时请按照相应的指导进行。下面列举一些在实际操作中可能存在的高危操作：

- 直连 DB 做数据操作，属于 G3 高危险操作。不能绕过应用系统直连 DB 做任何行为的数据变更。
- 历史数据清理，需要谨慎检查每个表的删除标志位。只能删除标志为 REMOVE 状态的数据，严禁删除 ONLINE 状态数据记录。

高危操作指导

下面介绍 Mongo 和 Redis 的高危操作指导。

Mongo

Mongo 为数据库，依赖 /data 目录挂载宿主机保持数据与状态。

> **重要**
>
> /data 下除 .log 日志文件外均为高危文件。

Mongo 为集群，若重启之后，会自动切换主备，建议先找到 primary 节点，然后重启 rinetd。具体操作如下：

1. 进入容器。
2. 执行 `shell: mongo` 。

```
Mongo> use admin
Mongo> db.auth('linke','LinkE2017')
Mongo> rs.status()
```

3. 找到 Primary 的 IP 地址。
4. 退出 Mongo。

5. 修改 `/data/rinetd.conf` ，把第二段 IP 改为 `primary ip` 。
6. 重启 `rinetd: ps aux | grep rinetd | awk '{print $2}' | kill Rinetd -c /data/rinetd.conf` 。

Redis

Redis 为缓存数据库，为有状态应用，依赖 /data 目录挂载宿主机保持数据与状态。

> 重要
>
> - /data 下除 log 日志文件外均为高危文件。
> - 其他应用在 `/home/admin/` 下，除了 logs 文件夹之外请勿删除或变更。

高危文件

- Antcodereids：/data/rinetd.conf (非空)。
- Antcodeweb
  - /home/admin/linkc-run/* (linkc 配置文件 & 执行文件)。
  - /etc/batman/batman.yaml (sshd & httpd 配置文件)。
- Antcodeproxy：/etc/batman/batman.yaml （proxy 配置文件）。
- Antcodeshard：/etc/batman/batman.yaml （shard 配置文件）。
- Antcodenode：/etc/batman/batman.yaml （node 配置文件）。
- Codecenter：/home/admin/release/run/* (程序 & 配置文件)。
- Mongo：/data 下所有文件，为数据库存储与配置。
- Redis：/data 下所有文件，为数据库存储与配置。
- 服务端运行相关文件：/home/admin/antb-run/*。
- Runner 配置文件：/etc/flash/runner/flash.tomAclinkelib: 不依赖文件操作。
- Aclinkecore: 不依赖文件操作。
- Aclinkeaci: 不依赖文件操作。
- Jenkins: /root/.jenkins 是高危文件。
- Linkeonex：包括 /data 整个文件夹。

4.日常运维

4.1. 监控

查看监控项

通过 RMS 实时监控订阅系统日志，做日常的运维监控和告警。

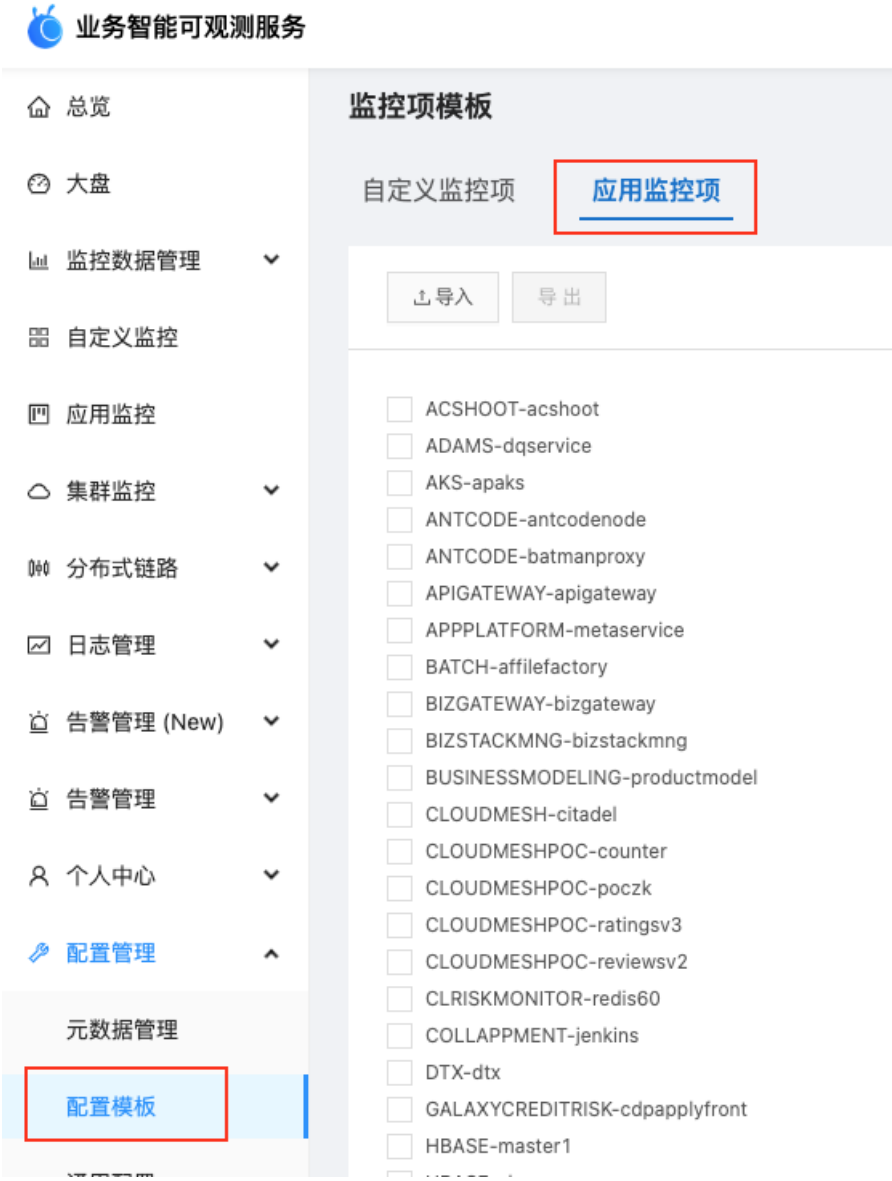
监控项说明

应用	分类	告警阈值	日志路径	说明
LINKEPORTAL-bahamutcloud	CPU 百分比	如果 CPU 百分比平均值连续 3 次大于 80%，系统则会报警。	关注 CPU 的使用情况	登录对应报警的 IP 容器中，检查 CPU 使用情况，查看物理机资源使用率，收集堆栈信息反馈给蚂蚁运维人员售后支持进行分析。
	1 分钟 Load 平均值	如果 1 分钟 Load 平均值连续 3 次大于 5 个，系统则会报警。	关注 CPU 的使用情况	Load 负载是根据 CPU 使用情况制定的，发生报警时应该先检查 CPU 的使用情况。
	磁盘使用率	如果磁盘使用率平均值连续 3 次大于 80%，系统则会报警。	关注磁盘的使用情况	登录对应报警 IP 容器，查看磁盘使用情况，找出占较多的目录，如果日志目录为 losf -L1，需要查看是否清除了日志但是空间没有释放。
	内存使用率	如果内存使用率平均值连续 3 次大于 80%，系统则会报警。	关注内存的使用情况	登录对应报警的 IP 容器，查看内存的使用情况，是否有进程在占用资源。如果内存持续高负载可以收集堆栈、dump 信息反馈给蚂蚁运维人员售后支持进行分析。

	端口	如果在端口 80 连续 3 次监听败，系统则会报警。	80 端口报警说明 nginx 服务没启动	登录对应报警的容器中，查看 nginx 进程是否存活，检查 80 端口。
	业务类	Error	`/home/admin/logs/bahamut/common-error.log`	`/home/admin/logs/bahamut/common-error.log` 日志为错误日志的报警。
		pv	`/home/admin/logs/tracelog/sofa-mvc-digest.log`	页面访问量数据。
		Service	`/home/admin/logs/tracelog/rpc-client-stat.log`	应用服务被调用。
		SQL	`/home/admin/logs/tracelog/zdal-db-stat.log`	SQL 耗时检测。

RMS 监控导入

在 RMS 控制台上，导入应用监控项配置模板：



导入文件如下几个产品的 Json 配置文件：

ANTCODE-rms-2209.json

LINKB-监控模版-2209..json

LINKEBASE-监控模版-2209..json

LINKECI-监控模版-2209..json

LINKEPORTAL-监控模版-2209..json

4.2. 巡检

4.2.1. 组件巡检

监听端口

- Runner 监控端口号 22。

- Antbuild 监控端口号 80。
- Mongo：27017~27018。
- Redis：6379~6378。
- 其他：80~80。

组件巡检

- **CPU**：如 CPU 占用超过 80% 持续 5 分钟，说明有异常，需要做重启处理。
- **内存**：如内存占用超过 80% 持续 5 分钟，说明有异常，需要做重启处理。
- **磁盘**：如磁盘占用超过 95%，说明日志文件过多，需清理日志。

核心进程巡检

Antcode-redis

Redis 服务：6379

Rinetd 服务：端口转发 6378 > 6379

SLB：前端 6379 > 后端 6378

关注：redis 进程、rinetd 进程和 `/data/rinetd.conf` 文件。

```
# ps -ef
UID          PID   PPID  C STIME TTY          TIME CMD
root           1      0  0 Apr16 ?        00:04:26 sh /root/entrypoint.sh
root          23      1  0 Apr16 ?        00:00:00 /usr/sbin/sshd
root          25      1  1 Apr16 ?        10:18:56 redis-server *:6379
root          27      1  2 Apr16 ?        19:54:09 rinetd -c /data/rinetd.conf
root          36      1  0 Apr16 ?        00:00:10 /usr/sbin/cron
root       41159      0  0 00:08 pts/0    00:00:00 /bin/sh -i
root       41171      1  0 00:08 ?        00:00:00 sleep 1
root       41172  41159  0 00:08 pts/0    00:00:00 ps -ef
root       63357     23  0 Apr20 ?        00:00:00 sshd: agent [priv]
agent      63372  63357  0 Apr20 ?        00:00:08 sshd: agent@notty
# cat /data/rinetd.conf
0.0.0.0 6378 11.158.76.159 6379
#
```

巡检步骤如下：

1. 通过 `ps -ef | grep redis` 检查 Redis 服务是否正常运行（端口 6379）。
2. 通过 `ps -ef | grep rinetd` 检查 rinetd 服务是否正常运行。检查 `/data/rinetd.conf` 内容是否为空。若无进程，则执行 `/usr/sbin/rinetd -c /data/rinetd.conf` 进行启动（也可以在云游上触发 antcoderedisinit）。
3. 健康检查：
   - 检查 `redis-cli {reids_host(宿主机 ip)} -p 6379` 是否可正常访问。
   - 检查 `redis-cli {redis slb} -p 6379` 是否可正常访问。(AntCodeweb 通过 SLB 访问)

Antcode-web

SLB：22 > 9997、80 > 80

服务通过 supervisor 进行管理。

核心进程：

```
sh-4.2# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Apr16 ?        00:00:04 /bin/bash /home/admin/start.sh
root        30     1  0 Apr16 ?        00:00:00 nginx: master process /opt/taobao/tengine/bin/tengine -c /opt/taobao/tengine/.running_conf/nginx-proxy.conf -
admin       31    30  0 Apr16 ?        00:00:00 /opt/taobao/install/cronolog/sbin/cronolog /home/admin/logs/cronolog/%Y/%m/%Y-%m-%d-error_log
admin       34    30  0 Apr16 ?        00:00:13 /opt/taobao/install/cronolog/sbin/cronolog /home/admin/logs/cronolog/%Y/%m/%Y-%m-%d-taobao-access_log
admin       35    30  0 Apr16 ?        00:00:00 rollback logs/eagleeye.log interval=60
admin       36    30  0 Apr16 ?        00:03:02 nginx: worker process
admin       37    30  0 Apr16 ?        00:03:00 nginx: worker process
root        79     1  0 Apr16 ?        00:08:19 /usr/bin/python /usr/bin/supervisord -c /etc/supervisord.conf
root        82    79  0 Apr16 ?        00:00:00 /usr/sbin/sshd -D -p 2022
admin       83    79  0 Apr16 ?        00:00:00 /bin/sh /home/admin/run.sh
admin       84    79  0 Apr16 ?        00:02:58 /usr/local/bin/batman httpd --config_file /etc/batman/batman.yaml
admin       85    79  0 Apr16 ?        00:04:04 /usr/local/bin/batman sshd --config_file /etc/batman/batman.yaml
admin       89    83  1 Apr16 ?        10:05:47 /opt/taobao/java/bin/java -server -agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005 -Xms1700
root       138     1  0 Apr16 ?        00:00:38 crond
agent      647     1  0 Apr16 ?        00:24:53 /opt/taobao/java/bin/java -cp /home/agent/xflush/xflush-agenth/system/*:/home/agent/xflush/xflush-agenth/lib/
root     78214     0  0 00:14 pts/0    00:00:00 /bin/sh -i
root     80308   138  0 00:19 ?        00:00:00 CROND
root     80323 80308  0 00:19 ?        00:00:00 /bin/sh -c /usr/bin/vsar --cron > /dev/null 2>&1
agent    80324     1  0 00:19 ?        00:00:00 /bin/sh /home/agent/xflush/xflush-agenth/bin/diver-monitor.sh all
root     80327 80323  0 00:19 ?        00:00:00 /usr/bin/vsar --cron
agent    80706 80324  0 00:20 ?        00:00:00 sleep 5
root     80707 78214  0 00:20 pts/0    00:00:00 ps -ef
sh-4.2#
```

端口：

```
sh-4.2# netstat -antp | grep LISTEN
tcp        0      0 0.0.0.0:35953           0.0.0.0:*               LISTEN      89/java
tcp        0      0 0.0.0.0:34393           0.0.0.0:*               LISTEN      89/java
tcp        0      0 0.0.0.0:9982            0.0.0.0:*               LISTEN      89/java
tcp        0      0 0.0.0.0:9997            0.0.0.0:*               LISTEN      85/batman
tcp        0      0 0.0.0.0:9998            0.0.0.0:*               LISTEN      84/batman
tcp        0      0 0.0.0.0:8081            0.0.0.0:*               LISTEN      89/java
tcp        0      0 0.0.0.0:8082            0.0.0.0:*               LISTEN      89/java
tcp        0      0 0.0.0.0:8084            0.0.0.0:*               LISTEN      89/java
tcp        0      0 0.0.0.0:12200           0.0.0.0:*               LISTEN      89/java
tcp        0      0 0.0.0.0:2022            0.0.0.0:*               LISTEN      82/sshd
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      30/nginx: master pr
tcp        0      0 0.0.0.0:8885            0.0.0.0:*               LISTEN      85/batman
tcp        0      0 0.0.0.0:8888            0.0.0.0:*               LISTEN      84/batman
tcp        0      0 0.0.0.0:19211           0.0.0.0:*               LISTEN      647/java
tcp        0      0 0.0.0.0:19212           0.0.0.0:*               LISTEN      647/java
tcp        0      0 0.0.0.0:19213           0.0.0.0:*               LISTEN      647/java
tcp        0      0 0.0.0.0:19214           0.0.0.0:*               LISTEN      647/java
tcp        0      0 0.0.0.0:19215           0.0.0.0:*               LISTEN      647/java
tcp        0      0 0.0.0.0:19216           0.0.0.0:*               LISTEN      647/java
tcp        0      0 127.0.0.1:9001          0.0.0.0:*               LISTEN      79/python
tcp        0      0 0.0.0.0:5005            0.0.0.0:*               LISTEN      89/java
sh-4.2#
```

关键文件：

```
Batman-httpd & batman-sshd: /etc/batman/batman.yaml
Antcodeweb: /home/admin/linkc-run/*
Supervisord: /etc/supervisord.conf
```

Log：

- /home/admin/logs/linkc/common-default.log （日常日志）
- /home/admin/logs/linkc/common-error.log （错误日志）
- /home/admin/logs/cronolog/** （nginx 日志）
- /home/admin/logs/batman-httpd.log
- /home/admin/logs/batman-sshd.log

巡检步骤如下：

1. 进程检查
   - 通过 `ps -ef | grep linkc` 检查 Java 服务进程是否存在。（可通过重启应用，或通过 supervisorctl 进行管理）
   - 通过 `ps -ef | grep batman-sshd batman-httpd` 关注进程是否存在。（supervisorctl 管理）。
   - 通过 `ps -ef | grep ngnix` 检查 nginx 服务。
2. 确认服务是否可访问： `curl http://localhost:8084/api/v3/user` （响应 401）。
3. 确认并链接 DB，DB 的相关配置可以在文件 `/home/admin/linkc-run/application.properties` 获取。

4. LinkC 服务启动异常检查。
   - LinkC 日志：`/home/admin/logs/linkc/*.log`
   - batman 日志：
     - `/home/admin/logs/batman-httpd.log`
     - `/home/admin/logs/batman-sshd.log`

Antcode-proxy

SLB：50551 > 50551

服务通过 supervisor 进行管理。

```
sh-4.2# ps -ef
UID          PID   PPID  C STIME TTY          TIME CMD
root           1      0  0 Apr16 ?        00:00:05 /bin/bash /home/admin/start.sh
root          41      1  0 Apr16 ?        00:11:22 /usr/bin/python /usr/bin/supervisord -c /etc/supervisord.conf
root          44     41  0 Apr16 ?        00:00:47 /usr/sbin/sshd -D
admin         45     41  0 Apr16 ?        00:44:40 /usr/local/bin/batman worker --config_file /etc/batman/batman.yaml
admin         47     41  0 Apr16 ?        00:06:53 /usr/local/bin/batman proxy --config_file /etc/batman/batman.yaml
root         111      1  0 Apr16 ?        00:01:03 crond
root      100725      0  0 00:35 pts/0    00:00:00 /bin/sh -i
root      100934    111  0 00:36 ?        00:00:00 CROND
agent     100945      1  0 00:36 ?        00:00:00 /bin/sh /home/agent/xflush/xflush-agenth/bin/diver-monitor.sh all
root      100946 100934  0 00:36 ?        00:00:00 /bin/sh -c /usr/bin/vsar --cron > /dev/null 2>&1
root      100949 100946  0 00:36 ?        00:00:00 /usr/bin/vsar --cron
root      101115     44  0 Apr20 ?        00:00:02 sshd: agent [priv]
agent     101117 101115  0 Apr20 ?        00:00:40 sshd: agent@notty
agent     101610 100945  0 00:36 ?        00:00:00 sleep 5
root      101611 100725  0 00:36 pts/0    00:00:00 ps -ef
sh-4.2#
```

端口：

```
sh-4.2# netstat -antp | grep LISTEN
tcp        0      0 0.0.0.0:2022            0.0.0.0:*               LISTEN      44/sshd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      44/sshd
tcp        0      0 0.0.0.0:8887            0.0.0.0:*               LISTEN      47/batman
tcp        0      0 0.0.0.0:8889            0.0.0.0:*               LISTEN      45/batman
tcp        0      0 127.0.0.1:9001          0.0.0.0:*               LISTEN      41/python
tcp        0      0 0.0.0.0:50551           0.0.0.0:*               LISTEN      47/batman
sh-4.2#
```

Log：

- `/home/admin/logs/batman-proxy.log`
- `/home/admin/logs/batman-worker.log1.2.4`

巡检步骤如下：

1. 进程检查

   通过 `ps -ef | grep batman` 关注进程 worker 和 proxy 是否存在。

2. 服务启动异常检查，路径如下：
   - `/home/admin/logs/batman-proxy.log`
   - `/home/admin/logs/batman-worker.log`

Antcode-shard

SLB（内网）：9996 > 9996

服务通过 supervisor 进行管理，依赖 DB。

```
sh-4.2# ps -ef
UID         PID   PPID  C STIME TTY          TIME CMD
root          1      0  0 Apr16 ?        00:00:06 /bin/bash /home/admin/start.sh
root         41      1  0 Apr16 ?        00:09:58 /usr/bin/python /usr/bin/supervisord -c /etc/supervisord.conf
root         44     41  0 Apr16 ?        00:00:50 /usr/sbin/sshd -D
admin        46     41  0 Apr16 ?        00:05:58 /usr/local/bin/batman shard --config_file /etc/batman/batman.yaml
root        100      1  0 Apr16 ?        00:01:00 crond
root       1190    100  0 00:41 ?        00:00:00 CROND
root       1195   1190  0 00:41 ?        00:00:00 /bin/sh -c /usr/bin/vsar --cron > /dev/null 2>&1
root       1202   1195  0 00:41 ?        00:00:00 /usr/bin/vsar --cron
agent      1203      1  0 00:41 ?        00:00:00 /bin/sh /home/agent/xflush/xflush-agenth/bin/diver-monitor.sh all
agent      1636   1203  0 00:41 ?        00:00:00 sleep 5
root       1637      0  0 00:41 pts/0    00:00:00 /bin/sh -i
root       1644   1637  0 00:41 pts/0    00:00:00 ps -ef
root     102414     44  0 Apr20 ?        00:00:02 sshd: agent [priv]
agent    102440 102414  0 Apr20 ?        00:00:29 sshd: agent@notty
sh-4.2#
```

端口：

```
sh-4.2# netstat -antp | grep LISTEN
tcp        0      0 127.0.0.1:9001          0.0.0.0:*               LISTEN      41/python
tcp        0      0 0.0.0.0:9994            0.0.0.0:*               LISTEN      46/batman
tcp        0      0 0.0.0.0:9996            0.0.0.0:*               LISTEN      46/batman
tcp        0      0 0.0.0.0:2022            0.0.0.0:*               LISTEN      44/sshd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      44/sshd
tcp        0      0 0.0.0.0:8884            0.0.0.0:*               LISTEN      46/batman
sh-4.2#
```

Log： `/home/admin/logs/batman-shard.log`

巡检步骤如下：

1. 进程检查

   通过 `ps -ef | grep batman` 检查进程 shard 是否存在。

2. 服务启动异常检查，路径为：`/home/admin/logs/batman-shard.log` 。

3. 确认 DB，从配置 `/etc/batman/batman.yaml` 中获取。

Antcode-node

SLB：无

Git 文件存储节点，注意磁盘容量。

```
sh-4.2# ps -ef
UID         PID   PPID  C STIME TTY          TIME CMD
root          1      0  0 Apr16 ?        00:00:04 /bin/bash /home/admin/start.sh
root         45      1  0 Apr16 ?        00:08:54 /usr/bin/python /usr/bin/supervisord -c /etc/supervisord.conf
root         48     45  0 Apr16 ?        00:00:52 /usr/sbin/sshd -D
admin        49     45  0 Apr16 ?        00:24:57 /usr/local/bin/batman worker --config_file /etc/batman/batman.yaml
root        114      1  0 Apr16 ?        00:01:02 crond
root      21433     48  0 Apr20 ?        00:00:02 sshd: agent [priv]
agent     21460  21433  0 Apr20 ?        00:00:40 sshd: agent@notty
git       21917     45  0 May07 ?        00:01:21 /usr/local/bin/batman node --config_file /etc/batman/batman.yaml
git       21926  21917  0 May07 ?        00:00:00 universal-ctags --_interactive=default --fields=* --languages=Basic,C,C#,C++,Clojure,Cobol,CSS,CUDA,D,E
root      34783    114  0 00:45 ?        00:00:00 CROND
root      34793  34783  0 00:45 ?        00:00:00 /bin/sh -c /usr/bin/vsar --cron > /dev/null 2>&1
agent     34795      1  0 00:45 ?        00:00:00 /bin/sh /home/agent/xflush/xflush-agenth/bin/diver-monitor.sh all
root      34798  34793  0 00:45 ?        00:00:00 /usr/bin/vsar --cron
root      35196      0  0 00:45 pts/0    00:00:00 /bin/sh -i
agent     35329  34795  0 00:45 ?        00:00:00 sleep 5
root      35330  35196  0 00:45 pts/0    00:00:00 ps -ef
sh-4.2#
```

端口：

```
sh-4.2# netstat -antp | grep LISTEN
tcp        0      0 0.0.0.0:8886            0.0.0.0:*               LISTEN      21917/batman
tcp        0      0 0.0.0.0:8889            0.0.0.0:*               LISTEN      49/batman
tcp        0      0 127.0.0.1:9001          0.0.0.0:*               LISTEN      45/python
tcp        0      0 0.0.0.0:9999            0.0.0.0:*               LISTEN      21917/batman
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      48/sshd
sh-4.2#
```

Git 数据存储路径：

```
sh-4.2# ls -al /home/git/
total 36
drwx------ 1 git  git  4096 Apr 17 13:26 .
drwxr-xr-x 1 root root 4096 Apr 16 16:16 ..
-rw-r--r-- 1 git  git    18 Mar 12  2018 .bash_logout
-rw-r--r-- 1 git  git   193 Mar 12  2018 .bash_profile
-rw-r--r-- 1 git  git   231 Mar 12  2018 .bashrc
drwx------ 2 git  git  4096 Apr 17 13:26 .ssh
drwxr-xr-x 1 git  git  4096 Apr 16 16:17 hooks
drwxr-xr-x 1 git  git  4096 Apr 17 13:26 logs
drwxr-xr-x 4 git  git  4096 Apr 10 11:47 repo
sh-4.2#
```

Log：

- `/home/admin/logs/hook.log`
- `/home/admin/logs/batman-node.log`

巡检步骤如下：

1. 进程检查
   - 通过 `ps -ef | grep batman` 关注进程 node 和 worker 是否存在。
   - 检查 `/home/git/repo` 目录是否存在。

   > **说明**
   >
   > `/home/git/repo` 目录必须存在，且 Git 用户拥有读写权限。

2. 服务启动异常检查，路径如下：
   - `/home/admin/logs/batman-node.log`
   - `/home/admin/logs/batman-worker.log`

Codecenter

SLB（内网）： 80 > 80

依赖 DB：

```
sh-4.1# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Apr14 ?        00:00:04 sh /opt/start.sh
root        18     1  0 Apr14 ?        00:00:00 sh /opt/entrypoint.sh
root       707    18  0 Apr14 ?        00:09:10 /usr/bin/python /usr/bin/supervisord -c /etc/supervisor/supervisord.conf
root       710   707  0 Apr14 ?        00:00:00 /usr/sbin/sshd -D -p 2022
root       752     1  0 Apr14 ?        00:00:38 /sbin/rsyslogd -i /var/run/syslogd.pid -c 5
498        754     1  0 Apr14 ?        00:00:00 shellinaboxd -u shellinabox -g shellinabox --cert=/var/lib/shellinabox --port=9191 --background=/var/run/shel
498        756   754  0 Apr14 ?        00:00:00 shellinaboxd -u shellinabox -g shellinabox --cert=/var/lib/shellinabox --port=9191 --background=/var/run/shel
admin     1897     1  0 Apr14 ?        00:00:00 bash /opt/software/cloudengine/bin/startup.sh /home/admin/mycodecenter-run/deploy /home/admin/mycodecenter-ru
admin     1910  1897  0 Apr14 ?        04:18:06 /usr/paas/java/bin/java -Drpc_bind_network_interface=eth0 -server -Xms1800m -Xmx1800m -Xmn680m -Xss256k -XX:M
root      2049     1  0 Apr14 ?        00:00:40 crond
root      2292     1  0 Apr14 ?        00:00:00 nginx: master process /opt/software/nginx/bin/t-alipay-tengine -c /home/admin/mycodecenter-run/tenginx-conf/t
admin     2293  2292  0 Apr14 ?        00:00:00 /opt/software/cronolog/sbin/cronolog /home/admin/logs/nginx/cronolog/%Y/%m/%Y-%m-%d-error.log
admin     2294  2292  0 Apr14 ?        00:00:00 /opt/software/cronolog/sbin/cronolog /home/admin/logs/nginx/cronolog/%Y/%m/%Y-%m-%d-access.log
admin     2295  2292  0 Apr14 ?        00:03:42 nginx: worker process
admin     2296  2292  0 Apr14 ?        00:02:06 nginx: worker process
agent     2537     1  0 Apr14 ?        00:30:28 /usr/paas/java/bin/java -cp /home/agent/xflush/xflush-agenth/system/*:/home/agent/xflush/xflush-agenth/lib/*
agent   107144     1  0 00:58 ?        00:00:00 /bin/sh /home/agent/xflush/xflush-agenth/bin/diver-monitor.sh all
root    107235     0  0 00:58 pts/0    00:00:00 /bin/sh -i
agent   107264 107144  0 00:58 ?        00:00:00 sleep 5
root    107266 107235  0 00:58 pts/0    00:00:00 ps -ef
sh-4.1# netstat -antp | grep LISTEN
tcp        0      0 0.0.0.0:41245           0.0.0.0:*               LISTEN      1910/java
tcp        0      0 127.0.0.1:19001         0.0.0.0:*               LISTEN      707/python
tcp        0      0 0.0.0.0:19211           0.0.0.0:*               LISTEN      2537/java
tcp        0      0 0.0.0.0:19212           0.0.0.0:*               LISTEN      2537/java
tcp        0      0 0.0.0.0:19213           0.0.0.0:*               LISTEN      2537/java
tcp        0      0 0.0.0.0:19214           0.0.0.0:*               LISTEN      2537/java
tcp        0      0 0.0.0.0:19215           0.0.0.0:*               LISTEN      2537/java
tcp        0      0 0.0.0.0:19216           0.0.0.0:*               LISTEN      2537/java
tcp        0      0 0.0.0.0:7001            0.0.0.0:*               LISTEN      1910/java
tcp        0      0 0.0.0.0:9191            0.0.0.0:*               LISTEN      754/shellinaboxd
tcp        0      0 0.0.0.0:1234            0.0.0.0:*               LISTEN      1910/java
tcp        0      0 0.0.0.0:9500            0.0.0.0:*               LISTEN      1910/java
tcp        0      0 0.0.0.0:9877            0.0.0.0:*               LISTEN      1910/java
tcp        0      0 0.0.0.0:36579           0.0.0.0:*               LISTEN      1910/java
tcp        0      0 0.0.0.0:9981            0.0.0.0:*               LISTEN      1910/java
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN      1910/java
tcp        0      0 100.88.174.64:12200     0.0.0.0:*               LISTEN      1910/java
tcp        0      0 0.0.0.0:2022            0.0.0.0:*               LISTEN      710/sshd
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      2292/nginx
tcp        0      0 0.0.0.0:8341            0.0.0.0:*               LISTEN      1910/java
sh-4.1#
```

巡检步骤如下：

1. 进程检查

   通过 `ps -ef | grep mycodecenter` 关注 Java 进程是否存在。

2. 服务启动异常检查，路径为：`/home/admin/logs/mycodecenter/*.log` 。

3. 确认 DB

   从配置 `/home/admin/mycodecenter-run/deploy/mycodecenter.ace/config/sofa config.properties` 中获取。

服务端 antbuild 运维

巡检步骤如下：

1. 查看状态是否正常。
   - 通过 `ps -ef | grep antb java` 检查 nginx 是否正常运行。
   - 检查 `curl -i http://localhost:80/api/v1/runners/count` 是否返回 200 OK。
2. 确认 linklog 是否可访问。

   通过 `/home/admin/antb-run/deploy/antb.ace/config/sofa-config.properties` 查看 linklog 地址 `link_log_url` 是否可以访问。

3. 确认并链接 DB。

   DB 的相关配置可以在文件 `/home/admin/antb-run/deploy/antb.ace/config/sofaconfig.properties` 获取。

Runner 运维

1. 查看 Runner 配置文件。

   查看 `/etc/flash/runner/flash.toml runner flash` 进程的配置文件，确认存在。

2. 查看 Runner 当前状态。

   查看 flash 进程

通过 `ps -ef | grep flash` 检查进程 `/bin/bash /home/admin/flash/runner_start.sh` 和进程 `usr/local/bin/flash runner --config=/etc/flash/runner/flash.toml` 。

查看 runner 进程日志

路径为： `tail -f flash-stderr---supervisor-zomFSs.log` 。

- 以下日志表示 Runner 运行正常：
  - `time="xx" level=info msg="start requestJob" builds=`
  - `time="xx" level=info msg="Checking for jobs... nothing" runner=a88cff4b`
- 以下日志表示 Runner 未注册：
  - `time="xx" level=warning msg="Verifying runner a88cff4b ... offLine" runner=a88cff4b`
  - `time="xx" level=info msg="Checking for jobs... forbidden" runner=a88cff4b`

runner 手动注册

路径为： `flash register --config /etc/flash/runner/flash.toml` ，日志如下：

- `INFO[0000] Registeringshshell...`
- `INFO[0000] Registeringbashshell...`
- `INFO[0000] Using config file: /etc/flash/runner/flash.toml`
- `WARN[0000] Verifying runner a88cff4b ... offLine runner=a88cff4b`
- `WARN[0000] Token a88cff4b configured but not valid, reregistering...`
- `INFO[0000] Registering runner succeeded, new runner token: a88cff4b api=RegisterRunner`

runner 手动下线

路径为： `flash register -u --config /etc/flash/runner/flash.toml` ，日志如下：

- `INFO[0000] Registeringshshell...`
- `INFO[0000] Registeringbashshell...`
- `INFO[0000] Using config file: /etc/flash/runner/flash.toml`
- `INFO[0000]Unregisteringrunnera88cff4b ...succeedapi=UnregisterRunner`
- `INFO[0000] Successfully unregistered a88cff4b`

4.2.2. 守夜人 HAS 巡检配置指南

前提条件

需联系产品运维团队提前在守夜人 Global 平台上导入守夜人巡检规则。

操作步骤

1. 导入导出巡检规则

i. 登录守夜人 Global 平台，在 **日常巡检 > 巡检规则** 页面导出前提条件中提前导入完成的巡检规则。

ii. 在守夜人 Local 站点的金融云租户下执行规则导入操作。

2. 编辑参数

针对 ANTCODE 产品的两个 Git 检测巡检任务，需要配置额外的运行参数。

> **说明**
>
> 如果该守夜人 Local 站点没有开放规则参数的编辑能力，则需要按守夜人系统的官方说明进行如下配置：切换到 ROC 租户，打开 **allowModifyKernel** 开关，导入的巡检规则便可具备配置参数的能力。

3. 创建巡检任务

登录守夜人 Global 平台，在 **日常巡检 > 巡检任务** 页面创建巡检任务。推荐创建两组巡检任务，分别配置ANTCODE 与 LinkE 的规则集。

> **说明**
>
> 守夜人系统的具体使用方法，可联系 TRaaS 团队相关负责人。

参数配置说明

通常在配置中，针对 ANTCODE 的两个 Git 检测操作需要配置额外的运行参数，根据检测协议的不同，本文分为两种参数说明，具体如下：

检测 ANTCODE 的 GIT HTTP 协议是否可以正常工作

> **说明**
>
> - 适用的产品范围为：LINKECI 和 LINKB。
> - 参数配置前需要巡检操作者事先在 ANTCODE 上初始化一个测试使用的代码仓库，并至少上传一个文件，然后从上述产品范围内的应用向 ANTCODE 发起 Git Clone 测试请求。

检测 GIT HTTP 协议需要配置以下 5 个运行参数：

- **GIT_HTTP_REPO_PREFIX**
- **GIT_HTTP_REPO_SUFFIX**
- **GIT_HTTP_USER**
- **GIT_HTTP_PWD**
- **GIT_HTTP_TOKEN**

> **说明**
>
> 若代码库 GIT HTTP 的 URL 为，则 GIT_HTTP_REPO_SUFFIX 应配置为 `code.passdev.alipay.net/antcloud/linke/sxy-aks-test.git`
>
> 而 GIT_HTTP_REPO_PREFIX 应配置为 `http://` 。

在 LinkE 控制台对代码仓库的 GIT HTTP 密码与访问 Token 进行配置，示例如下：

- 代码仓库的 GIT HTTP 密码配置示例

- 访问 Token 的配置示例

完成上述配置后，需要在守夜人 Local 站点上按实际情况进行配置，如下图所示：

检测 ANTCODE 的 Git SSH 协议是否可以正常工作

> **说明**
>
> - 适用的产品范围为：LINKEBASE 和 LINKEPORTAL。
> - 参数配置前需要巡检操作者事先在 ANTCODE 上初始化一个测试使用的代码仓库，并至少上传一个文件，然后从上述产品范围内的应用向 ANTCODE 发起 Git Clone 测试请求。

检测 Git SSH 协议需要配置以下 2 个运行参数：

- **GIT_SSH_HOST**
- **GIT_SSH_REPO_URL**

> **说明**
>
> 若代码库的 GIT_SSH_REPO_URL 为 `antcloud@code.passdev.alipay.net:linke/sxy-aks-test.git`，则 GIT_SSH_HOST 应配置为 `code.passdev.alipay.net`。

示例如下：

注意事项

为了确保 Git SSH Clone 顺利进行，还需要在代码仓库的以下两处页面配置公钥，否则巡检脚本没有权限测试代码。

- **巡检使用用户的 SSH 公钥**

  > **说明**
  >
  > 此公钥的权限跟随着用户。

  公钥内容如下：

  ```
  ssh-rsa
  AAAAB3NzaC1yc2EAAAADAQABAAABgQD0Z8WJXAu3ZUbjJ9DWSGJE1WQpANNOga1pmGRtWvti5Pt0v7
  f2r/Sw2fCrdZF+i2uiC1wdevqDnZPNZ47b+tJEPNo/Ytz1T/NJK0XHJxH0ClZHZhc5bb8kWAjN5IPZBabBn6
  VFipOKcbAX24BMXeeD0sWz890ScaKzG8F4T27klxkh8DGGf5MLGT8u4Jomt3unVzM7RjLiVGzRWyPrideJl
  9L3wLZ+DMrQwM8C+yOOqEISX0JP/DESYBgzAd0CeHPw2cVRPgx/ayHr82qj0gG/FXF659Wi0ie6CfJqNJa
  E/1UsPFdlQilAUkTobqcN1QJyUIL8IYP6Hwasvi16BQLxumiyZD5m9CngnJpaP9k957nBDoUJCsxCw7LAKQ
  2Fetfr65+1vOeFdcfqsgK7j16vEG36uQigGm6fNz4qJgcV/hhW2V6o+baYZ5ViJNzwjNEumrNmocFmQc4jXJ
  nWVWe+DHYWyKvSwBxo2YGPTfYobNHqk45N9KtCb63BfMrdDg8= key1@linke-test
  ```

  配置入口为 **代码服务 > 个人密钥设置 > SSH 密钥**，如下图所示：

- **仓库级别的部署 SSH 公钥**

> **说明**
>
> 此公钥拥有指定仓库的只读权限。

公钥内容如下：

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQCx2gqNmvN0CJvP1hSWbuI82NfRsidhneLb/WY6l11a8+9twLin
gMd9YdxN2xDpiZlXek8ZthWr0uupfJTwYLWDzJZdrsOIk6q2gYDVLok+zTsFWzvQBD+NBMa1HGyW2zQo
e97eO41sHpiVRu10sUt0/gGZmMzR28x0Hku37nYPkA48BHwzV4Q6ctOQKcv9weexfy4QDLaJ7hMyAFXS
X+WomXsg4v1CbIaIQx4xHHY6e1IBtUvYlmlImvTfEbKPsMAwz/Jzf3vkzRstDN5Uqu9AB1/ZOr4VOjKWOdt
pZYzGrumPhFLHRhL25hjFM9N1WmVIk3nQuNv7mu0dDPHBYaMbInFwrRj5gDHxMAWq+FGbe16lFe5+U/
hzAVTD0rMPQVf9jken88HJF/tuaT8a9/xbhR39fAIVxm7RiI09tR0F74WFO4cnBOeumQeapBSMJFfiDK+oy7
lLg5Xcfbvj/vpwi9zUBf7/eThBefu9vw8sy/zmqTzVlWemHhn5g0Pc3GE= key2@linke-test
```

配置入口为 **代码服务 > 我的仓库 > 权限与设置 > 部署公钥**，如下图所示：

4.3. 日志

系统默认日志

- **日志文件名和说明：**app-default.log
- **日志文件路径：** `/home/admin/logs/` 。
- **日志查询方式：**通过 RMS 订阅，或登录容器查询。
- **日志格式：**参见 监控。
- **归档策略：**所有业务日志，采取按天归档的策略。
- **日志示例：**参见 监控。

LINKB 组件

日志文件名：/tmp/flash-stderr-*.log。

日志说明：Runner 的执行日志。

日志文件路径： `/tmp/flash-stderr-*.log` 。

日志查询方式： `tail -f /tmp/flash-stderr-*.log` 。

日志格式：

- 以下日志表示 Runner 运行正常：
  - `time="xx" level=info msg="start requestJob" builds=`
  - `time="xx" level=info msg="Checking for jobs... nothing" runner=a88cff4b`

- 以下日志表示 Runner 未注册：
  - `time="xx" level=warning msg="Verifying runner a88cff4b ... offLine" runner=a88cff4b`
  - `time="xx" level=info msg="Checking for jobs... forbidden" runner=a88cff4b`

归档策略：使用 supervisor 自动管理文档，默认 50M 自动归档。

日志示例：

- `time="2020-05-15T17:29:57+08:00" level=info msg="Checking for jobs...nothing"runner=a88cff4b`
- `time="2020-05-15T17:30:00+08:00" level=info msg="Feeding runners to channel"builds=a88cff4b`
- `time="2020-05-15T17:30:00+08:00" level=info msg="start requestJob" builds=a88cff4b`
- `time="2020-05-15T17:30:00+08:00" level=info msg="Checking for jobs...nothing"runner=a88cff4b`

LINKQ 组件

服务调用摘要（主要监控采集）

- **日志文件名：**tcc-spi-digest.log。
- **日志文件路径：** `/home/admin/logs/` 。
- **日志查询方式：**通过 RMS 订阅，或登录容器查询。
- **日志格式：**参见 监控。
- **归档策略：**所有业务日志，采取按天归档的策略。
- **日志示例：**参见 监控。

服务操作详情

- **日志文件名：**tcc-spi-action.log。
- **日志文件路径：** `/home/admin/logs/` 。
- **日志查询方式：**通过 RMS 订阅，或登录容器查询。
- **日志格式：**参见 监控。
- **归档策略：**所有业务日志，采取按天归档的策略。
- **日志示例：**参见 监控。

服务查询详情

- **日志文件名：**tcc-spi-view.log。
- **日志文件路径：** `/home/admin/logs/` 。
- **日志查询方式：**通过 RMS 订阅，或登录容器查询。
- **日志格式：**参见 监控。
- **归档策略：**所有业务日志，采取按天归档的策略。
- **日志示例：**参见 监控。
- LINKQ 组件中，用例中心前台用户交互操作，主要集中在 2 个 tcc digest 日志文件。

- `tcc-api-digest.log` 只在 linkqweb 系统的这个日志文件才会有内容。记录了前台用户的操作行为，日志格式如下。（默认路径：`/home/admin/logs/fastq-easycase/tcc-api digest.log`）

```
$ tail -f tcc-api-digest.log
2020-05-09 14:36:54,483;0ba03cc8158900621402213129020 2;0.1;80596$长坤;TCC;VIEW;reportList;ok;331 ms
2020-05-09 14:37:02,276;0ba03cc8158900622225513329020 2;0.1;80596$长坤;TCC;VIEW;reportList;ok;12 ms
```

系统日志通过分号（；）分段内容，日志分段内容说明如下：

分段名称	说明
操作时间	系统操作执行的时间，精确到毫秒。
Sofa TracerID	SOFA 框架生成，标识一个请求操作。
Sofa RPCID	SOFA 框架生成，标识一个系统间调用。
用户工号	从 LinkU 系统查询的工号。
用户名称	当前操作的用户名称 LinkU 系统记录。
系统标识	有两种系统内部标识，TCC 和 TC2 。
操作分类	有两种分类：VIEW 和 ACTION。 ■ VIEW：表示只读接口调用。 ■ ACTION：表示数据操作类接口调用。
接口名称	业务接口名称。
接口返回状态	有两种分类：ok 和 err。 ■ ok 表示操作正常。 ■ err 接口返回失败（含参数校验不通过）。
执行耗时（ms）	本次操作的耗时。

- `tcc-spi-digest.log` 只在 easycaseapps 系统的这个日志文件才会有内容。记录系实际处理的服务行为，日志格式如下。（默认路径：`/home/admin/logs/fastqeasyapps/tcc-spi-digest.log`）

```
$ tail -f tcc-spi-digest.log
2020-05-08 13:47:09,575;0ba03cc815889168295556627790202;0.1.1;80596$长坤;TC2;VIEW;GetSampleUrl;ok;0 ms
2020-05-08 13:47:09,648;0ba03cc815889168295554627690202;0.1.1;80596$长坤;TC2;VIEW;TestCaseViewListForWeb;ok;86 ms
2020-05-08 13:47:09,660;0ba03cc815889168295066626990202;0.1.1;80596$长坤;TC2;VIEW;ListTestCaseNode;ok;146 ms
2020-05-08 13:47:12,055;0ba03cc815889168315466628090202;0.1.1;80596$长坤;TC2;VIEW;ListTestCaseNode;ok;501 ms
2020-05-08 13:47:16,107;0ba03cc815889168360626283902 02;0.1.1;80596$长坤;TC2;VIEW;TestLibVersions;ok;26 ms
2020-05-08 13:47:16,411;0ba03cc815889168361666628690202;0.1.1;80596$长坤;TC2;VIEW;TestCaseViewListForWeb;ok;225 ms
2020-05-08 13:47:17,253;0ba03cc815889168369536628990202;0.1.1;80596$长坤;TC2;VIEW;TestCaseViewListForWeb;ok;292 ms
```

- **Mock 测试平台的关键日志**（目录 `/home/admin/logs/amock/` 下）：
  - `mock-invoke.log API`：调用相关业务日志。
  - `mock-spi.log SPI`：服务的详细日志，SPI 服务调用时打印。
  - `mock-groovy-class.log`：自建 groovy 类（加验签类、报文解析类、工具类）相关日志。
  - `mock-rule.log`：规则校验、匹配相关业务日志。
  - `common-error.log`：业务出现问题时抛出异常打印的日志文件。

LINKECI 组件

- **Aclinkelib**
  - 运行正常日志：`/home/admin/logs/aclinkelib/app-default.log`
  - 错误日志：`/home/admin/logs/aclinkelib/common-error.log`
- **Aclinkecore**
  - 运行正常日志：`/home/admin/logs/aclinkecore/common-default.log`
  - 运行错误日志：`/home/admin/logs/aclinkecore/common-error.log`
- **Aclinkeaci**
  - 运行正常日志：`/home/admin/logs/linke-aci/common-default.log`
  - 运行错误日志：`/home/admin/logs/linke-aci/common-error.log`

LINKEPORTAL 组件

- **Bahamut**
  - 运行正常日志：`/home/admin/logs/bahamut/app-default.log`
  - 运行错误日志：`/home/admin/logs/bahamut/common-error.log`
- **Linkt**
  - 运行正常日志：/home/admin/logs/linkt/app-default.log
  - 运行错误日志：`/home/admin/logs/linkt/common-error.log`
- **deploycore**
  - 运行正常日志：`/home/admin/logs/linkw/app-default.log`
  - 运行错误日志：`/home/admin/logs/linkw/common-error.log`

4.4. 升级/重启

4.4.1. 有持久化存储的应用

LINKEBASE-linkemongo

升级或者重启时由于已经建立过 Mongo 集群，所以要从已有的副本集进行创建，容器启动时会校验容器内 `/data/db` 目录存在数据，以该目录为数据路径启动 `rs_linke` 副本集。

为了防止 primary 节点挂掉无法启动（由于节点、数据盘之类的原因导致启动异常）而导致 Mongo 集群无法回到原数据状态，需要维持 primary 节点存活状态才能正常使用。

> 说明
>
> 如果 primary 节点挂掉，重新启动的 secondary 节点可能无法加入到副本集 Node 中。

linkemongo 的升级重启务必选择灰度分组的方式，在集群正常状态下一台或两台 Mongo 下线时，primary 节点会自动切换为剩下的存活节点，因此分组发布时重启的 Mongo 容器节点能作为 secondary 通过 primary 的 Mongo 容器加入到 `rs_linke` 副本集当中继续维持三节点。

LINKEBASE-linkuredis & ANTCODE.antcoderedis

无特殊需要，重启方式可选择直接重启容器，如果容器能进入也可以直接在容器内终止 `/root/entrypoint.sh` 进程使其自动重启。

> 说明
>
> 升级/重启期间如果没有进行主备切换，服务运行状态可能受到影响。

LINKEBASE-linkenexus

重启方式选择直接重启容器即可，由于已经持久化路径下建立过 Nexus 数据，升级或者重启时 Nexus 启动脚本会自动选择 restart 方式保留原有数据和主备关系。

ANTCODE-antcodenode

重启时可选择直接重启容器或者进入容器内部使用 supervisorctl 重启对应服务。

> 重要
>
> 升级/重启主节点期间代码服务不可用。

4.4.2. 普通应用

ANTCODE

antcode 1.32 及以后的版本，antcode 和 batman 的应用镜像合并为同一个，通过启动参数来控制启动哪些服务，启动参数 **ANTCODE_WEB** 为 true 表示启动 linkcweb（java）进程，启动参数 **BATMAN_UNITS** 用来选择启动哪些 batman 单元，因此容器的运维上 antcode 所有应用基本一致，不同点如下：

- 只有 antcodenode 才需要启动内置 Redis。
- antcodeweb 需要启动 Java web 进程，可以通过如下参数进行：
  - 通过 **LINKC_JVM_XmxXms** 调整 Java 进程启动的堆内存大小。
  - 通过 **DRUID_<xxx>** 配置 Druid 数据源监控。
  - 通过 **LINKC_SWAGGER_ENABLE** 开启/关闭 swagger api 文档的访问。
  - 通过 **PAGE_VERSION** 调整前端页面版本，在正式完成 v2 版本升级之前会在解决方案中修改版本为 v1，您也可以通过 `/home/admin/bin/page_version.sh` 来切换版本。

```
[root@y15016lac-antcodeweb-1 /home/admin]
#ll webpage
lrwxrwxrwx 1 root root 21 Oct 13 10:04 webpage -> /home/admin/webpagev1

[root@y15016lac-antcodeweb-1 /home/admin]
#cat /home/admin/bin/page_version.sh
#!/bin/sh

function change_version() {
    if [ -n "$2" ]; then
        ln -snf "$2"/webpage"$1" "$2"/webpage
    else
        ln -snf /home/admin/webpage"$1" /home/admin/webpage
    fi
}

if [ -n "$2" ]; then
    PAGE_VERSION=$2
fi

if [ "$PAGE_VERSION" = "v2" ]; then
    change_version v2 $1
else
    change_version v1 $1
fi

[root@y15016lac-antcodeweb-1 /home/admin]
#bash bin/page_version.sh

[root@y15016lac-antcodeweb-1 /home/admin]
#ll webpage
lrwxrwxrwx 1 root root 21 Oct 14 15:53 webpage -> /home/admin/webpagev1

[root@y15016lac-antcodeweb-1 /home/admin]
#
```

LINKECI-acijenkinscloud

LinkE 内置 Jenkins，用于执行流水线 CI 任务，通过内置脚本启动，jenkinsslb 地址使用 8080 端口可以登录控制台，默认密码为 `admin/ciAdmin`，如果容器异常可直接重启。

```
sh-4.1# ps -ef
UID          PID   PPID  C STIME TTY          TIME CMD
root           1      0  0 May12 ?        00:00:00 bash /home/jenkins/start_jenkins.sh
root          33      1  0 May12 ?        00:00:07 /sbin/rsyslogd -i /var/run/syslogd.pid -c 5
root          64      1  0 May12 ?        00:00:24 crond
root          72      1  0 May12 ?        00:00:00 sudo java -server -Xms1500m -Xmx1500m -Xmn400m -Xss256k -XX:PermSi
root          73     72  1 May12 ?        12:55:13 java -server -Xms1500m -Xmx1500m -Xmn400m -Xss256k -XX:PermSize=12
root         150      0  0 May12 ?        00:00:00 /bin/sh -i
root         157    150  0 May12 ?        00:00:00 bash
root        1219      0  0 May12 ?        00:00:00 /bin/sh -i
root        1226   1219  0 May12 ?        00:00:00 bash
root        1346   1226  0 May12 ?        00:00:00 bash
root       49235      0  0 15:07 ?        00:00:00 /bin/sh -i
root       49241  49235  0 15:07 ?        00:00:00 ps -ef
```

LINKEPORTAL-linkeonex

LinkE 页面转发 oneconsole 的 Nginx 应用容器，由容器内置脚本启动，出现 LinkE 控制台无法打开时可以检查该容器 80 端口是否正常，如异常可直接重启容器。

```
sh-4.2# ps -ef
UID          PID   PPID  C STIME TTY          TIME CMD
root           1      0  0 Jun23 ?        00:00:00 sh /root/entrypoint.sh
root          11      1  0 Jun23 ?        00:00:00 crond start
root          23      1  0 Jun23 ?        00:00:00 nginx: master process nginx -c /root/onex.conf -g daemon off;
root          25     23  0 Jun23 ?        00:00:00 nginx: worker process
root       24520      0  0 15:08 ?        00:00:00 /bin/sh -i
root       24528  24520  0 15:08 ?        00:00:00 ps -ef
sh-4.2#
```

4.4.3. 其他应用

其他应用均为 SOFA 工程，容器入口为 `/opt/entrypoint` ，容器内通过 supervisor 启动进程，出现异常时可以直接重启容器或者进入容器通过 supervisorctl 来操作，supervisorctl 进入交互式之后可以直接 `restart defaultpaas-global` 来重启 SOFA 应用。

```
[root@y150161ac-linkucloud-1 admin]# ps -ef
UID          PID   PPID  C STIME TTY          TIME CMD
root           1      0  0 6月11 ?        00:00:01 sh /opt/entrypoint.sh
root         770      1  0 6月11 ?        00:05:15 /usr/bin/python /usr/bin/supervisord -c /etc/supervisor/supervisord.conf
root         774    770  0 6月11 ?        00:00:00 /usr/sbin/sshd -D -p 2022
root         778    770  0 6月11 ?        00:00:28 /usr/sbin/crond -n
root        1594    770  0 6月11 ?        00:00:20 /usr/sbin/rsyslogd -c5 -f /etc/rsyslog.conf -N debug -n
admin       7585      1  0 6月11 ?        00:00:00 bash /opt/software/cloudengine/bin/startup.sh /home/admin/linku-run/deploy /home/admin/linku-run/work
admin       7598   7585  0 6月11 ?        01:26:36 /usr/paas/java/bin/java -Drpc_bind_network_interface=eth0 -server -Xms1800m -Xmx1800m -Xmn680m -Xss256k -XX:Me
agent       8808      1  0 6月11 ?        00:07:57 /usr/paas/java/bin/java -cp /home/agent/xflush/xflush-agenth/system/*:/home/agent/xflush/xflush-agenth/lib/* -
root        9125      1  0 6月11 ?        00:00:00 nginx: master process /opt/software/nginx/bin/tengine -c /home/admin/linku-run/tenginx-conf/t-alipay-tengine.c
admin       9126   9125  0 6月11 ?        00:00:00 /opt/software/cronolog/sbin/cronolog /home/admin/logs/nginx/cronolog/%Y/%m/%Y-%m-%d-error.log
admin       9127   9125  0 6月11 ?        00:00:26 /opt/software/cronolog/sbin/cronolog /home/admin/logs/nginx/cronolog/%Y/%m/%Y-%m-%d-access.log
admin       9128   9125  0 6月11 ?        00:00:00 rollback logs/eagleeye.log interval=60
admin       9129   9125  0 6月11 ?        00:03:25 nginx: worker process
admin       9130   9125  0 6月11 ?        00:03:23 nginx: worker process
agent     115145      1  0 14:43 ?        00:00:00 /bin/sh /home/agent/xflush/xflush-agenth/bin/diver-monitor.sh all
root      115367      0  0 14:43 ?        00:00:00 /bin/sh -i
root      115374 115367  0 14:43 ?        00:00:00 bash
agent     115408 115145  0 14:43 ?        00:00:00 sleep 5
root      115409 115374  0 14:43 ?        00:00:00 ps -ef
[root@y150161ac-linkucloud-1 admin]# supervisorctl -c /etc/supervisor/supervisord.conf
change_password                  FATAL     Exited too quickly (process log may have details)
crond                            RUNNING   pid 778, uptime 12 days, 21:09:28
defalutpaas-global               EXITED    Jun 11 05:35 PM
fix-cron-notrun                  FATAL     Exited too quickly (process log may have details)
obproxy                          FATAL     Exited too quickly (process log may have details)
rsyslog                          RUNNING   pid 1594, uptime 12 days, 21:09:26
secure_agent                     FATAL     Exited too quickly (process log may have details)
sshd                             RUNNING   pid 774, uptime 12 days, 21:09:28
supervisor> exit
[root@y150161ac-linkucloud-1 admin]# su -c /etc/supervisor/supervisord.conf
su            sudo          sudoedit       sudoreplay     sulogin        sum            supervisorctl  supervisord    sushell        suspend
[root@y150161ac-linkucloud-1 admin]# supervisorctl -c /etc/supervisor/supervisord.conf restart crond
crond: stopped
crond: started
[root@y150161ac-linkucloud-1 admin]#
```

4.5. 扩容

当 LinkE 存在性能容量不足，或者有扩容需求时，可按照本文内容进行操作。

4.5.1. 有持久化存储的应用

LINKEBASE-linkemongo

- **副本集扩容**

  在三节点的基础上再扩容副本节点，如果是同个环境中可以直接扩容后重启 mongo 容器或者手动添加副本节点，如果不是当前同一个环境（比如在备机房进行灾备，需要通过 linkemongo 的容器进行启动或者 Mongo 的脚本启动 Mongo server）那么只能用手动添加为备节点的方式。

  增加一台容器后，需要到当前 Mongo 容器内部操作（或者在能访问 mongo slb 的网络中访问操作）操作步骤如下：

  i. 执行 以下命令进入到 mongo primary 节点。

  ```
  mongo --host mongomaster_slb --port 27018/27017 -u linke -p LinkE2017 --authenticationDatabase admin
  ```

  ii. 执行 `rs.status()` 查看已有副本节点。

  iii. 执行 add 命令，例如 `rs.add({"_id": 3, "host": "11.124.111.161:27017"});` 。

  > 说明
  >
  > `id` 和 `host` 与已有的节点不能重复，新增节点和端口需要和现有集群网络互通。

- **数据容量扩容**

  所有应用容器所在宿主机对应的磁盘扩容或者 pvc 扩容。

LINKEBASE-linkuredis 和 ANTCODE.antcoderedis

- **slave 节点数量扩容**

  因为是固定的三个节点主从架构，无法再自动扩容 slave 节点，如有额外扩容备节点需要，需要新启动一个 Redis 容器，然后在容器内部通过 `redis-cli -a $REDIS_PASSWORD` 进入 Redis 终端后执行 `SLAVEOF <master_ip> 6379` ，使其成为 slave。

- **数据容量扩容**

  所有应用容器所在宿主机对应的磁盘扩容或者 pvc 扩容。

LINKEBASE-linkenexus

- **容器数量扩容**

  Nexus 的写入操作比较少（远少于读取操作），扩容容器可以只增加读取节点（只能写入到一台主节点，但是可以向多个备节点同步），负载均衡 8081 端口分流读流量。

- **数据容量扩容**

  所有应用容器所在宿主机对应的磁盘扩容或者 pvc 扩容。

ANTCODE-antcodenode

- **容器扩容**

  扩容容器数量后，需要从解决方案中强制重发 antcode 产品，否则 container_ips 无法更新从而导致无法注册新节点，除非手动在数据库表中进行注册。

  登录 LinkE 的 mysql 实例查询 batmanshard 数据库的 node 表数据。

> **说明**
>
> IP 字段值需要和 antcodenode 容器 IP 值对应。

```
select * from node;
```

new-antstack-plus. batmanshard

```
1  SELECT * FROM node;
```

Message　Result 1　Status

	created	updated	node_type	pod_id	hostname	ip	status
ode	2021-03-17 20:22:33	2021-03-17 20:22:33	0	1	asp-antcode-antcodenode-0.antcode-antcodenode-svc.asp-antcode	11.166.180.172	1
ode	2021-03-17 20:22:33	2021-03-17 20:22:33	2	1	asp-antcode-antcodenode-1.antcode-antcodenode-svc.asp-antcode	100.88.227.82	1

node 表数据中有两条 status=1 的记录。

- 一条对应容器 `antcodenode-0 node_type=0` 表示主节点。
- 另一条对应容器 `antcodenode-1 node_type=2` 表示备节点。

手动注册

手动注册时需要插入一条记录：hostname 为新容器的 host，ip 为新容器的 ip，`status=1` 表示线状态（下线状态为 0），根据主备关系设置 `node_type` 为 0 或 2，`pod_id` 对于新增集群分片（即新增 master 节点时）需要 +1，如已经存在 `pod_id=1` 的则新增记录 `pod_id` 为 2，以此类推，同时需要在 pod 表中注册一条 pod id 的记录。

单行详情　新建　删除　提交修改　导出数据　【表格数据可以编辑】

	id	description	created	updated
1	1	1	2017-12-04 14:31:11	2017-12-04 14:31:17
2	2	2	2021-10-14 15:37:30	2021-10-14 15:37:33

自动注册

流程同手动注册。

- **数据容量扩容**

  所有应用容器所在宿主机对应的磁盘扩容或者 pvc 扩容。

4.5.2. 其他应用

直接扩容容器数量即可，SLB 会自动挂载多个后端。

5.应急维护

5.1. 流程定义

应急维护的定义

在出现重大故障时，最短时间内进行业务恢复的处理定义为应急处理操作。

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

应急维护的流程和原则

如遇到本章节下覆盖的故障，建议按照文中指导进行故障排查和修复。

若故障排查和修复遇到困难，可联系技术支持寻求帮助。如遇到下文未覆盖的故障，请直接联系技术支持。

5.2. 异常处理

现象	可能的原因	解决方法	验证方法
MongoDB 数据库无法写入。	磁盘空间占满。	1. 清理日志文件。 2. 重启容器。 如果未解决，说明是数据文件过大，请联系运维人员给机器扩容。	进入容器，shell 执行 MongoDB，登录后执行 `db.getCollection('test').insert({a:'1'})`，执行成功则表示成功恢复。
触发的任务一直处于 create 状态，任务一直不消费，无日志。	• 日志平台不可访问。 • Runner 异常，未注册。 • 服务端应用宕机。	1. 登录 Runner 机器查看 Runner 运行态，如果显示 offline，则进行手动注册。 2. 重启服务端应用。 3. 查看服务端日志，common-error 日志与 linklogpass 相关日志是否报错。 如有异常，请联系 Linklog 平台运维人员。	• 检查工作台是否可以查看日志。 • Runner 实时打印请求任务的日志。 • 服务端 Java 和 Nginx 进程存在，并 健康检查返回 200 OK。
流水线列表没数据。	• Java 内存溢出 • 底层异常导致应用宕机	通过基础设施运维工作台，重启容器。	重启容器后检查流水线是否有数据。

现象	可能的原因	解决方法	验证方法
流水线列表有数据，但是触发流水线一直是 abort。	• Java 内存溢出 • 底层异常导致应用宕机	通过基础设施运维工作台，重启容器。	重启容器后检查流水线进程是否恢复正常。
UT/IT 组件一直 pending。	• Java 内存溢出或者 Jenkins 内存溢出 • 底层异常导致应用宕机	通过基础设施运维工作台，重启容器。	重启容器后检查 UT/IT 组件是否恢复正常。
无法访问平台或者访问失败。	• Java 内存溢出 • 底层异常导致应用宕机	通过基础设施运维工作台，重启容器。	检查前台操作产品是否恢复。

5.3. 主备切换

有持久化存储的应用

LINKEBASE-linkemongo

linkemongo 为三副本集群架构，如果部分容器宕机会自动完成主备切换，人工操作可检查 slb 地址连接 mongo 是否正常，或者进入 mongo 容器执行以下命令，检查容器是否是否正常。

```
mongo --host 127.0.0.1 --port 27018 -u linke -p LinkE2017 --authenticationDatabase admin
```

LINKEBASE-linkenexus

linkenexus 有主备特性目前无法自动切换，未切换的情况下第一台容器为主，第二台容器为备，主容器宕机则无法正常提供服务，需要先确保数据安全（例如手动在 Pod 对应的机器上备份 `/home/t4/nexus-data` 的目录）再进行容器重启，在备容器正常的情况下也可以在备份数据后先将两台容器内的 rinetd 服务转发地址切换到备容器：

```
# 修改 rinetd 中转发的目标地址到被容器：8080 127.0.0.1 linkenexus-1.xxx 8081
vim /etc/rinetd.conf
```

ANTCODE-antcodenode

antcodenode 有主备特性目前无法自动切换，未切换的情况下第一台容器为主，第二台容器为备，主容器宕机则无法正常提供服务，需要先确保数据安全（例如手动在 pod 对应的机器上备份 `/home/t4/data/repo` 的目录）再进行容器重启，在备容器正常的情况下也可以在备份数据后先将两台容器进行主备切换。

登录 LinkE 的 MySQL 实例查询 batmanshard 数据库的 Node 表数据，IP 字段值需要为IP 地址，和 antcodenode 容器 IP 地址对应。

```
select * from node;
```

new-antstack-plus.. batmanshard

```
1  SELECT * FROM node;
```

Message　Result 1　Status

	created	updated	node_type	pod_id	hostname	ip	status
ode	2021-03-17 20:22:33	2021-03-17 20:22:33	0	1	asp-antcode-antcodenode-0.antcode-antcodenode-svc.asp-antcode	11.166.180.172	1
ode	2021-03-17 20:22:33	2021-03-17 20:22:33	2	1	asp-antcode-antcodenode-1.antcode-antcodenode-svc.asp-antcode	100.88.227.82	1

有两条 status=1 的记录，一条对应容器 `antcodenode-0 node_type` ，数值为 0；另一条对应容器 `antcodenode-1 node_type` ，数值为 2，主备切换时将 0 和 2 的 `node_type` 进行互换即可。

LINKEBASE-linkuredis & ANTCODE.antcoderedis

暂不支持自动切换主备，如果确认 Redis 访问异常（默认情况下第一台为主，第二台为备），切换主备时有以下两种方案：

- 进入容器修改 `/data/rinetd.conf` ，将目标地址修改为正常的备容器。
- 直接重启容器。

LINKECI-acijenkinscloud

暂无主备切换方案。

其他应用

无主备关系，如有异常可下线异常机器，SLB 会自动切流。

5.4. 数据备份

需要备份的数据文件夹

代码仓库数据存储

涉及容器：Antcodeode

路径：/home/git/repo

图示：

```
[root@passdev-antcode-antcodenode-0 /home/git/repo]
#pwd
/home/git/repo

[root@passdev-antcode-antcodenode-0 /home/git/repo]
#ll
total 116
drwxr-x---    3 git git  4096 Feb 14 10:03 alipaycn
drwxr-x--- 1312 git git 69632 May  5 16:16 antcloud
drwxr-x---    5 git git  4096 Feb 14 10:07 antftroc
drwxr-x---    3 git git  4096 Jan 19 16:11 default
drwxr-x---    3 git git  4096 Feb 14 10:10 linketest
drwx------    2 git git 16384 Jan  4 16:28 lost+found
drwxr-x---   11 git git  4096 Apr 11 17:50 passdev
drwxr-x---    4 git git  4096 Mar 15 15:28 public
drwxr-x---    3 git git  4096 Feb 18 17:04 superstar
```

MongoDB 数据存储

涉及容器：linkemongo

路径：/data/db

图示：

```
root@passdev-linkebase-linkemongo-0:/data/db# pwd
/data/db
root@passdev-linkebase-linkemongo-0:/data/db# ls
WiredTiger                                collection-560--2295297756984718516.wt
WiredTiger.lock                           collection-562--2295297756984718516.wt
WiredTiger.turtle                         collection-564--2295297756984718516.wt
WiredTiger.wt                             collection-567--2295297756984718516.wt
WiredTigerLAS.wt                          collection-569--2295297756984718516.wt
_mdb_catalog.wt                           collection-572--2295297756984718516.wt
collection-0--2295297756984718516.wt      collection-574--2295297756984718516.wt
collection-0--8753282369845961205.wt      collection-576--2295297756984718516.wt
collection-104--2295297756984718516.wt    collection-578--2295297756984718516.wt
collection-107--2295297756984718516.wt    collection-580--2295297756984718516.wt
```

不需要备份的数据文件夹

普通应用

没有挂载持久化存储郑，无数据备份需求。

redis

LinkE 产品内置了一个 redis 应用（linkuredis / antcoderedis），但其为缓存数据，丢失后可重建。

5.5. 网络异常

主要是网络问题，检查 SLB 和容器的服务监听端口。

- 若容器端口正常，则检查 SLB 是否挂载后端服务器。
- 若容器可登录但端口不正常，则检查容器是否宕机，并对容器进行重启。
- 若容器无法登录则检查对于的物理机器是否宕机（如 oom），进行机器重启，在对有状态应用进行重启时需要确保持久化数据的安全，必要时先进行备份或者确认已有备份数据。

5.6. 数据容灾

有持久化存储的应用

LINKEBASE-linkemongo

数据备份

数据所在的宿主机磁盘（宿主机数据目录 `/home/t4/data/mongo` ）做快照或者其他类似备份策略。

数据迁移/恢复

- 如果是数据故障或者需要恢复某一天的数据，由于持久化数据落在底座机器的磁盘上，基于底座的磁盘备份能力来做数据容灾。
  - 对于阿里云底座，会要求部署时设置每天凌晨 3:00 的数据磁盘快照（保留 30 天）的备份策略。
  - 对于非阿里云底座，目前需要按照底座机器数据盘备份策略来设置，需要和现场交付团队确认。
- 对于备份的数据盘，如果需要恢复数据，需要先停服，并下线所有容器。

- 对于阿里云底座，基于当天的快照创建一块云盘，然后将此磁盘再挂载到容器对应的 ECS 上（路径不冲突，比如恢复盘挂载到 `/hom2/t4/data/mongo_bak1` ），然后将路径下的数据复制到容器 `/home/t4/data/mongo` ，但是复制前需要对当前的 `/home/t4/data/mongo` 数据进行 CP 备份，避免冲刷掉当前数据后无法恢复。
- 对于非阿里云底座，依照底座机器的备份策略来回拷数据。

- 三台容器依次操作完当前数据备份和数据回拷后，上线容器，然后进入容器中执行 `mongo --host mongomaster_slb --port 27018/27017 -u linke -p LinkE2017 --authenticationDatabase admin` ，再执行 `show dbs` 查看是否存在 bahamut、aclinkelib、aci 以及 LinkA 的数据集。然后登录 LinkE 控制台查看是否符合预期。

LINKEBASE-linkuredis & ANTCODE-antcoderedis

数据备份

数据所在的宿主机磁盘（宿主机数据目录 `/home/t4/data/mongo` ）做快照或者其他类似备份策略。

数据迁移/恢复

- 如果是数据故障或者需要恢复某一天的数据，由于持久化数据落在底座机器的磁盘上，基于底座的磁盘备份能力来做数据容灾。
  - 对于阿里云底座，会要求部署时设置每天凌晨 3:00 的数据磁盘快照（保留 30 天）的备份策略。
  - 对于非阿里云底座，目前需要按照底座机器数据盘备份策略来设置，需要和现场交付团队确认。
- 对于备份的数据盘，如果需要恢复数据，需要先停服，下线所有容器：
  - 对于阿里云底座，基于当天的快照创建一块云盘，然后将此磁盘再挂载到容器对应的 ECS 上（路径不冲突，比如恢复盘挂载到 `/hom2/t4/data/mongo_bak1` ），然后将路径下的数据复制到容器 `/home/t4/data/mongo` ，但是复制前需要对当前的 `/home/t4/data/mongo` 数据进行 CP 备份，避免冲刷掉当前数据后无法恢复。
  - 对于非阿里云底座，依照底座机器的备份策略来回拷数据。
- 三台容器依次操作完当前数据备份和数据回拷后，上线容器，然后进入容器中执行 `mongo --host mongomaster_slb --port 27018/27017 -u linke -p LinkE2017 --authenticationDatabase admin` ，再执行 `show dbs` 查看是否存在 bahamut、aclinkelib、aci 以及 LinkA 的数据集。然后登录 LinkE 控制台查看是否符合预期。

LINKEBASE-linkenexus

数据备份

数据所在的宿主机磁盘（宿主机数据目录 `/home/t4/nexus-data` ）做快照或者其他类似备份策略。

数据迁移/恢复

数据容灾基本同 LINKEBASE-linkemongo（也需要停服操作），不同的是 Nexus 数据所在路径为 `/home/t4/nexus-data` ，重启容器后登录 Nexus 控制台查看是否符合预期。

ANTCODE-antcodenode

数据备份

数据所在的宿主机磁盘（宿主机数据目录 `/home/t4/data/repo` ）做快照或者其他类似备份策略。

数据迁移/恢复

数据容灾基本同 LINKEBASE-linkemongo（也需要停服操作），不同的是：

- 如果备容器未出现异常，检查容器内 `/home/git/repo` 目录下的文件是否符合预期（与主节点文件目录不一致时基本应该是未受损的），如果符合预期则可以基于该备节点直接拷贝数据来恢复异常前的数据，如果是采用备份磁盘来恢复数据的话，请务必先备份容器内当前的数据目录，以免无法恢复当前数据。
- antcodenode 数据所在路径为 `/home/t4/data/repo` ，重启容器后登录 LinkE 控制台查看代码服务页面是否符合预期。
- antcodenode 的磁盘数据是按照租户以及群组（命名空间）、代码库级别隔离的，不同的代码库对于 `/home/t4/data/repo` 下租户 ID 目录下 `<group_name>/<repository_name>.git` ，可以只针对目标代码库或者群组进行数据恢复，按需进行。

5.7. 容器宕机

LINKEBASE-linkemongo

宕机情况下还有一台容器存活则不会影响线上服务，重启宕机容器即可。

LINKEBASE-linkuredis & ANTCODE-antcoderedis

宕机情况下还有一台容器存活则不会影响线上服务，重启宕机容器即可。

LINKEBASE-linkenexus

> 说明
>
> linkenexus 有主备特性目前无法自动切换。

在未切换的情况下第一台容器为主，第二台容器为备，主容器宕机则无法正常提供服务，需要先确保数据安全（如手动在 pod 对应的机器上备份 `/home/t4/nexus-data` 的目录）再进行容器重启，在备容器正常的情况下也可以在备份数据后先将两台容器内的 rinetd 服务转发地址切换到备容器，命令如下：

```
# 修改 rinetd 中转发的目标地址到被容器：8080 127.0.0.1 linkenexus-1.xxx 8081
vim /etc/rinetd.conf
```

ANTCODE-antcodenode

在 1.32 及以后的版本中，antcodenode 有主备故障监控和切换能力，未切换的情况下第一台容器为主，第二台容器为备，主容器宕机则无法正常提供服务，如果手动切换主备先确保数据安全（如手动在 pod 对应的机器上备份 `/home/t4/data/repo` 的目录）再进行容器重启，在备容器正常的情况下也可以在备份数据后先将两台容器进行主备切换，操作如下：

登录 LinkE 的 mysql 实例查询 batmanshard 数据库的 node 表数据。

> 说明
>
> IP 字段值需要和 antcodenode 容器 IP 值对应。

```
select * from node;
```

new-antstack-plus.　batmanshard

```
1  SELECT * FROM node;
```

Message　Result 1　Status

	created	updated	node_type	pod_id	hostname	ip	status
ode	2021-03-17 20:22:33	2021-03-17 20:22:33	0	1	asp-antcode-antcodenode-0.antcode-antcodenode-svc.asp-antcode	11.166.180.172	1
ode	2021-03-17 20:22:33	2021-03-17 20:22:33	2	1	asp-antcode-antcodenode-1.antcode-antcodenode-svc.asp-antcode	100.88.227.82	1

有两条 status=1 的记录，一条对应容器 `antcodenode-0 node_type` 应为 0 ，另一条对应容器 `antcodenode-1 node_type` 应为 2，主备切换时将 0 和 2 的 `node_type` 进行互换即可。

5.8. 断电恢复

依据云游重启即可。

> **说明**
>
> 重启有状态应用时确认对数据无影响。

6.附录：基础术语

AntFinancial Kubernetes Engine（AKE）

AKE（AntFinancial Kubernetes Engine）是在开源 Kubernetes 和 Docker 基础上蚂蚁自研的容器云平台，通过蚂蚁自研组件进行了能力增强。对外提供统一的计算、存储、网络资源抽象，屏蔽物理机和虚拟机等各种环境的差异。

云游（Yunyou）

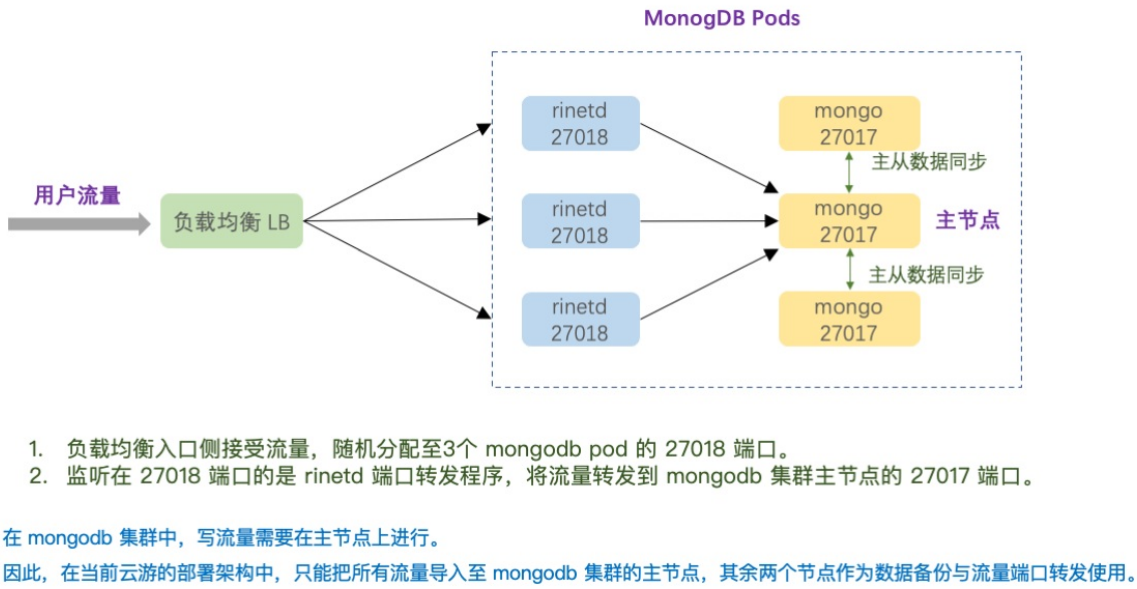
云游是一个 web 应用，主要提供的功能包括 AKE 集群管理（创建集群、节点管理、网络管理、容器管理等），AntStack 产品和 AKE 集群管控的中台。

7.LinkE 自带数据库运维

7.1. MongoDB 集群

MongoDB 集群架构

LinkE 默认在云游中创建 3 个 Pod 组成一个集群（非社区版本的 mongo operator 方案），其架构如下：



集群名称为 `rs_linke（replicaSet = rs_linke）` 。

架构特点

- 负载均衡入口侧接受流量，随机分配至 3 个 mongodb pod 的 27018 端口。
- 监听在 27018 端口的是 rinetd 端口转发程序，将流量转发到 mongodb 集群主节点的 27017 端口。

设计理念

在 mongodb 集群中，写流量需要在主节点上进行，而云游创建的 LB 只能随机分配流量。

因此，在当前云游的部署架构中，只能把所有流量导入至 mongodb 集群的主节点，其余两个节点作为数据备份与流量端口转发使用。

常见运维操作

手动加入从节点

应用场景

mongodb 在部署时没有一台一台升级，或者因内存限制而重启。此时集群状态无法自动同步，需要人工介入，重新手动组建集群。

操作步骤

> **说明**
>
> 假设 mongo0、mongo1 仍维持集群，其中 mongo0 是主节点，mongo2 掉线。

1. **在 mongo2 停止 mongodb 进程。**

   可以使用以下三种方案依次尝试：

   i. 在 mongo2 pod 可登录 mongo 控制台时，使用如下命令停服。

   ```
   use admin; db.shutdownServer();
   ```

   ii. 使用 mongod --shutdown 命令。

   ```
   mongod  --shutdown  --dbpath /data/db
   ```

   iii. 使用 Kill 命令。

2. **备份 mongo2 原 `/data/db` 数据。**

   使用 webshell 登录 mongod2 pod。

   ```
   cd /data
   mkdir backup

   # 备份原存储数据至 backup 文件夹
   mv db configdb backup
   # 若 mv 命令无法继续，使用 cp -r 命令

   # 删除 /data 文件夹下除 backup 之外的文件
   rm -rf /data/mongo.log /data/mongo.log* /data/*.pid
   # rm -rf /data/db /data/configdb

   # 创建新的数据库文件夹
   mkdir -p /data/db /data/configdb

   # 把 data 与 configdb 这两个文件夹赋予 mongodb 用户权限
   chown -R mongodb:mongodb /data/db /data/configdb
   ```

   > **重要**
   >
   > 需要确定文件夹权限，否则 mongodb 会因权限问题启动失败。

```
# ls -al
total 180804
drwxr-xr-x 5 root    root         4096 Mar  7 22:36 .
drwxr-xr-x 1 root    root         4096 Mar  7 22:35 ..
drwxr-xr-x 2 mongodb mongodb      4096 Mar  7 18:37 configdb
drwxr-xr-x 4 mongodb mongodb     36864 Mar 21 21:32 db
drwx------ 2 root    root        16384 Mar  7 18:37 lost+found
-rw-r--r-- 1 root    root    184764502 Mar 21 21:33 mongo.log
-rw-r--r-- 1 root    root         4698 Mar  7 18:39 mongo.log.2022-03-07T10-39-28
-rw-r--r-- 1 root    root        14783 Mar  7 18:52 mongo.log.2022-03-07T10-55-05
-rw-r--r-- 1 root    root       269553 Mar  7 22:35 mongo.log.2022-03-07T14-36-46
-rw-r--r-- 1 root    root            3 Mar  7 18:39 pid
-rw-r--r-- 1 root    root           88 Mar  7 22:40 rinetd.conf
#
```

3. **在 mongo2 中启动新的 mongodb 进程。**

```
mongod --dbpath /data/db --keyFile /etc/keyFile --replSet rs_linke --logpath /data/mongo.log --bind_ip 0.0.0.0 --fork
```

> **说明**
>
> 启动后使用 `ps -ef | grep mongo` 查看进程状态。若启动失败，查看 `/data/mongo.log` 日志文件。

4. **换到 mongo0 节点添加从节点（mongodb 集群现有主节点）。**

   执行 mongo 命令，登录控制台。

```
// 认证
use admin
p="LinkE2017"
db.auth("linke", p)

// 查看集群状态
rs.status()

// 假设三台节点的 host 为
mongo_host0="asp-linkebase-linkemongo-0.linkebase-linkemongo-svc.asp-linkebase.svc.cluster.local:27017"
mongo_host1="asp-linkebase-linkemongo-1.linkebase-linkemongo-svc.asp-linkebase.svc.cluster.local:27017"
mongo_host2="asp-linkebase-linkemongo-2.linkebase-linkemongo-svc.asp-linkebase.svc.cluster.local:27017"

// 先手动移除问题节点
rs.remove(mongo_host2)

// 再手动增加当前新节点
rs.add(mongo_host2)
```

5. **查看集群状态。**

> **说明**
>
> 此步骤大约需要等待 30s。

在 mongo0/2 节点上执行如下命令：

```
// 认证
use admin
p="LinkE2017"
db.auth("linke", p)

// 查看集群状态
rs.status()
```

完成上述操作后，您可以通过以下操作来验证手动组建集群是否符合预期。

- 端口转发程序配置文件 `/data/rinetd.conf` 是否符合预期。

  在 mongo pod 中，主进程是一个 shell 程序，每隔 30 秒检查集群状态，若主节点切换，则更新此配置文件并重启 rinetd 程序应用新配置。

  此外，也需要在 pod 中输命令 `mongo --port 27018` 看看是否可以连接到主节点上。
- 在新加入的从节点中，执行命令 `ls -l /data/db`，查看数据文件是否开始同步。

新组建集群方案

应用场景

一般适用于使用 mongodump 全量数据后重建数据库场景。

操作步骤

1. **创建 mongodb 数据库集群管理员。**
   i. 选择一个 mongo pod 新启动的 mongod 进程。
   ii. 创建 LinkE 账户并设置为 mongodb 数据库集群管理员。

   > **说明**
   >
   > - 必须先备份并移出当前 pod 中 `/data` 文件夹上先前的 mongodb 数据文件，否则执行不成功。
   > - 第一次新节点的启动，可不指定 `replSet = rs_linke` 。

iii. 执行 mongo 命令，登录控制台。

> **重要**
>
> 由于新启动的节点是一台全新节点，账号与密码都未设置，而旧节点都已初始化完成，所以需要结合实际情况操作命令。

```
// 创建 linke 用户
use admin
p="LinkE2017"
db.createUser(
 {
   user: "linke",
   pwd: p,
   roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
 }
)

// 如果 linke 用户存在，直接认证
/*
use admin
p="LinkE2017"
db.auth("linke", p)
 */

// 为 linke 用户赋予 mongodb 集群管理权限
target_db = "admin"
db.grantRolesToUser(
  "linke",
  [ { role: "userAdminAnyDatabase", db: target_db },
   { role: "dbAdminAnyDatabase", db: target_db },
   { role: "readWriteAnyDatabase", db: target_db },
      { role: "clusterAdmin", db: target_db } ]
)
```

2. 手动初始化新集群。

假设全新节点为 mongo0，执行如下命令：

```
// 认证
use admin
p="LinkE2017"
db.auth("linke", p)

// 查看集群状态
rs.status()
rs.isMaster()

// 假设三台节点的 host 为
mongo_host0="asp-linkebase-linkemongo-0.linkebase-linkemongo-svc.asp-linkebase.svc.cluster.local:27017"
mongo_host1="asp-linkebase-linkemongo-1.linkebase-linkemongo-svc.asp-linkebase.svc.cluster.local:27017"
mongo_host2="asp-linkebase-linkemongo-2.linkebase-linkemongo-svc.asp-linkebase.svc.cluster.local:27017"

/*
rs_config={ _id: 'rs_linke', members: [
      {"_id": 0, "host": mongo_host0 },
      {"_id": 1, "host": mongo_host1 },
      {"_id": 2, "host": mongo_host2 }]
}
 */

// 第一次初始化，集群则只有一个节点，指定 replica_set = rs_linke
rs_config={ _id: 'rs_linke', members: [
      {"_id": 0, "host": mongo_host0 }]
}
rs.initiate(rs_config)

// 若 initiate 失败，则还可以使用 reconfig 方法
// rs.reconfig(rs_config, {"force" : true})
```

3. **在其他 mongo 节点上启动 mongodb 服务。**

   手动在 mongo0 上添加副本集节点，执行命令如下：

```
// 手动增加/移除节点
rs.add(mongo_hostx)
rs.remove(mongo_hostx)

// 其他操作备忘
/*
// 在 Primary 节点上设置节点权重
cfg = rs.conf()
// cfg.members[0].host = mongo_host0
cfg.members[1].priority = 5
rs.reconfig(cfg)
rs.reconfig(cfg, {"force" : true})
 */
```

4. **使用 mongorestore 导入先前备份的数据**

具体操作请参考官方 mongodb 数据库运维工具集的 使用文档。

设置 mongodb 的内存缓存大小

应用场景

通常应用于 mongodb 数据库节点停止后重启不成功的场景。

> **说明**
>
> 具体现象是重启几分钟后被停止，然后又重启，循环往复。 查看 `/data/mongo.log` 日志，显示停止的原因是 cannot allocate memory，即无法分配新内存。

该现象的原因是 mongodb 启动时没指定缓存设置参数，cachesize 默认限制大小如下图所示，具体请参考 MongoDB 官方 FAQ。

> Starting in MongoDB 3.4, the default WiredTiger internal cache size is the larger of either:
>
> - 50% of (RAM - 1 GB), or
> - 256 MB.
>
> For example, on a system with a total of 4GB of RAM the WiredTiger cache will use 1.5GB of RAM (`0.5 * (4 GB - 1 GB) = 1.5 GB`). Conversely, a system with a total of 1.25 GB of RAM will allocate 256 MB to the WiredTiger cache because that is more than half of the total RAM minus one gigabyte (`0.5 * (1.25 GB - 1 GB) = 128 MB < 256 MB`).

而 LinkE 自带的 mongodb 以容器化形式部署，而容器技术其实就相当于宿主机上运行的一个进程，mongodb 程序探测到的内存大小实际是宿主机内存。

> **说明**
>
> 假设宿主机内存为 64G内存，那么 cacheSize 被自动设置成 30G。 如果磁盘中存储的数据文件过多（超过 pod memory limit 大小），受 mongodb 的 mmap 加载机制影响，极容易在初始化全量数据时触发 memory limit 上限，然后被告知为 cannot allocate memory。

操作步骤

启动参数增加 cachesize 设置。

- 如果 Pod 的内存规格为 4G，则可设置 cacheSize 为 1.5G。

```
mongod --wiredTigerCacheSizeGB 1.5 --dbpath /data/db --keyFile /etc/keyFile --replSet rs_linke --lo
gpath /data/mongo.log --bind_ip 0.0.0.0 --fork
```

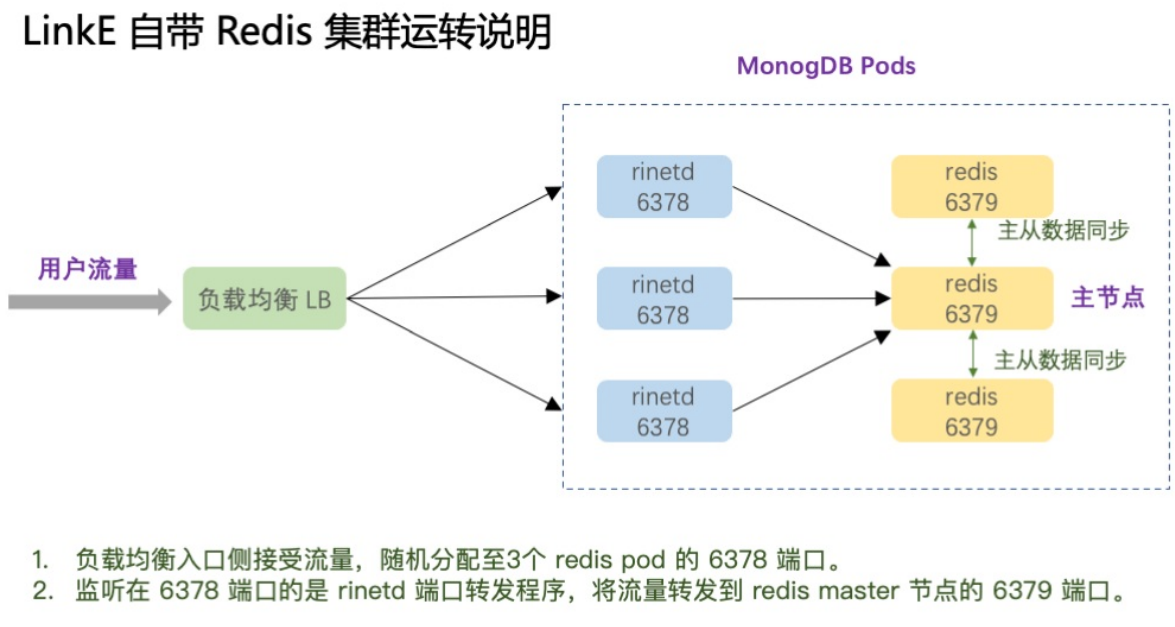
- 如果 Pod 的内存规格增加到 12G，则可设置 cacheSize 为 5G。

```
mongod --wiredTigerCacheSizeGB 5 --dbpath /data/db --keyFile /etc/keyFile --replSet rs_linke --log
path /data/mongo.log --bind_ip 0.0.0.0 --fork
```

7.2. Redis 主从运维架构

Redis 服务架构

LinkE 默认在云游中创建 2~3个 Pod 组成一个集群（非社区版本的 redis operator 方案），其架构如下（与 mongodb 集群方案类似）：



在 LinkE 的部署架构中，LinkEBase 产品与 Antcode 产品均自带了一组 Redis 应用服务，分别为 LinkU（用户管理模块）和 antcodeweb / batmannode （代码服务模块）提供数据缓存服务。

架构特点

- 负载均衡入口侧接受流量，随机分配至每个 redis pod 的 6378 端口。
- 监听在 6378 端口的是 rinetd 端口转发程序，将流量转发到 redis master 节点的 6379 端口。

设计理念

在 Redis 集群中，写流量需要在主节点上进行，而云游 LB 仅能支持将流量均分到每个 Pod 上。

因此，在当前云游的部署架构中，采取的是通过 TCP 端口转发手段，将所有流量导入至 redis master 节点，其余 slave 节点作为数据备份使用。

常见运维操作

Redis 运行架构说明

> **说明**
>
> - 假设 redis pods 存在两个，一主一从，redis0 pod 是 master 节点，redis1 pod 是 slave 节点。
> - 目录 `/data/` 是 pod 挂载的外部存储资源。

默认在一个 redis pod 下启动两个 redis server 实例和一个 redis 哨兵，如下图所示：

```
[root@saas-linkebase-linkuredis-0 /tmp]
#ps -ef | grep redis
root           9       1  0 Mar07 ?        00:13:15 redis-server 0.0.0.0:7788
root          94       1  0 Mar07 ?        00:18:55 redis-server 0.0.0.0:6379
root         115       1  0 Mar07 ?        00:22:13 redis-sentinel *:26379 [sentinel]
root      120628 120466  0 17:40 pts/1    00:00:00 grep --color=auto redis
```

三个 redis 进程可分为两组：

- **redis-7788**：辅助 redis server，用以两个 Pod 之间的临时数据同步，宕机后也不影响主线功能。
- **redis-6378**：主 redis server，配合哨兵协同工作。

辅助 redis-7788 实例使用的配置文件为 `/tmp/redis.conf`，非对外提供服务的主进程。

```
[root@saas-linkebase-linkuredis-0 /tmp]
#ll
total 16
-rw-r--r-- 1 root root 202 Mar  7 22:35 appendonly.aof
-rw-r--r-- 1 root root 171 Mar  7 23:35 dump.rdb
-rw-r--r-- 1 root root 152 Mar  7 22:35 redis.conf
-rw-r--r-- 1 root root 978 Mar  7 23:35 redis.log
-rw-r--r-- 1 root root   0 Jun 11  2019 test.lock

[root@saas-linkebase-linkuredis-0 /tmp]
#cat redis.conf
bind 0.0.0.0
daemonize yes
logfile "/tmp/redis.log"
dir /tmp
masterauth 123456
requirepass 123456
protected-mode yes
appendonly yes
slave-read-only yes
```

辅助 redis-server
监听 7788 端口

手动配置 master / slave 节点

> **说明**
>
> 此操作主要聚焦于 redis-6378 服务实例，在启动前，需要将该实例的配置文件 `/data/redis/redis.conf` 复制到 `/usr/local/redis/redis_slave.conf` 文件下。

将新节点手动加入 redis 集群前，应当先确认以下几个配置文件是否符合预期。

> **说明**
>
> 最主要的是确认 master 节点的 IP 地址是否正确。

- redis master 节点启动配置文件

```
[root@saas-linkebase-linkuredis-0 /root]
#cat /usr/local/redis/redis_slave.conf
bind 0.0.0.0
daemonize yes
logfile "/data/redis/redis.log"
dir /data/redis
replicaof 100.83.19.218 6379
masterauth 123456
requirepass 123456
protected-mode yes
appendonly yes
slave-read-only yes

[root@saas-linkebase-linkuredis-0 /root]
#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
41: eth0@if42: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:12:34:0a:8e:46 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 100.83.19.218/24 brd 100.83.19.255 scope global eth0
       valid_lft forever preferred_lft forever
```

redis master 节点: 100.83.19.218
redis slave 节点: 100.89.19.215

- redis slave 节点启动配置文件

```
[root@saas-linkebase-linkuredis-1 /]
#cat /usr/local/redis/redis_slave.conf
bind 0.0.0.0
daemonize yes
logfile "/data/redis/redis.log"
dir /data/redis
replicaof 100.83.19.218 6379
masterauth 123456
requirepass 123456
protected-mode yes
appendonly yes
slave-read-only yes

[root@saas-linkebase-linkuredis-1 /]
#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
73: eth0@if74: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:12:34:6a:1e:a3 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 100.83.19.215/24 brd 100.83.19.255 scope global eth0
       valid_lft forever preferred_lft forever
```

redis master 节点: 100.83.19.218
redis slave 节点: 100.89.19.215

- redis master 节点的 sentinel 哨兵进程启动配置文件

```
[root@saas-linkebase-linkuredis-0 /root]
#cat /data/redis/      如果要重置 redis，此文件夹下除 config 文件的数据要删除。
appendonly.aof     last_hosts         redis.conf         redis_name         sentinel.conf      sentinel.log
dump.rdb           password_last.txt  redis.log          sentinel/          sentinel.conf.tmp

[root@saas-linkebase-linkuredis-0 /root]
#cat /data/redis/sentinel.conf
daemonize yes                                   redis master 节点: 100.83.19.218
logfile "/data/redis/sentinel.log"              redis slave 节点: 100.89.19.215
dir "/data/redis/sentinel"
sentinel monitor mymaster_100.83.19.218 100.83.19.218 6379 1
sentinel auth-pass mymaster_100.83.19.218 ali88

# Generated by CONFIG REWRITE
protected-mode no
pidfile "/var/run/redis.pid"
port 26379
user default on nopass ~* &* +@all
sentinel myid 5d236ddc7078ac63d28b71f8c7628750239d38c3 <------ redis sentinel id 需要匹配
sentinel config-epoch mymaster_100.83.19.218 0
sentinel leader-epoch mymaster_100.83.19.218 0
sentinel current-epoch 0
sentinel known-replica mymaster_100.83.19.218 100.83.19.215 6379
sentinel known-sentinel mymaster_100.83.19.218 100.83.19.215 26379 8b461bd86ac6713f9a1c58fec012b67d2d6ede9a
```

- redis slave 节点的 sentinel 哨兵进程启动配置文件

```
[root@saas-linkebase-linkuredis-1 /]
#cat /data/redis/sentinel.conf
daemonize yes                                   redis master 节点: 100.83.19.218
logfile "/data/redis/sentinel.log"              redis slave 节点: 100.89.19.215
dir "/data/redis/sentinel"
sentinel monitor mymaster_100.83.19.218 100.83.19.218 6379 1
sentinel auth-pass mymaster_100.83.19.218 ali88

# Generated by CONFIG REWRITE
protected-mode no
pidfile "/var/run/redis.pid"
port 26379
user default on nopass ~* &* +@all
sentinel myid 8b461bd86ac6713f9a1c58fec012b67d2d6ede9a <------ redis sentinel id 需要匹配
sentinel config-epoch mymaster_100.83.19.218 0
sentinel leader-epoch mymaster_100.83.19.218 0
sentinel current-epoch 0
sentinel known-replica mymaster_100.83.19.218 100.83.19.215 6379
sentinel known-sentinel mymaster_100.83.19.218 100.83.19.218 26379 5d236ddc7078ac63d28b71f8c7628750239d38c3
```

确认无误后，在新 Pod 节点输入以下命令组建 redis 主从节点。

```
redis-server /data/redis/redis.conf
redis-sentinel /data/redis/sentinel.conf
```

> **说明**
>
> 辅助 redis-server 实例的启动方式如下：`redis-server /tmp/redis.conf --port 7788` 可选启动，主要用于保存 redis 重启前的 host 信息，即 `last_hosts` 。

完成后，新 Pod 的 redis-6379 实例会变为从节点，在主节点的 redis 控制台上输入 `info Replication` 查看主从节点状态是否符合预期，如下图所示：

```
#redis-cli -a ali88
Warning: Using a password with '-a' or '-u' option on the command line
127.0.0.1:6379> info Replication
# Replication
role:master
connected_slaves:1
slave0:ip=100.88.140.14,port=6379,state=online,offset=182673880,lag=0
master_failover_state:no-failover
master_replid:f482094347d2f693bed03f4932d9594ecb435c7f
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:182673880
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:181625305
repl_backlog_histlen:1048576
```

redis-server 当前节点的状态

如果新节点的 redis 启动不成功，或者加入 master 节点不成功，则应当检查 `/data/redis/redis.log` 日志文件，必要时可先备份再清除 `/data/redis` 下 `appendonly.aof` 与 `dump.rdb` 两个文件，然后再启动 redis-server 实例。

> **说明**
>
> LinkE 主要使用 redis 进行数据缓存，因此 redis 内存数据丢失后，会从数据库中重新读取相关信息后重建缓存。

8.ANTCODE 代码仓库运维

8.1. ANTCODE 代码仓库高可用说明书

背景信息

适用版本

本文档基于 ANTCODE 1.43.0 版本进行介绍。

高可用能力介绍

- 默认的部署架构为双容器一主一备（或一主一从），代码库数据实时同步，备节点不提供客户端的流量读写，从节点可提供读服务不提供写服务，主备/主从节点可手动切换（此步骤需要操作数据库）。
- 数据备份基于数据磁盘的快照做日级别冷备，定时按天冷备份（底座磁盘有备份恢复能力则有，反之则无）

> **说明**
>
> 代码文件存储在 ntcodenode pod 的 `/home/git` 目录下，为持久化存储卷（PVC）。

原理介绍

代码存储节点的工作原理

如下图所示，标识为 antcodenode 的节点，是代码仓库的存储节点。初次部署后默认有两个，形成一组存储节点。

passdev-antcode-antcodenode-0 linke/privateantcode:6f3acc7712ca64938109f3a7baf420db4ce8f89f_20220308141218_u	11.166.198.103	11.166.80.141	FIN-PAAS-DEV(当前)	AntCode代码服务(AntCode) ANTCODE	antcodenode
passdev-antcode-antcodenode-1 linke/privateantcode:6f3acc7712ca64938109f3a7baf420db4ce8f89f_20220308141218_u	11.158.115.229	100.88.161.35	FIN-PAAS-DEV(当前)	AntCode代码服务(AntCode) ANTCODE	antcodenode
passdev-antcode-antcodeweb-0 linke/privateantcode:3d2d496409e6e43f269b20f068a1326ad9a49d05_20220424234731	100.88.181.115	11.166.80.141	FIN-PAAS-DEV(当前)	AntCode代码服务(AntCode) ANTCODE	antcodeweb
passdev-antcode-codecenter-0 linke/mycodecenter:master_517eb45d65e53d8860f82cc646f009c9eafeea69_20220421140558	11.166.197.45	11.166.80.141	FIN-PAAS-DEV(当前)	AntCode代码服务(AntCode) ANTCODE	codecenter
passdev-antcode-batmanproxy-0 linke/privateantcode:6f3acc7712ca64938109f3a7baf420db4ce8f89f_20220308141218_u	11.166.197.51	11.166.80.141	FIN-PAAS-DEV(当前)	AntCode代码服务(AntCode) ANTCODE	batmanproxy

名词解释

术语	解释
Node	是代码仓库的存储节点。node 的身份有主节点、从节点、备节点。
主节点	master，`node_type=0`，接受读写流量。
从节点	mirror，`node_type=1`，分流只读流量。
备节点	backup，`node_type=2`，数据备份使用，不承载读写流量。
Pod	该 Pod 不是 k8s pod，而是代表 node group，也就是一组 node。 说明 在商业版本 Antcode 中，默认只部署一主一备两个节点组成一组 nodes。
Shard	数据分片存储规则，目前只配置一条规则 default_pod。

Antcode 架构示意与节点寻址流程

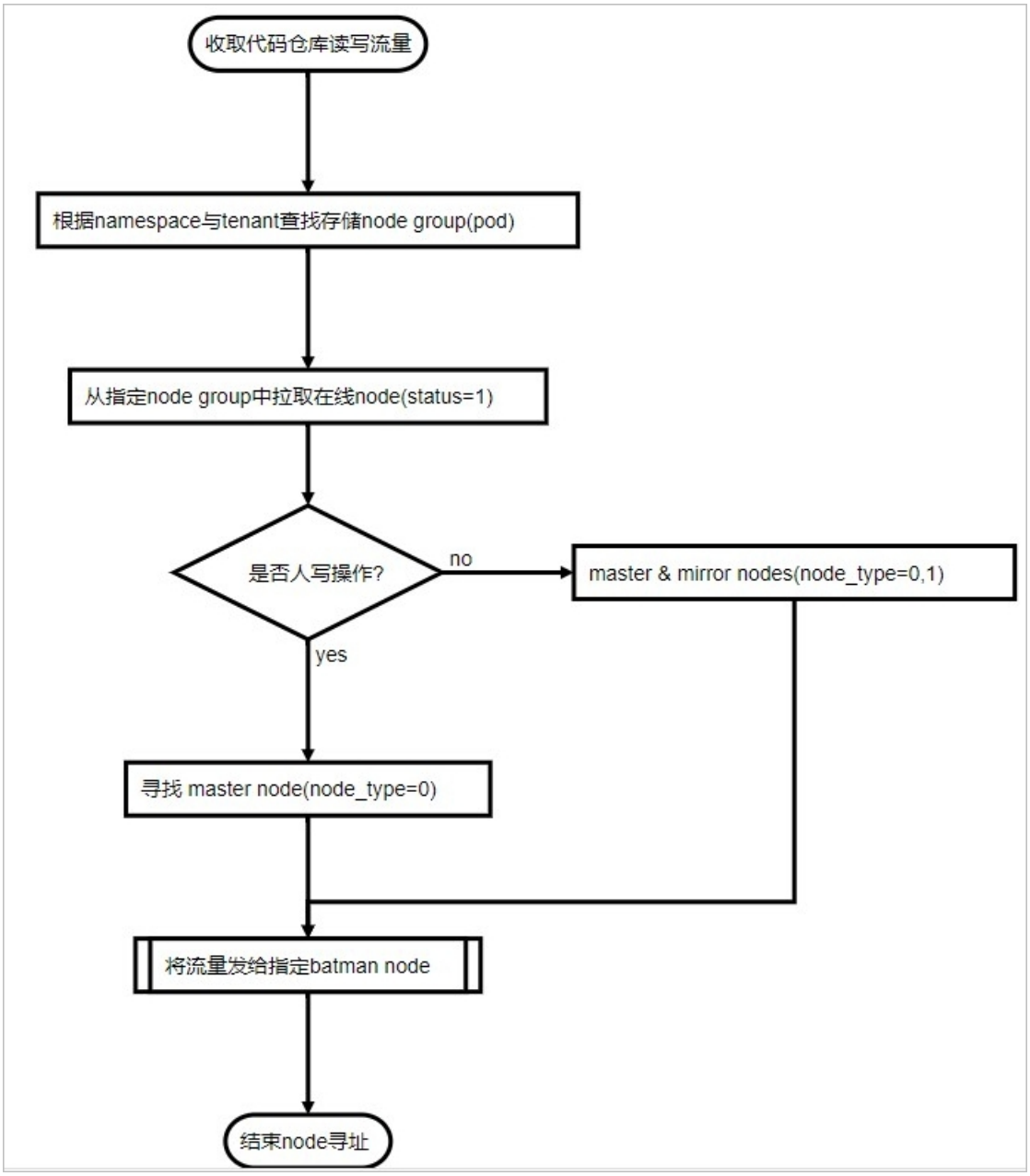
- Antcode 架构示意表

参数名称	说明
↓	
antcodeweb、batman-sshd、batman-httpd	流量入口，接受页面端 API 请求、GIT HTTP 与 SSH 请求，并且进行鉴权。
↓	
batman-proxy	○ proxy：Git 流量处理，统一处理 GIT HTTP 与 SSH 的代码读写流量。 ○ shard：Git 节点寻址，将流量分发到代码实际存储的节点。
↓	
batman-shard	

参数名称	说明
↓	
batman-node0、batman-node1（a node group - pod）	Git Server 代码仓库实际存储节点，可支持多组 node 存储。 说明 代码存储节点的扩容，实质上也只需要扩容 node 节点。

如上表所示，batman-proxy 与 batman-shard 进行代码读写流量的存储决策，即按照 tenant 和 namespace 维度（租户和代码群组）来分配流量，这就表示某一个租户名下某 namespace 中所有代码仓库均存储在一起。

- Antcode 节点寻址流程图



表数据示例

下图中每个代码群组的 namespace 所指向的存储 pod（注意此处 pod 是指 node group，不是 k8s pod 概念）的 pod_id 都为 1，代表所有仓库都存储在默认的 node group 里。

```
4 SELECT * FROM `namespace`
5 LIMIT 20;
```

执行历史　执行结果1 ×

导出文件　单行详情　开启编辑　部署周期任务　数据可视化　请输入　过滤

序号	id	namespace_path	code_namespace_id	created	updated	pod_id	region_id
1	1	jianbentest	1	2017-12-05 10:08:01	2017-12-05 10:08:01	1	0
2	2	wenchang	2	2017-12-05 10:34:06	2017-12-05 10:34:06	1	0
3	3	linkcode	3	2017-12-05 11:38:08	2017-12-05 11:38:08	1	0
4	4	grp001	4	2017-12-05 13:19:15	2017-12-05 13:19:15	1	0
5	5	LinkTest	5	2017-12-05 16:54:22	2017-12-05 16:54:22	1	0
6	6	muheng	6	2017-12-05 20:19:07	2017-12-05 20:19:07	1	0
7	7	hanxin.hx	7	2017-12-20 20:55:10	2017-12-20 20:55:10	1	0
8	8	group1220	8	2017-12-20 22:01:39	2017-12-20 22:01:39	1	0

下图是 pod_id = 1 的 node group，一主一备。

```
8 SELECT * FROM `node`
9 LIMIT 20;
```

执行历史　执行结果1 ×

导出文件　单行详情　开启编辑　部署周期任务　数据可视化

序号	id	description	created	updated	node_type
1	33	batman node master	2021-10-12 15:53:35	2022-03-29 21:43:00	0
2	34	batman node backup	2021-10-12 15:53:35	2021-10-12 15:53:35	2
3	35	pre backup	2021-10-12 16:11:14	2021-12-10 16:58:04	2
4	43	node	2021-10-12 15:53:35	2021-10-12 15:53:35	1

在 Antcode 寻址流程图 中，最关键的一点是第一步：**根据 namespace 与 tenant 查找存储 node group(pod)**。这是 batman 分片决策，默认只有一条路由 default_pod，如下图所示， 路由 default_pod 的 value 值为 1，代表默认所有新增的 namespace 下的代码均存储在 pod_id = 1 的 node_group 中。

```
6 SELECT * FROM `shard_config`
7 LIMIT 20;
```

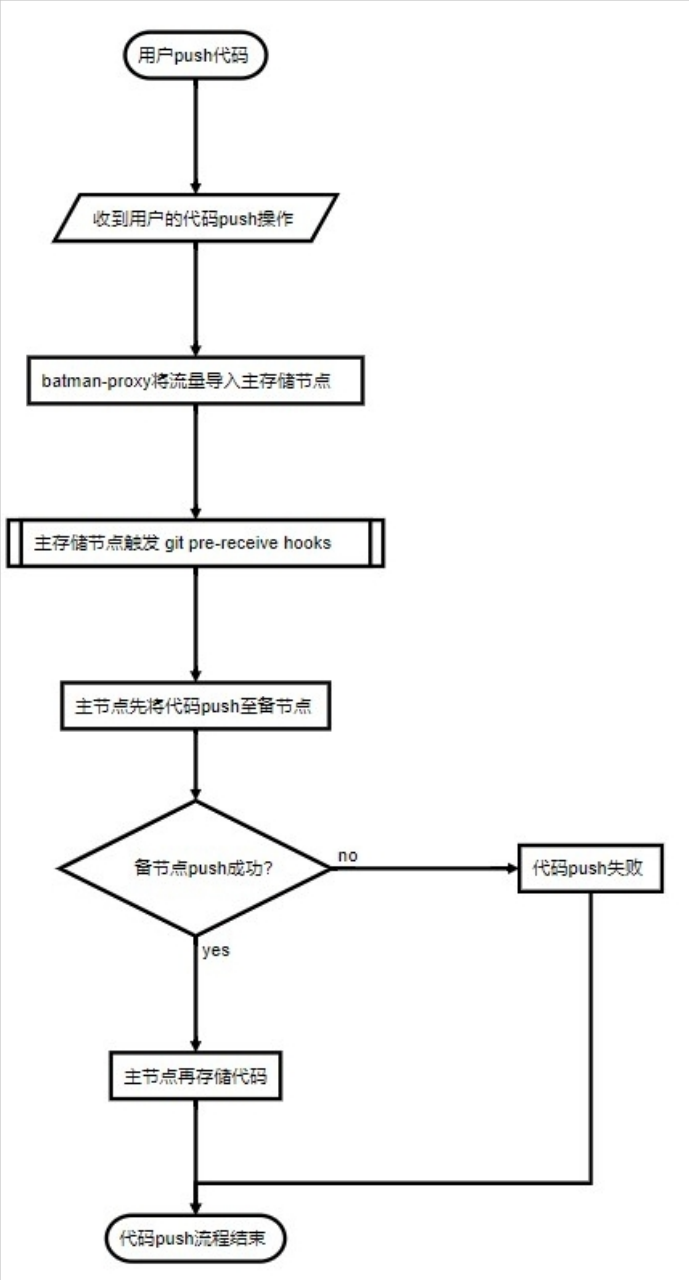
执行历史　执行结果1 ×

导出文件　单行详情　开启编辑　部署周期任务　数据可视化

序号	id	name	value	created
1	1	default_pod	1	2017-12-04 14:35:29

代码主从同步原理

基于 Git Server Hooks 机制的逻辑同步，而非基于磁盘数据拷贝的物理同步，运转机制如下图所示。



8.2. 代码仓库存储节点宕机的处理方式

本文将通过假设 a1、b1 两个实例，详细介绍处理节点宕机的具体操作步骤。

假设代码仓库存储服务（antcodenode）有两个 pod，其中 a1 实例部署在 x1 节点上，b1 实例部署上 y1 节点上。

如果 x1 节点宕机，a1 实例会宕机，处理步骤如下：

步骤一：确认 a1 存储实例的角色

查询 batmanshard 数据库的 node 表。

> 说明
>
> - SQL： `select * from node`
> - 关键列：pod_id、node_type、status。

id	description	created	updated	node_type	pod_id	hostname	ip	status
1	batman node	2022-01-04 08:29:48	2022-01-04 08:29:48	2	2	passdev-antcode-antcodenode-1.antcode-antcodenode-svc.passdev-antcode	11.158.115.229	1
2	batman node	2022-01-04 08:29:51	2022-01-04 08:29:51	0	2	passdev-antcode-antcodenode-0.antcode-antcodenode-svc.passdev-antcode	11.166.198.103	1
3	batman node	2022-04-25 09:24:16	2022-04-25 09:24:16	0	3	passdev-antcode-antcodenode-0.antcode-antcodenode-svc.passdev-antcode	11.166.198.103	-1
4	batman node	2022-04-25 09:28:29	2022-04-25 09:28:29	0	3	passdev-antcode-antcodenode-0.antcode-antcodenode-svc.passdev-antcode	11.166.198.103	-1

如上图所示：

- pod_id 相同的 node 为一组存储节点，通常为两个。

  一般情况下，a1/b1 属于同一组代码存储 node，所以它们的 pod_id 是相同的。
- status=1 的 node 是上线状态，其他则是下线状态。

  如上图所示，显示有一组 node 已下线，属于先前部署遗留的脏数据。
- node_type 标明了 node 在这一组存储节点的角色。
  - 主节点：master， `node_type=0` ，接受读写流量。
  - 从节点：mirror， `node_type=1` ，分流只读流量。
  - 备节点：backup， `node_type=2` ，数据备份使用，不承载读写流量。

根据 ip / hostname 信息，通过 a1 实例的 node_type 值判断其是不是主节点。

- 如果不是主节点，则不需要处理，等待宕机服务器恢复，a1 实例重启即可。
- 如果宕机的 a1 实例是存储主节点（node_type=0），需要订正数据库，将备节点切换成主节点。

步骤二：将备节点切换成主节点

目前主节点 a1 已宕机，此时需要将原备节点 b1 切换成主节点。

在数据库中进行以下操作：

1. 将 a1 的 node_type 修改为 2。
2. 将 b1 的 node_type 修改为 0。

> 说明
>
> SQL： `update node set node_type = 0 where id = xxx`

待 a1 恢复后，它将以备节点运转。

步骤三：增加节点

如果服务器 x1 不可恢复，需要增加节点，在云游上对 antcodenode 进行扩容。

假设新扩容实例 c1 运行在 z1 服务器上，c1 启动后会自动在数据库 batmanshard node 表中新添加一条数据，但其 `node_type` 与 `pod_id` 值通常不符合预期，需要手动订正。

假设 c1 需要与原 b1 形成一组新 node，则操作步骤为：

1. 使用物理备份方式，手动将两个节点的代码仓库数据同步。

   将 b1 `/home/git` 目录下所有文件（代码）打包复制到 c1 实例的 `/home/git` 下。

   > **说明**
   >
   > 可用 rsync 命令，在 b1 实例上操作，例如：`rsync -alrtDP -z /home/git {c1_ip}:/home` 。

2. 更新 c1 实例在 node 表的记录。

   > **说明**
   >
   > - pod_id 必须设置。
   > - SQL：`update node set node_type = xx` 、 `pod_id = xx where id = xxx` 。
   > - 参数按实际需求填写，c1 的 pod_id 应与 b1 同步。