解决方案实践

滴普科技数据智能平台解决方案实践

文档版本 1.1

发布日期 2024-04-23

HUAWEI

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 方案概述

应用场景

企业数据中台建设

随着企业信息化日臻完善以及DT时代的来临，数据量不断加速增长，企业信息化建设开始出现诸多发展瓶颈和痛点

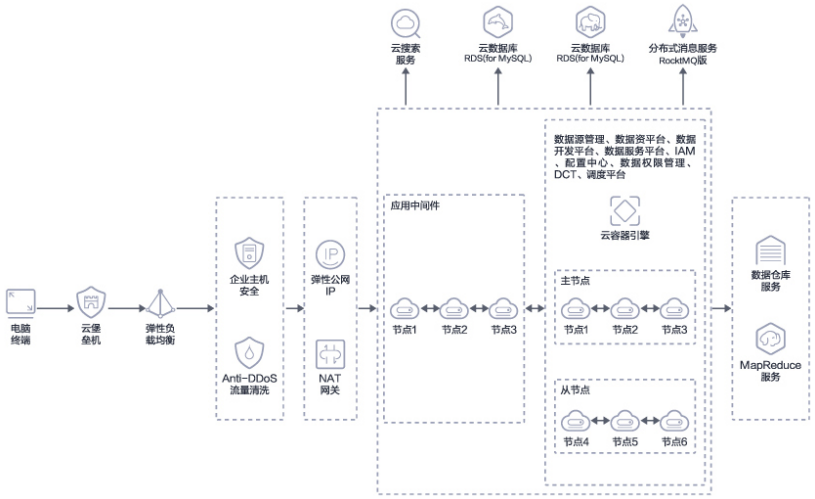
- 实时性要求越来越高，旧有的企业数据管理和分析体系无法支持业务系统的柔性响应、精准服务、快速迭代等需要，迫切需要建设数据中台，以更好的实现数据汇聚链接，驱动企业决策。
- 大数据发展至今，技术组件纷繁复杂，而企业缺乏统一的开发和管理工具，导致数据中台的建设存在成本高、周期长、建设运维复杂等痛点。
- 随着信息化的深入，在传统烟囱式IT建设方式下，企业独立采购或者自建的各种企业信息系统，在内部形成诸多数据孤岛。

通过本方案实现的业务效果：

- 平台组件统一部署，开发工具开箱即用，可视化开发维护易使用，数据任务智能化监控，异常告警及时处理。
- 非浸入式地多种不同存储引擎之间实时同步，进行数据统一汇聚，解决数据孤岛问题，基于WAL架构下的CKP异常自动保存技术，实现断点续传，面对再复杂的网络状况，也能保证数据传输的稳定性。
- 兼容多种技术组件，屏蔽技术细节，支持多人协作，从而降低系统建设复杂度，提高建设效率，缩短上线时间。
- 提供统一的企业数据共享服务，对数据消费方实行严格的数据权限管控，同时监控和分析服务来评估数据价值。
- 提供数据安全能力，基于多租户技术保障业务隔离和数据隔离，使用数据权限保证，从数据采集、数据存储、数据传输到数据使用的各个环节进行权限控制。
- 提供规范的数据模型、加工及服务能力，保障建模流程规范可控，使企业数据标准化。

方案架构

图 1-1 架构图



方案描述：

- 基于华为云计算底座构建基于开源、易于开发、面向业务的大数据平台，实现数据的集中管理；
- 统一数据标准，统一统计口径，实现各业务系统间数据打通、端到端的数据处理及应用；
- 打造支撑企业战略发展的核心指标体系，提供了精准、高效的数据指标服务，切实有效地帮助管理者提升业务洞察和辅助决策能力。

方案优势

- 实时高性能数据同步：基于CDC机制，实现复网络环境下的实时数据实时同步，轻松应对海量异构数据源，全面覆盖各类数据同步场景
- 技术兼容开放：兼容开源生态，支持多种计算引擎并可随心切换；兼容多种存储标准，满足多模态数据存储需求
- 简单易用多人协作：提供敏捷开发IDE，通过简单SQL编译即可让企业拥有大数据开发能力；多人协作设计、开发提高效率，缩短开发周期
- 生态开放：对外提供数据基础设施的各项能力，通过OpenAPI，助力实现多种大数据应用业务场景。
- 全流程数据质量监控：支持ETL全流程的事前、事中、事后数据质量检查和告警，形成数据质量报告，帮助客户提升数据质量。
- 一站式数据开发：覆盖数据采集、建模、离线/实时开发、运维监控、服务开发等环节，数据开发者只需专注于业务开发；
- 支持复杂调度场景：支持ETL流程多层嵌套、跨项目、跨流程依赖调度，支持不同周期时间的调度。

2 资源和成本规划

表 2-1 云服务资源清单

序号	资源名称	配置规格	命名示例	Region	数量	说明
1	VPC	默认配置即可 有需要可自定义网段	vpc-default	/	1	所有资源建议公用一个VPC
	NAT网关	最小配置规格	nat-e2bf	/	/	配置内网专用服务器能上外网
2	安全组	根据需要开通入方向端口	Daas-Webserver	/	1	确保所有服务器集群所在安全组可以访问数据源。
3	ECS	s6.small.1 1vCPUs | 1GB | 40GB通用型SSD	ecs-test	/	1	用作跳板机
4	CCE鲲鹏集群	Master 2核4G NODE 8核32G	Daas-2021	/	3maser 3node	应用服务器K8S容器平台
5	中间件	8核32G 100GB磁盘	Daas-yx	/	6	RedisGraph Rocketmq Zookeeper
6	RDS（PostgreSQL）	2核4G 100GB	Rds-972	/	1	PostgreSQL
7	云搜索服务css	4核8G 100GB	css-2ef5	/	1	elasticsearch

序号	资源名称	配置规格	命名示例	Region	数量	说明
8	GaussDB(for MySQL)	8核32G 100GB	Daas-mysql	/	1	Mysql应用数据库
9	CDH集群	16核64G 150GB	Daas-cdh	/	6	cdh大数据集群

表 2-2 软件信息

软件	版本	说明
Docker	18.09.9	docker容器
Kubernetes	容器集群	华为cce容器集群
MySQL	8.0	GaussDB华为云数据库
Redis	6.0.0	平台用缓存
RocketMQ	4.4	平台用消息中间件
Zookeeper	3.6.1	平台用服务注册中心
PostgreSQL	11.2	华为云RDS服务
Kong	1.1.0	平台用网关
Konga	0.14.1	Kong dashboard
Apollo	v1.9.0	Apollo配置中心
Eureka	2.0.0	平台用spring cloud服务注册中心
elasticsearch	7.7.1	华为云Css搜索服务
RedisGraph	2.2.0	图数据库
Jdk	1.8.0_251	Java程序工具
dolphinscheduler	/	调度平台
deepexi-daas-security	/	Daas-安全项目
deepexi-daas-metrics	/	Daas-指标项目
deepexi-daas-develop	/	Daas-开发平台项目
deepexi-daas-bigdata-develop	/	Daas-大数据开发项目
deepexi-daas-datasource	/	Daas-数据源项目
deepexi-daas-data-service	/	Daas-数据服务项目
deepexi-daas-client-iam-admin	/	IAM-租户版本服务管理平台

软件	版本	说明
deepexi-daas-client-iam-openapi	/	IAM-租户版本服务-开放接口
deepexi-daas-client-iam-sso	/	IAM-租户版本服务-单点登录
deepexi-daas-management-web	/	Daas管理平台-总管理平台(前端)
deepexi-daas-service-web	/	Daas管理平台-数据/业务建模(前端)
deepexi-daas-dev-web	/	Daas管理平台-开发平台(前端)
deepexi-client-iam-front	/	IAM-管理端(前端)

表 2-3 成本规划

云资源	规格	数量	每月费用（元）
弹性公网IP	带宽费用: 独享 | 全动态BGP | 按带宽计费 | 1Mbit/s弹性公网IP费用: 1个	1	86.25
企业主机安全	规格: 企业版	1	90.00
Anti-DDoS流量清洗	DDOS原生基础防护	1	0.00
弹性云服务器	规格: X86计算 | 通用入门型 | t6.2xlarge.4 | 8核 | 32GB 镜像: CentOS | CentOS 7.6 64bit 系统盘: 高IO | 200GB 弹性公网IP: 全动态BGP | 独享 | 按带宽计费 | 5Mbit/s	3	2,597.70
云容器引擎	产品分类: CCE容器集群 | 混合集群 | 集群管理规模50节点 | 高可用 含3个worknode	1	3,728.20
NAT网关	NAT网关 产品规格 小型NAT网关	1	306.00
弹性云服务器	规格: X86计算 | 通用入门型 | t6.2xlarge.4 | 8核 | 32GB 镜像: CentOS | CentOS 7.6 64bit 系统盘: 高IO | 500GB 弹性公网IP: 全动态BGP | 独享 | 按带宽计费 | 5Mbit/s	3	2,912.70

云资源	规格	数量	每月费用（元）
弹性云服务器	规格: X86计算 | 通用入门型 | t6.4xlarge.1 | 8核 | 16GB 镜像: CentOS | CentOS 7.6 64bit 系统盘: 通用型SSD | 500GB 弹性公网IP: 全动态BGP | 独享 | 按带宽计费 | 5Mbit/s	6	5,751.60
弹性云服务器	规格: X86计算 | 通用入门型 | t6.2xlarge.2 | 8核 | 16GB 镜像: CentOS | CentOS 7.6 64bit 系统盘: 通用型SSD | 500GB 弹性公网IP: 全动态BGP | 独享 | 按带宽计费 | 5Mbit/s	3	3,013.80
云搜索服务	规格: X86计算 | 计算密集型 | 4 vCPUs | 8 GB 存储类型: 超高I/O | 100GB 带宽类型: 低带宽 | 5Mbit/s	1	2,399.16
云数据库 RDS(for MySQL)	MySQL | 8.0 | 主备 | 通用型 | 2核4GB SSD云盘 | 40GB	1	470.00
云容器引擎	产品分类: CCE容器集群 | 混合集群 | 50节点 | 是	1	1,262.40
企业主机安全	规格: 企业版	1	90.00
Anti-DDoS流量清洗	DDOS原生基础防护	1	0.00
云数据库 RDS(for PostgreSQL)	4核，8G 200G SSD存储 华南区，主备, 版本：12	1	1,125.00
弹性公网IP	带宽费用: 独享 | 全动态BGP | 按带宽计费 | 5Mbit/s 弹性公网IP费用: 1个	1	115.00
弹性云服务器	规格: X86计算 | 通用入门型 | t6.2xlarge.4 | 8核 | 32GB 镜像: CentOS | CentOS 7.9 64bit 系统盘: 高IO | 40GB 数据盘: 通用型SSD | 200GB 弹性公网IP: 全动态BGP | 独享 | 按带宽计费 | 5Mbit/s	4	3,395.60

云资源	规格	数量	每月费用（元）
弹性云服务器	规格: X86计算 | 通用入门型 | t6.2xlarge.2 | 8核 | 16GB 镜像: CentOS | CentOS 7.9 64bit 系统盘: 高IO | 40GB 数据盘: 通用型SSD | 200GB 弹性公网IP: 全动态BGP | 独享 | 按带宽计费 | 5Mbit/s	8	5,652.80
弹性负载均衡 ELB	动态BGP|固定带宽|5Mbps 公网IP	1	309.36
小型NAT网关	小型NAT网关	1	306.00

3 实施步骤

3.1 华为云服务实例创建

3.1.1 云服务 cce 集群鲲鹏

步骤1 登录华为云控制台，搜索cce集群

图 3-1 登录控制台

步骤2 单击购买kubernetes，选择购买鲲鹏集群

图 3-2 单击 1

图 3-3 单击 1

图 3-4 单击 3

步骤3 选择完成配置后单击下一步创建节点，选择node节点规格8核32G内存系统磁盘40GB，数据磁盘100GB

图 3-5 创建节点 1

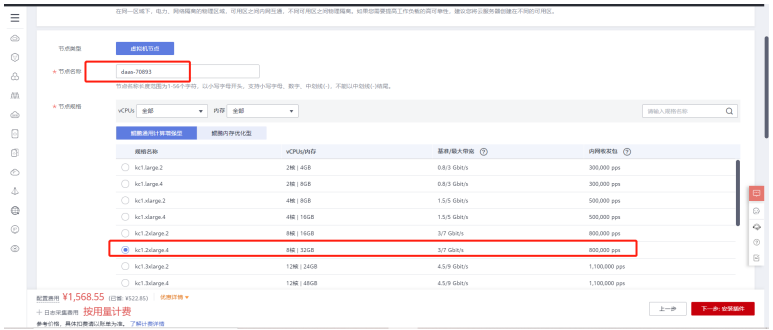
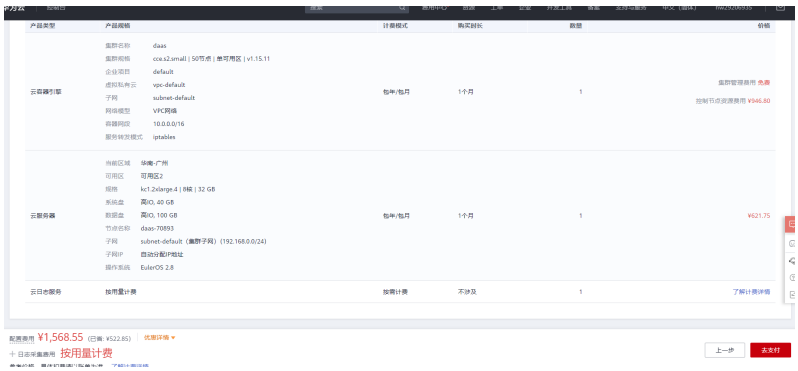


图 3-6 创建节点 2

步骤4 单击下一步创建，安装插件，选择默认即可，单击去支付完成集群创建

图 3-7 集群创建 1

图 3-8 集群创建 2



----结束

3.1.2 GaussDB(for MySQL)

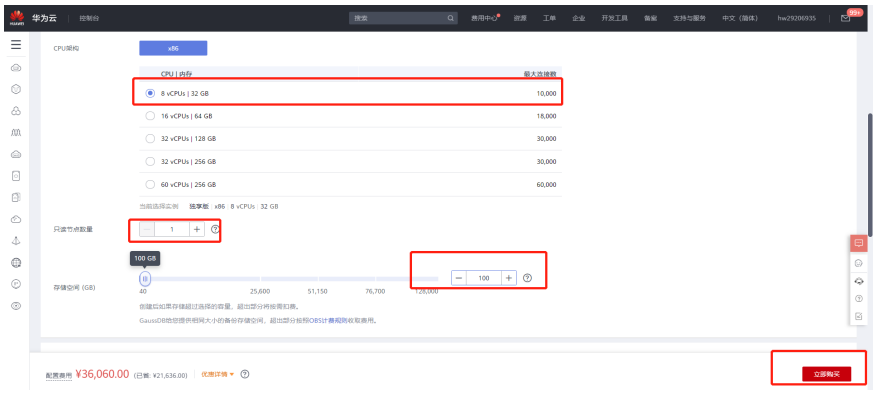
步骤1 创建数据库实例

图 3-9 创建数据库实例 1

图 3-10 创建数据库实例 2

步骤2 选择购买配置信息

图 3-11 选择购买配置信息



步骤3 选择vpc 信息与子网信息需要与cce创建一致

图 3-12 选择 vpc 信息

步骤4 选择数据库版本以及购买时长信息，单击立即购买按钮

图 3-13 立即购买

步骤5 查看创建信息后单击去支付按钮

图 3-14 单击按钮

----结束

3.1.3 RDS（PostgreSQL）

步骤1 单击控制台RDS购买PostgreSQL

图 3-15 单击控制台

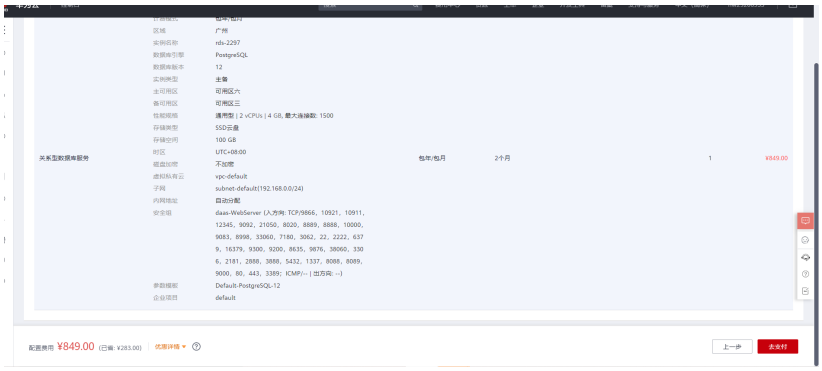
步骤2 选择内存 磁盘配置

图 3-16 选择内存 磁盘配置

步骤3 配置网络安全组，单击购买

图 3-17 单击购买 1

图 3-18 单击购买 2



----结束

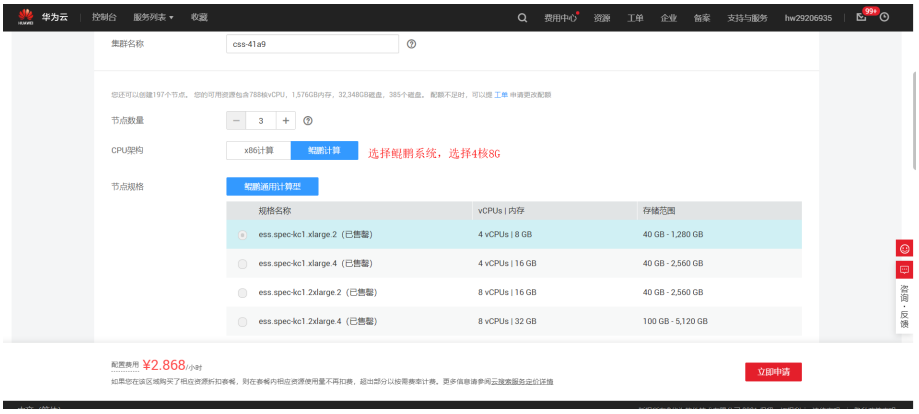
3.1.4 云搜索服务 CSS

步骤1 创建集群

图 3-19 创建集群

步骤2 选择鲲鹏版本创建，4核8G内存，存储节点100GB，节点数量3个

图 3-20 版本创建



步骤3 设置密码，选择安全组，单击立即申请创建css集群

图 3-21 设置密码

步骤4 创建完成后如图所示

图 3-22 创建完成

步骤5 购买资源包，单击如图所示按钮购买折扣套餐

图 3-23 购买资源包 1

图 3-24 购买资源包 2

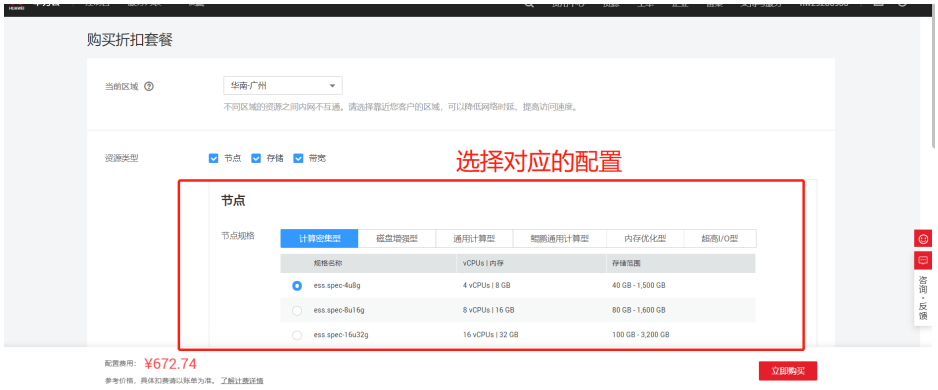
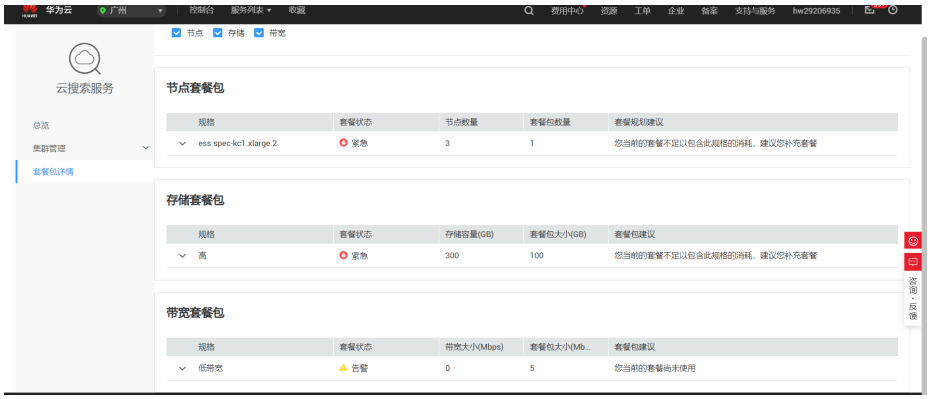


图 3-25 购买资源包 3

步骤6 单击购买按钮，购买后的资源包可以用来抵扣集群使用量

图 3-26 购买 4



----结束

3.2 VPC&安全组创建

3.2.1 VPC 创建

步骤1 打开华为云控制台，搜索虚拟私有云VPC

步骤2 创建私有VPC

图 3-27 创建私有 VPC1

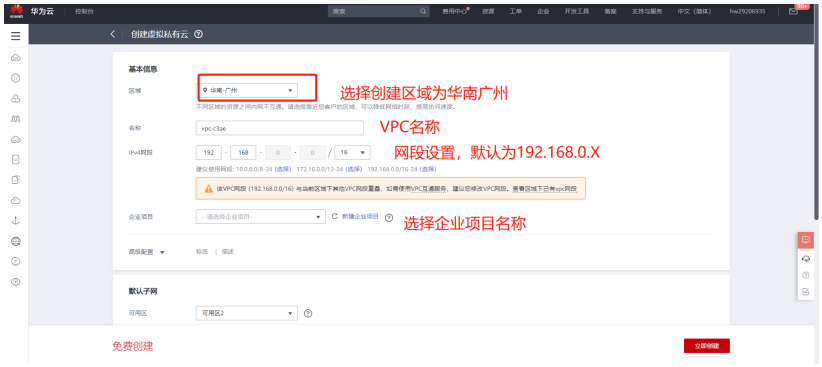


图 3-28 创建私有 VPC2

步骤3 单击创建，创建成功后如图所示

图 3-29 创建成功

步骤4 至此，创建VPC完成

----结束

3.2.2 创建网络安全组

步骤1 单击VPC网络控制台，找到访问控制-安全组

图 3-30 安全组

步骤2 单击创建安全组

图 3-31 创建安全组

步骤3 查看创建的安全组

图 3-32 查看安全组

----结束

3.3 CDH 部署

主机映射

```
hostnamectl set-hostname cluster1-hadoop01-yx-daas-deepexi
vim /etc/hosts
# 192.168.0.225   cluster1-hadoop01-yx-daas-deepexi
# 192.168.0.21    cluster1-hadoop02-yx-daas-deepexi
# 192.168.0.232   cluster1-hadoop03-yx-daas-deepexi
# 192.168.0.60    cluster2-hadoop01-yx-daas-deepexi
# 192.168.0.97    cluster2-hadoop02-yx-daas-deepexi
# 192.168.0.252   cluster2-hadoop03-yx-daas-deepexi
```

关闭防火墙

1. 查看状态
   ```
   systemctl status firewalld
   systemctl status iptables
   ```
2. 停止
   ```
   systemctl stop firewalld
   systemctl stop iptables
   ```
3. 禁止开机启动
   ```
   systemctl disable firewalld
   systemctl disable iptables
   ```

Ntp 配置

1. yum install ntp

   配置master服务器：

   ```
   vim /etc/ntp.conf
   # 允许内网其他机器同步时间，如果不添加该约束默认允许所有IP访问本机同步服务
   # IP为本局域网内的子网IP
   restrict 192.168.100.0 mask 255.255.255.0 nomodify notrap
   #配置和上游标准时间同步
   server 210.72.145.44  # 中国国家授时中心
   server 0.cn.pool.ntp.org
   server 1.cn.pool.ntp.org
   server 2.cn.pool.ntp.org
   server 3.cn.pool.ntp.org
   #如果外部时间服务器不可用，NTP Server以本地时间作为时间服务器
   server 127.127.1.0  # local clock
   fudge 127.127.1.0 stratum 10
   ```
2. 配置slaves 服务器
   ```
   #配置和上游标准时间同步
   Server 192.168.100.45
   #配置允许上游时间服务器主动修改本机的时间
   restrict 192.168.100.45 nomodify notrap noquery
   timedatectl  set-ntp  yes
   ```
3. 查看当前时间和NTP服务的同步状态

```
timedatectl  set-ntp  yes
timedatectl
ntpstat
```

SSH 免密登录配置

1. 设置config
   ```
   vim /etc/ssh/sshd_config
   PermitRootLogin yes            # 允许root用户登录
   PasswordAuthentication no      # 不使用密码登录
   systemctl restart sshd.service
   ```
2. 只在其中一台服务器生成密钥
   ```
   ssh-keygen -t rsa
   ```
3. 生成authorized_keys
   ```
   cat /root/.ssh/id_rsa.pub >> authorized_keys
   ```
4. 下载 私匙id_rsa 和 authorized_keys 文件,并上传到其他节点~/.ssh目录下
   ```
   chmod 700 id_rsa
   mkdir /root/.ssh
   ```
5. 验证
   ```
   ssh cluster1-hadoop02-yx-daas-deepexi
   exit
   ```

JDK 安装

1. 必须安装在 /usr/java/
   ```
   tar –zxvf jdk-8u181-linux-x64.tar.gz –C /usr/java/
   ```
2. 配置环境变量
   ```
   vim /etc/profile
   export JAVA_HOME=/usr/java/ jdk-1.8.0_181
   export PATH=$PATH:$ JAVA_HOME/bin
   ```

jdbc 上传

上传jdbc到/usr/share/java/目录下，重命名为mysql-connector-java.jar

```
mkdir -p /usr/share/java/
cp mysql-connector-java-8.0.15-bin.jar /usr/share/java/mysql-connector-java.jar
```

禁用透明大页面压缩

1. 查看
   ```
   cat /sys/kernel/mm/transparent_hugepage/defrag
   ```
2. 修改
   ```
   echo never > /sys/kernel/mm/transparent_hugepage/defrag
   echo never > /sys/kernel/mm/transparent_hugepage/enabled
   ```
3. 将命令写入开机启动
   ```
   vim /etc/rc.local
   ```

Swapping 交换区设置

1. 查看
   ```
   cat /proc/sys/vm/swappiness
   ```
2. 修改
   ```
   echo &quot;vm.swappiness = 0&quot; &gt;&gt; /etc/sysctl.conf
   ```

3. 生效
```
sysctl -p
```

Tunned 系统调优服务禁用

1. 查看tuned状态
```
systemctl status tuned
```
2. 查看活动的服务
```
tuned-adm list
```
3. 关闭tuned服务
```
tuned-adm off
```
4. 确定没有活动的服务（No current active profile）
```
tuned-adm list
```
5. 关闭并禁止开启启动
```
systemctl stop tuned
systemctl disable tuned
```

MySQL 安装（离线）

1. 查看并卸载系统自带的Mariad
```
rpm –qa | grep mariadb
rpm -e --nodeps mariadb-libs-5.5.52-1.el7.centos.x86_64
```
2. 上传rpm包
```
mysql-community-client-8.0.15-1.el7.x86_64.rpm
mysql-community-common-8.0.15-1.el7.x86_64.rpm
mysql-community-libs-8.0.15-1.el7.x86_64.rpm
mysql-community-server-8.0.15-1.el7.x86_64.rpm
mysql-community-libs-compat-8.0.15-1.el7.x86_64.rpm
```
3. 安装
```
rpm -ivh mysql-community-client-8.0.15-1.el7.x86_64.rpm
```
4. 顺序
```
common -> libs -> client -> server -> libs-compat
```
5. 如果提示缺少libaio依赖
```
yum install libaio
```
6. 查看状态
```
systemctl status mysqld
```
7. 备份数据配置文件并修改
```
cp /etc/my.cnf{,.bak}
vim /etc/my.cnf
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
transaction-isolation = READ-COMMITTED
key_buffer_size = 32M
max_allowed_packet = 32M
thread_stack = 256K
thread_cache_size = 64
max_connections = 5000
binlog_expire_logs_seconds = 10
max_binlog_size = 200M
log_bin=/var/lib/mysql/mysql_binary_log
server_id=225
binlog_format = ROW
read_buffer_size = 2M
read_rnd_buffer_size = 16M
sort_buffer_size = 8M
join_buffer_size = 8M
innodb_file_per_table = 1
```

```
innodb_flush_log_at_trx_commit  = 2
innodb_log_buffer_size = 64M
innodb_buffer_pool_size = 4G
innodb_thread_concurrency = 8
innodb_flush_method = O_DIRECT
innodb_log_file_size = 512M
#lower_case_table_names = 1
[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
sql_mode=STRICT_ALL_TABLES
```

8. 启动和开机启动

```
sudo systemctl start mysqld
sudo systemctl enable mysqld
```

9. 查看初始密码

```
systemctl status mysqld -l
```

10. 登录

```
mysql –uroot –p
输入密码
```

11. 修改密码

```
alter user 'root'@'localhost' identified by '123456';
```

12. 修改权限

```
use mysql;
update user set host = &amp;quot;%&amp;quot; where user = 'root';
flush privileges;
quit;
# 如果提示密码简单
alter user 'root'@'localhost' identified by 'Abcd123456.';
set global validate_password.policy=0;
set global validate_password.length=4;
alter user 'root'@'localhost' identified by '123456';
```

13. 修改连接限制

```
vim /usr/lib/systemd/system/mysqld.service
LimitNOFILE = 65535
sysctl –p
```

14. 登录MySQL建库

```
create database scm default character set utf8 default collate utf8_general_ci;
create database amon default character set utf8 default collate utf8_general_ci;
create database rman default character set utf8 default collate utf8_general_ci;
create database hue default character set utf8 default collate utf8_general_ci;
create database metastore default character set utf8 default collate utf8_general_ci;
create database sentry default character set utf8 default collate utf8_general_ci;
create database nav default character set utf8 default collate utf8_general_ci;
create database navms default character set utf8 default collate utf8_general_ci;
create database oozie default character set utf8 default collate utf8_general_ci;
```

15. 创建用户

```
create user 'scm'@'%' identified by '123456';
create user 'amon'@'%' identified by '123456';
create user 'rman'@'%' identified by '123456';
create user 'hue'@'%' identified by '123456';
create user 'hive'@'%' identified by '123456';
create user 'sentry'@'%' identified by '123456';
create user 'nav'@'%' identified by '123456';
create user 'navms'@'%' identified by '123456';
create user 'oozie'@'%' identified by '123456';
```

16. 修改密码验证方式

```
alter user 'scm'@'%' identified with mysql_native_password by '123456';
alter user 'amon'@'%' identified with mysql_native_password by '123456';
alter user 'rman'@'%' identified with mysql_native_password by '123456';
alter user 'hue'@'%' identified with mysql_native_password by '123456';
alter user 'hive'@'%' identified with mysql_native_password by '123456';
alter user 'sentry'@'%' identified with mysql_native_password by '123456';
alter user 'nav'@'%' identified with mysql_native_password by '123456';
```

```
alter user 'navms'@'%' identified with mysql_native_password by '123456';
alter user 'oozie'@'%' identified with mysql_native_password by '123456';
```

17. 授权

```
grant all on scm.* to 'scm'@'%';
grant all on amon.* to 'amon'@'%';
grant all on rman.* to 'rman'@'%';
grant all on hue.* to 'hue'@'%';
grant all on metastore.* to 'hive'@'%';
grant all on sentry.* to 'sentry'@'%';
grant all on nav.* to 'nav'@'%';
grant all on navms.* to 'navms'@'%';
grant all on oozie.* to 'oozie'@'%';
flush privileges;
```

18. 验证

```
show grants for 'scm'@'%';
show grants for 'amon'@'%';
show grants for 'rman'@'%';
show grants for 'hue'@'%';
show grants for 'hive'@'%';
show grants for 'sentry'@'%';
show grants for 'nav'@'%';
show grants for 'navms'@'%';
show grants for 'oozie'@'%';
```

Cloudera Manager（6.x 安装方式相同）

1. 离线下载CM安装包上传到服务器，主节点上传server和daemons，从节点上传agent和daemons

```
cloudera-manager-agent-6.2.1-1426065.el7.x86_64.rpm
cloudera-manager-daemons-6.2.1-1426065.el7.x86_64.rpm
cloudera-manager-server-6.2.1-1426065.el7.x86_64.rpm
```

2. 安装CM

```
yum install bind-utils psmisc cyrus-sasl-plain cyrus-sasl-gssapi fuse portmap fuse-libs /lib/lsb/init-functions httpd mod_ssl openssl-devel python-psycopg2 MySQL-python libxslt
rpm -ivh cloudera-manager-daemons-6.0.1-610811.el7.x86_64.rpm
rpm -ivh cloudera-manager-server-6.0.1-610811.el7.x86_64.rpm
```

3. 修改服务主机（所有agent）

```
vim /etc/cloudera-scm-agent/config.ini
server_host=master # server主机host
server_port=7182
```

4. 设置CM数据库

```
sudo /opt/cloudera/cm/schema/scm_prepare_database.sh mysql scm scm
# 输入scm密码
```

5. 启动server

```
sudo systemctl start cloudera-scm-server
```

6. 查看启动情况

```
tail -f /var/log/cloudera-scm-server/cloudera-scm-server.log
```

7. 查看server状态

```
sudo systemctl status cloudera-scm-server
显示以下信息说明启动完成
INFO WebServerImpl:com.cloudera.server.cmf.WebServerImpl: Started Jetty server.
```

8. 登录web

```
web：http://master:7180
账号密码：admin
```

CDH 集群部署

1. 单集群部署，上传CDH离线安装包

```
CDH-6.0.1-1.cdh6.0.1.p0.590678-el7.parcel
CDH-6.0.1-1.cdh6.0.1.p0.590678-el7.parcel.sha256
manifest.json
```

2. 上传到 /opt/cloudera/parcel-repo/，没有则手动创建，修改.sha256文件后缀

```
mv CDH-6.0.1-1.cdh6.0.1.p0.590678-el7.parcel.sha256 CDH-6.0.1-1.cdh6.0.1.p0.590678-el7.parcel.sha
```

3. 修改用户和组

```
chown cloudera-scm:cloudera-scm /opt/cloudera/parcel-repo/*
```

4. 登录界面

图 3-33 登录界面 1

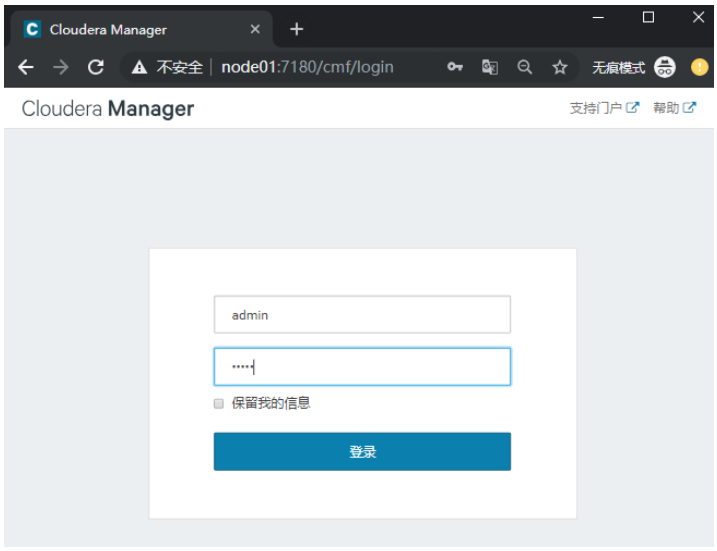


图 3-34 登录界面 2

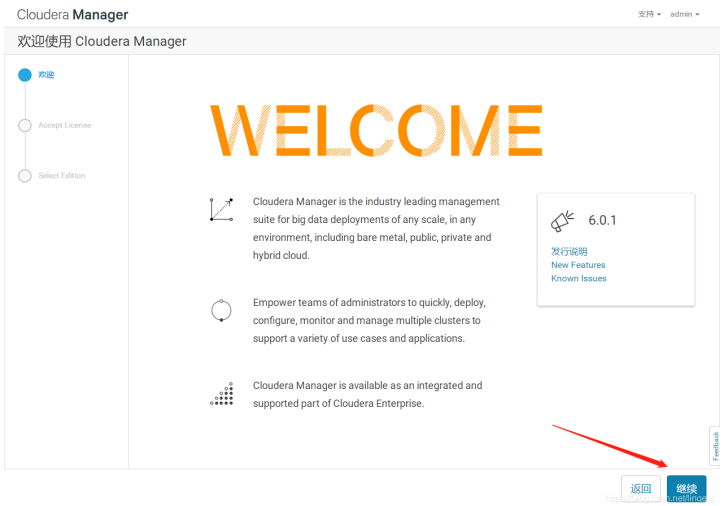
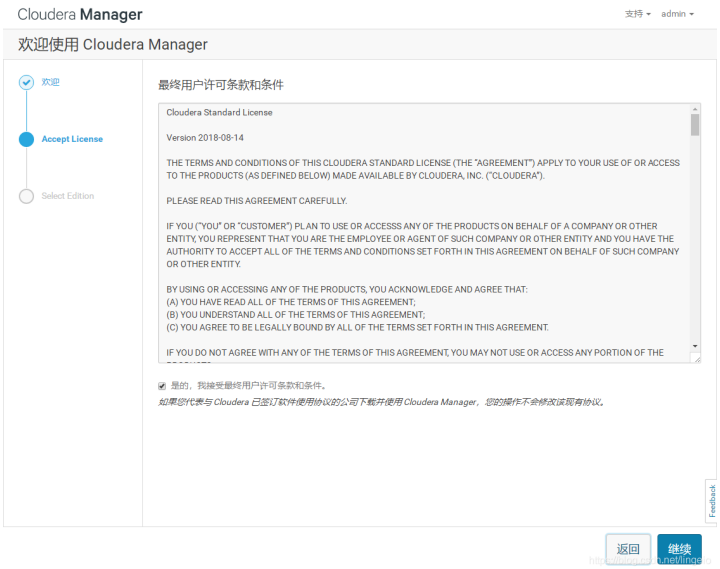


图 3-35 登录界面 3



5. 选择版本，免费版即可

图 3-36 选择版本 1

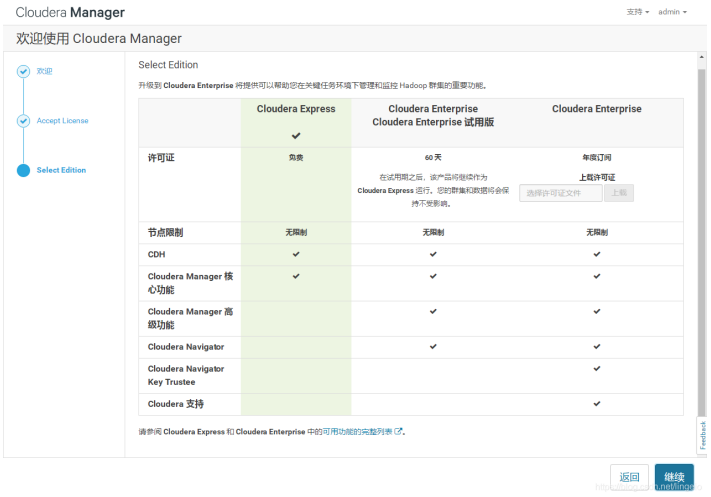
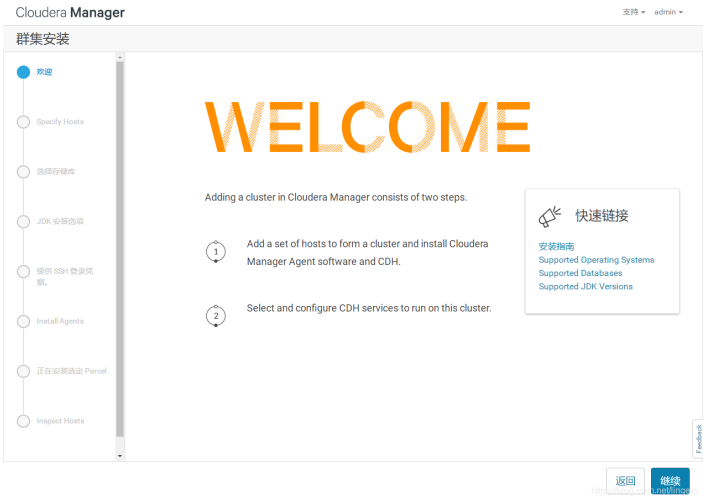


图 3-37 选择版本 2



6. 单击继续，可输入和搜索主机，也可通过“当前管理的主机”看到已连接到server的agent，直接选择即可

图 3-38 当前管理的主机 1

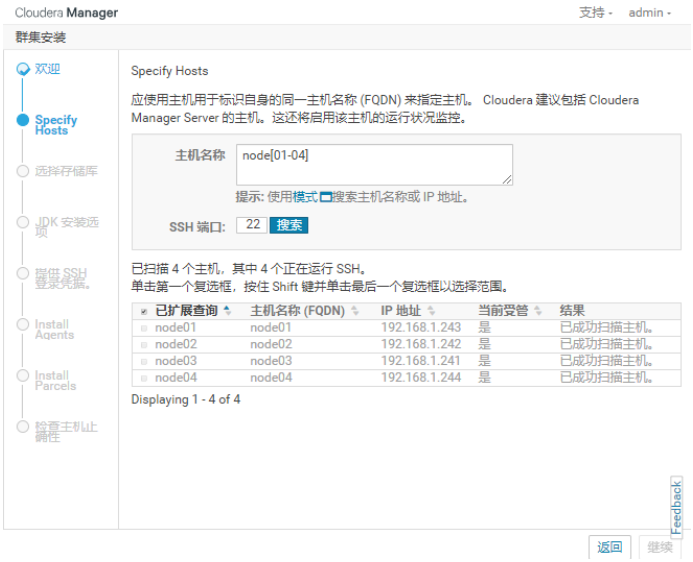


图 3-39 当前管理的主机 2

图 3-40 当前管理的主机 3

图 3-41 当前管理的主机 4

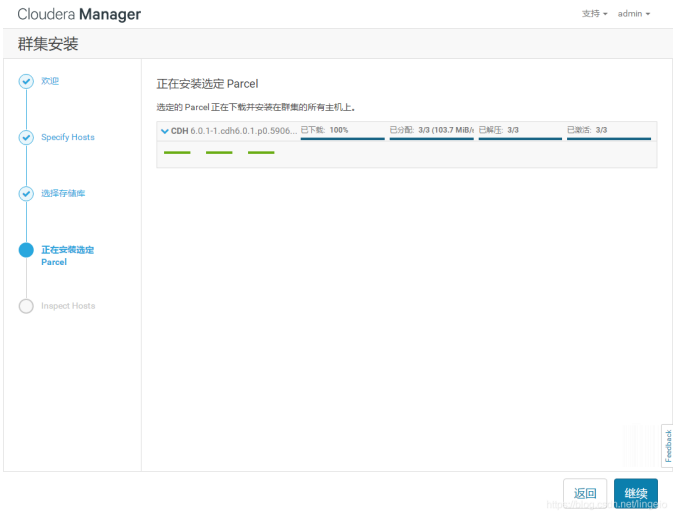


图 3-42 当前管理的主机 5

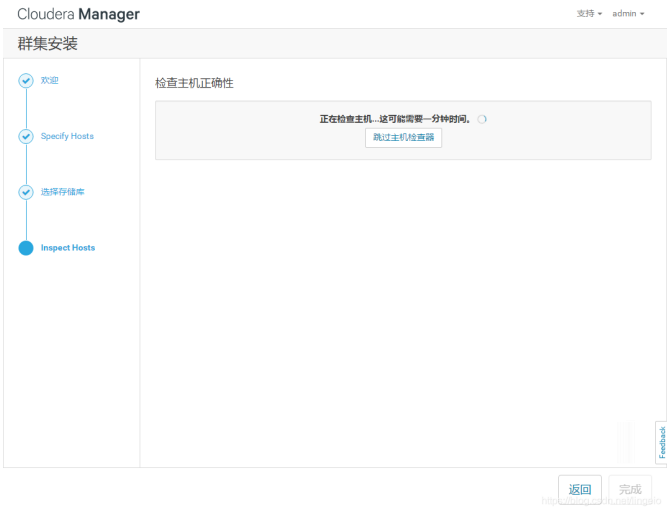


图 3-43 当前管理的主机 6

7. 集群安装

图 3-44 集群安装 1

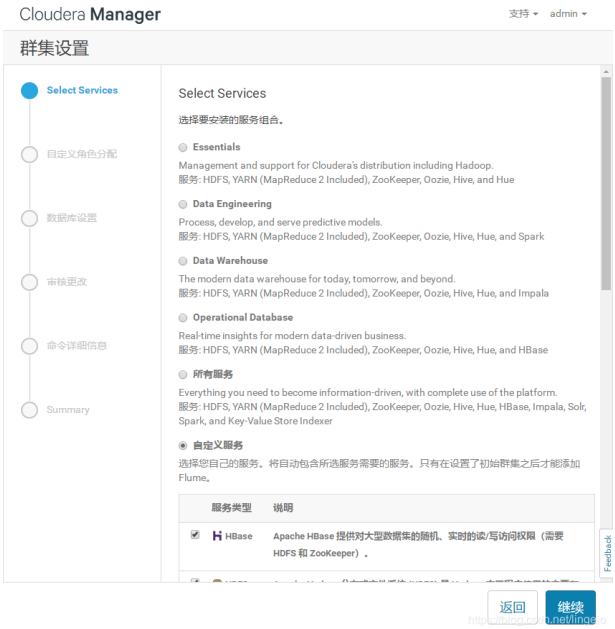


图 3-45 集群安装 2

图 3-46 集群安装 3

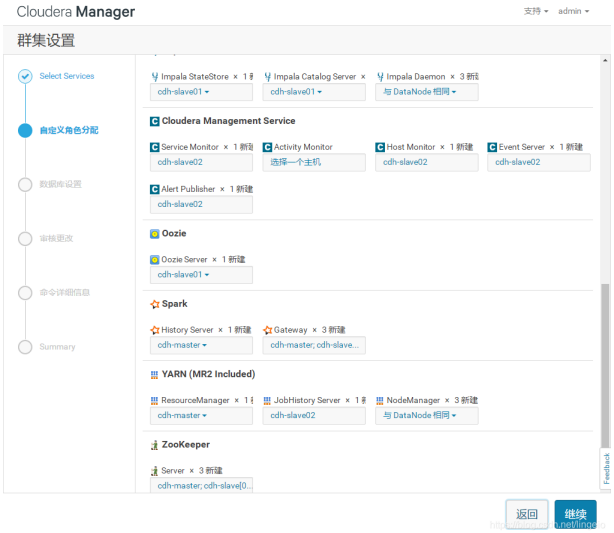


图 3-47 集群安装 4

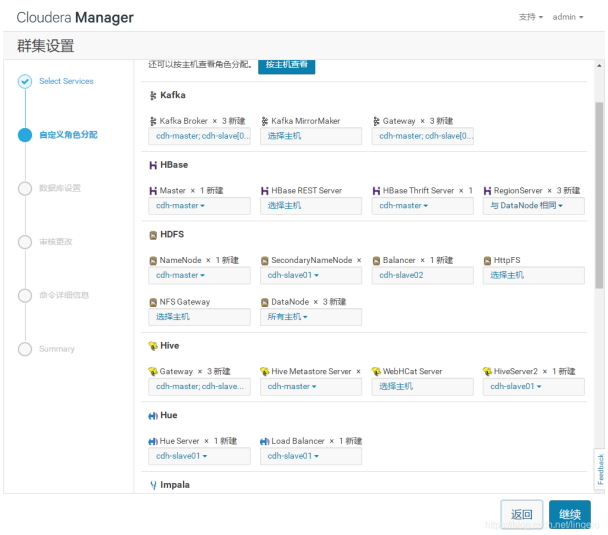


图 3-48 集群安装 5

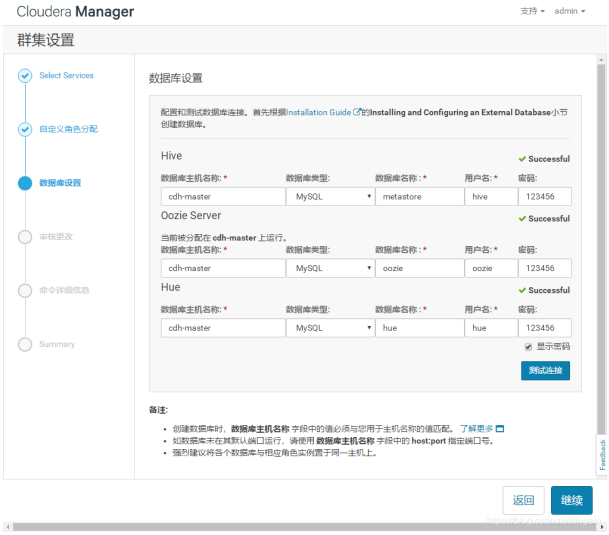


图 3-49 集群安装 6

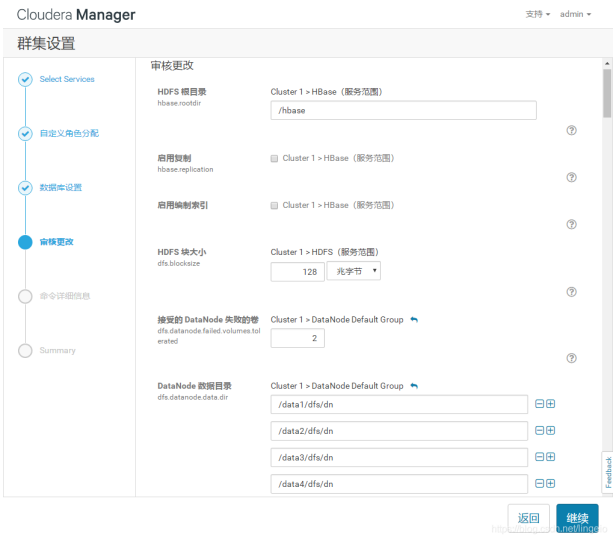


图 3-50 集群安装 7

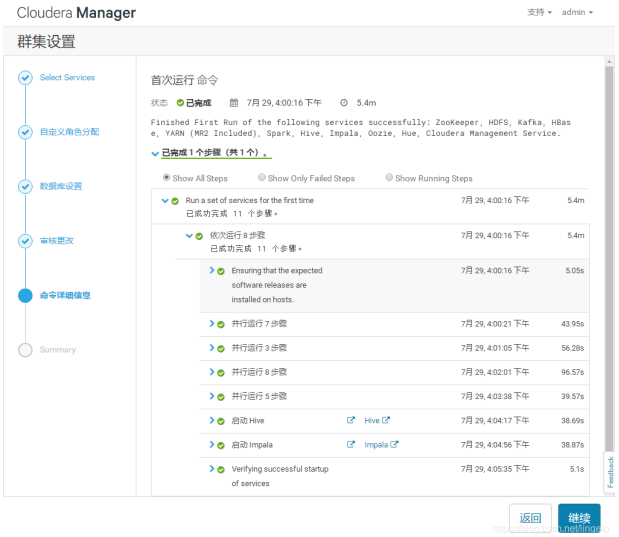
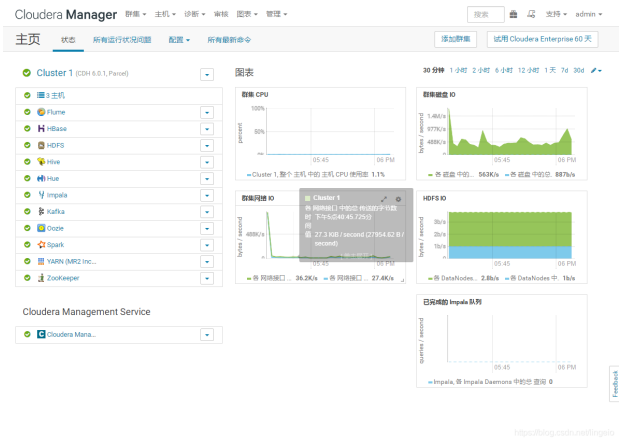


图 3-51 集群安装 8



8. 安装Httpd（集群中一台服务器上即可）
```
yum install –y httpd
```
9. 创建cloudera目录
```
mkdir -p /var/www/html/cloudera
```
10. 复制安装包到 /var/www/html/cloudera
```
allkeys.asc
cloudera-manager-agent-6.2.1-1426065.el7.x86_64.rpm
cloudera-manager-daemons-6.2.1-1426065.el7.x86_64.rpm
cloudera-manager-server-6.2.1-1426065.el7.x86_64.rpm
cloudera-manager-server-db-2-6.2.1-1426065.el7.x86_64.rpm
enterprise-debuginfo-6.2.1-1426065.el7.x86_64.rpm
RPM-GPG-KEY-cloudera
```
11. 安装createrepo服务，用于创建自定义仓库
```
yum install -y yum-utils createrepo
```
12. 生成rpm元数据
```
cd /var/www/html/cloudera
createrepo .
```
13. 浏览器验证

    http://master/cloudera

14. 制作Cloudera Manager的repo源

```
vim /etc/yum.repos.d/cm6.1.1_repo
[cm6.2.1_repo]
name = cm6.2.1_repo
baseurl=http://master/cloudera
enable = true
gpgcheck = false
```

15. 重启CM

```
systemctl start cloudera-scm-server
```

16. 登录web

    web：http://master:7180

    账号密码：admin

图 3-52 登录 web

17. 选择服务器步骤与离线安装相同，从Paecel选择开始不同

图 3-53 选择 1

18. 选择自定义存储库，输入上面配置的cloudera的http地址

图 3-54 选择 2

19. Parcel选择，单击“更多选项”,单击“-”删除其它所有地址，输入http://master/cloudera，单击“保存更改”

图 3-55 选择 3

图 3-56 选择 4

20. 单击继续，选择jdk，上面已经安装jdk，取消勾选即可

图 3-57 选择 5

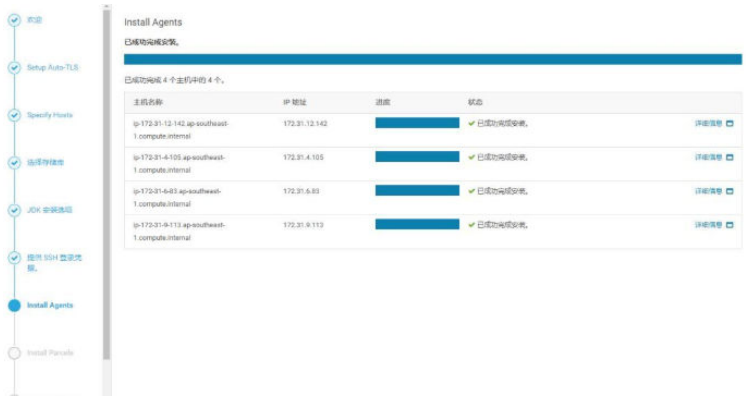
21. 单击继续，配置ssh账号密码

图 3-58 配置 ssh 账号密码

22. 单击“继续”，进入下一步，安装cm agent相关到各个节点

图 3-59 下一步 1

图 3-60 下一步 2



23. Agent安装完以后，大数据组件安装与单集群离线安装相同

3.4 Redis 部署

登录云服务器 ECS

创建命名空间

```
# kubectl create ns applife
# kubectl create ns daas
# kubectl create ns daas-web
# kubectl create ns kong
# kubectl create ns openkube
# kubectl create ns nfs
```

RedisGraph 安装

安装路径约定，安装机器：中间件服务器

安装目录：/opt/redis-6.0.0

配置文件：/opt/redis-6.0.0/config/redis.conf 和 /opt/redis-6.0.0/config/sentinel/sentinel.conf

数据目录：/data/redis-6.0.0/data

日志目录：/data/redis-6.0.0/logs

插件目录：/opt/redis-6.0.0/modules/redisgraph.so

安装 Redis 主从

1. 解压，上传压缩包redis-6.0.0-linux.tar.gz到/soft/redis并解压

```
# tar -vxf redis-6.0.0-linux.tar.gz
# mv redis-6.0.0 /opt
# mkdir /opt/redis-6.0.0/bin
# cd /opt/redis-6.0.0/src
# cp redis-benchmark redis-check-aof redis-check-rdb redis-cli redis-sentinel redis-server -t ../bin/
```

2. 创建数据目录及授权

```
# mkdir -p /data/redis-6.0.0/{data,logs}
# chmod 755 /opt/redis-6.0.0/modules/redisgraph.so
```

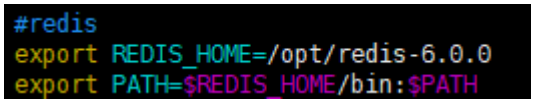
说明

redisgraph.so要授权可执行，否则会导致redis启动失败。

3. 配置环境变量

```
# vim /etc/profile
export REDIS_HOME=/opt/redis-6.0.0
export PATH=$REDIS_HOME/bin:$PATH
```

图 3-61 配置环境变量



4. 新增主redis配置文件

说明

/opt/redis-6.0.0/config/redis.conf文件不存在，注意文件不要有特殊空格字符，踩过坑！

```
# vim /opt/redis-6.0.0/config/redis.conf
bind 0.0.0.0
port 6379
daemonize yes
pidfile /var/run/redis.pid
loglevel notice
logfile /data/redis-6.0.0/logs/redis.log
dir /data/redis-6.0.0/data
masterauth Mypwd@123456
requirepass Mypwd@123456
appendonly yes
loadmodule /opt/redis-6.0.0/modules/redisgraph.so
# 确保有1个从节点写入，且延时不超过10s，否则主节点会停止写入请求（防止数据丢失）
min-replicas-to-write 1
min-replicas-max-lag 10
```

5. 新增从redis配置文件

在安装redis-slave机器上，重复上面5.4.2.1 到5.4.2.4过程，并使用下面配置。

说明

如果/opt/redis-6.0.0/config/redis.conf文件不存在就新增。

```
# vim /opt/redis-6.0.0/config/redis.conf
bind 0.0.0.0
port 6379
daemonize yes
pidfile /var/run/redis.pid
loglevel notice
logfile /data/redis-6.0.0/logs/redis.log
dir /data/redis-6.0.0/data
masterauth Mypwd@123456
requirepass Mypwd@123456
appendonly yes
loadmodule /opt/redis-6.0.0/modules/redisgraph.so
# 确保有1个从节点写入，且延时不超过10s，否则主节点会停止写入请求（防止数据丢失）
min-replicas-to-write 1
min-replicas-max-lag 10
#配置连接主redis
replicaof 192.168.0.82 6379
```

6. 启动redis-server

分别在主从创建redis-server系统启动文件

```
# vim /etc/systemd/system/redis-server.service
[Unit]
Description=redis
```

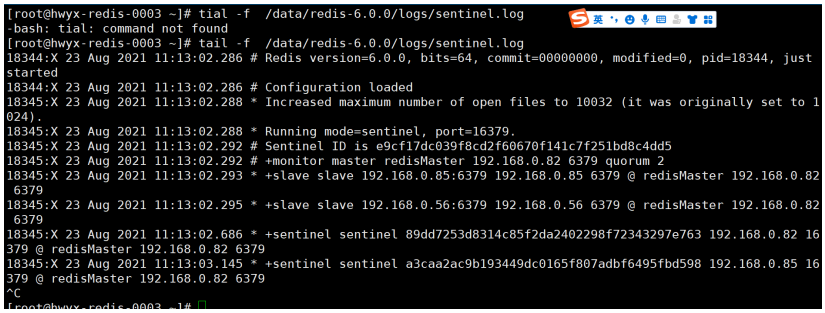
```
After=network.target
[Service]
Type=forking
PIDFile=/var/run/redis.pid
ExecStart=/opt/redis-6.0.0/bin/redis-server /opt/redis-6.0.0/config/redis.conf
ExecStop=/opt/redis-6.0.0/bin/redis-cli -h 127.0.0.1 -p 6379 shutdown
PrivateTmp=true
[Install]
WantedBy=multi-user.target
```

7. 重新加载systemctl配置
```
# systemctl daemon-reload
```
8. 设置跟随linux启动，并启动es
```
# systemctl start redis-server && systemctl enable redis-server
```
9. 查看redis角色
```
# redis-cli -h 192.168.0.82 -p 6379 -a Mypwd@123456 INFO | grep role
```

配置哨兵

1. 创建哨兵数据目录
```
# mkdir /data/redis-6.0.0/data/sentinel
```
2. 新增配置文件
```
# vim /opt/redis-6.0.0/config/sentinel/sentinel.conf
```
3. 分别在主从redis创建sentinel配置文件（3个实例一样）
```
port 16379
daemonize yes
pidfile /var/run/redis-sentinel.pid
logfile /data/redis-6.0.0/logs/sentinel.log
dir /data/redis-6.0.0/data/sentinel
sentinel deny-scripts-reconfig yes
#配置主redis的地址，“2”表示集群中有2个Sentinel认为master退出了就切换
sentinel monitor redisMaster 192.168.0.82 6379 2
sentinel auth-pass redisMaster Mypwd@123456
sentinel config-epoch redisMaster 1
sentinel leader-epoch redisMaster 1
```
4. 启动哨兵，分别在主从redis创建redis-sentinel系统启动文件
```
# vim /etc/systemd/system/redis-sentinel.service
[Unit]
Description=The redis-sentinel Process Manager
After=syslog.target network.target
[Service]
Type=forking
PIDFile=/var/run/redis-sentinel.pid
ExecStart=/opt/redis-6.0.0/bin/redis-sentinel /opt/redis-6.0.0/config/sentinel/sentinel.conf
ExecStop=/opt/redis-6.0.0/bin/redis-cli -h 127.0.0.1 -p 16379 shundown
PrivateTmp=true
[Install]
WantedBy=multi-user.target
```
5. 重新加载systemctl配置
```
# systemctl daemon-reload
```
6. 设置跟随linux启动，并启动es
```
# systemctl start redis-sentinel && systemctl enable redis-sentinel
```
7. 查看启动日志
```
# cat /data/redis-6.0.0/logs/sentinel.log
```

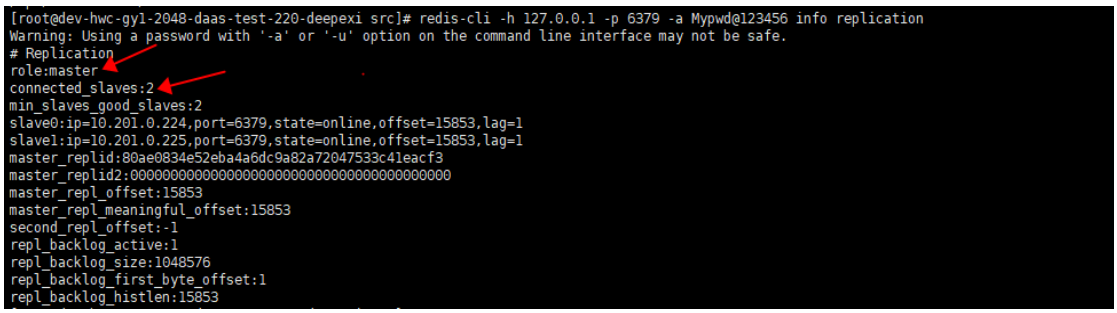
图 3-62 查看启动日志



8. 检查redis主从状态

```
# redis-cli -h 127.0.0.1 -p 6379 -a Mypwd@123456 info replication
```

图 3-63 查看启动日志




```
# Replication
role:master          #角色master或者slave
connected_slaves:2    #连接的slave个数
min_slaves_good_slaves:2
slave0:ip=192.168.0.228 ,port=6379,state=online,offset=288218,lag=0    #slave1节点信息
slave1:ip=192.168.0.56,port=6379,state=online,offset=288078,lag=1    #slave2节点信息
master_replid:2ac8c63f0185a38189010b192e42b761e69f549a
master_replid2:4490ff668a9d41d5636e6a382ff1e80ca31892cf
master_repl_offset:288218
master_repl_meaningful_offset:288218
second_repl_offset:69703
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:288218
```

3.5 Rocketmq 部署

3.5.1 安装配置 JDK、上传并解压

安装配置 JDK

1. 执行安装jdk命令

```
yum install java-1.8.0-openjdk
```

2. 配置环境变量

```
# vim /etc/profile
#JDK
export JAVA_HOME=/usr/lib/jvm/java
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JAVA_HOME/jre/lib/rt.jar
export PATH=$PATH:$JAVA_HOME/bin
```

图 3-64 配置环境变量

```
export JAVA_HOME=/usr/lib/jvm/java
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JAVA_HOME/jre/lib/rt.jar
export PATH=$PATH:$JAVA_HOME/bin

export REDIS_HOME=/opt/redis-5.0.5
export PATH=$REDIS_HOME/bin:$PATH
```

```
# source /etc/profile
```

上传并解压

1. 上传安装包到服务器/soft/rocketmq

```
# cd /soft/rocketmq
# unzip rocketmq-all-4.4.0-bin-release.zip
# mv rocketmq-all-4.4.0-bin-release /opt/rocketmq-4.4.0
```

2. 创建目录

```
# mkdir -p /data/rocketmq-4.4.0/data/store-master/{commitlog,consumequeue,index}
# mkdir -p /data/rocketmq-4.4.0/data/store-slave/{commitlog,consumequeue,index}
# mkdir -p /data/rocketmq-4.4.0/logs
```

3.5.2 RocketMq 配置

1. 节点1主192.168.0.190节点配置broker-master

```
# vim /opt/rocketmq-4.4.0/conf/broker-master.conf
brokerClusterName=SyncCluster
brokerName=broker-01
brokerId=0
brokerIP1=192.168.0.190  #本节点ip地址
namesrvAddr=192.168.0.190:9876;192.168.0.228:9876;192.168.0.168:9876  #配置所有节点的namesrv地址
defaultTopicQueueNums=4
autoCreateTopicEnable=true
autoCreateSubscriptionGroup=true
listenPort=10911
deleteWhen=04
fileReservedTime=72
mapedFileSizeCommitLog=1073741824
mapedFileSizeConsumeQueue=300000
destroyMapedFileInterval=120000
redeleteHangedFileInterval=120000
diskMaxUsedSpaceRatio=88
storePathRootDir=/data/rocketmq-4.4.0/data/store-master
storePathCommitLog=/data/rocketmq-4.4.0/data/store-master/commitlog
maxMessageSize=65536
flushCommitLogLeastPages=4
flushConsumeQueueLeastPages=2
flushCommitLogThoroughInterval=10000
flushConsumeQueueThoroughInterval=60000
checkTransactionMessageEnable=false
sendMessageThreadPoolNums=128
pullMessageThreadPoolNums=128
brokerRole=SYNC_MASTER
flushDiskType=ASYNC_FLUSH
```

2. 节点1从192.168.0.190节点配置broker-slave

```
# vim /opt/rocketmq-4.4.0/conf/broker-slave.conf
brokerClusterName=SyncCluster
brokerName=broker-01
brokerId=1
#本节点ip地址
brokerIP1=192.168.0.190
#配置所有节点的namesrv地址
namesrvAddr=192.168.0.190:9876;192.168.0.228:9876;192.168.0.168:9876
```

```
defaultTopicQueueNums=4
autoCreateTopicEnable=true
autoCreateSubscriptionGroup=true
listenPort=10921
deleteWhen=04
fileReservedTime=72
mapedFileSizeCommitLog=1073741824
mapedFileSizeConsumeQueue=300000
destroyMapedFileInterval=120000
redeleteHangedFileInterval=120000
diskMaxUsedSpaceRatio=88
storePathRootDir=/data/rocketmq-4.4.0/data/store-slave
storePathCommitLog=/data/rocketmq-4.4.0/data/store-slave/commitlog
maxMessageSize=65536
flushCommitLogLeastPages=4
flushConsumeQueueLeastPages=2
flushCommitLogThoroughInterval=10000
flushConsumeQueueThoroughInterval=60000
checkTransactionMessageEnable=false
sendMessageThreadPoolNums=128
pullMessageThreadPoolNums=128
brokerRole=SLAVE
flushDiskType=ASYNC_FLUSH
```

3. 节点2主192.168.0.228节点配置broker-master

```
# vim /opt/rocketmq-4.4.0/conf/broker-master.conf
brokerClusterName=SyncCluster
brokerName=broker-02
brokerId=0
#本节点ip地址
brokerIP1=192.168.0.228
#配置所有节点的namesrv地址
namesrvAddr=192.168.0.190:9876;192.168.0.228:9876;192.168.0.168:9876
defaultTopicQueueNums=4
autoCreateTopicEnable=true
autoCreateSubscriptionGroup=true
listenPort=10911
deleteWhen=04
fileReservedTime=72
mapedFileSizeCommitLog=1073741824
mapedFileSizeConsumeQueue=300000
destroyMapedFileInterval=120000
redeleteHangedFileInterval=120000
diskMaxUsedSpaceRatio=88
storePathRootDir=/data/rocketmq/store-master
storePathCommitLog=/data/rocketmq/store-master/commitlog
maxMessageSize=65536
flushCommitLogLeastPages=4
flushConsumeQueueLeastPages=2
flushCommitLogThoroughInterval=10000
flushConsumeQueueThoroughInterval=60000
checkTransactionMessageEnable=false
sendMessageThreadPoolNums=128
pullMessageThreadPoolNums=128
brokerRole=SYNC_MASTER
flushDiskType=ASYNC_FLUSH
```

4. 节点2从192.168.0.228节点配置broker-slave

```
# vim /opt/rocketmq-4.4.0/conf/broker-slave.conf
brokerClusterName=SyncCluster
brokerName=broker-02
brokerId=1
#本节点ip地址
brokerIP1=192.168.0.228
#配置所有节点的namesrv地址
namesrvAddr=192.168.0.190:9876;192.168.0.228:9876;192.168.0.168:9876
defaultTopicQueueNums=4
autoCreateTopicEnable=true
autoCreateSubscriptionGroup=true
listenPort=10921
```

```
deleteWhen=04
fileReservedTime=72
mapedFileSizeCommitLog=1073741824
mapedFileSizeConsumeQueue=300000
destroyMapedFileInterval=120000
redeleteHangedFileInterval=120000
diskMaxUsedSpaceRatio=88
storePathRootDir=/data/rocketmq-4.4.0/data/store-slave
storePathCommitLog=/data/rocketmq-4.4.0/data/store-slave/commitlog
maxMessageSize=65536
flushCommitLogLeastPages=4
flushConsumeQueueLeastPages=2
flushCommitLogThoroughInterval=10000
flushConsumeQueueThoroughInterval=60000
checkTransactionMessageEnable=false
sendMessageThreadPoolNums=128
pullMessageThreadPoolNums=128
brokerRole=SLAVE
flushDiskType=ASYNC_FLUSH
```

5. 节点3主192.168.0.168节点配置broker-master

```
# vim /opt/rocketmq-4.4.0/conf/broker-master.conf
brokerClusterName=SyncCluster
brokerName=broker-03
brokerId=0
#本节点ip地址
brokerIP1=192.168.0.168
#配置所有节点的namesrv地址
namesrvAddr=192.168.0.190:9876;192.168.0.228:9876;192.168.0.168:9876
defaultTopicQueueNums=4
autoCreateTopicEnable=true
autoCreateSubscriptionGroup=true
listenPort=10911
deleteWhen=04
fileReservedTime=72
mapedFileSizeCommitLog=1073741824
mapedFileSizeConsumeQueue=300000
destroyMapedFileInterval=120000
redeleteHangedFileInterval=120000
diskMaxUsedSpaceRatio=88
storePathRootDir=/data/rocketmq/store-master
storePathCommitLog=/data/rocketmq/store-master/commitlog
maxMessageSize=65536
flushCommitLogLeastPages=4
flushConsumeQueueLeastPages=2
flushCommitLogThoroughInterval=10000
flushConsumeQueueThoroughInterval=60000
checkTransactionMessageEnable=false
sendMessageThreadPoolNums=128
pullMessageThreadPoolNums=128
brokerRole=SYNC_MASTER
flushDiskType=ASYNC_FLUSH
```

6. 节点3从192.168.0.168节点配置broker-slave

```
# vim /opt/rocketmq-4.4.0/conf/broker-slave.conf
brokerClusterName=SyncCluster
brokerName=broker-03
brokerId=1
#本节点ip地址
brokerIP1=192.168.0.168
#配置所有节点的namesrv地址
namesrvAddr=192.168.0.190:9876;192.168.0.228:9876;192.168.0.168:9876
defaultTopicQueueNums=4
autoCreateTopicEnable=true
autoCreateSubscriptionGroup=true
listenPort=10921
deleteWhen=04
fileReservedTime=72
mapedFileSizeCommitLog=1073741824
mapedFileSizeConsumeQueue=300000
```

```
destroyMapedFileInterval=120000
redeleteHangedFileInterval=120000
diskMaxUsedSpaceRatio=88
storePathRootDir=/data/rocketmq/store-slave
storePathCommitLog=/data/rocketmq/store-slave/commitlog
maxMessageSize=65536
flushCommitLogLeastPages=4
flushConsumeQueueLeastPages=2
flushCommitLogThoroughInterval=10000
flushConsumeQueueThoroughInterval=60000
checkTransactionMessageEnable=false
sendMessageThreadPoolNums=128
pullMessageThreadPoolNums=128
brokerRole=SLAVE
flushDiskType=ASYNC_FLUSH
```

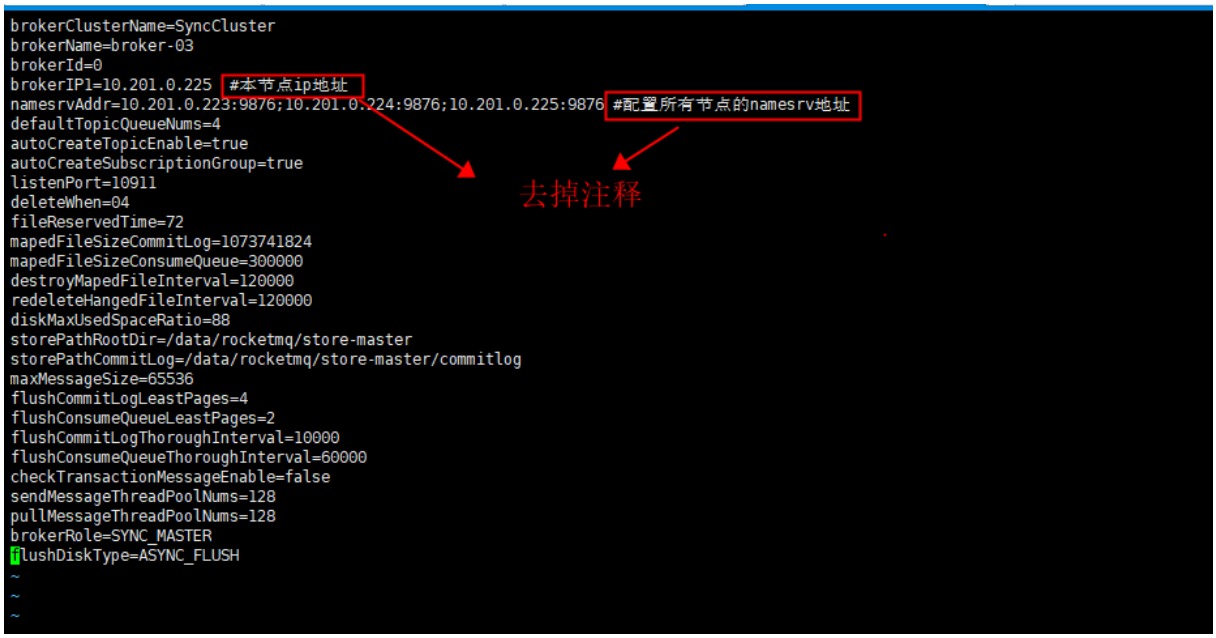
3.5.3 配置 JVM 启动参数

1. 默认的jvm启动参数内存都比较大，可以根据机器情况适当调整，不然有可能会启动失败；有3个脚本需要修改（注意内存**单位m不能缺少**，否则会启动失败）：

```
# vim /opt/rocketmq-4.4.0/bin/runserver.sh
JAVA_OPT=&quot;${JAVA_OPT} -server -Xms512m -Xmx512m -Xmn640m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=320m&quot;
# vim /opt/rocketmq-4.4.0/bin/runbroker.sh
JAVA_OPT=&quot;${JAVA_OPT} -server -Xms512m -Xmx512m -Xmn256m&quot;
# vim /opt/rocketmq-4.4.0/bin/tools.sh
JAVA_OPT=&quot;${JAVA_OPT} -server -Xms512m -Xmx512m -Xmn256m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m&quot;
```

2. 注意事项：注意去掉master/slave这些注册，特别注意IP后面不能有空格

图 3-65 注意事项



3.5.4 启动 RocketMq

1. 创建rocketmq-**namesrv**

```
# vim /etc/systemd/system/rocketmq-namesrv.service
[Unit]
Description=rocketmq-namesrv
Documentation=http://mirror.bit.edu.cn/apache/rocketmq/
After=network.target
```

```
[Service]
Type=sample
Environment=JAVA_HOME=/opt/jdk1.8.0_131
User=root
ExecStart=/opt/rocketmq-4.4.0/bin/mqnamesrv > /data/rocketmq-4.4.0/logs/mqnamesrv.log 2>&1
ExecReload=/bin/kill -s HUP $MAINPID
ExecStop=/bin/kill -s QUIT $MAINPID
Restart=0
LimitNOFILE=65536
[Install]
WantedBy=multi-user.target
```

2. 创建broker-master

```
# vim /etc/systemd/system/rocketmq-master.service
[Unit]
Description=rocketmq-master
Documentation=http://mirror.bit.edu.cn/apache/rocketmq/
After=network.target
[Service]
Type=sample
Environment=JAVA_HOME=/opt/jdk1.8.0_131
User=root
ExecStart=/opt/rocketmq-4.4.0/bin/mqbroker -c /opt/rocketmq-4.4.0/conf/broker-master.conf >/data/
rocketmq-4.4.0/logs/mqbroker-master.log 2>&1
ExecReload=/bin/kill -s HUP $MAINPID
ExecStop=/bin/kill -s QUIT $MAINPID
Restart=0
LimitNOFILE=65536
[Install]
WantedBy=multi-user.target
```

3. 创建broker-slave

```
# vim /etc/systemd/system/rocketmq-slave.service
[Unit]
Description=rocketmq-slave
Documentation=http://mirror.bit.edu.cn/apache/rocketmq/
After=network.target
[Service]
Type=sample
Environment=JAVA_HOME=/opt/jdk1.8.0_131
User=root
ExecStart=/opt/rocketmq-4.4.0/bin/mqbroker -c /opt/rocketmq-4.4.0/conf/broker-slave.conf >/data/
rocketmq-4.4.0/logs/mqbroker-slave.log 2>&1
ExecReload=/bin/kill -s HUP $MAINPID
ExecStop=/bin/kill -s QUIT $MAINPID
Restart=0
LimitNOFILE=65536
[Install]
WantedBy=multi-user.target
```

4. 设置开机自启

   由于rocketmq有启动顺序依赖，因此要写入rc

```
# vim /etc/rc.d/rc.local
systemctl start rocketmq-namesrv
systemctl start rocketmq-master
systemctl start rocketmq-slave
```

图 3-66 设置开机自启

```
#!/bin/bash
# THIS FILE IS ADDED FOR COMPATIBILITY PURPOSES
#
# It is highly advisable to create own systemd services or udev rules
# to run scripts during boot instead of using this file.
#
# In contrast to previous versions due to parallel execution during boot
# this script will NOT be run after all other services.
#
# Please note that you must run 'chmod +x /etc/rc.d/rc.local' to ensure
# that this script will be executed during boot.

touch /var/lock/subsys/local

systemctl start rocketmq-namesrv
systemctl start rocketmq-master
systemctl start rocketmq-slave
```

5. 启动服务

   重新加载systemctl配置

   ```
   # systemctl daemon-reload
   ```

   启动顺序namesrv -> master -> slave.

   ```
   # systemctl start rocketmq-namesrv && systemctl enable rocketmq-namesrv
   # systemctl start rocketmq-master && systemctl enable rocketmq-master
   # systemctl start rocketmq-slave && systemctl enable rocketmq-slave
   ```

   说明

   如果要停止，顺序与启动顺序相反。

6. 检查状态

   ```
   # systemctl status rocketmq-namesrv
   # systemctl status rocketmq-master
   # systemctl status rocketmq-slave
   ```

图 3-67 检查状态

```
[root@dev-hwc-gy1-2048-daas-test-220-deepexi conf]# systemctl status rocketmq-namesrv
● rocketmq-namesrv.service - rocketmq-nameserver
   Loaded: loaded (/etc/systemd/system/rocketmq-namesrv.service; disabled; vendor preset: disabled)
   Active: active (running) since Fri 2021-02-26 16:54:26 CST; 31s ago
     Docs: http://mirror.bit.edu.cn/apache/rocketmq/
 Main PID: 15269 (mqnamesrv)
   CGroup: /system.slice/rocketmq-namesrv.service
           ├─15269 /bin/sh /opt/rocketmq-4.4.0/bin/mqnamesrv > /data/rocketmq-4.4.0/logs/mqnamesrv.log 2>&1
           ├─15273 sh /opt/rocketmq-4.4.0/bin/runserver.sh org.apache.rocketmq.namesrv.NamesrvStartup > /data/rocketmq-4.4.0/logs/mqnamesrv.log 2>&1
           └─15275 /opt/jdk1.8.0_131/bin/java -server -Xms512m -Xmx512m -Xmn256m -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -XX:+UseConcMarkSwe

Feb 26 16:54:26 dev-hwc-gy1-2048-daas-test-220-deepexi systemd[1]: Started rocketmq-nameserver.
Feb 26 16:54:26 dev-hwc-gy1-2048-daas-test-220-deepexi mqnamesrv[15269]: Java HotSpot(TM) 64-Bit Server VM warning: Using the DefNew young collector
Feb 26 16:54:26 dev-hwc-gy1-2048-daas-test-220-deepexi mqnamesrv[15269]: Java HotSpot(TM) 64-Bit Server VM warning: UseCMSCompactAtFullCollection is
Feb 26 16:54:27 dev-hwc-gy1-2048-daas-test-220-deepexi mqnamesrv[15269]: The Name Server boot success. serializeType=JSON
[root@dev-hwc-gy1-2048-daas-test-220-deepexi conf]# systemctl status rocketmq-master
● rocketmq-master.service - rocketmq-master
   Loaded: loaded (/etc/systemd/system/rocketmq-master.service; disabled; vendor preset: disabled)
   Active: active (running) since Fri 2021-02-26 16:54:35 CST; 30s ago
     Docs: http://mirror.bit.edu.cn/apache/rocketmq/
 Main PID: 15325 (mqbroker)
   CGroup: /system.slice/rocketmq-master.service
           ├─15325 /bin/sh /opt/rocketmq-4.4.0/bin/mqbroker -c /opt/rocketmq-4.4.0/conf/broker-master.conf >/data/rocketmq-4.4.0/logs/mqbroker-master
           ├─15329 sh /opt/rocketmq-4.4.0/bin/runbroker.sh org.apache.rocketmq.broker.BrokerStartup -c /opt/rocketmq-4.4.0/conf/broker-master.conf >/
           └─15332 /opt/jdk1.8.0_131/bin/java -server -Xms512m -Xmx512m -Xmn256m -XX:+UseG1GC -XX:G1HeapRegionSize=16m -XX:G1ReservePercent=25 -XX:In

Feb 26 16:54:35 dev-hwc-gy1-2048-daas-test-220-deepexi systemd[1]: Started rocketmq-master.
Feb 26 16:54:36 dev-hwc-gy1-2048-daas-test-220-deepexi mqbroker[15325]: The broker[broker-01, 10.201.0.223:10911] boot success. serializeType=JSON an
[root@dev-hwc-gy1-2048-daas-test-220-deepexi conf]# systemctl status rocketmq-slave
● rocketmq-slave.service - rocketmq-slave
   Loaded: loaded (/etc/systemd/system/rocketmq-slave.service; disabled; vendor preset: disabled)
   Active: active (running) since Fri 2021-02-26 16:54:39 CST; 29s ago
     Docs: http://mirror.bit.edu.cn/apache/rocketmq/
 Main PID: 15414 (mqbroker)
   CGroup: /system.slice/rocketmq-slave.service
           ├─15414 /bin/sh /opt/rocketmq-4.4.0/bin/mqbroker -c /opt/rocketmq-4.4.0/conf/broker-slave.conf >/data/rocketmq-4.4.0/logs/mqbroker-slave.l
           ├─15418 sh /opt/rocketmq-4.4.0/bin/runbroker.sh org.apache.rocketmq.broker.BrokerStartup -c /opt/rocketmq-4.4.0/conf/broker-slave.conf >/d
           └─15421 /opt/jdk1.8.0_131/bin/java -server -Xms512m -Xmx512m -Xmn256m -XX:+UseG1GC -XX:G1HeapRegionSize=16m -XX:G1ReservePercent=25 -XX:In

Feb 26 16:54:39 dev-hwc-gy1-2048-daas-test-220-deepexi systemd[1]: Started rocketmq-slave.
Feb 26 16:54:40 dev-hwc-gy1-2048-daas-test-220-deepexi mqbroker[15414]: The broker[broker-01, 10.201.0.223:10921] boot success. serializeType=JSON an
```

7. 看到上面结果，说明启动成功。
8. 附加：另一种启动方式

   分别启动3个节点的namesrv和broker，启动顺序namesrv ---> broker。

   启动namesrv

   ```
   # nohup sh /opt/rocketmq-4.4.0/bin/mqnamesrv > /data/rocketmq-4.4.0/logs/mqnamesrv.log 2>&1 &
   ```

   启动broker-master

   ```
   # nohup sh /opt/rocketmq-4.4.0/bin/mqbroker -c /opt/rocketmq-4.4.0/conf/broker-master.conf >/data/
   rocketmq-4.4.0/logs/mqbroker-master.log 2>&1 &
   ```

   启动broker-slave

```
# nohup sh /opt/rocketmq-4.4.0/bin/mqbroker -c /opt/rocketmq-4.4.0/conf/broker-slave.conf >/data/rocketmq-4.4.0/logs/mqbroker-slave.log 2>&1 &
```

说明

停止顺序则与启动顺序相反。

3.5.5 部署 rocketmq-console 控制台

部署rocketmq-console控制台（前提是服务器先安装好jdk，如无请参考后面安装jdk部分）

新建目录mkdir /opt/rocketmq-console，并上传安装包rocketmq-console-ng-2.0.0.jar到该目录。

1. rocketmq-console，创建目录，并上传rocketmq-console

```
# mkdir /opt/rocketmq-console
# mkdir /data/rocketmq-console/
# cd /opt/rocketmq-console
# cp /soft/rocketmq/rocketmq-console-ng-2.0.0.jar ./
# vim /etc/systemd/system/rocketmq-console.service
[Unit]
Description=rocketmq-console
After=syslog.target network.target remote-fs.target nss-lookup.target
[Service]
Type=simple
ExecStart=/opt/jdk1.8.0_251/bin/java -jar /opt/rocketmq-console/rocketmq-console-ng-2.0.0.jar --server.port=38060 --rocketmq.config.namesrvAddr=192.168.0.190:9876;192.168.0.228:9876;192.168.0.168:9876 --rocketmq.config.isVIPChannel=false
Restart=always
PrivateTmp=true
[Install]
WantedBy=multi-user.target
```

2. 重新加载systemctl配置

```
# systemctl daemon-reload
```

3. 设置跟随linux启动，并启动rocketmq-console

```
# systemctl start rocketmq-console && systemctl enable rocketmq-console
```

4. 或用以下方式启动服务

```
#nohup java -jar ./rocketmq-console-ng-2.0.0.jar --server.port=&amp;quot;38060&amp;quot; --rocketmq.config.namesrvAddr=&amp;quot;192.168.0.190:9876;192.168.0.228:9876;192.168.0.168:9876&amp;quot; &amp;gt;/data/rocketmq-console/mqconsole.log 2&amp;gt;&amp;amp;1 &amp;amp;
```

5. 访问mq控制台：

   http://124.71.107.141:38060/#/cluster

图 3-68 访问 mq 控制台 1

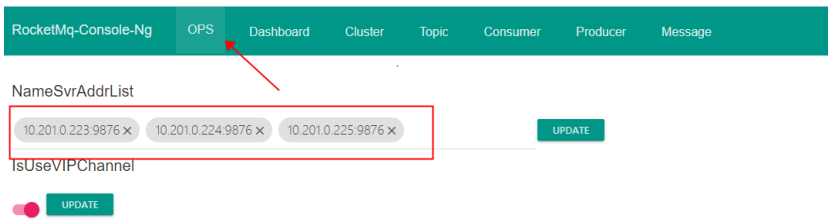
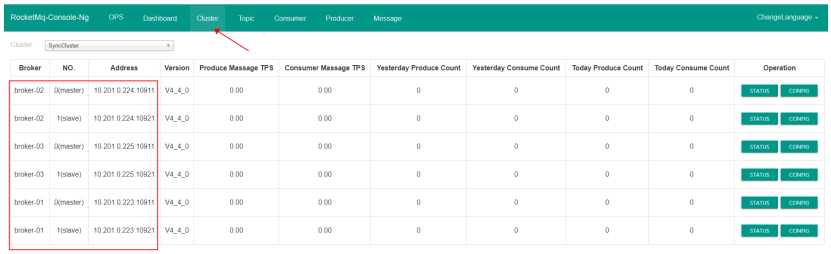


图 3-69 访问 mq 控制台 2



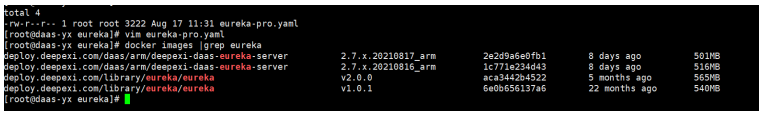
3.6 Eureka 部署

导入镜像

上传编译arm版本的镜像deploy.deepexi.com/daas/arm/deepexi-daas-eureka-server:2.7.x.20210817_arm，安装到服务器，然后导入镜像

deepexi-daas-eureka- server:2.7.x.20210817_arm

图 3-70 导入镜像

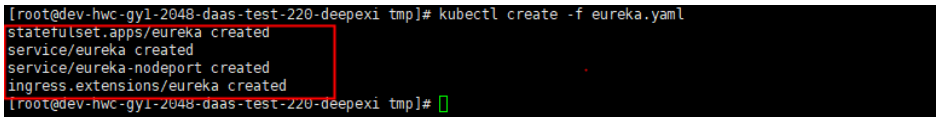


创建 Eureka 集群

1. 通过k8s创建，上传eureka-pro.yaml文件到k8s master，然后执行：

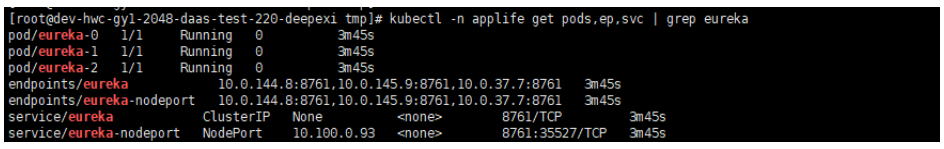
```
# kubectl create -f eureka-pro.yaml
```

图 3-71 执行 1




```
# kubectl -n applife get pods,ep,svc | grep eureka
```

图 3-72 执行 2



2. 看到上面提示/结果，表示创建成功。

   附：eureka.yaml文件内容

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
name: deepexi-daas-eureka-server
```

```
namespace: applife
labels:
app: deepexi-daas-eureka-server
spec:
serviceName: deepexi-daas-eureka-server
replicas: 3
podManagementPolicy: &amp;quot;Parallel&amp;quot;
selector:
matchLabels:
app: deepexi-daas-eureka-server
template:
metadata:
namespace: applife
labels:
app: deepexi-daas-eureka-server
openkube.deepexi.cloud/logtype: java
spec:
containers:
- name: deepexi-daas-eureka-server
imagePullPolicy: Always
image: deploy.deepexi.com/daas/arm/deepexi-daas-eureka-server:2.7.x.20210817_arm
command: [&amp;quot;/bin/sh&amp;quot;,&amp;quot;-c&amp;quot;,&amp;quot;java -Djava.security.egd=file:/dev/./urandom -
Duser.timezone=Asia/Shanghai -Denv=DEV  -XX:+HeapDumpOnOutOfMemoryError -jar /home/
demo.jar --spring.security.user.name=admin --spring.security.user.password=deepexi --server.port=8761
--eureka.client.register-with-eureka=true --eureka.client.fetch-registry=true --
eureka.client.serviceUrl.defaultZone=http://admin:deepexi@deepexi-daas-eureka-server-0.deepexi-
daas-eureka-server.applife.svc.cluster.local:8761/eureka/,http://admin:deepexi@deepexi-daas-eureka-
server-1.deepexi-daas-eureka-server.applife.svc.cluster.local:8761/eureka/,http://
admin:deepexi@deepexi-daas-eureka-server-2.deepexi-daas-eureka-server.applife.svc.cluster.local:8761/
eureka/ --eureka.instance.hostname=
$MY_POD_NAME.$MY_IN_SERVICE_NAME.$MY_POD_NAMESPACE.$MY_K8S_DNS_NAME"]
env:
- name: TZ
value: Asia/Shanghai
- name: MY_K8S_DNS_NAME
value: svc.cluster.local
- name: MY_IN_SERVICE_NAME
value: deepexi-daas-eureka-server
- name: MY_NODE_NAME
valueFrom:
fieldRef:
fieldPath: spec.nodeName
- name: MY_POD_NAME
valueFrom:
fieldRef:
fieldPath: metadata.name
- name: MY_POD_NAMESPACE
valueFrom:
fieldRef:
fieldPath: metadata.namespace
- name: MY_POD_IP
valueFrom:
fieldRef:
fieldPath: status.podIP
#volumeMounts:
#- name: nfs-pvc
#mountPath: &amp;quot;/etc/daas/kerberos&amp;quot;
#volumes:
#- name: nfs-pvc
#persistentVolumeClaim:
#claimName: daas-claim
---
apiVersion: v1
kind: Service
metadata:
name: deepexi-daas-eureka-server
namespace: applife
labels:
app: deepexi-daas-eureka-server
```

```
spec:
type: ClusterIP
ports:
- port: 8761
targetPort: 8761
name: http
protocol: TCP
selector:
app: deepexi-daas-eureka-server
---
apiVersion: v1
kind: Service
metadata:
name: deepexi-daas-eureka-server-nodeport
namespace: applife
labels:
app: deepexi-daas-eureka-server
spec:
type: NodePort
ports:
- port: 8761
nodePort: 38761
name: http
protocol: TCP
targetPort: 8761
selector:
app: deepexi-daas-eureka-server
```

3.7 Zookeeper 部署

前置条件

安装JDK

3台机器2888和3888端口能互通

安装目录约定

- 安装目录：/opt/zookeeper-3.6.1
- 配置文件：/opt/zookeeper-3.6.1/conf/zoo.cfg
- 数据目录：/data/zookeeper-3.6.1/data
- 日志目录：/data/zookeeper-3.6.1/logs

安装配置

三台zookeeper配置，解压并创建目录

上传安装apache-zookeeper-3.6.1-bin.tar.gz包到**3**台服务器，分别都解压并创建目录

```
# cd /soft/zookeeper/
# tar -vxf apache-zookeeper-3.6.1-bin.tar.gz
# mv apache-zookeeper-3.6.1-bin /opt/zookeeper-3.6.1
# mkdir -p /data/zookeeper-3.6.1/{data,logs}
```

1. Zookeeper节点一配置

   在节点一机器执行以下操作

   添加集群节点身份标识

   ```
   # echo &quot;1&quot; &gt;/data/zookeeper-3.6.1/data/myid
   ```

新增配置集群信息

```
# vim /opt/zookeeper-3.6.1/conf/zoo.cfg
clientPort=2181
tickTime=2000
initLimit=20
syncLimit=10
dataDir=/data/zookeeper-3.6.1/data
dataLogDir=/data/zookeeper-3.6.1/logs
#2888是leader与flower之间的访问端口(心跳端口)，3888是投票选举leader端口(数据端口)
#配置3个zookeeper节点地址
server.1=192.168.0.190:2888:3888
server.2=192.168.0.228:2888:3888
server.3=10.201.0.168:2888:3888
#开启四字命令
4lw.commands.whitelist=*
```

配置日志目录

```
# sed -i 's/ZOO_LOG_DIR\=\&quot;\$ZOOKEEPER_PREFIX\/logs\&quot;/ZOO_LOG_DIR\=\&quot;\/data
\/zookeeper-3.6.1\/logs\&quot;/g' /opt/zookeeper-3.6.1/bin/zkEnv.sh
```

2. Zookeeper节点二配置

在节点二机器执行以下操作

添加集群节点身份标识

```
# echo &quot;2&quot; &gt;/data/zookeeper-3.6.1/data/myid
```

新增配置集群信息

```
# vim /opt/zookeeper-3.6.1/conf/zoo.cfg
clientPort=2181
tickTime=2000
initLimit=20
syncLimit=10
dataDir=/data/zookeeper-3.6.1/data
dataLogDir=/data/zookeeper-3.6.1/logs
#2888是leader与flower之间的访问端口(心跳端口)，3888是投票选举leader端口(数据端口)
#配置3个zookeeper节点地址
server.1=192.168.0.190:2888:3888
server.2=192.168.0.228:2888:3888
server.3=10.201.0.168:2888:3888
#开启四字命令
4lw.commands.whitelist=*
```

配置日志目录

```
# sed  -i 's/ZOO_LOG_DIR\=\&quot;\$ZOOKEEPER_PREFIX\/logs\&quot;/ZOO_LOG_DIR\=\&quot;\/data
\/zookeeper-3.6.1\/logs\&quot;/g' /opt/zookeeper-3.6.1/bin/zkEnv.sh
```

3. Zookeeper节点三配置

在节点三机器执行以下操作

添加集群节点身份标识

```
# echo &quot;3&quot; &gt;/data/zookeeper-3.6.1/data/myid
```

新增配置集群信息

```
# vim /opt/zookeeper-3.6.1/conf/zoo.cfg
clientPort=2181
tickTime=2000
initLimit=20
syncLimit=10
dataDir=/data/zookeeper-3.6.1/data
dataLogDir=/data/zookeeper-3.6.1/logs
#2888是leader与flower之间的访问端口(心跳端口)，3888是投票选举leader端口(数据端口)
#配置3个zookeeper节点地址
server.1=192.168.0.190:2888:3888
server.2=192.168.0.228:2888:3888
server.3=10.201.0.168:2888:3888
#开启四字命令
4lw.commands.whitelist=*
```

配置日志目录

```
# sed  -i 's/ZOO_LOG_DIR\=\&quot;\$ZOOKEEPER_PREFIX\/logs\&quot;/ZOO_LOG_DIR\=\&quot;\/data\/zookeeper-3.6.1\/logs\&quot;/g' /opt/zookeeper-3.6.1/bin/zkEnv.sh
```

启动 zookeeper

1. 3个节点分别创建zookeeper系统启动文件

```
# vim /etc/systemd/system/zookeeper.service
[Unit]
Description=zkServer
After=network.target remote-fs.target nss-lookup.target
ConditionPathExists=/opt/zookeeper-3.6.1/conf/zoo.cfg
[Service]
Type=forking
Environment=JAVA_HOME=/opt/jdk1.8.0_131
ExecStart=/opt/zookeeper-3.6.1/bin/zkServer.sh start
ExecStop=/opt/zookeeper-3.6.1/bin/zkServer.sh stop
PrivateTmp=true
[Install]
WantedBy=multi-user.target
```

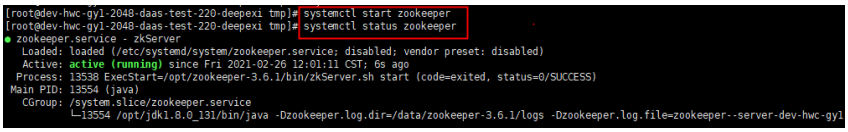
2. 重新加载systemctl配置

```
# systemctl daemon-reload
```

3. 设置跟随linux启动，并启动zookeeper

```
# systemctl start zookeeper && systemctl enable zookeeper
```

图 3-73 启动

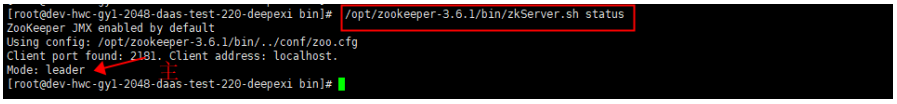


4. 查看状态

```
# /opt/zookeeper-3.6.1/bin/zkServer.sh status
```

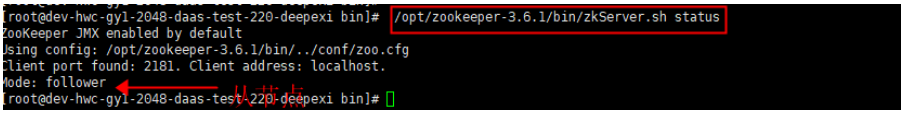
主节点

图 3-74 启动



从节点

图 3-75 启动



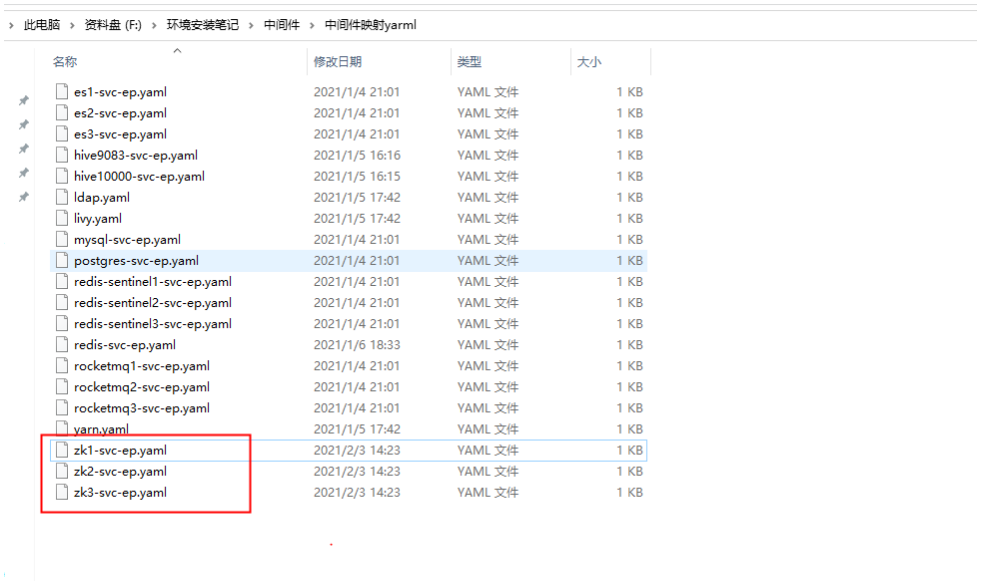
3.8 id-generator 部署

配置 zookeeper

在使用id-generator前，要先确保已经安装zookeeper，因为id-generator依赖zookeeper。

上传zookeeper的yaml文件到k8s宿主机

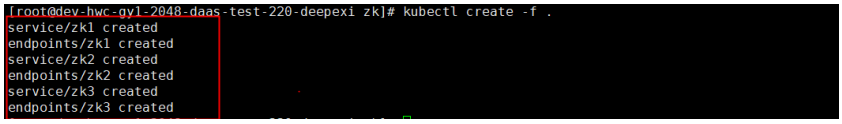
图 3-76 启动



创建endpoint

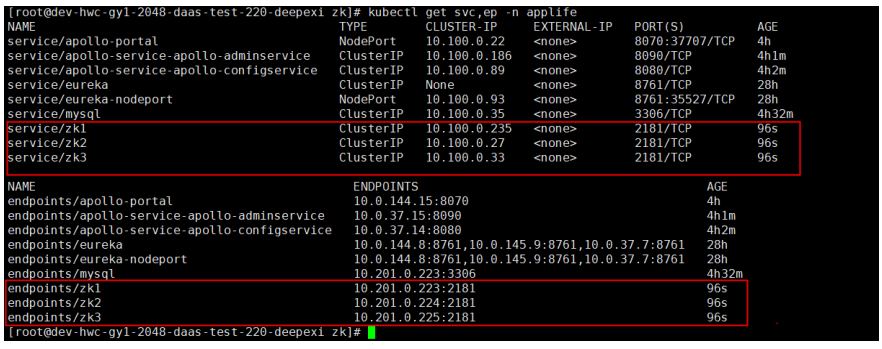
```
# kubectl create -f .
```

图 3-77 创建 1




```
# kubectl get svc,ep -n applife
```

图 3-78 创建 2

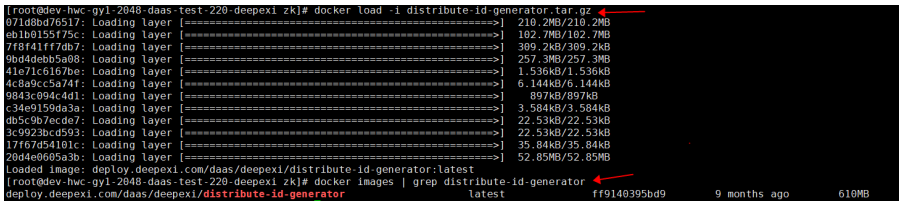


导入镜像

上传distribute-id-generator.tar.gz到k8s宿主机，执行

```
# docker load -i distribute-id-generator.tar.gz
```

图 3-79 执行



看到上面结果，表明镜像导入成功。

创建应用副本

通过k8s master或者华为云cce控制台导入yaml文件。

图 3-80 导入 yaml 文件


```
# kubectl apply -f deepexi-daas-distribute-id-generator.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
name: deepexi-distribute-id-generator-deploy
namespace: daas
spec:
replicas: 1
template:
metadata:
labels:
app: deepexi-distribute-id-generator-deploy
openkube.deepexi.cloud/logtype: java
spec:
containers:
- name: deepexi-distribute-id-generator-deploy
imagePullPolicy: Always
image: swr.cn-south-1.myhuaweicloud.com/daas/distribute-id-generator:v2.7.x_20210817_arm64
command: [&quot;/bin/sh&quot;]
args: [&quot;-c&quot;,&quot;cd /home; java -Djava.security.egd=file:/dev/./urandom -Duser.timezone=Asia/
Shanghai -XX:+HeapDumpOnOutOfMemoryError -jar ./demo.jar --leaf.snowflake.zk.address=zk1.applife --
dubbo.registry.address=zookeeper://zk1.applife?backup=zk2.applife,zk3.applife --server.port=38103&quot;]
imagePullSecrets:
- name: default-secret
selector:
matchLabels:
app: deepexi-distribute-id-generator-deploy
---
apiVersion: v1
kind: Service
metadata:
name: deepexi-distribute-id-generator-service
namespace: daas
spec:
type: NodePort
ports:
- port: 38103
nodePort: 38103
```

```
targetPort: 38103
selector:
app: deepexi-distribute-id-generator-deploy
```

验证

1. 在k8s宿主机上，查看pod的地址，

   说明

   类型为ClusterIp是不能通过外网访问的

   ```
   # kubectl get svc -n daas
   ```

   图 3-81 查看 pod 的地址

   ```
   NAME                                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)           AGE
   deepexi-daas-bigdata-develop-service  ClusterIP   10.247.231.73    <none>        30009/TCP         5d22h
   deepexi-daas-datasource-service       ClusterIP   10.247.220.45    <none>        30007/TCP         4d22h
   deepexi-daas-iam-admin-service        ClusterIP   10.247.26.5      <none>        30017/TCP         31h
   deepexi-daas-iam-openapi-service      ClusterIP   10.247.127.84    <none>        30018/TCP         31h
   deepexi-daas-iam-sso-service          ClusterIP   10.247.147.171   <none>        30016/TCP         31h
   deepexi-daas-inspector-service        ClusterIP   10.247.201.91    <none>        30019/TCP         5d22h
   deepexi-daas-log-service              ClusterIP   10.247.202.143   <none>        30029/TCP         6d3h
   deepexi-daas-metadata2-service        ClusterIP   10.247.85.132    <none>        30028/TCP         4d23h
   deepexi-daas-metrics-service          ClusterIP   10.247.207.140   <none>        30004/TCP         4d22h
   deepexi-daas-model-service            ClusterIP   10.247.186.75    <none>        30026/TCP         2d2h
   deepexi-daas-security-service         ClusterIP   10.247.217.6     <none>        30006/TCP         6h23m
   deepexi-daas-server-service           ClusterIP   10.247.1.67      <none>        30023/TCP         5d6h
   deepexi-daas-service-admin-service    ClusterIP   10.247.248.217   <none>        30005/TCP         5d6h
   deepexi-daas-service-datasource-service ClusterIP 10.247.23.0      <none>        30024/TCP         5d6h
   deepexi-daas-service-oauth-service    ClusterIP   10.247.194.145   <none>        30025/TCP         5d6h
   deepexi-distribute-id-generator-service NodePort  10.247.23.210    <none>        38103:38103/TCP   8d
   deepexi-message-admin-service         NodePort    10.247.79.6      <none>        30033:30033/TCP   31h
   [root@daas-yx generator]#
   ```

2. 在K8s宿主机上，执行：

   ```
   # curl http://192.168.0.69:38103/distribute-id-generator/api/v1/snowflake/id/32
   ```

   图 3-82 执行

   ```
   deepexi-message-admin-service            NodePort    10.247.79.6      <none>        30033:30033/TCP   31h
   [root@daas-yx generator]# curl http://192.168.0.69:38103/distribute-id-generator/api/v1/snowflake/id/32
   1430470811109564495[root@daas-yx generator]#
   ```

   看到上面返回的结果，说明部署成功，否则检查一下zookeeper配置是否正确。

3.9 Apollo 部署

上传apollo-adminservice.tar.gz、apollo-configservice.tar.gz、apollo-portal.tar.gz到/soft/apollo下

导入镜像

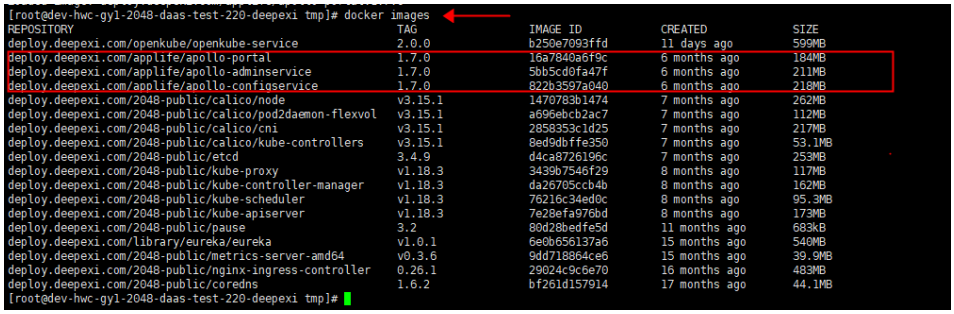
```
#cd /soft/apollo
# docker load -i apollo-adminservice.tar.gz
# docker load -i apollo-configservice.tar.gz
# docker load -i apollo-portal.tar.gz
```

图 3-83 导入镜像 1

```
[root@dev-hwc-gyl-2048-daas-test-220-deepexi tmp]# docker load -i apollo-adminservice.tar.gz
f1b5933fe4b5: Loading layer [==================================================>]  5.796MB/5.796MB
9b9b7f3d56a0: Loading layer [==================================================>]  3.584kB/3.584kB
edd61588d126: Loading layer [==================================================>]  80.28MB/80.28MB
35a6d6e58f1c: Loading layer [==================================================>]  13.56MB/13.56MB
c40ca755ba53: Loading layer [==================================================>]   54.5MB/54.5MB
d6827e67c353: Loading layer [==================================================>]  61.11MB/61.11MB
Loaded image: deploy.deepexi.com/applife/apollo-adminservice:1.7.0
[root@dev-hwc-gyl-2048-daas-test-220-deepexi tmp]# docker load -i apollo-configservice.tar.gz
25dcda0eccfa: Loading layer [==================================================>]  13.56MB/13.56MB
148f5b1307e4: Loading layer [==================================================>]  57.81MB/57.81MB
24654231cf6e: Loading layer [==================================================>]  64.78MB/64.78MB
Loaded image: deploy.deepexi.com/applife/apollo-configservice:1.7.0
[root@dev-hwc-gyl-2048-daas-test-220-deepexi tmp]# docker load -i apollo-portal.tar.gz
e5aab5b16d77: Loading layer [==================================================>]  13.56MB/13.56MB
d6c819098c25: Loading layer [==================================================>]  41.72MB/41.72MB
8d29d1b74f25: Loading layer [==================================================>]  46.33MB/46.33MB
```

```
# docker images
```

图 3-84 导入镜像 2



```
[root@dev-hwc-gy1-2048-daas-test-220-deepexi tmp]# docker images
REPOSITORY                                                TAG                 IMAGE ID            CREATED             SIZE
deploy.deepexi.com/openkube/openkube-service              2.0.0               b250e7093ffd        11 days ago         599MB
deploy.deepexi.com/applife/apollo-portal                  1.7.0               16a7840a6f9c        6 months ago        184MB
deploy.deepexi.com/applife/apollo-adminservice            1.7.0               5bb5cd0fa47f        6 months ago        211MB
deploy.deepexi.com/applife/apollo-configservice           1.7.0               822b3597a040        6 months ago        218MB
deploy.deepexi.com/2048-public/calico/node                v3.15.1             1470783b1474        7 months ago        262MB
deploy.deepexi.com/2048-public/calico/pod2daemon-flexvol  v3.15.1             a696ebcb2ac7        7 months ago        112MB
deploy.deepexi.com/2048-public/calico/cni                 v3.15.1             2858353c1d25        7 months ago        217MB
deploy.deepexi.com/2048-public/calico/kube-controllers    v3.15.1             8ed9dbffe350        7 months ago        53.1MB
deploy.deepexi.com/2048-public/etcd                       3.4.9               d4ca8726196c        7 months ago        253MB
deploy.deepexi.com/2048-public/kube-proxy                 v1.18.3             3439b7546f29        8 months ago        117MB
deploy.deepexi.com/2048-public/kube-controller-manager    v1.18.3             da26705ccb4b        8 months ago        162MB
deploy.deepexi.com/2048-public/kube-scheduler             v1.18.3             76216c34ed0c        8 months ago        95.3MB
deploy.deepexi.com/2048-public/kube-apiserver             v1.18.3             7e28efa976bd        8 months ago        173MB
deploy.deepexi.com/2048-public/pause                      3.2                 80d28bedfe5d        11 months ago       683kB
deploy.deepexi.com/library/eureka/eureka                  v1.0.1              6e0b656137a6        15 months ago       540MB
deploy.deepexi.com/2048-public/metrics-server-amd64       v0.3.6              9dd718864ce6        15 months ago       39.9MB
deploy.deepexi.com/2048-public/nginx-ingress-controller   0.26.1              29024c9c6e70        16 months ago       483MB
deploy.deepexi.com/2048-public/coredns                    1.6.2               bf261d157914        17 months ago       44.1MB
[root@dev-hwc-gy1-2048-daas-test-220-deepexi tmp]#
```

看到上面3个镜像，说明导入成功。

初始化数据库

登录Mysql，执行apollo建库脚本

```
# mysql -h192.168.0.230 -P3306 -uroot -pdaas2020
```

由于阿波罗是后面重新打的新版本包，数据库初始化脚本需要从官网下载最新的1.9版本

```
mysql> source /soft/apollo/apolloconfigdb.sql
mysql> source /soft/apollo/apolloportaldb.sql
```

导入 yaml

1. 修改mysql连接地址

图 3-85 修改 mysql 连接地址

2. 修改为apollo数据库地址

图 3-86 修改为 apollo 数据库地址

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql
  namespace: applife
spec:
  ports:
    - port: 3306
---
kind: Endpoints
apiVersion: v1
metadata:
  name: mysql
  namespace: applife
subsets:
  - addresses:
      - ip: 10.201.0.223
    ports:
      - port: 3306
```

3. 创建mysql数据源代理：

```
# kubectl create -f mysql-svc-ep.yaml
```

4. 导入yaml文件

图 3-87 导入 yaml 文件

导入顺序：configerservice->adminservice->portal，如下：

表 3-1 导入顺序

文件名称
apollo-service-apollo-configservice-cm.yaml
apollo-service-apollo-configservice-deployment.yaml
apollo-service-apollo-configservice-svc.yaml
apollo-service-apollo-adminservice-cm.yaml
apollo-service-apollo-adminservice-deployment.yaml
apollo-service-apollo-adminservice-svc.yaml
apollo-portal-cm.yaml
apollo-portal-deployment.yaml
apollo-portal-svc.yaml

说明

一定要按照顺序，先导入xxx-**cm**，然后在导入xxx-**deployment**和xxx-**svc**.

依次导入，导入成功后，可以看到下面3个apollo服务

图 3-88 apollo 服务

```
-rw-r--r-- 1 root root       110 Jan  1  1970 repositories
[root@daas-yx apollo]# vim mysql-svc-ep.yaml
[root@daas-yx apollo]# kubectl get pod -n applife
NAME                                          READY   STATUS    RESTARTS   AGE
apollo-portal-794b8d45dc-jprvp                1/1     Running   0          8d
apollo-service-apollo-adminservice-6656549ddc-mvfpc   1/1     Running   0          8d
apollo-service-apollo-configservice-7d8f5b8d5f-fxslt   1/1     Running   0          8d
deepexi-daas-eureka-server-0                  1/1     Running   0          8d
deepexi-daas-eureka-server-1                  1/1     Running   0          8d
deepexi-daas-eureka-server-2                  1/1     Running   0          8d
[root@daas-yx apollo]#
```

附录

1. 如果不希望通过endpoint使用数据源，将xxx-**cm**文件中的数据库连接改为Ip地址即可。

图 3-89 图示 1

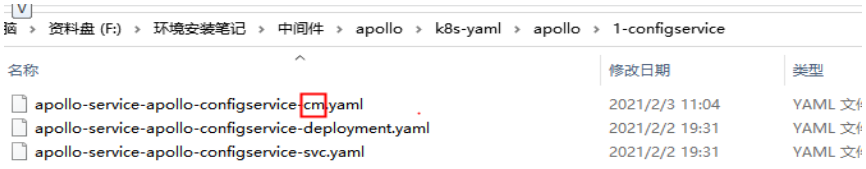


图 3-90 图示 2

图 3-91 图示 3

2. 修改为IP地址即可

图 3-92 图示 4

3.10 NFS 部署

参考资料

https://jimmysong.io/kubernetes-handbook/practice/using-nfs-for-persistent-storage.html

https://github.com/kubernetes-retired/external-storage/tree/master/nfs-client/deploy

安装 NFS-Server

1. 安装nfs
   ```
   yum install -y nfs-common nfs-utils
   ```
2. 创建数据目录
   ```
   # mkdir -p /data/nfs
   ```
3. 访问授权

   添加nfs-server访问白名单及文件存储路径
   ```
   # vim /etc/exports
   /data/nfs 192.168.0.0/24(no_root_squash,rw,sync,no_subtree_check)
   ```

   说明

   如果允许所有访问，用/data/nfs *(no_root_squash,rw,sync,no_subtree_check)
4. 加载配置
   ```
   # exportfs -rv
   ```
5. 启动NFS和rpcbind
   ```
   # systemctl enable nfs && systemctl enable rpcbind && systemctl start rpcbind nfs
   ```

图 3-93 启动 NFS 和 rpcbind1

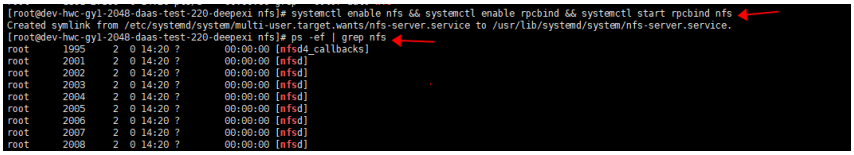
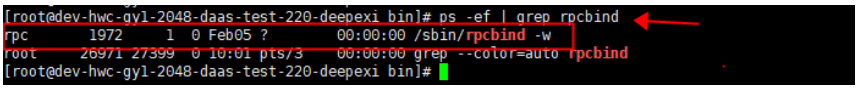


图 3-94 启动 NFS 和 rpcbind2



6. 查看 RPC 服务的注册状况

```
# rpcinfo -p 192.168.0.69
```

图 3-95 查看 RPC 服务的注册状况

```
[root@dev-hwc-gy1-2048-daas-test-220-deepexi data]# rpcinfo -p 10.201.0.223
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  47518  status
    100024    1   tcp  48506  status
    100005    1   udp  20048  mountd
    100005    1   tcp  20048  mountd
    100005    2   udp  20048  mountd
    100005    2   tcp  20048  mountd
    100005    3   udp  20048  mountd
    100005    3   tcp  20048  mountd
    100003    3   tcp   2049  nfs
    100003    4   tcp   2049  nfs
    100227    3   tcp   2049  nfs_acl
    100003    3   udp   2049  nfs
    100003    4   udp   2049  nfs
    100227    3   udp   2049  nfs_acl
    100021    1   udp  39815  nlockmgr
    100021    3   udp  39815  nlockmgr
    100021    4   udp  39815  nlockmgr
    100021    1   tcp  43787  nlockmgr
    100021    3   tcp  43787  nlockmgr
    100021    4   tcp  43787  nlockmgr
```

7. showmount测试

```
# showmount -e 192.168.0.69
```

图 3-96 showmount 测试

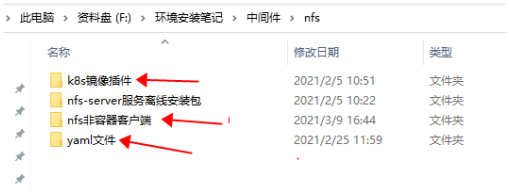
```
Last login: Thu Aug 26 15:38:48 2021 from 192.168.0.44
[root@daas2021-46750-nsw30 ~]# showmount -e 192.168.0.69
Export list for 192.168.0.69:
/data/nfs *
[root@daas2021-46750-nsw30 ~]#
```

K8s 配置 NFS-Client

上传“k8s镜像插件”客户端到K8s的所有Node节点服务器的/soft/nfs，并安装。

上传“yaml文件”到k8s的控制节点服务器。

图 3-97 图示 1



1. 安装nfs-client

   上传“nfs非容器客户端”到3台node节点，

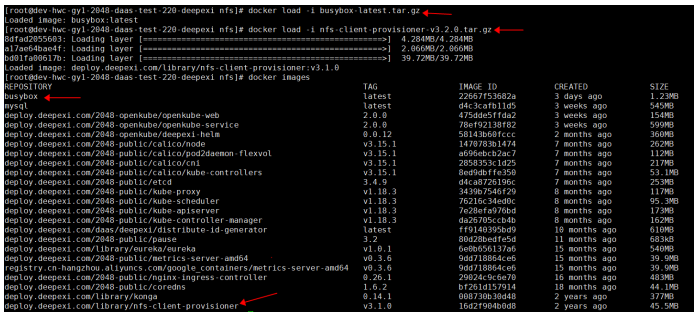
```
yum install -y nfs-common nfs-utils rpcbind
```

2. Node导入镜像

   上传“k8s镜像插件”到3台node节点，并导入镜像

```
# docker load -i busybox-latest.tar.gz
# docker load -i nfs-client-provisioner-v3.2.0.tar.gz
```

图 3-98 图示 2



3. Master创建命名空间

```
# kubectl create namespace nfs
```

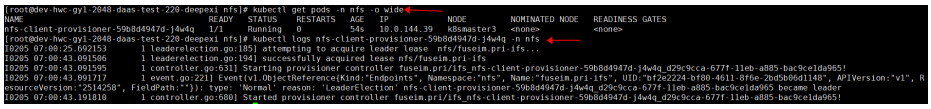
4. Master创建NFS服务K8s插件

```
# kubectl apply -f rbac.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
name: nfs-client-provisioner
# replace with namespace where provisioner is deployed
namespace: nfs
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: nfs-client-provisioner-runner
rules:
  - apiGroups: [&quot;&quot;]
    resources: [&quot;persistentvolumes&quot;]
    verbs: [&quot;get&quot;, &quot;list&quot;, &quot;watch&quot;, &quot;create&quot;,
&quot;delete&quot;]
  - apiGroups: [&quot;&quot;]
    resources: [&quot;persistentvolumeclaims&quot;]
    verbs: [&quot;get&quot;, &quot;list&quot;, &quot;watch&quot;, &quot;update&quot;]
  - apiGroups: [&quot;storage.k8s.io&quot;]
    resources: [&quot;storageclasses&quot;]
    verbs: [&quot;get&quot;, &quot;list&quot;, &quot;watch&quot;]
  - apiGroups: [&quot;&quot;]
    resources: [&quot;events&quot;]
    verbs: [&quot;create&quot;, &quot;update&quot;, &quot;patch&quot;]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: run-nfs-client-provisioner
subjects:
- kind: ServiceAccount
name: nfs-client-provisioner
# replace with namespace where provisioner is deployed
namespace: nfs
roleRef:
kind: ClusterRole
name: nfs-client-provisioner-runner
apiGroup: rbac.authorization.k8s.io
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: leader-locking-nfs-client-provisioner
# replace with namespace where provisioner is deployed
namespace: nfs
rules:
  - apiGroups: [&quot;&quot;]
    resources: [&quot;endpoints&quot;]
    verbs: [&quot;get&quot;, &quot;list&quot;, &quot;watch&quot;, &quot;create&quot;,
```

```
&amp;quot;update&amp;quot;, &amp;quot;patch&amp;quot;]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
name: leader-locking-nfs-client-provisioner
# replace with namespace where provisioner is deployed
namespace: nfs
subjects:
- kind: ServiceAccount
name: nfs-client-provisioner
# replace with namespace where provisioner is deployed
namespace: nfs
roleRef:
kind: Role
name: leader-locking-nfs-client-provisioner
apiGroup: rbac.authorization.k8s.io
# kubectl apply -f class.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
name: managed-nfs-storage
provisioner: fuseim.pri/ifs # or choose another name, must match deployment's env
PROVISIONER_NAME'
parameters:
  archiveOnDelete: &amp;quot;false&amp;quot;
# kubectl apply -f deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
name: nfs-client-provisioner
labels:
app: nfs-client-provisioner
# replace with namespace where provisioner is deployed
namespace: nfs
spec:
replicas: 1
strategy:
type: Recreate
selector:
matchLabels:
app: nfs-client-provisioner
template:
metadata:
labels:
app: nfs-client-provisioner
spec:
serviceAccountName: nfs-client-provisioner
containers:
- name: nfs-client-provisioner
image: kopkop/nfs-client-provisioner-arm64:v3.1.0-k8s1.11
volumeMounts:
- name: nfs-client-root
mountPath: /persistentvolumes
env:
- name: PROVISIONER_NAME
value: fuseim.pri/ifs
- name: NFS_SERVER
value: 192.168.0.69   #NFS服务器IP地址
- name: NFS_PATH
value: /data/nfs       #NFS服务器上文件存储路径
volumes:
- name: nfs-client-root
nfs:
server: 192.168.0.69     #NFS服务器IP地址
path: /data/nfs          #NFS服务器上文件存储路径
```

看到以下信息，表明deployment创建成功

图 3-99 图示 3

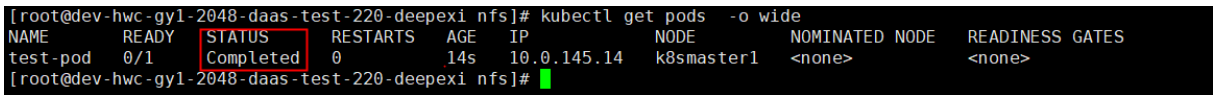


5. 测试

   创建NFS服务存储目录虚拟映射

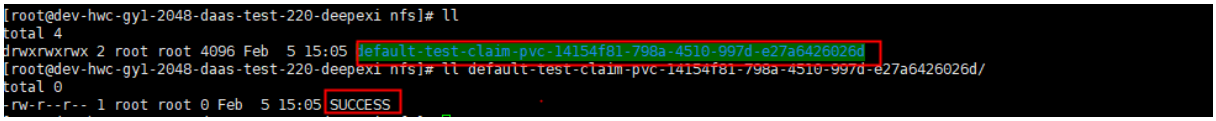
```
# kubectl apply -f test-claim.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: test-claim
annotations:
volume.beta.kubernetes.io/storage-class: &amp;quot;managed-nfs-storage&amp;quot;
spec:
accessModes:
- ReadWriteMany
resources:
requests:
storage: 1000Mi #资源大小限制
# kubectl apply -f test-pod.yaml
kind: Pod
apiVersion: v1
metadata:
name: test-pod
spec:
containers:
- name: test-pod
image: busybox:latest
command:
- &amp;quot;/bin/sh&amp;quot;
args:
- &amp;quot;-c&amp;quot;
- &amp;quot;touch /data/nfs/SUCCESS &amp;amp;&amp;amp; exit 0 || exit 1&amp;quot;
volumeMounts:
- name: nfs-pvc
mountPath: &amp;quot;/data/nfs&amp;quot; #挂载到镜像中的路径
restartPolicy: &amp;quot;Never&amp;quot;
volumes:
- name: nfs-pvc
persistentVolumeClaim:
claimName: test-claim #存储目录虚拟映射，跟test-claim.yaml的名称一致
```

图 3-100 图示 4



到NSF-Server服务器(192.168.0.69)所在机器的/data/nfs，可以看到创建了一个随机默认目录和SUCCESS文件

图 3-101 图示 5



6. 容器挂载

   创建k8s客户端服务，上传daas-claim.yaml到服务器，并执行

图 3-102 图示 6


```
# kubectl create ns daas
# kubectl apply -f daas-claim.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
name: daas-claim
namespace: daas
annotations:
volume.beta.kubernetes.io/storage-class: &amp;quot;managed-nfs-storage&amp;quot;
spec:
accessModes:
- ReadWriteMany
resources:
requests:
storage: 1000Mi #资源大小限制
```

挂载容器指定目录（/etc/daas/kerberos）到NFS服务器

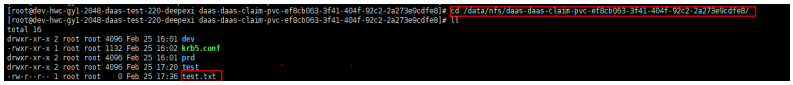
图 3-103 图示 7

```
volumeMounts:
- name: nfs-pvc
mountPath: &amp;quot;/etc/daas/kerberos&amp;quot; #容器目录
volumes:
- name: nfs-pvc
persistentVolumeClaim:
claimName: daas-claim #NFS客户端服务名称
```

进入NFS服务器，看到test.txt，表明容器到NFS的挂载已成功

```
# cd /data/nfs/daas-daas-claim-pvc-xxx
```

图 3-104 图示 8



3.11 DaaS 部署

3.11.1 登录访问 DaaS 系统与容器编排文件总览

登录访问 DaaS 系统

http://139.159.196.135/daas-management/index.html#/login

租户：daas

账号：daas

密码：abcd1234

容器编排文件总览

前端页面（5个）

统一账号管理前端页面

deepxi-daas-iam-front.yaml

Daas管理平台前端

deepxi-daas-base-application-web.yaml

deepxi-daas-dev-web.yaml

deepxi-daas-management-web.yaml

deepxi-daas-service-web.yaml

后端服务（17个）

统一账号后端接口域

deepxi-daas-iam-admin.yaml

统一账号开放平台域

deepxi-daas-iam-openapi.yaml

统一账号单点登录域

deepxi-daas-iam-sso.yaml

开发平台和大数据处理

deepexi-daas-bigdata-develop.yaml

数据服务工程

deepexi-daas-data-service.yaml

数据服务平台-数据源服务

deepexi-daas-data-service-platform-datasource.yaml

数据服务平台-鉴权服务

deepexi-daas-data-service-platform-oauth.yaml

数据源服务

deepexi-daas-datasource.yaml

统一分布式ID生成器

deepexi-daas-distribute-id-generator.yaml

数据质量

deepexi-daas-inspector.yaml

元数据收集服务

deepexi-daas-metadata-collector.yaml

元数据处理服务

deepexi-daas-metadata-server.yaml

元数据web服务

deepexi-daas-metadata-web.yaml

指标系统

deepexi-daas-metrics.yaml

数据安全

deepexi-daas-security.yaml

服务器指令执行服务

deepexi-daas-server.yaml

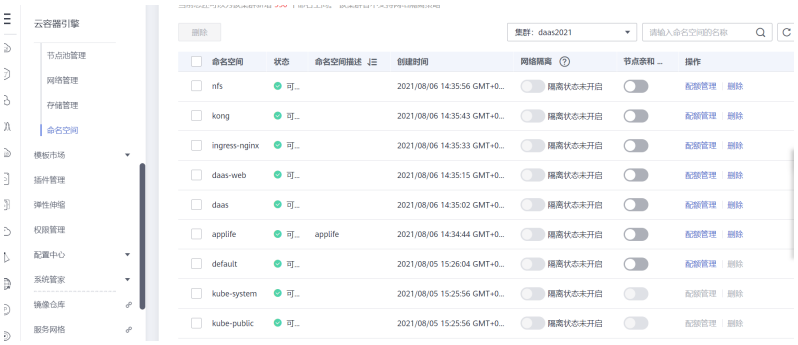
开发平台

deepxi-daas-develop.yaml

3.11.2 CCE 导入服务

1. 命名空间介绍
   applife:中间件服务或中间件映射服务；
   daas: DAAS后台服务；
   daas-web: DAAS前台服务；
   kong: 网关服务.

   图 3-105 命名空间介绍



2. 导入后台服务
打开kubectl控制机器进入/sort/daas-yaml/sh目录

图 3-106 导入后台服务

```
total 136
-rw-r--r-- 1 root root 5916 Apr 30 14:15 a.zip
-rw-r--r-- 1 root root  933 Jul 27 15:42 deepexi-daas-base-application-web-template.yaml
-rw-r--r-- 1 root root 2196 Aug 26 15:16 deepexi-daas-bigdata-develop-template.yaml
-rw-r--r-- 1 root root 5610 Jul 27 16:17 deepexi-daas-data-service-template.yaml
-rw-r--r-- 1 root root 5150 Mar 24 11:29 deepexi-daas-data-service-template.yaml_bak
-rw-r--r-- 1 root root 2234 Aug 26 15:12 deepexi-daas-datasource-template.yaml
-rw-r--r-- 1 root root 5695 Jul 27 16:33 deepexi-daas-develop-template.yaml
-rw-r--r-- 1 root root  850 Jul 27 16:36 deepexi-daas-dev-web-template.yaml
-rw-r--r-- 1 root root 2161 Aug 26 14:48 deepexi-daas-inspector-template.yaml
-rw-r--r-- 1 root root 2096 Aug 26 14:41 deepexi-daas-log-template.yaml
-rw-r--r-- 1 root root  899 Jul 27 16:45 deepexi-daas-management-web-template.yaml
-rw-r--r-- 1 root root 2220 Aug 26 14:30 deepexi-daas-metadata2-template.yaml
-rw-r--r-- 1 root root 5772 Jul 27 16:48 deepexi-daas-metadata-collector-template.yaml
-rw-r--r-- 1 root root 2735 Aug 26 11:30 deepexi-daas-metadata-server-template.yaml
-rw-r--r-- 1 root root 5718 Jul 27 16:53 deepexi-daas-metadata-web-template.yaml
-rw-r--r-- 1 root root 2157 Aug 26 14:23 deepexi-daas-metrics-template.yaml
-rw-r--r-- 1 root root 2072 Aug 26 14:15 deepexi-daas-model-template.yaml
-rw-r--r-- 1 root root  864 Jul 27 17:00 deepexi-daas-model-web-template.yaml
-rw-r--r-- 1 root root 2164 Aug 26 14:11 deepexi-daas-security-template.yaml
-rw-r--r-- 1 root root 2121 Aug 26 11:58 deepexi-daas-server-template.yaml
-rw-r--r-- 1 root root 2063 Aug 26 11:55 deepexi-daas-service-admin-template.yaml
-rw-r--r-- 1 root root 2689 Aug 26 11:22 deepexi-daas-service-datasource-template.yaml
-rw-r--r-- 1 root root 2061 Aug 26 11:22 deepexi-daas-service-oauth-template.yaml
-rw-r--r-- 1 root root  874 Jul 27 17:04 deepexi-daas-service-web-template.yaml
-rwxr-xr-x 1 root root 1301 Mar 12 18:14 deploy.sh
drwxr-xr-x 2 root root 4096 Aug 25 11:12 dockerfile
-rw-r--r-- 1 root root 1796 Aug  3 17:57 message_admin.yaml
-rw-r--r-- 1 root root 1414 Aug  4 15:36 temp.tar.gz
[root@daas-yx sh]# pwd
/sort/daas-yaml/sh
[root@daas-yx sh]#
```

陆续把容器编排文件导入

```
Kubectl  apply -f .
```

如果出现下running状态提示，表面容器已启动，否则检查报错信息

图 3-107 导入后台服务 2

```
/sort/daas-yaml/sh
[root@daas-yx sh]# kubectl get pod -n daas
NAME                                                READY   STATUS    RESTARTS   AGE
deepexi-daas-bigdata-develop-deploy-777d5fb9dd-lf2n9   1/1     Running   0          99m
deepexi-daas-datasource-deploy-6f89bbdb5-pcbbk      1/1     Running   0          104m
deepexi-daas-iam-admin-deploy-69887994df-vphkk      1/1     Running   0          5h10m
deepexi-daas-iam-openapi-deploy-54f5b768f-56wdj     1/1     Running   0          5h10m
deepexi-daas-iam-sso-deploy-7c4867967b-wrbtx        1/1     Running   0          5h9m
deepexi-daas-inspector-deploy-6fc86b6f96-slq4p      1/1     Running   0          128m
deepexi-daas-log-deploy-78b7c56b6b-272qw            1/1     Running   0          134m
deepexi-daas-metadata2-deploy-5c46bb68b-42hft       1/1     Running   0          145m
deepexi-daas-metrics-deploy-7f74c57c78-qnp7c        1/1     Running   0          153m
deepexi-daas-model-deploy-bc4c7b74b-ps4j6           1/1     Running   0          161m
deepexi-daas-security-deploy-ffb778cb9-2zgbj        1/1     Running   0          165m
deepexi-daas-server-deploy-67bc6c5f9-8jm85          1/1     Running   0          4h57m
deepexi-daas-service-admin-deploy-579d4bd6f-mb9nn   1/1     Running   0          5h1m
deepexi-daas-service-datasource-deploy-77bfc64f4-6hjrp   1/1     Running   0          5h6m
deepexi-daas-service-oauth-deploy-657957ff-jn7mp    1/1     Running   0          5h6m
deepexi-distribute-id-generator-deploy-7fbcfdf66f-jcl2p   1/1     Running   0          9d
deepexi-message-admin-deploy-57445b677c-nwnqg       1/1     Running   0          5h9m
[root@daas-yx sh]#
```

查看服务启动日志，选择某一个服务，“查看日志”

```
kubectl logs -n daas deepexi-daas-security-deploy-ffb778cb9-2zgbj -f --tail=200
```

图 3-108 查看日志

```
2021-08-26T15:44:42.869+0800: 5579.915: [GC (Allocation Failure) 2021-08-26T15:44:42.869+0800: 5579.915: [DefNew: 140452K->5339K(152576K), 0.0147622 secs] 229624K->94511K(
491200K), 0.0148153 secs] [Times: user=0.01 sys=0.00, real=0.02 secs]
2021-08-26 15:47:43.112  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26 15:52:43.113  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26T15:56:25.213+0800: 6282.259: [GC (Allocation Failure) 2021-08-26T15:56:25.213+0800: 6282.259: [DefNew: 141019K->5387K(152576K), 0.0141102 secs] 230191K->94558K(
491200K), 0.0141852 secs] [Times: user=0.01 sys=0.00, real=0.01 secs]
2021-08-26 15:57:43.113  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26 16:00:00.000  INFO 1 --- [ool-18-thread-1] c.d.d.s.scheduling.SentryPermissionTask  : Sentry权限同步开始，时间：Thu Aug 26 16:00:00 CST 2021
2021-08-26 16:00:00.005  INFO 1 --- [ool-18-thread-1] c.d.d.s.scheduling.SentryPermissionTask  : Sentry权限同步结束，时间：Thu Aug 26 16:00:00 CST 2021
2021-08-26 16:02:43.113  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26 16:07:43.114  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26T16:07:53.417+0800: 6970.463: [GC (Allocation Failure) 2021-08-26T16:07:53.417+0800: 6970.463: [DefNew: 141067K->5441K(152576K), 0.0139981 secs] 230238K->94613K(
491200K), 0.0140735 secs] [Times: user=0.02 sys=0.00, real=0.02 secs]
2021-08-26 16:12:43.114  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26 16:17:43.114  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26T16:19:29.032+0800: 7666.078: [GC (Allocation Failure) 2021-08-26T16:19:29.032+0800: 7666.078: [DefNew: 141121K->5794K(152576K), 0.0147063 secs] 230293K->94965K(
491200K), 0.0147830 secs] [Times: user=0.01 sys=0.00, real=0.01 secs]
2021-08-26 16:22:43.115  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26 16:27:43.115  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26T16:31:04.070+0800: 8361.115: [GC (Allocation Failure) 2021-08-26T16:31:04.070+0800: 8361.116: [DefNew: 141474K->5431K(152576K), 0.0147910 secs] 230645K->94603K(
491200K), 0.0148748 secs] [Times: user=0.01 sys=0.00, real=0.02 secs]
2021-08-26 16:32:43.115  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26 16:37:43.116  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26T16:42:39.410+0800: 9056.456: [GC (Allocation Failure) 2021-08-26T16:42:39.410+0800: 9056.456: [DefNew: 141111K->5861K(152576K), 0.0152014 secs] 230283K->95033K(
491200K), 0.0152787 secs] [Times: user=0.01 sys=0.00, real=0.02 secs]
2021-08-26 16:42:43.116  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26 16:47:43.117  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26 16:52:43.117  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
2021-08-26T16:54:21.436+0800: 9758.482: [GC (Allocation Failure) 2021-08-26T16:54:21.436+0800: 9758.482: [DefNew: 141541K->5447K(152576K), 0.0147291 secs] 230713K->94619K(
491200K), 0.0148087 secs] [Times: user=0.01 sys=0.00, real=0.02 secs]
2021-08-26 16:57:43.117  INFO 1 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver      : Resolving eureka endpoints via configuration
```

3. 导入前台服务
同导入后台服务一样，打开kubectl机器

图 3-109 导入前台服务 1

```
total 732
-rw-r--r-- 1 root root    946 Aug 17 16:08 deepexi-daas-base-application-web-template.yaml
-rw-r--r-- 1 root root    701 Aug 17 10:46 default.conf
drwxrwxrwx 6 root root   4096 Aug 17 10:46 dist
-rw-r--r-- 1 root root 727790 Aug 17 11:01 dist.tar.gz
-rw-r----- 1 root root    225 Aug 17 16:05 Dockerfile
-rw-r--r-- 1 root root    642 Aug 17 10:46 nginx.conf
[root@daas-yx front]#
```

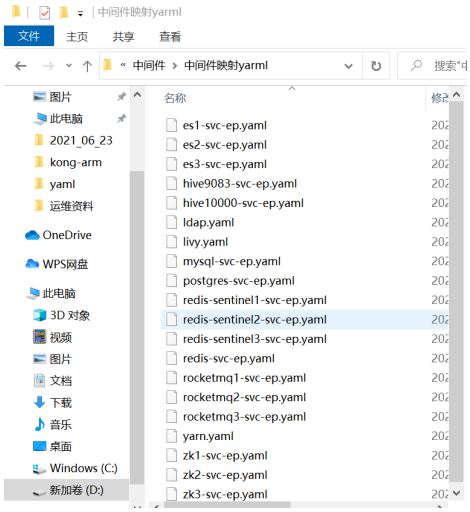
陆续导入容器编排文件即可
结果如图

图 3-110 导入前台服务 2

```
[root@daas-yx front]# kubectl get pod -n daas-web
NAME                                                READY   STATUS    RESTARTS   AGE
deepexi-daas-base-application-web-deploy-fcdf798c9-sk4cr   1/1     Running   0          9d
deepexi-daas-dev-web-deploy-689796c88-9sljp         1/1     Running   0          8d
deepexi-daas-iam-front-deploy-786c8cfd4b-qpsp4      1/1     Running   0          2d6h
deepexi-daas-management-web-deploy-74c75787b6-8f5xn 1/1     Running   0          8d
deepexi-daas-model-web-deploy-65cf655786-jwr55      1/1     Running   0          8d
deepexi-daas-service-web-deploy-77d4d48676-bpdzg    1/1     Running   0          7d23h
[root@daas-yx front]#
```

4. 导入中间件服务
打开kubectl控制机器
陆续导入中间件(映射)容器编排文件

图 3-111 导入中间件服务 1



结果如下：

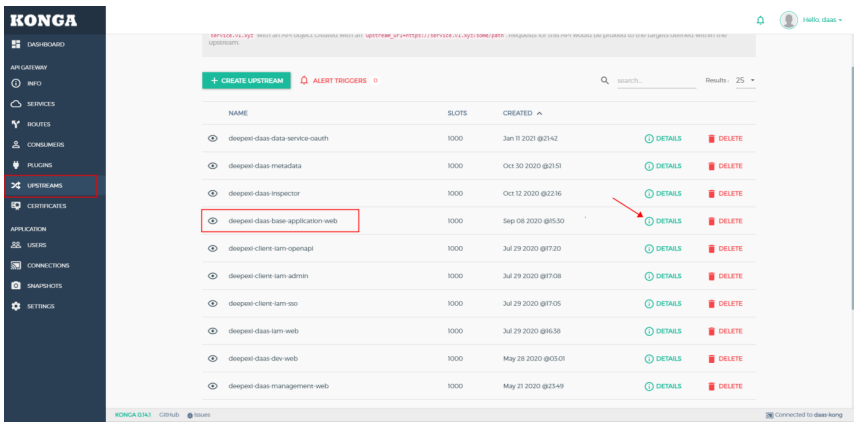
图 3-112 导入中间件服务 2

```
deepexi-daas-service-web-deploy-77d4d48676-bpdzg          1/1     Running   0          7d23h
[root@daas-yx front]# kubectl get ep -n applife
NAME                                ENDPOINTS                                                  AGE
apollo-portal                       172.16.2.33:8070                                           8d
apollo-service-apollo-adminservice  172.16.2.168:8090                                          8d
apollo-service-apollo-configservice 172.16.1.155:8080                                          8d
deepexi-daas-eureka-server          172.16.1.154:8761,172.16.2.166:8761,172.16.2.31:8761       9d
deepexi-daas-eureka-server-nodeport 172.16.1.154:8761,172.16.2.166:8761,172.16.2.31:8761       9d
es1                                 192.168.0.5:9200                                           6d21h
es2                                 192.168.0.202:9200                                         6d21h
es3                                 192.168.0.177:9200                                         6d21h
hive1                               192.168.0.232:10000                                        6h7m
hive2                               192.168.0.252:9083                                         6h3m
livy                                192.168.0.232:8998                                         6h1m
mysql                               192.168.0.230:3306                                         9d
postgres                            192.168.0.15:5432                                          7d5h
redis                               192.168.0.82:6379                                          7d5h
redis-sentinel1                     192.168.0.82:16379                                         6d6h
redis-sentinel2                     192.168.0.85:16379                                         6d6h
redis-sentinel3                     192.168.0.56:16379                                         6d6h
rocketmq1                           192.168.0.190:9876                                         7d5h
rocketmq2                           192.168.0.228:9876                                         7d5h
rocketmq3                           192.168.0.168:9876                                         7d5h
yarn                                192.168.0.232:8088                                         5h50m
zk1                                 192.168.0.190:2181                                         10d
zk2                                 192.168.0.228:2181                                         10d
zk3                                 192.168.0.168:2181                                         10d
[root@daas-yx front]#
```

3.11.3 前端服务 Kong 配置

1. 管理平台基础页面（deepxi-daas-base-application-web）

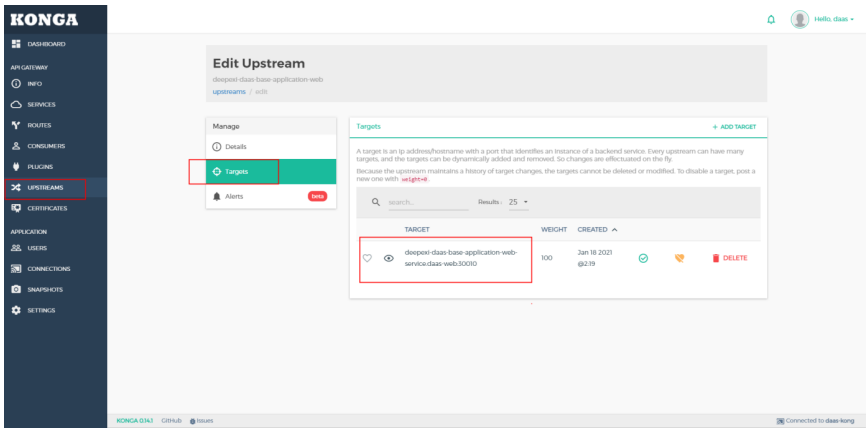
图 3-113 管理平台基础页面



2. 指定为K8s服务名称

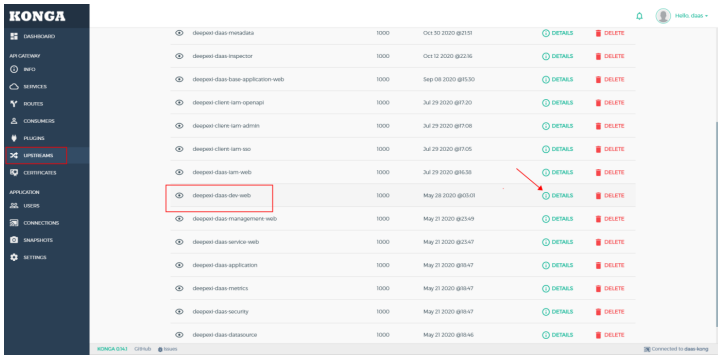
   deepexi-daas-base-application-web-service.daas-web:30010，注意要加上命名空间“.daas-web”

图 3-114 指定为 K8s 服务名称



3. 管理平台数据开发页面（deepxi-daas-dev-web）

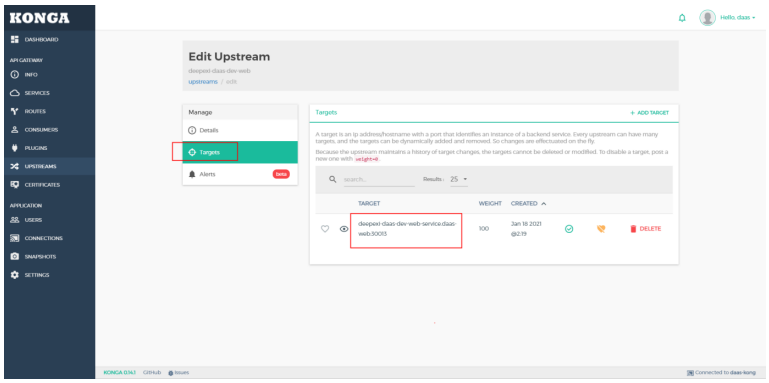
图 3-115 管理平台数据开发页面 1



4. 指定为K8s服务名称

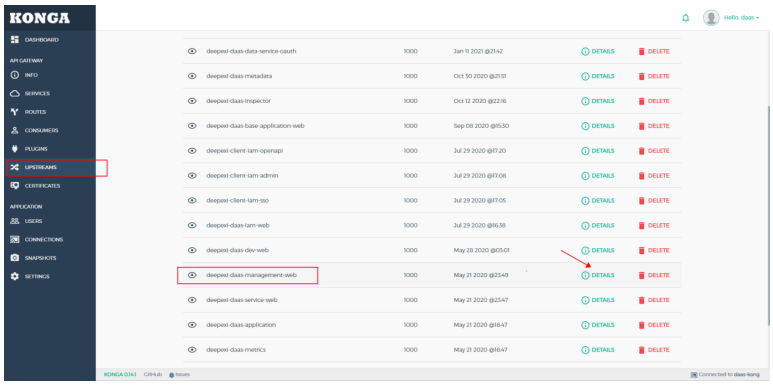
   deepexi-daas-dev-web-service.daas-web:30013，注意要加上命名空间“.daas-web”

图 3-116 指定为 K8s 服务名称



5. 管理平台数据管理页面（deepxi-daas-management-web）

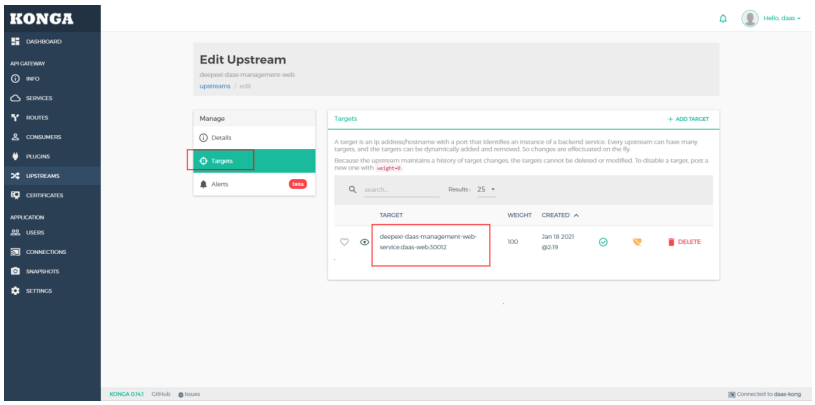
图 3-117 管理平台数据管理页面



6. 指定为K8s服务名称

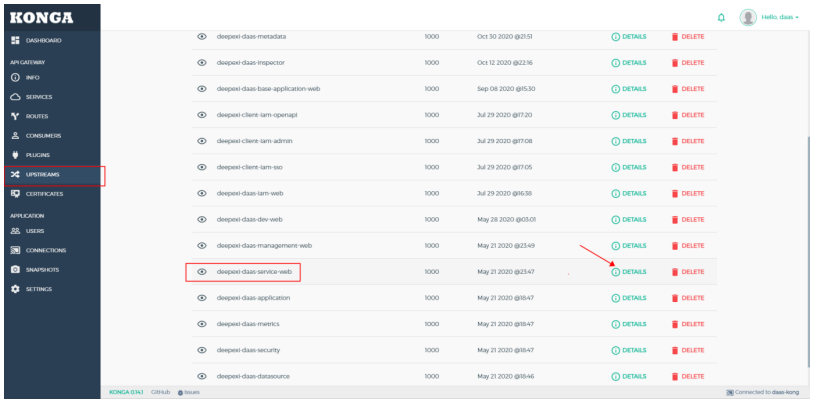
   deepexi-daas-management-web-service.daas-web:30012，注意要加上命名空间“.daas-web”

图 3-118 指定为 K8s 服务名称 1



7. 管理平台数据服务页面（deepxi-daas-service-web）

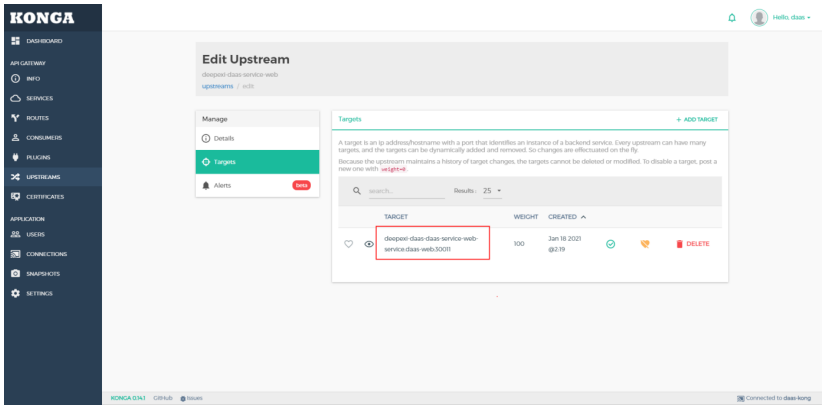
图 3-119 管理平台数据服务页面



8. 指定为K8s服务名称

   deepexi-daas-daas-service-web-service.daas-web:30011，注意要加上命名空间“.daas-web”

图 3-120 指定为 K8s 服务名称 2



3.11.4 IAM 后端服务配置

1. 镜像版本

   iam-sso：deepexi-client-iam-admin:4.6.4-beta_arm64

iam-openapi：deepexi-client-iam-openapi:4.6.4-beta_arm64

iam-admin：deepexi-client-iam-admin:4.6.4-beta_arm64

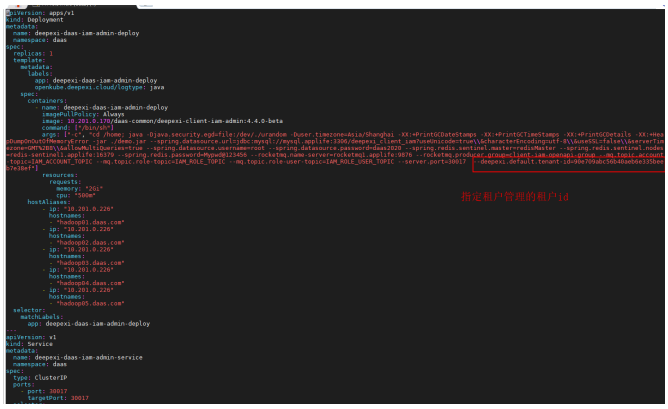
2. 启动配置

iam-admin、iam-openapi这两个服务的启动配置需要指定租户管理员的租户id。不指定默认值为：4e086791212649d79f30ec0527599aee

需要添加的配置项：

```
--deepexi.default.tenant-id=?
```

图 3-121 启动配置



deepexi.default.tenant-id这个值在deepexi_client_iam数据库执行下面sql可以查询：

```
select tenant_id from deepexi_client_iam.account_account_info where is_main = 1 and enterprise_code = 'admin' and is_deleted = 0
```

3. apollo配置

security服务apollo 配置中，下面这几项需要配置正确。（使用最新iam-4.4.0初始化sql初始化iam的数据库，配置项的值如下）初始化sql参考：

https://deepexi.yuque.com/docs/share/550fe007-a9db-4496-95fa-c4d8a14bf369?#

```
# 角色组id
role.groups.id = 1（角色组id默认为1）
# 应用系统功能组id
system.function.id = 1
# 应用项目功能组id
project.function.id = 2
# 权限中心元素id
permission.center.id = 12
# 账户管理元素id
user.manager.id = 17
# 角色管理元素id
role.manager.id = 14
# 权限管理元素id
permission.manager.id = 54
```

以上配置来源通过查询deepexi_client_iam数据库相关表得到，相关sql如下：

select id from permission_role_group where name = '系统角色组';

select id from permission_function_group where name = '系统功能组';

select id from permission_function_group where name = '项目功能组';

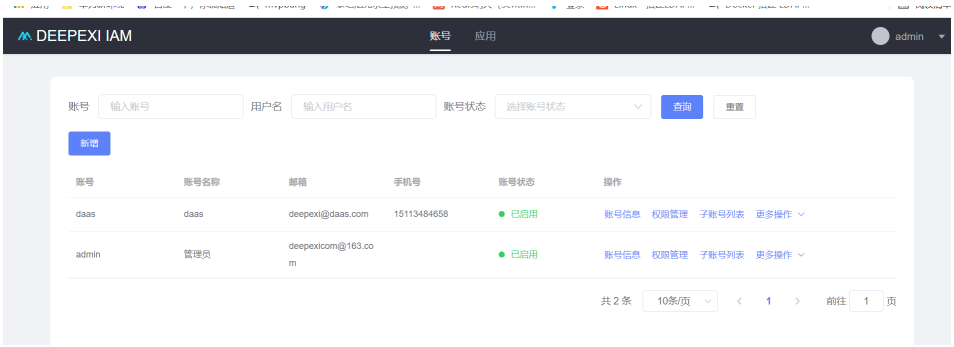
select id from permission_element_resource where name = '权限中心';

select id from permission_element_resource where name = '账户管理';

select id from permission_element_resource where name = '角色管理';
select id from permission_element_resource where name = '授权管理';
如果上述几项配置不正确，登录后菜单将加载失败。

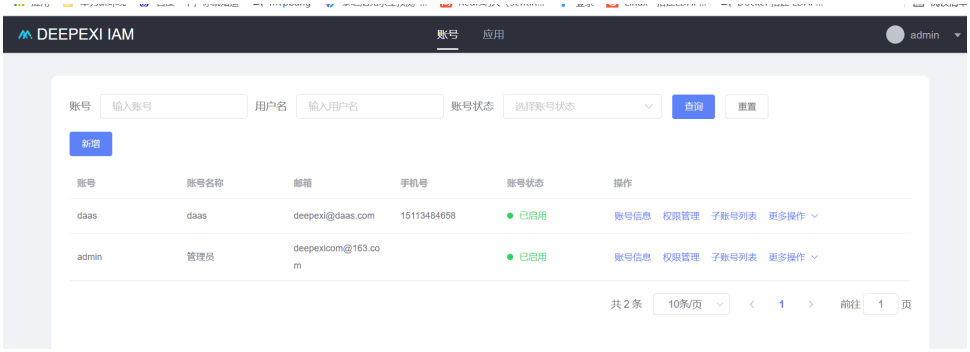
4. 登录访问Iam系统
http://139.159.196.135/iam-console/index.html#/login
daas租户账号：daas/abcd1234，租户管理员账号：admin/abcd1234
租户账号登录只能管理自己租户，效果如下：

图 3-122 登录访问 Iam 系统 1



租户管理员账号登录可以管理所有租户及应用，效果如下：

图 3-123 登录访问 Iam 系统 2



说明

如果使用admin(租户管理员)账号登录，存在多个租户时，页面账号列表却只有admin账号。要去检查iam-admin、iam-openapi启动配置的--deepexi.default.tenant-id这个值是否配置正确（参考7.5.2章节）。

5. 问题排查

a. DAAS系统登录不上
确认主账号在IAM是否可以正常登录。（确保账号密码是正确的）
确认登录时，daas security服务是否调用 iam-sso 成功。（查看 iam 日志可以知道。ps: 没调用成功也会提示账号密码错误 ）

b. 登录成功，报错：该租户并未订阅该产品，请先订阅后再访问
使用admin账号登录IAM中查看租户是否被授权了daas产品。如果确定了已经授权但还是报同样的错，则需要查看admin-openapi的日志分析问题。

图 3-124 登录成功 1

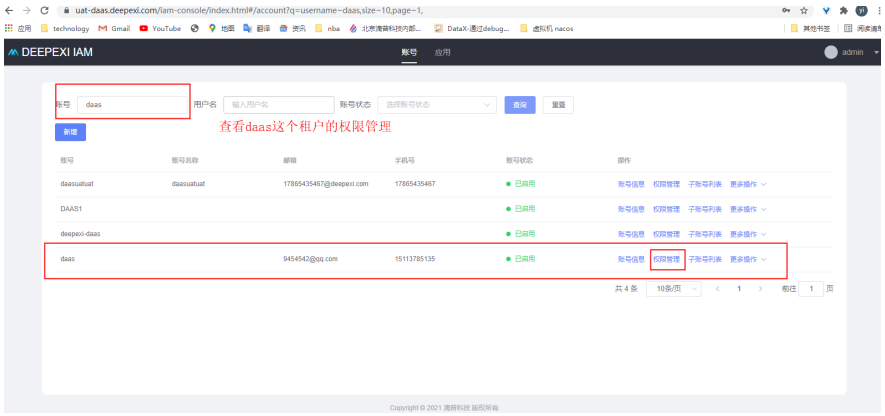


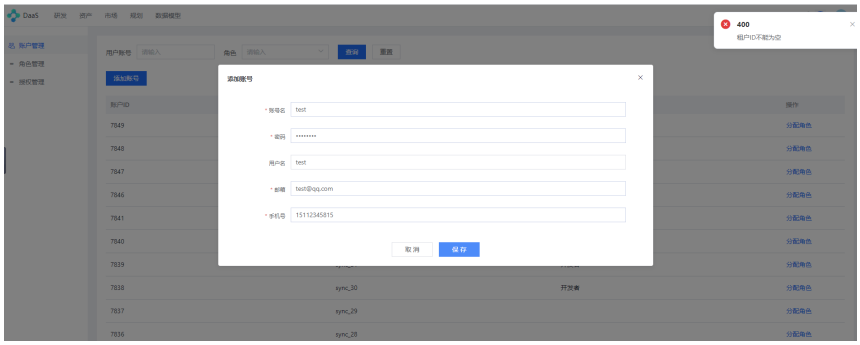
图 3-125 登录成功 2

c. IAM登录admin账号，看不到所有用户
参考7.5.2，将租户ID在启动脚本配置上

d. 非主账号，登录成功，菜单访问不了
需要登录主账号去给这个子账号配置权限

e. 账号创建报错
访问下面接口会返回"401"，"token失效请重新登录"
http://139.159.196.135/deepexi-daas-security/api/v1/users/currentUser?tenantId=985e029870ba47399933a6bf5e08eb3d&userId=10002&projectId=&businessSegmentsId=&_=1616033853562
确认security服务apollo配置，iam.signingKey = d20a0382a5f840f7afbd3f4df92ecaf8 是否为这个值。需要修改成这个值。security服务JWT解析编码要用到这个值

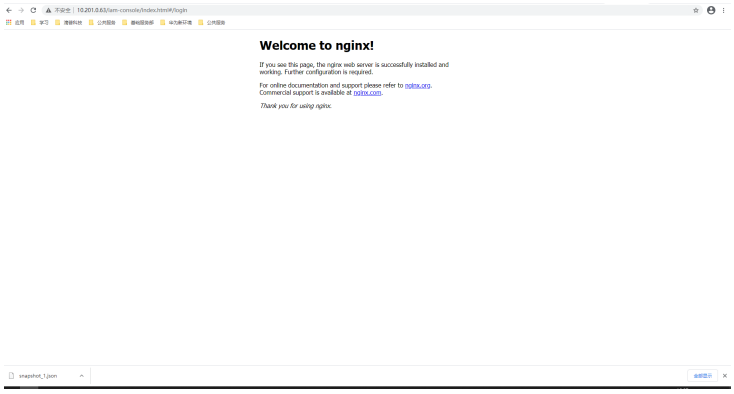
图 3-126 账号创建报错



f. 访问iam出现nginx首页

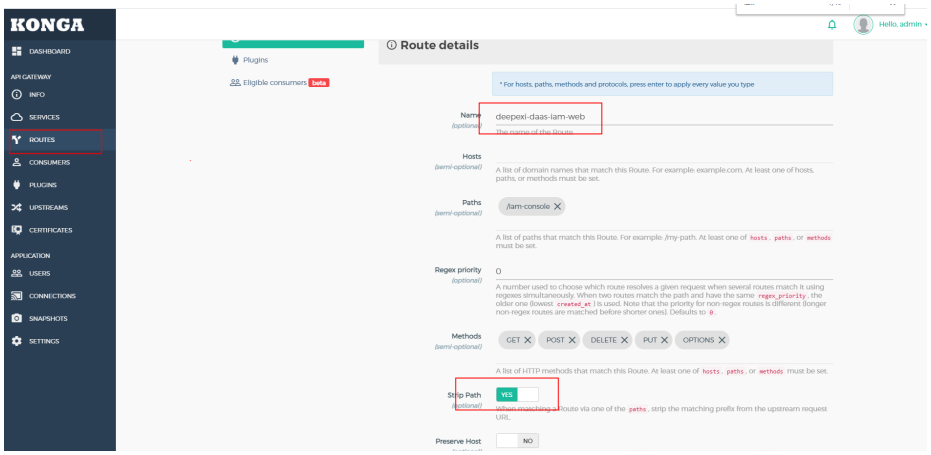
如果访问Iam出现以下页面，需要设置konga配置

图 3-127 访问 iam 出现 nginx 首页



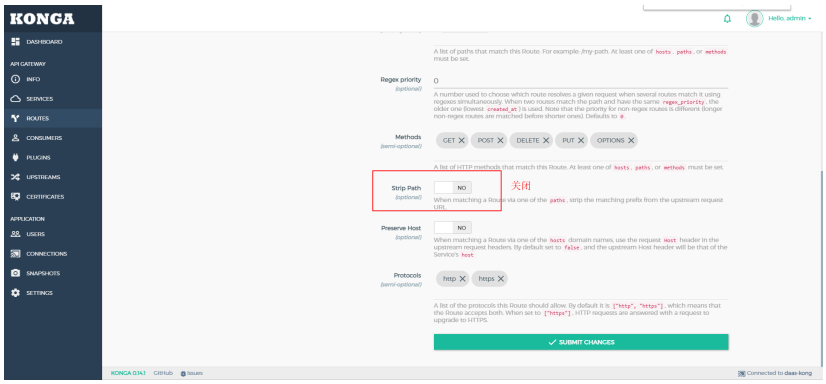
打开“konga > ROUTES”，找到deepexi-daas-iam-web配置，关闭“Strip Path”

图 3-128 图示 1



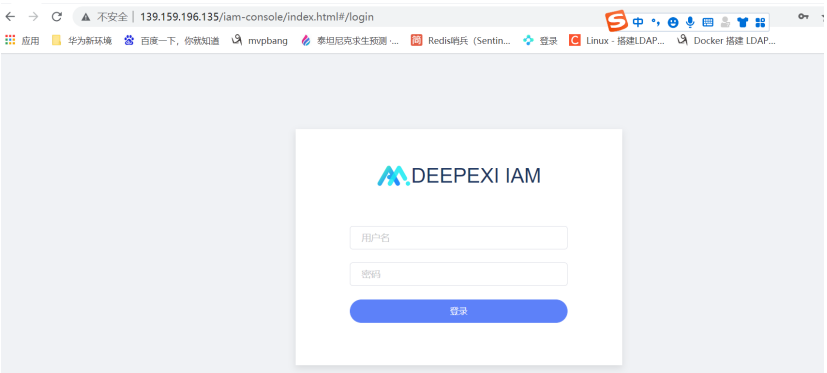
结果如下：

图 3-129 图示 2



保存，重新访问iam-console

图 3-130 图示 3



3.12 DolphinScheduler 部署

前置条件说明

- Zookeeper需安装好
- JDK先配置好(1.8)
- gcc 需安装好(4.8.*)
- 安装大数据组件客户端(HDFS\YARN\HIVE\FLINK\SPARK)
- 安装datax环境

安装机器

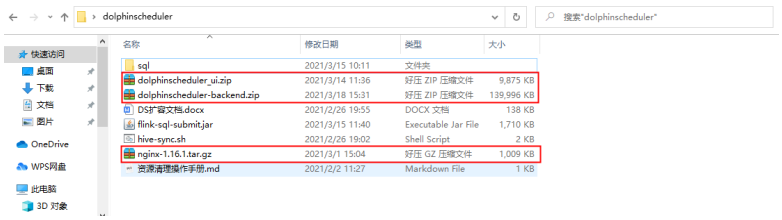
前提

1. CDH中需要有两个Hadoop集群，每个集群下部署一套DS (CDH机器允许部署的情况)
2. 如果是POC，可以只用一套DS
3. CDH机器不允许部署的情况，参考https://deepexi.yuque.com/docs/share/cbb7e60f-7188-41ef-8a72-219f269c07aa?#，将部署机器加入CDH集群管理即可

文件上传

解压dolphinscheduler.zip, 将dolphinscheduler_ui.zip、dolphinscheduler-backend.zip、nginx-1.16.1.tar.g上传至/data/daas/目录下并且解压

图 3-131 解压



配置 dolphinscheduler 用户并授权

```
# useradd dolphinscheduler
# echo &amp;quot;dolphinscheduler&amp;quot; | passwd --stdin dolphinscheduler
# sed -i '$adolphinscheduler  ALL=(ALL)  NOPASSWD: NOPASSWD: ALL' /etc/sudoers
# sed -i 's/Defaults    requirett/#Defaults    requirett/g' /etc/sudoers
# su dolphinscheduler
```

配置sudo免密

```
# ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
# cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
# chmod 600 ~/.ssh/authorized_keys
# exit
```

授权ds源目录

```
# chown -R dolphinscheduler:dolphinscheduler /data/daas/dolphinscheduler-backend
```

授权ds安装目录

```
# mkdir -p /data/daas/dolphinscheduler
# chown -R dolphinscheduler:dolphinscheduler /data/daas/dolphinscheduler
```

如果是ds是集群（多master 多worker）的话，需要配置机器间免密登录，需要执行以下步骤

在安装DS的机器执行以下命令

```
# su dolphinscheduler
```

获取公钥

```
# less ~/.ssh/id_rsa.pub
```

图 3-132 获取公钥

将生成的公钥复制到另外安装机器的 ~/.ssh/authorized_keys文件。

```
# vim ~/.ssh/authorized_keys
```

粘贴并保存，其他机器操作类似

配置 hdfs 用户

```
# su hdfs    如果出现：This account is currently not available. 说明 hdfs 是不可用的；如果没有该用户，则需要创建 # useradd hdfs，再执行以下步骤
# vi /etc/passwd 找到hdfs用户，将/sbin /nologin改成/bin/bash
# su hdfs    正常切到hdfs用户即可
# exit
# usermod -a -G dolphinscheduler hdfs 将hdfs用户加入dolphinscheduler组
```

配置 datax 环境检测

各个机器配置datax环境，解压datax.tar.gz到指定目录，例如/opt/soft/datax，将datax配置到环境变量

```
# vim /etc/profile 加入 : export DATAX_HOME=/opt/soft/datax
# source /etc/profile
```

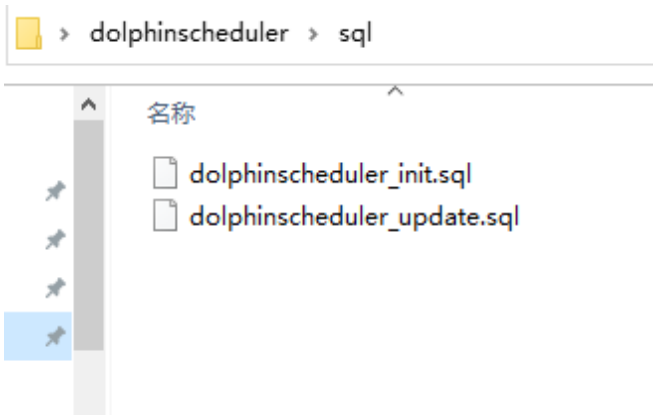
建立 JDK 软连接

```
# sudo ln -s /data/daas/jdk1.8.0_251/bin/java /usr/bin/java
```

创建数据库

创建数据库，通过工具执行/dolphinscheduler/sql文件夹中的sql文件(两份都需要执行)，这里以账号为root，密码为daas2020

图 3-133 创建数据库



修改配置

1. 找到 hdfs-site.xml 和 core-site.xml 文件，一般在 /etc/hadoop/conf 目录下，复制到/data/daas/dolphinscheduler-backend/conf目录下

   # cp /etc/hadoop/conf/hdfs-site.xml /etc/hadoop/conf/core-site.xml /data/daas/dolphinscheduler-backend/conf

2. 修改运行参数,修改 `conf/env` 目录下的 `.dolphinscheduler_env.sh` 环境变量(以相关用到的软件都安装在/opt/soft下为例)

   ```
   [dolphinscheduler@tools]#  vi /data/daas/dolphinscheduler-backend/
   conf/env/.dolphinscheduler_env.sh
   export HADOOP_HOME=/opt/cloudera/parcels/CDH/lib/hadoop
   export HADOOP_CONF_DIR=/opt/cloudera/parcels/CDH/lib/hadoop/etc/hadoop
   export SPARK_HOME=/opt/cloudera/parcels/CDH/lib/spark
   export HIVE_HOME=/opt/cloudera/parcels/CDH/lib/hive
   export JAVA_HOME=/usr/java/jdk1.8.0_251-amd64
   export PYTHON_HOME=/usr/bin/python2.7
   export FLINK_HOME=/opt/flink
   export DATAX_HOME=/opt/soft/datax
   PATH=$HADOOP_HOME/bin:$SPARK_HOME/bin:$JAVA_HOME/bin:$HIVE_HOME/bin:$FLINK_HOME/
   bin:$PATH
   ```

3. 修改一键部署脚本 install.sh中的各参数，特别注意以下参数的配置（这个配置很重要，如果设置不当，dolphinscheduler是启动不了的，输入后要多次检查）

   ```
   [root@tools]# vi /data/daas/dolphinscheduler-backend/install.sh
   dbtype=&quot;mysql&quot;
   #数据库地址
   dbhost=&quot;10.57.4.14:3306&quot;
   #数据库名字
   dbname=&quot;dolphinscheduler&quot;
   #数据库用户
   username=&quot;root&quot;
   #密码
   ```

```
passowrd=&quot;daas2020&quot;
#改成上面已经授权了的路径
installPath=&quot;/data/daas/dolphinscheduler&quot;
#安装用户,就是前面linux新建的用户
deployUser=&quot;dolphinscheduler&quot;
#改成对应的zookeeper地址和端口，如果是集群部署，用逗号隔开
zkQuorum=&quot;10.201.0.1112:2181,10.201.0.1113:2181&quot;
#部署ds的机器，用逗号隔开，所有worker机器都需要写上，如果只有一台机器则写一个ip
ips=&quot;10.201.0.112,10.201.0.113&quot;
#master服务部署在哪台机器上，所有master机器都需要写上并且用逗号隔开
masters=&quot;10.201.0.112&quot;
#worker服务部署在哪台机器上，所有worker机器都需要写上并且用逗号隔开
workers=&quot;10.201.0.112&quot;
#报警服务部署在哪台机器上（一般在master）
alertServer=&quot;10.201.0.112&quot;
#后端api服务部署在在哪台机器上（只在一台机器部署就可以）
apiServers=&quot;10.201.0.112&quot;
#邮件协议
mailProtocol=&quot;SMTP&quot;
# mail server host
# 邮件配置，以qq邮箱为例
mailServerHost=&quot;smtp.exmail.qq.com&quot;
# mail server port
#邮件端口
mailServerPort=&quot;25&quot;
# sender
#发送者
mailSender=&quot;daas@deepexi.com&quot;
#发送用户
mailUser=&quot;&quot;daas@deepexi.com&quot;
# sender password
#邮箱密码
mailPassword=&quot;daas2020&quot;
 # TLS协议的邮箱设置为true，否则设置为false
starttlsEnable=&quot;true&quot;
# 邮件服务地址值，参考上面 mailServerHost
sslTrust=&quot;smtp.mxhichina.com&quot;
# 开启SSL协议的邮箱配置为true，否则为false。注意: starttlsEnable和sslEnable不能同时为true
sslEnable=&quot;false&quot;
# 这里必须改为HDFS
resUploadStartupType=&quot;HDFS&quot;
#hdfs地址
defaultFS=&quot;hdfs://10.201.0.112:8020&quot;
#yarn集群的所有ip，用逗号隔开，例如&quot;10.201.0.112,10.201.0.113&quot;，如果不是集群，则不用填
yarnHaIps=
#如果yarn是集群的则不用填，如果不是集群，则填ip，例如&quot;10.201.0.112&quot;
singleYarnIp=
#hdfs目录
hdfsPath=&quot;/dolphinscheduler&quot;
# 注意，如果开启了 kerberos ，这个选项为空
hdfsRootUser=&quot;hdfs&quot;
```

执行脚本

```
# su dolphinscheduler 切换到dolphinscheduler用户，执行install.sh脚本，然后到各个机器上执行jps命令，查看服务是否启动成功
```

前端 UI 部署（在线、离线二选一进行部署）

- **在线部署**

  进入dolphinscheduler-ui目录下执行，切换到root用户 sh ./install-dolphinscheduler-ui.sh，执行后，会在运行中请键入前端端口，默认端口是8888，或者键入其他端口，然后会让键入跟前端ui交互的api-server的ip，接着是让键入跟前端ui交互的api-server的port，接着是操作系统选择，最后等待部署完成

- **离线部署**

  a. 安装nginx前首先要确认系统中安装了gcc、pcre-devel、zlib-devel、openssl-devel

  b. 切换到root用户下，上传nginx-1.16.1.tar.gz到/data/daas

  c. # tar -zxvf nginx-1.16.1.tar.gz

  # cd nginx-1.16.1

  #./configure --without-http_rewrite_module --without-http_gzip_module --prefix=/usr/local/nginx /usr/local/nginx是nginx的安装目录

  # make & make install

  # cd /usr/local/nginx/sbin

  # ./nginx

  d. 修改nginx.conf配置

```
# cd /usr/local/nginx/conf
# vim nginx.conf
server {
listen       8888; # 访问端口(自行修改)
server_name  ip; # api服务的ip
location / {
root   /data/daas/dolphinscheduler_ui/dist; # 前端解压的dist目录地址(自行修改)
index  index.html index.html;
}
client_max_body_size 50m;
location /dolphinscheduler {
proxy_pass http://ip:12345;# API服务的接口地址(自行修改)
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header x_real_ipP $remote_addr;
proxy_set_header remote_addr $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_http_version 1.1;
proxy_connect_timeout 4s;
proxy_read_timeout 30s;
proxy_send_timeout 12s;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection &amp;quot;upgrade&amp;quot;;
}
}
# cd /usr/local/nginx/sbin
```

  重启nginx

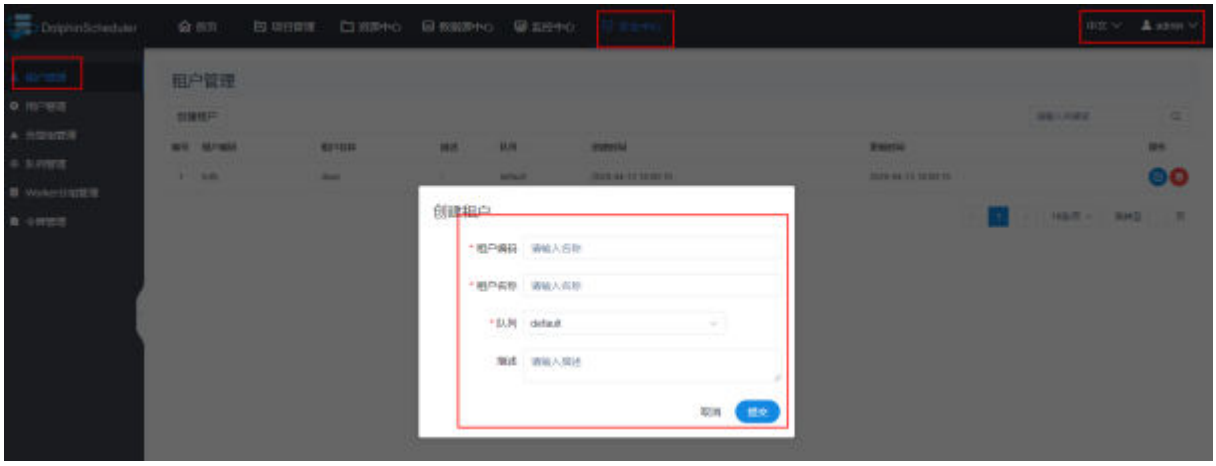
```
# ./nginx -s reload
```

  e. 浏览器访问http://ip:8888，出现登录页面即可

租户用户创建

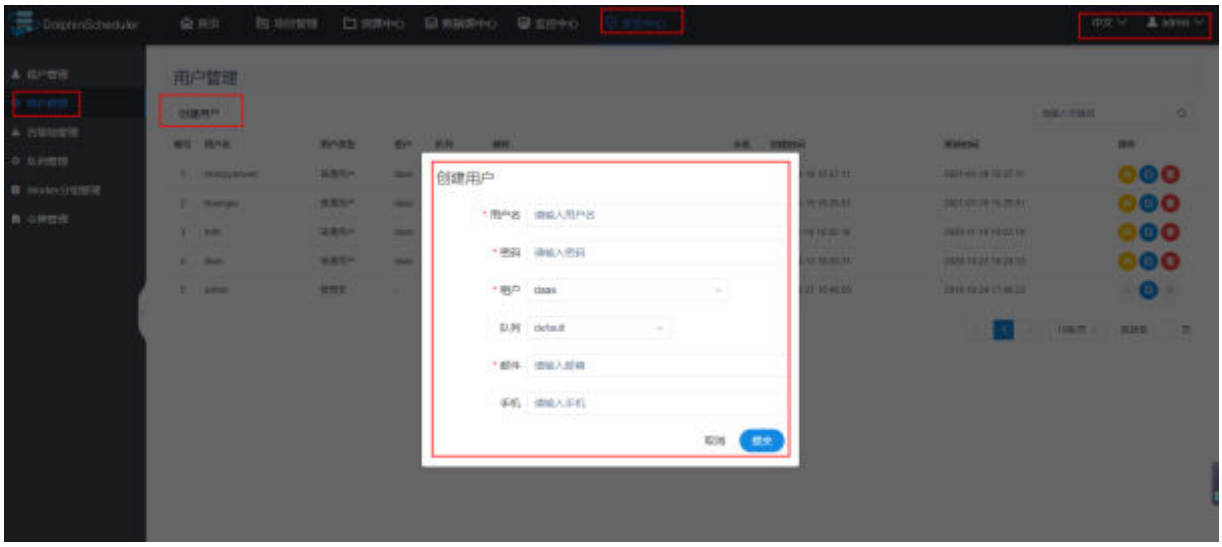
1. 使用admin/dolphinscheduler123 登录，到租户管理中创建hdfs租户，编码一定要是hdfs

图 3-134 图示 1



2. 创建用户，并退出admin账户，登录新创建的用户,再到资源中心上传文件，将flink-sql-submit.jar，hive-sync.sh上传上去即可。

图 3-135 图示 2



3.13 Livy 部署

前置条件说明

1. JDK先配置好(1.8)
2. 安装大数据组件客户端(HDFS\YARN\HIVE\SPARK)

安装机器

前提

1. CDH中需要有两个Hadoop集群，每个集群下部署一个livy (CDH机器允许部署的情况)

2. CDH机器不允许部署的情况，参考https://deepexi.yuque.com/docs/share/cbb7e60f-7188-41ef-8a72-219f269c07aa?#，将部署机器加入CDH集群管理即可

安装包解压

将安装包解压到opt目录下

```
[root@tools]# unzip apache-livy-0.7.0-incubating-bin.zip -d /opt
```

修改 livy 配置文件

```
[root@tools]# cd /opt/apache-livy-0.7.0-incubating-bin/conf/
[root@tools]# cp livy.conf.template livy.conf
[root@tools]# vim livy.conf
```

修改配置项（非kerberos直接复制以下配置粘贴保存）：

livy.spark.master = yarn

livy.spark.deployMode = cluster

livy.environment = production

livy.impersonation.enabled = true

livy.server.csrf_protection.enabled true

livy.server.port = 8998

livy.server.session.timeout = 3600000

livy.server.recovery.mode = recovery

livy.server.recovery.state-store=filesystem

livy.server.recovery.state-store.url=/tmp/livy

kerberos集群需增加的配置（kerberos用户以及认证文件找相关运维人员要）：

livy.server.launch.kerberos.keytab = /etc/daas/kerberos/prd/kafka.keytab

livy.server.launch.kerberos.principal = kafka/hadoop03.daas.com@DAAS.COM

livy.server.access-control.enabled = false

livy.server.auth.type = kerberos

livy.server.auth.kerberos.principal = HTTP/hadoop1.daas.com@DAAS.COM

livy.server.auth.kerberos.keytab =/etc/daas/kerberos/dev/http.keytab

livy.server.auth.kerberos.name-rules = DEFAULT

修改 livy 环境变量配置文件

```
[root@tools]# cp livy-env.sh.template livy-env.sh
[root@tools]# vim livy-env.sh
```

修改配置项（以下配置需根据具体环境改动）：

export JAVA_HOME=/usr/java/default

export HADOOP_HOME=/opt/cloudera/parcels/CDH/lib/hadoop

export SPARK_CONF_DIR=/opt/cloudera/parcels/CDH/lib/spark/conf

export SPARK_HOME=/opt/cloudera/parcels/CDH/lib/spark

export HADOOP_CONF_DIR=/opt/cloudera/parcels/CDH/lib/hadoop/etc/hadoop

export LIVY_LOG_DIR=/var/log/livy

export LIVY_PID_DIR=/var/run/livy

export LIVY_SERVER_JAVA_OPTS="-Xmx8g"

修改 livy 黑名单配置文件、目录权限修改

```
[root@tools]# cp spark-blacklist.conf.template spark-blacklist.conf
[root@tools]#chmod -R 775 /opt/apache-livy-0.7.0-incubating-bin
```

配置数据规则引擎运行环境

```
hadoop fs -mkdir /griffin/
hadoop fs -put -f /opt/griffin-measure.jar /griffin/
hadoop fs -mkdir -p /home/spark_conf/
hadoop fs -put -f /opt/cloudera/parcels/CDH/lib/hive/conf/hive-site.xml /home/spark_conf/
```

启动 livy

```
[root@tools]# sh  /opt/apache-livy-0.7.0-incubating-bin/bin/livy-server start
```

livy 其它命令相关

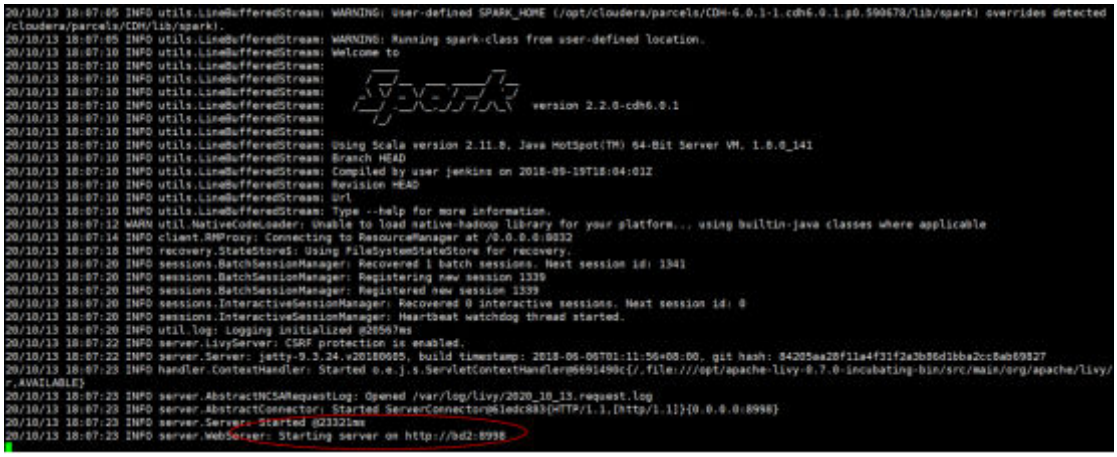
停止命令：sh bin/livy-server stop

状态命令：sh bin/livy-server status

日志查看命令：tail -200f /var/log/livy/livy-root-server.out

显示以下界面无报错则正常：

图 3-136 livy 其它命令相关



3.14 Daas-develop-agent 部署

服务介绍

- 背景：原dass开发平台中实时任务是采用api接口的方式去停止yarn任务，停止速度慢。据客户方滔博反馈，当任务运行比较久是任务数据比较多，停止起来速度非常慢，严重影响体验，必须优化，因此采用在yarn部署agent，通过agent执行flink命令行的方式去停止yarn任务。
- 定位：本项目现期功能是用来停止yarn实时任务，未来不排除拓展功能。

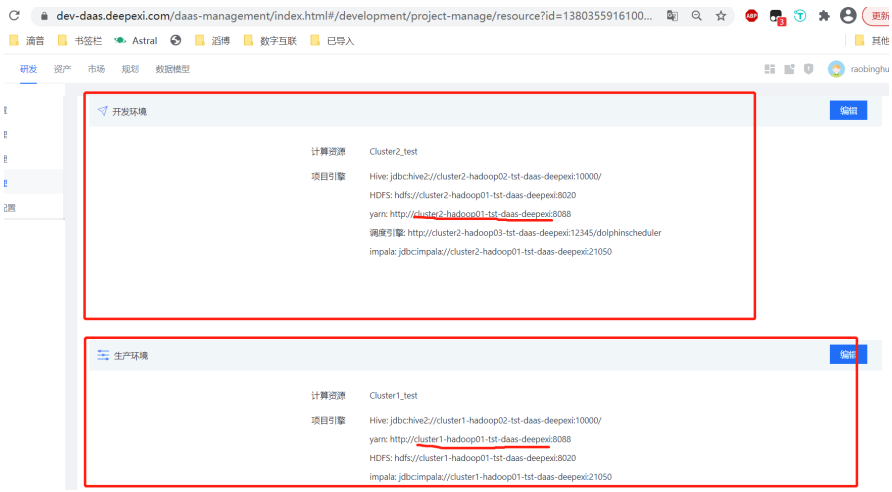
环境要求

JDK已安装好(1.8)

安装机器

项目使用的开发环境、生产环境yarn资源所在机器，如图：

图 3-137 安装机器



上传 jar 包

步骤1 登录项目中配置的yarn资源所在机器,即上一步骤所指机器

步骤2 创建目录

1. cd /data
2. mkdir daas-develop-agent
3. cd daas-develop-agent

步骤3 上传deepexi-daas-develop-agent-1.0.0-SNAPSHOT.jar至当前目录

----结束

修改配置

1. 新建启动脚本start.sh，内容如下：

```
nohup java -Denv=dev-test -Dserver.port=9099 -Drocketmq.nameServer=&quot;10.54.16.9:9876&quot; -Drocketmq.topic.group.tag.realtime.job.operate=cluster2-hadoop01-tst-daas-deepexi -jar deepexi-daas-develop-agent-1.0.0-SNAPSHOT.jar &gt; dev-test-log &amp;
# uat环境
nohup java -Denv=uat -Dserver.port=9098 -Drocketmq.nameServer=&quot;10.54.16.39:9876;10.54.16.39:9877&quot; -Drocketmq.topic.group.tag.realtime.job.operate=cluster2-hadoop01-tst-daas-deepexi -jar deepexi-daas-develop-agent-1.0.0-SNAPSHOT.jar &gt; uat-log &amp;
# prd环境
nohup java -Denv=prd -Dserver.port=9097 -Drocketmq.nameServer=&quot;10.54.16.31:9876;10.54.16.31:9877&quot; -Drocketmq.topic.group.tag.realtime.job.operate=cluster2-hadoop01-tst-daas-deepexi -jar deepexi-daas-develop-agent-1.0.0-SNAPSHOT.jar &gt; prd-log &amp;
```

2. 配置项说明

   a. -Drocketmq.nameServer="10.54.16.9:9876"

      对应环境的rocketmq的nameServer地址

   b. -Drocketmq.topic.group.tag.realtime.job.operate=cluster2-hadoop01-tst-daas-deepexi

      cluster2-hadoop01-tst-daas-deepexi：为当环境的yarn资源域名（无域名则使用ip）替换“.”为“-”的结果（如果域名为hadoop1.daas.com —> hadoop1-daas-com）

3. 配置附加说明

   因为集群资源现在是各个环境共用的，开发/测试/生产/...各个环境都可以使用此集群的yarn，因此在配置的时候需要把所有集群环境都配置上。启动时会启动多个实例来处理不同环境的任务。配置例如下图：

   图 3-138 配置附加说明

```
# agent项目启动
# 开发测试环境
nohup java -Denv=dev-test -Dserver.port=9099 -Drocketmq.nameServer="10.54.16.9:9876" -Drocketmq.topic.group.tag.realtime.job.operate=cluster2-hadoop01-tst-daas-deepexi -jar deepexi-daas-develop-agent-1.0.0-SNAPSHOT.jar > dev-test-log &

# uat环境
nohup java -Denv=uat -Dserver.port=9098 -Drocketmq.nameServer="10.54.16.39:9876;10.54.16.39:9877" -Drocketmq.topic.group.tag.realtime.job.operate=cluster2-hadoop01-tst-daas-deepexi -jar deepexi-daas-develop-agent-1.0.0-SNAPSHOT.jar > uat-log &

# prd环境
nohup java -Denv=prd -Dserver.port=9097 -Drocketmq.nameServer="10.54.16.31:9876;10.54.16.31:9877" -Drocketmq.topic.group.tag.realtime.job.operate=cluster2-hadoop01-tst-daas-deepexi -jar deepexi-daas-develop-agent-1.0.0-SNAPSHOT.jar > prd-log &

# 演练环境
nohup java -Denv=yl -Dserver.port=9096 -Drocketmq.nameServer="10.57.4.14:9876" -Drocketmq.topic.group.tag.realtime.job.operate=cluster2-hadoop01-tst-daas-deepexi -jar deepexi-daas-develop-agent-1.0.0-SNAPSHOT.jar > yl-log &

# 华为环境
nohup java -Denv=hw -Dserver.port=9094 -Drocketmq.nameServer="rocketmq1.applife:9876;rocketmq2.applife:9876;rocketmq3.applife:9876" -Drocketmq.topic.group.tag.realtime.job.operate=cluster2-hadoop01-tst-daas-deepexi -jar deepexi-daas-develop-agent-1.0.0-SNAPSHOT.jar > hw-log &
```

   在客户的环境上部署时，通常来说同一套yarn资源不会存在开发/测试/生产/...各个环境共用的问题，因此只需要配置对应环境信息即可，启动时只启一个实例。

4. 新建启动脚本stop.sh，内容如下：

```
pid=$(ps -ef | grep &quot;deepexi-daas-develop-agent&quot; | grep -v grep | awk '{print $2}')

if [ -n &quot;$pid&quot; ]; then
        kill -9 $pid;
        echo &quot;程序停止成功！ &quot;;
```

```
else
        echo &amp;quot;程序不存在！&amp;quot;
fi
```

启动

```
sh start.sh
```

3.15 Flink 部署

flink1.12 jar 下载

cd /opt

wget https://archive.apache.org/dist/flink/flink-1.12.0/flink-1.12.0-bin-scala_2.11.tgz

解压

tar -zxvf flink-1.12.0-bin-scala_2.11.tgz

修改配置文件

```
vim conf/flink-conf.yaml
```

把值修改成以下内容

说明

cluster2-hadoop01-tst-daas-deepexi:8020只是本案例的，具体替换成集群Namenode端口

state.backend: rocksdb

state.checkpoints.dir: hdfs://cluster2-hadoop01-tst-daas-deepexi:8020/daas/flink/checkpoints/

state.savepoints.dir: hdfs://cluster2-hadoop01-tst-daas-deepexi:8020/daas/flink/savepoints/

state.backend.incremental: true

state.checkpoints.num-retained: 3

jobmanager.archive.fs.dir: hdfs://cluster2-hadoop01-tst-daas-deepexi:8020/daas/flink/flink-jobs/

historyserver.web.address: cluster2-hadoop01-tst-daas-deepexi

historyserver.web.port: 8082

historyserver.archive.fs.dir: hdfs://cluster2-hadoop01-tst-daas-deepexi:8020/daas/flink/flink-jobs/

修改环境变量

```
vim /etc/profile
export FLINK_HOME=/opt/flink
export PATH=$PATH:$FLINK_HOME/bin
```

分发服务器

把修改好的 flink文件 分发到集群所有机子

测试是否成功

```
flink  run -m yarn-cluster  /opt/flink/examples/batch/WordCount.jar
```

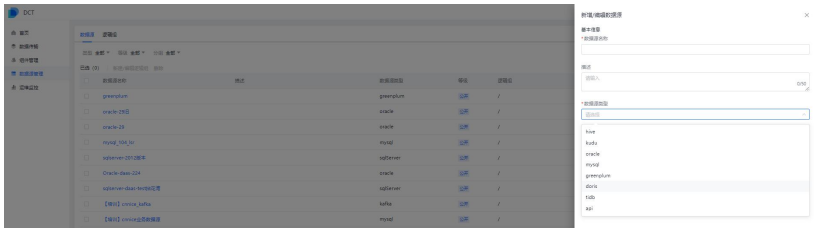
3.16 产品操作

3.16.1 数据集成

数据源管理

支持通过手动配置或批量导入的方式接入数据源

图 3-139 数据源管理

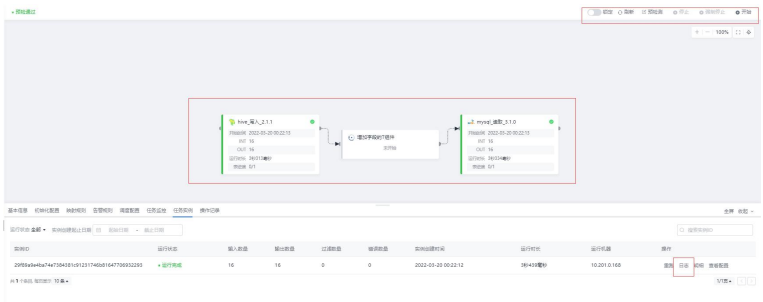


任务设计

- 拖拉拽设计

  支持组件化拖拉拽方式快速完成数据同步任务的设计，用户在完成设计后，可通过预检测确认设计无误，通过开始、停止、刷新、查看日志等操作调试任务。

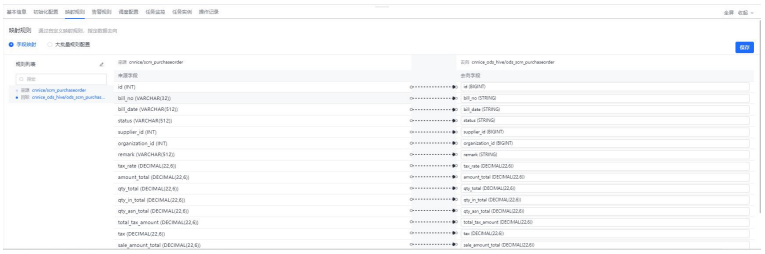
  图 3-140 拖拉拽设计



- 字段自动映射

配置好源表及目标表后，同名字段将自动映射，并支持用户手动调整映射关系。

图 3-141 字段自动映射



- 调度配置

  支持手动启动、定时启动两种方式，实现离线数据同步任务的调度。其中，定时启动方式下，可设置生效日期，及年、月、周、日、时、分、秒等不同颗粒度的调度频率及起调时间点，并生成 cron 表达式。

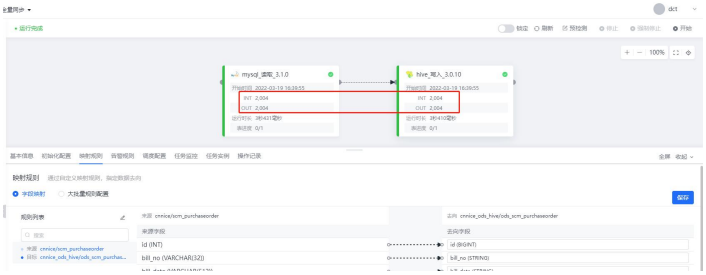
图 3-142 调度配置

数据同步模式

- 全量同步

  离线同步模式中，在不设置任务前置过滤 SQL 的情况下，同步任务将执行对源端数据向目标端数据的全量同步。在组件上可以看到同步的数据量及运行时长等信息。

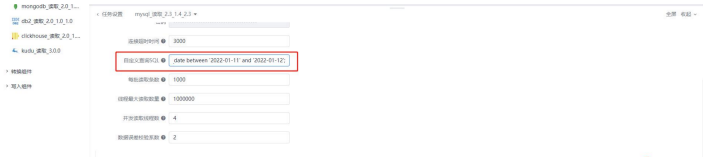
图 3-143 全量同步



- 数据补录

  当发生源端历史数据变更等情况时，通过在读取组件自定义 SQL，可以添加针对日期字段进行过滤的 WHERE 条件，可以实现针对特定时间段的数据补录需要。

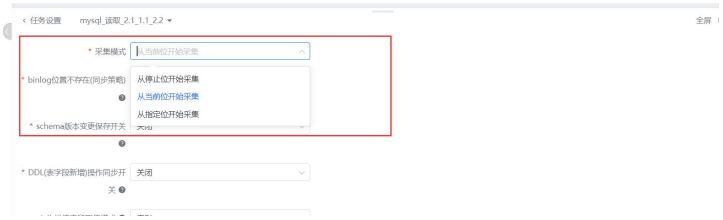
图 3-144 数据补录



- 增量同步

  支持以当前位置、停止位置、指定位置作为采集起始点的不同模式的实时增量同步。其中，基于停止位置，可实现断点续传，使得网络中断等意外情况发生并恢复后，数据依旧可以完整无误的进行传输。

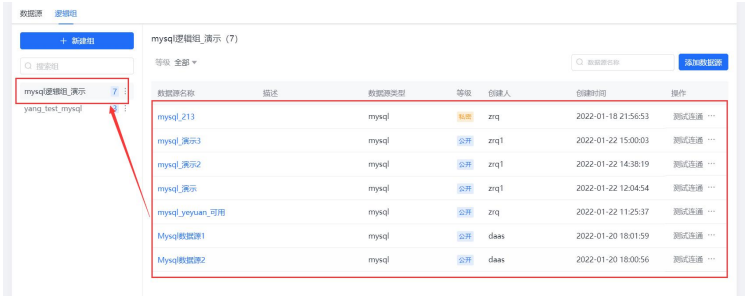
图 3-145 增量同步



- 数据整合分发

  针对分库分表数据整合场景及数据分发场景，可基于数据源逻辑组，通过多对一、一对多任务来实现批量任务的创建及运行。

图 3-146 数据整合分发



任务运维

1. 任务级监控

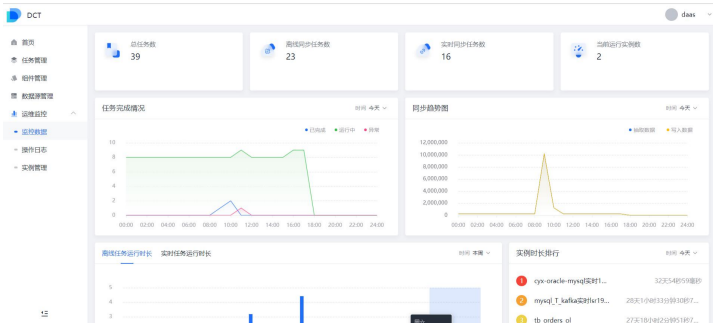
   任务监控支持查看任务运行状态、同步耗时、输入输出的记录数及趋势、磁盘占用及剩余空间信息、CPU 负载及内存使用性能曲线等信息。

图 3-147 任务级监控

2. 平台级监控

   运维监控是针对所有数据集成任务进行的统一监控，包括数据监控、操作日志查看、实例管理等功能。其中，监控数据可以查看包含总任务数、离线任务数、实时任务数、当前运行实例、任务完成总体曲线、整体同步趋势、任务运行时长趋势及排行、资源使用情况等信息。

图 3-148 平台级监控



3.16.2 数据开发与治理

3.16.2.1 数据源管理与数据标准管理

数据源管理

支持对 OLAP 数据源、关系型数据源、NoSQL 数据库、文件数据源、消息队列数据源等异构数据源类型进行接入，屏蔽不同数据库之间的差异，实现数据源的统一管理。

图 3-149 数据源管理 1

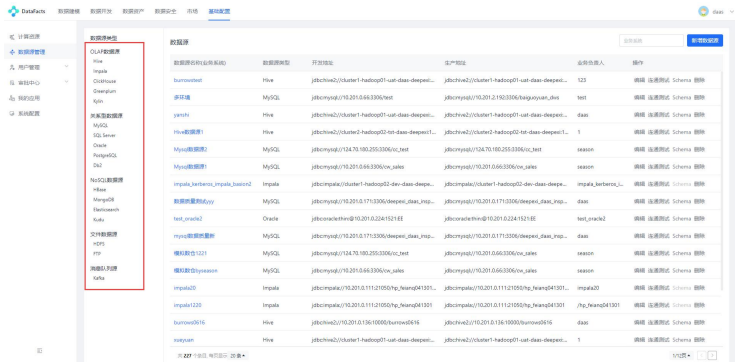
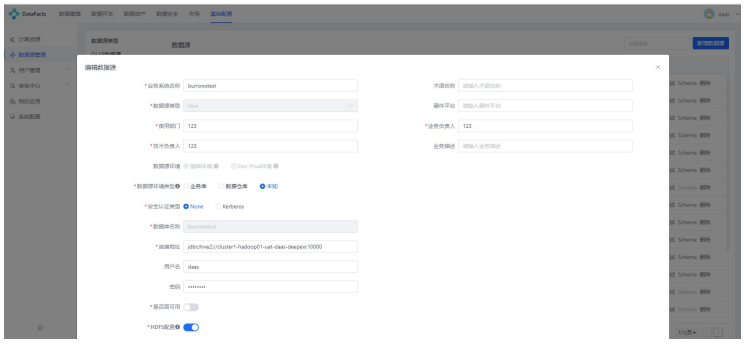


图 3-150 数据源管理 2

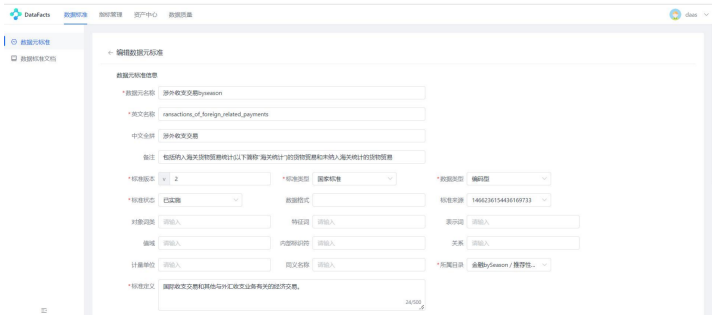


数据标准管理

- 数据元标准

通过“数据标准”，可以对数据元标准进行统一的规范化定义，包括标准名称、标准版本、标准类型、标准来源、数据类型、数据格式、值域范围、计量单位、标准定义及标准代码集等内容。

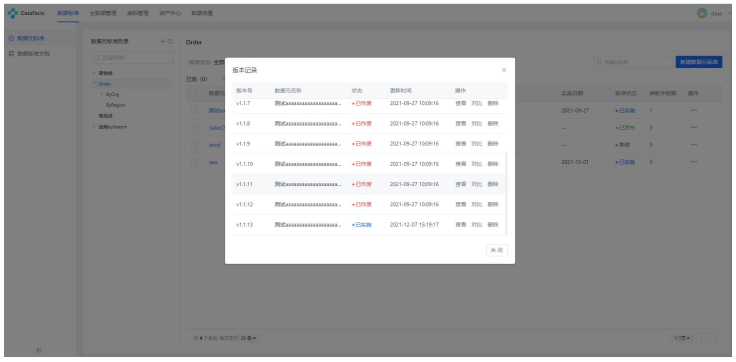
图 3-151 数据标准管理 1



- 标准版本管理

  支持数据标准的版本查看及对比。

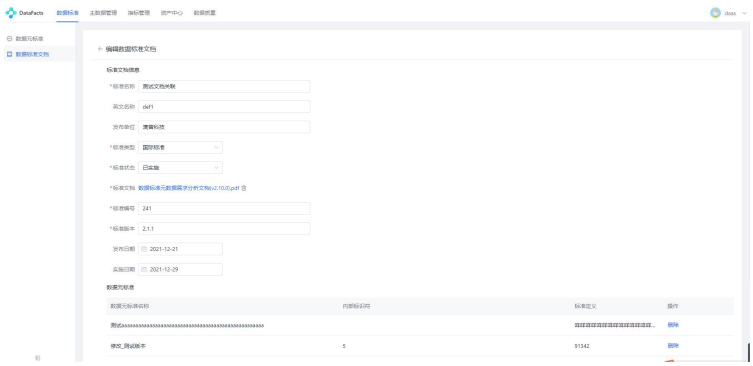
图 3-152 标准版本管理



- 数据标准文档管理

  支持数据标准来源文档的上传及管理。创建数据标准文件，需上传相关文档，明确该标准文件的类型、实施状态、版本号等关键信息。创建后，可下载文件及关联标准文档所规定的数据元标准。其中，实施状态包含草稿、已发布、已实施、已作废等状态。

图 3-153 数据标准文档管理



3.16.2.2 数据建模

1. 数据分层分域

   支持数据按如 ODS、DWD、DWS、ADS 等数据分层设置，支持数据按照主题、部门等进行数据域设置。为模型设计从业务和技术的角度提供分类分层框架，是数仓规划的关键基础。

图 3-154 数据分层分域 1

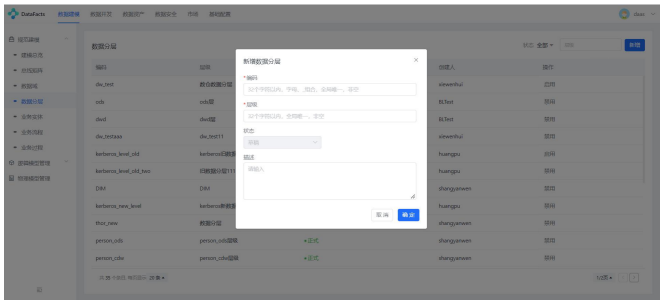
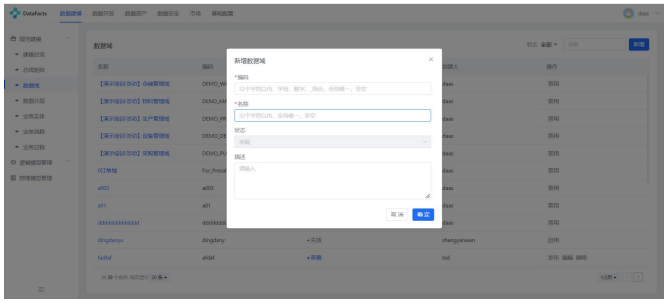


图 3-155 数据分层分域 2



2. 业务实体管理

   基于对企业各业务线的业务实体调研，对不同业务实体所涉及的数据维度进行汇总，如时间、日期、地区、部门等。

图 3-156 业务实体管理 1

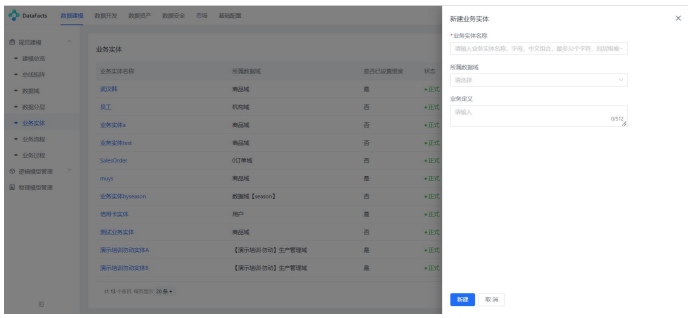
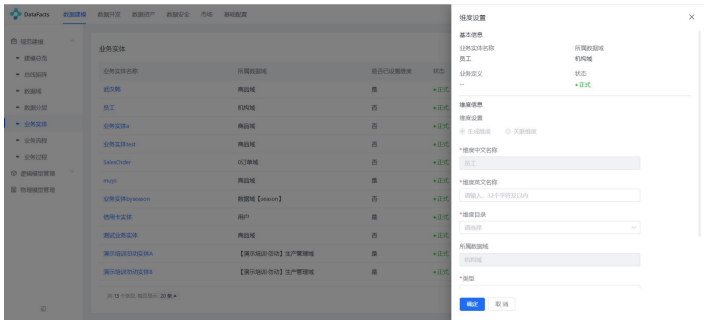


图 3-157 业务实体管理 2



3. 业务流程及过程管理

   基于对企业业务流程及业务过程调研，对需要构建的事实模型进行汇总。其中，业务流程包含多个业务过程。

图 3-158 业务流程及过程管理 1

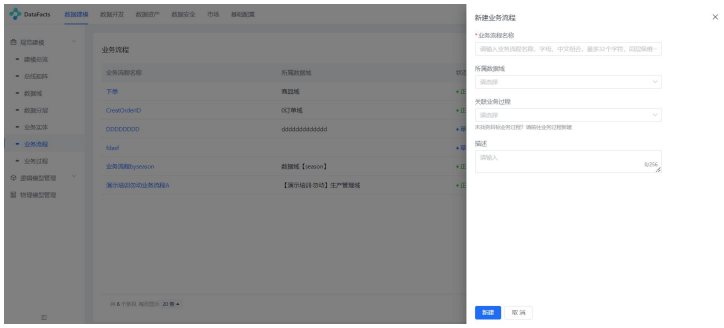
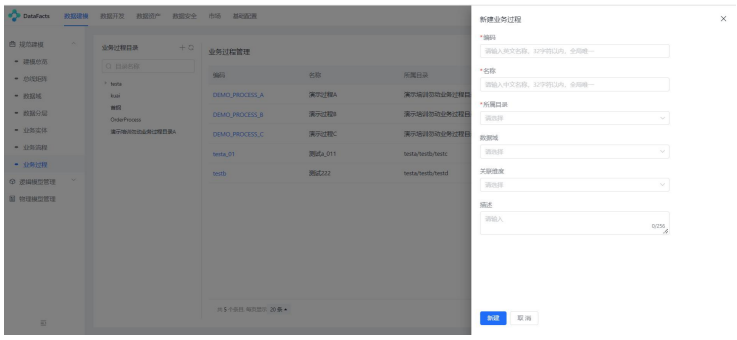


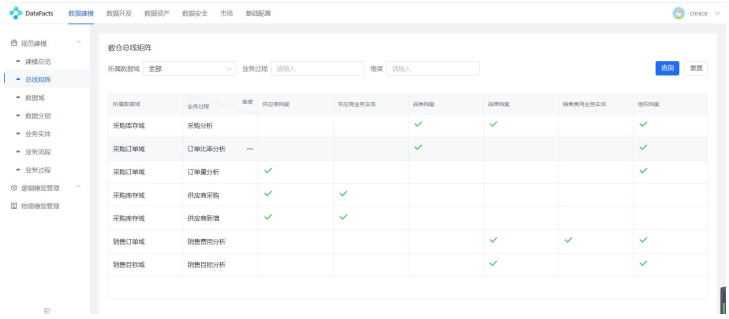
图 3-159 业务流程及过程管理 2



4. 总线矩阵

   完成业务实体、业务流程、业务过程的构建后，平台会形成由待填充的事实逻辑模型和维度逻辑模型构成的总线矩阵。模型设计人员可以方便的按照矩阵的指引，完成维度表、事实表等逻辑模型设计。

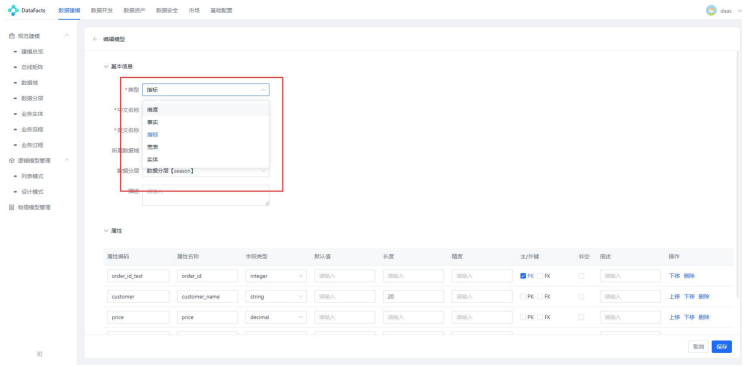
图 3-160 总线矩阵



5. 逻辑模型创建

   在逻辑模型设计中，用户可以定义维度、事实等类型的逻辑模型，新增属性，添加属性编码、名称、字段类型等参数，完成逻辑模型设计，并为其分配到指定的数据域及数据分层中。

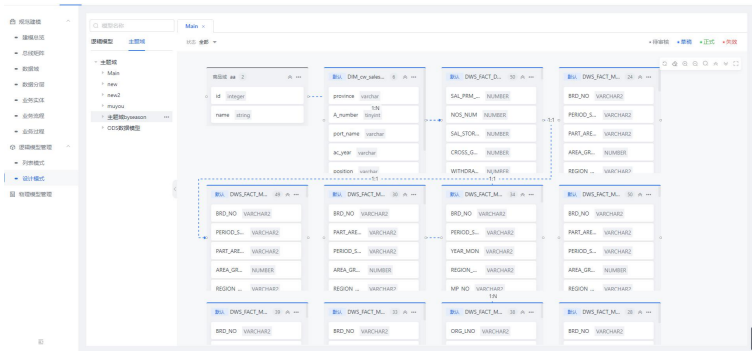
图 3-161 逻辑模型创建



6. 可视化设计

   在设计模式中，可以通过图形化拖拉拽方式，按照数仓规划，配置维度模型和事实模型之间的关联关系，以便于数据开发人员在数据开发过程中，能够准确的理解各模型之间的关系并高效准确的设计 ETL 任务。

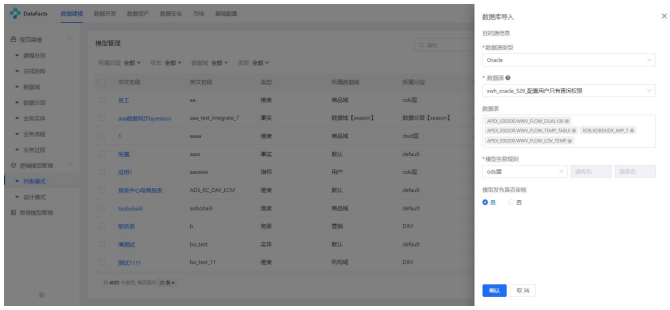
图 3-162 可视化设计



7. 逆向工程

   支持通过数据库导入的方式，快速将外部数据库中的表逆向为逻辑模型，导入到DataFacts平台中，帮助用户在数据同步、ETL 任务迁移等场景中，复用已有的逻辑模型，节省重复建设成本。

图 3-163 逆向工程



8. 免SQL物理化

   通过配置化方式，依照数仓规划指引，快速将过审的逻辑模型物理化到数仓位置，降低技术门槛。

图 3-164 免 SQL 物理化

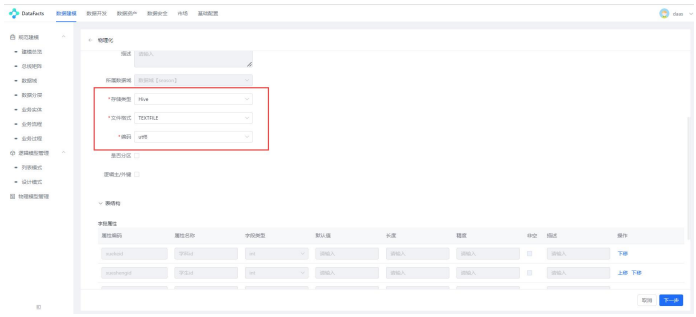
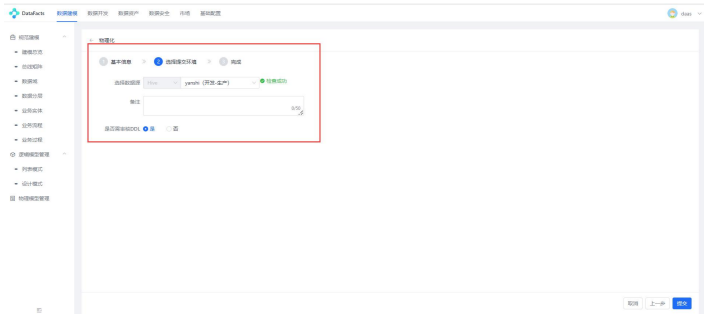


图 3-165 免 SQL 物理化 2

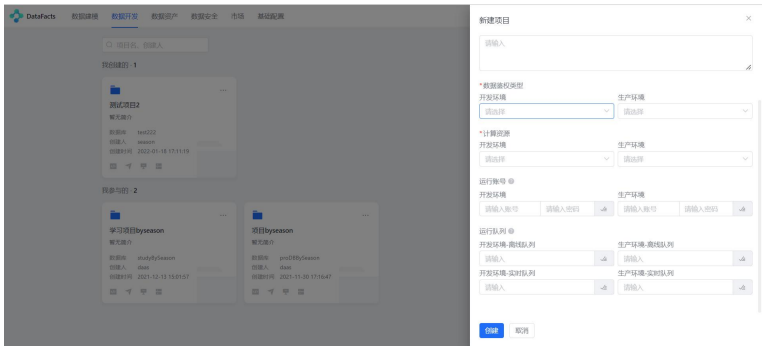


3.16.2.3 数据开发

1. 项目创建

   数据开发项目需为开发环境和生产环境配置数据鉴权模式，选择计算资源组，同时可以指定运行账号及运行队列。

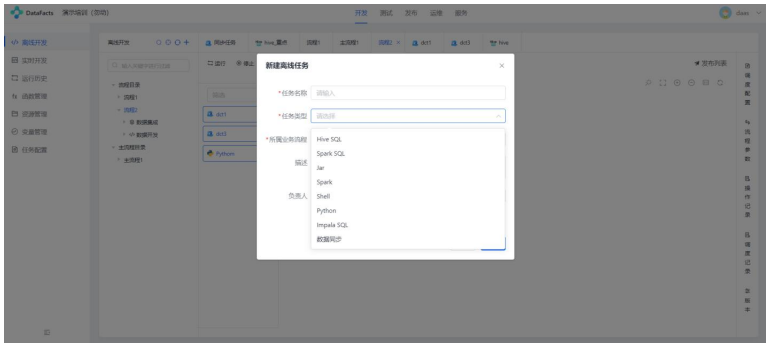
图 3-166 项目创建



2. 离线任务开发

   离线数据开发支持 HiveSQL、ImpalaSQL、SparkSQL、Python、Shell、Jar、Spark 等任务类型。

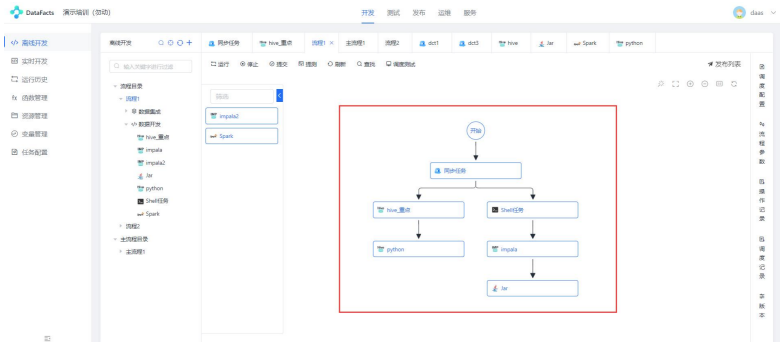
图 3-167 离线任务开发



3. 流程设计

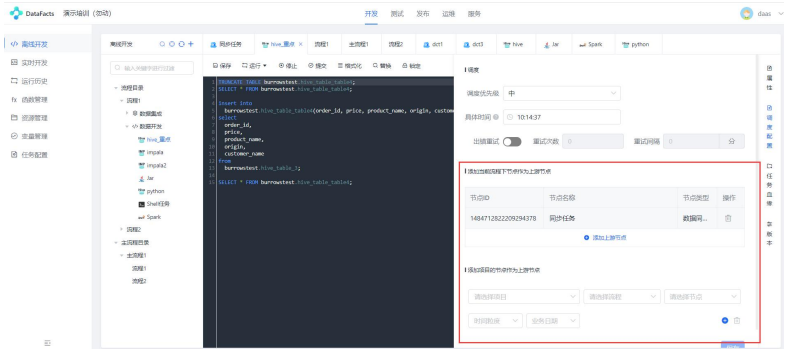
   任务运行以流程为核心，支持拖拉拽方式可视化编排，将多个不同类型的任务按照 DAG原则组织为其中各个子节点。

图 3-168 流程设计 1



除了拖拉拽方式外，在具体的任务节点中，也可以配置化的方式添加本项目上游节点、跨项目依赖的上游节点。

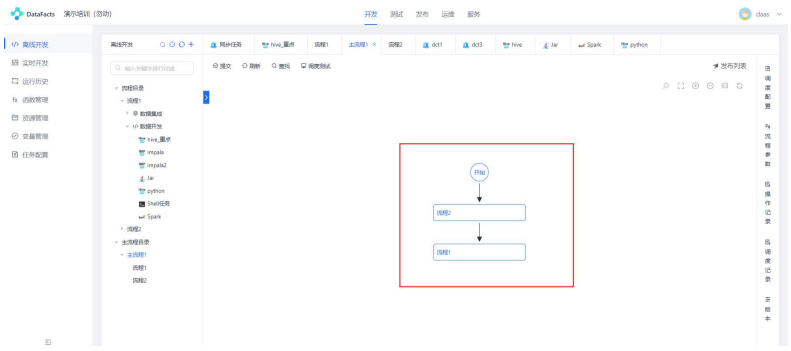
图 3-169 流程设计 2



4. 主流程设计

对于更加复杂的情况，同样可以通过拖拉拽方式可视化编排的方式，将项目内多个流程组织为主流程的形式满足复杂任务的依赖调度。

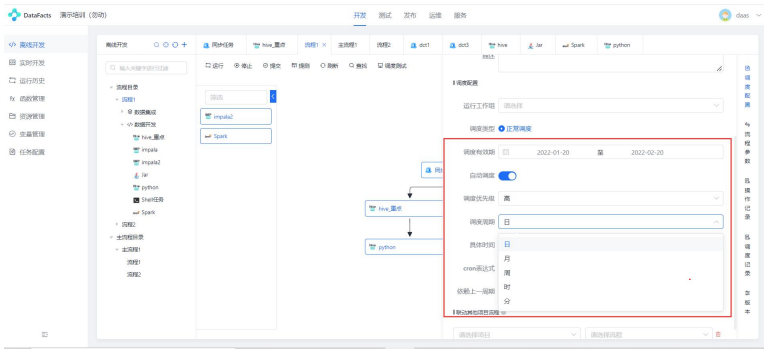
图 3-170 主流程设计



5. 任务调度

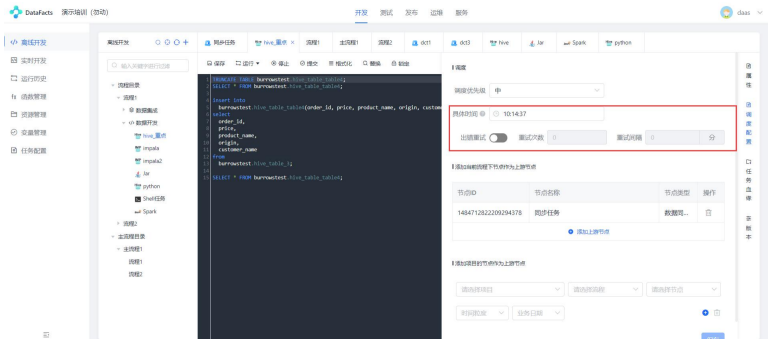
主流程、流程支持日、月、周、时、分等不同精度的周期调度方式。当设置为自动调度模式时，主流程或流程在发布到生产环境并启动后，将在有效期内，按给定的调度周期内的具体时间进行周期性调度。

图 3-171 任务调度 1



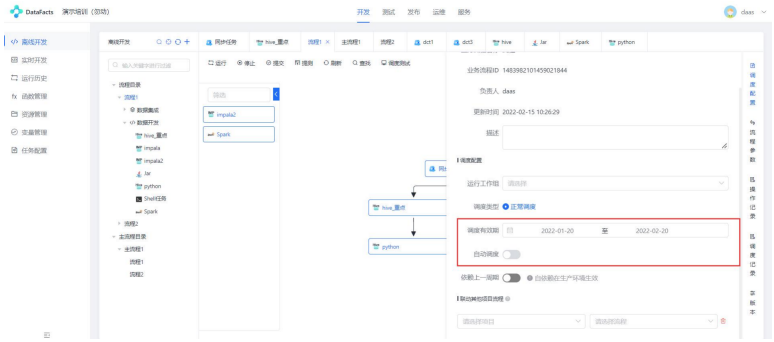
主流程、流程外、任务节点均可设置具体运行时间。因此，同一流程下，不同任务节点在服从上下游顺序逻辑的前提下，均可按照自身的运行需要灵活启动。同时，可以对任务节点，可以设置出错重试次数和间隔时间，以实现任务调度过程中的容错处理。

图 3-172 任务调度 2



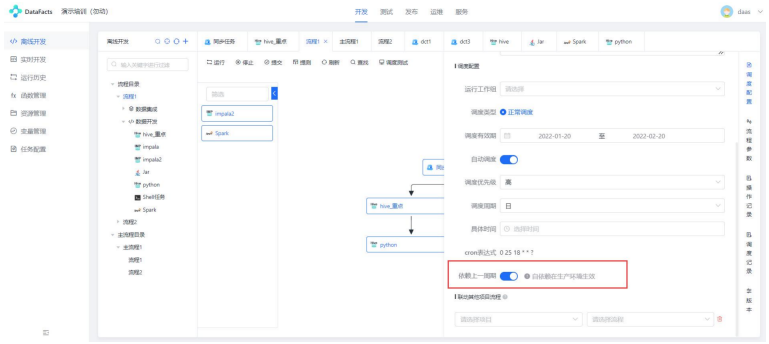
如果不启动自动调度，主流程或流程在发布到生产环境并启动后，可以在有效期内通过手动调度方式运行。

图 3-173 任务调度 3



同时，流程还支持自依赖、跨周期依赖设置，以支持复杂的调度场景需要。

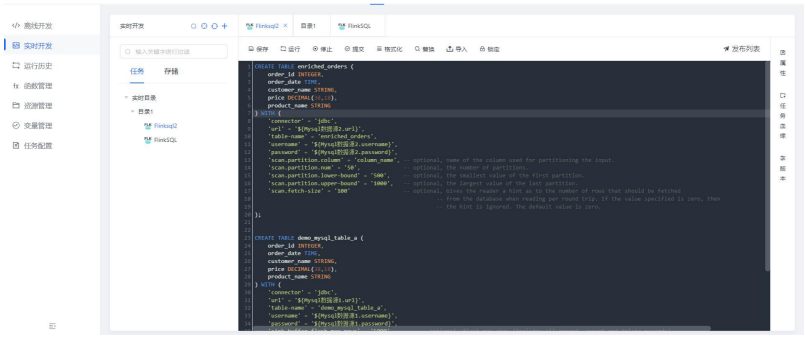
图 3-174 任务调度 4



6. 实时任务开发

   FlinkSQL 任务可以通过模板快速生成 sink 和 source 代码。

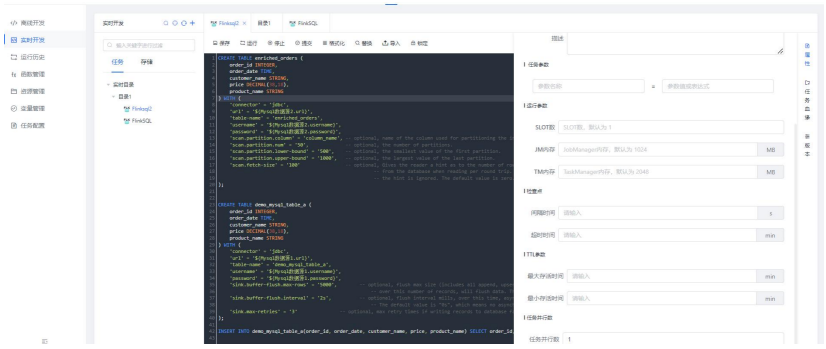
图 3-175 实时任务开发



7. 实时任务调度

   支持任务参数、运行参数、检查点、TTL 参数、任务并行数。支持数据血缘配置和版本管理。

图 3-176 实时任务调度

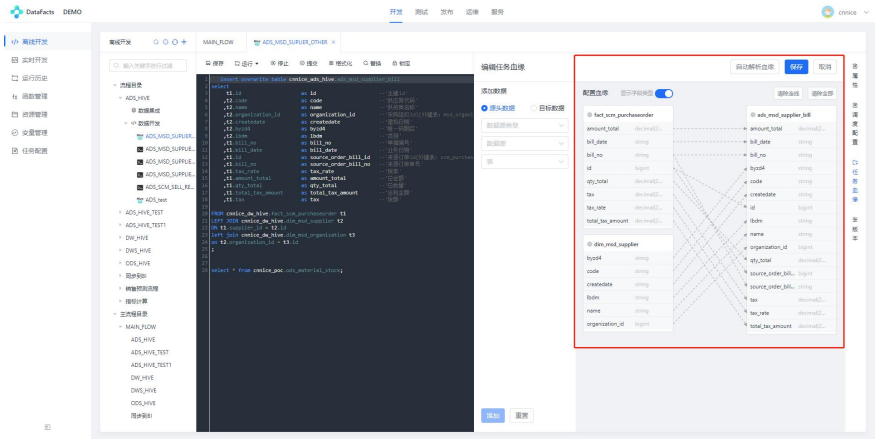


3.16.2.4 数据血缘、资源管理、函数管理和变量管理

数据血缘

在数据采集和数据开发任务中，支持手工配置数据血缘及基于 SQL 解析自动生成数据血缘。在任务发布至生产环境后，即可形成数据血缘影响关系图，直观展示元数据上下游关系，查看到来源于各个相关的数据开发任务的血缘分析、影响分析。

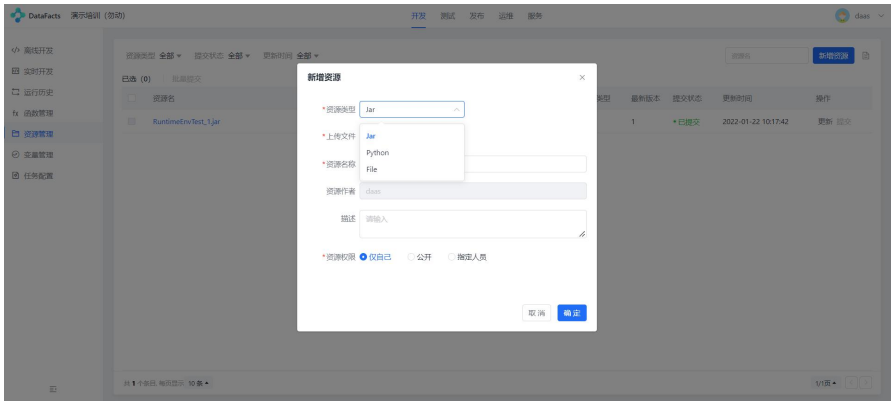
图 3-177 数据血缘



资源管理

函数资源及外部代码资源，可以在资源管理中上传。资源支持 Jar、Python、File 等类型。

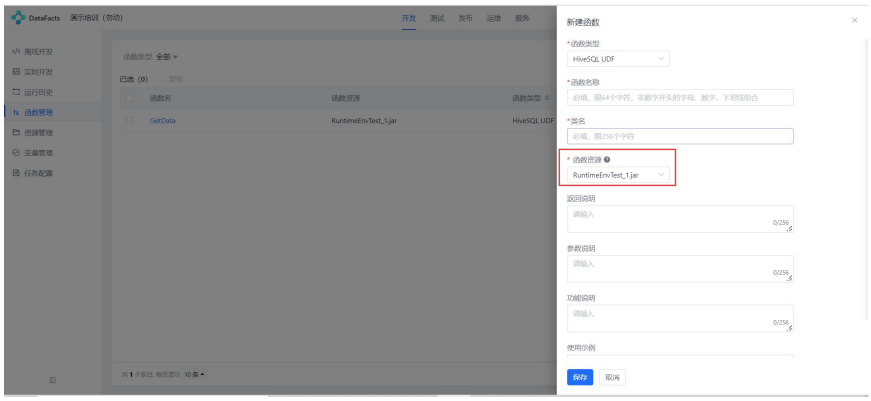
图 3-178 资源管理



函数管理

对于复杂的代码实现场景，可以进行 UDF 封装为函数资源，通过资源管理上传后，通过函数管理进行引用管理，而后在 SQL 或脚本编辑器中进行调用，从而实现代码的高可复用，节省重复编写成本。

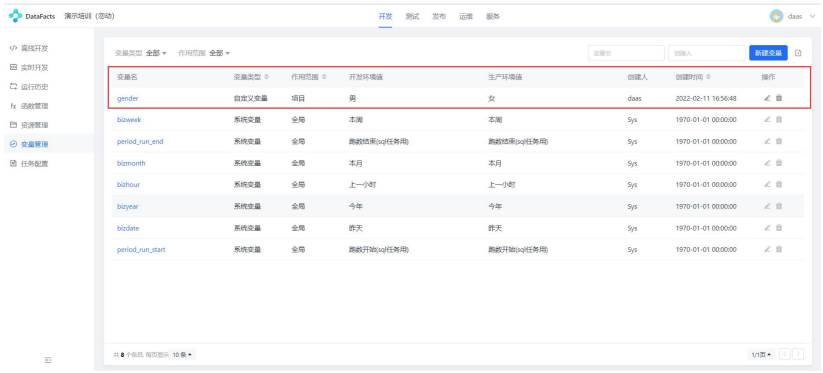
图 3-179 函数管理



变量管理

通过变量管理功能，用户根据开发环境、生产环境的需要，对同一参数设置不同的值，引用变量的任务将根据运行环境自动适配变量值，简化开发流程。

图 3-180 变量管理



3.16.2.5 数据发布 CI/CD、任务运维

数据发布 CI/CD

支持数据发布 CI/CD。数据开发团队完成任务开发后，可以将流程提交到测试环境中进行验证、调整，任务在测试环境的运行过程中，不影响用户继续完成其他开发任务。对于通过测试的任务，数据工程师可以提交到发布流程中，开发 Leader 则可通过查看任务、比对版本来审核数据任务。审核通过后，运维工程师方可进一步发布到生产环境。

图 3-181 数据发布 CI/CD1

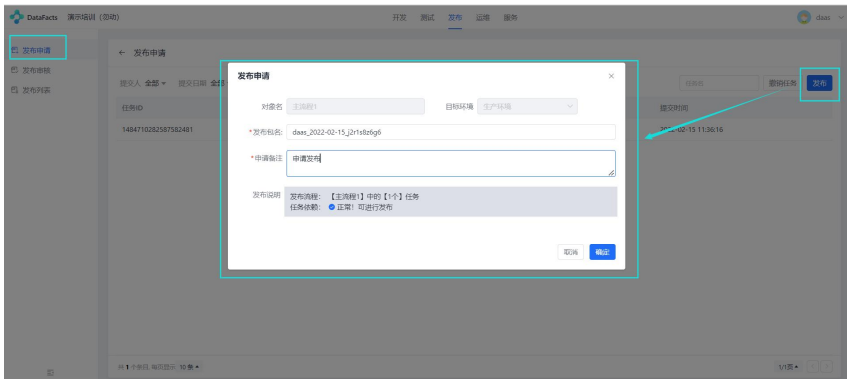


图 3-182 数据发布 CI/CD2

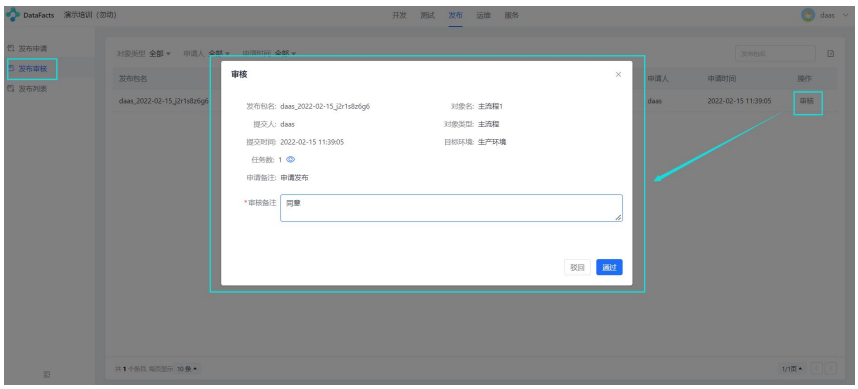


图 3-183 数据发布 CI/CD3

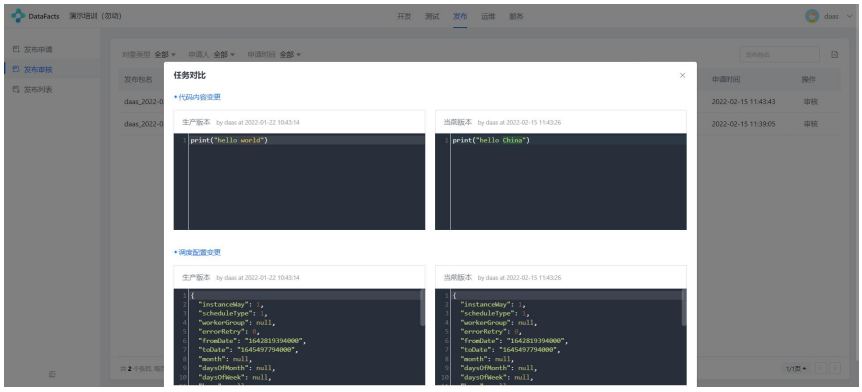
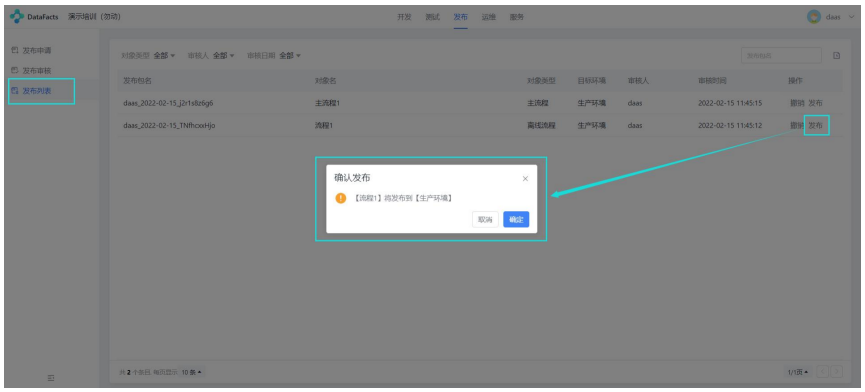


图 3-184 数据发布 CI/CD4

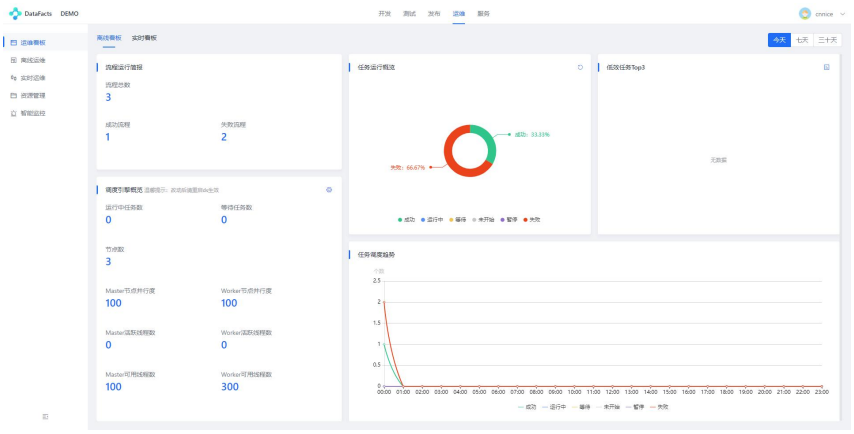


任务运维

1. 运维看板

   支持总览离线任务、实时任务的运行状态、运行时长，及调度引擎的运行情况，帮助数据开发工程师及时发现问题，解决问题，同时达到充分利用资源，快速优化任务的目的。

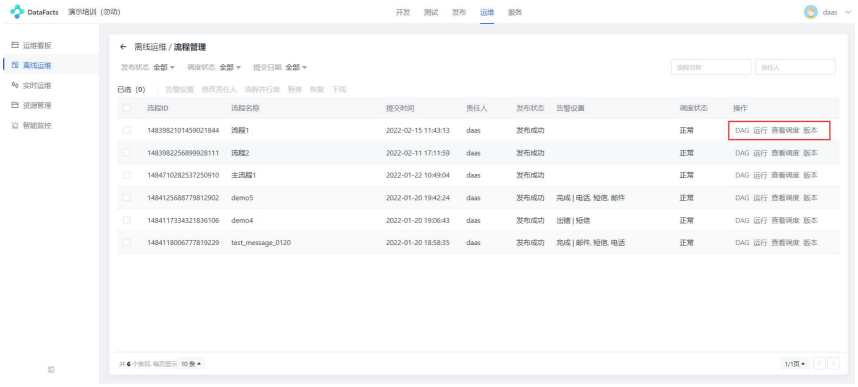
   图 3-185 运维看板



2. 离线任务运维

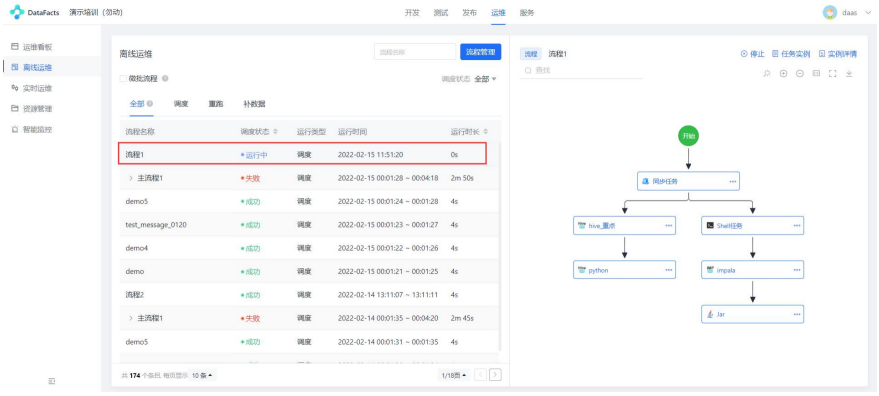
离线运维支持运维工程师对生产环境上的流程进行运行管理，支持重跑、错误重试、停止、置成功、置失败、补数据、配置质量规则、查看日志等操作。流程发布到生产环境后，运维工程师可以在流程管理中运行流程，并支持版本回滚。

图 3-186 离线任务运维 1



每次启动流程，都会生成相应的流程运行实例。每个实例都可以查看到其对应流程版本的 DAG 及对应任务节点版本的代码、资源。

图 3-187 离线任务运维 2

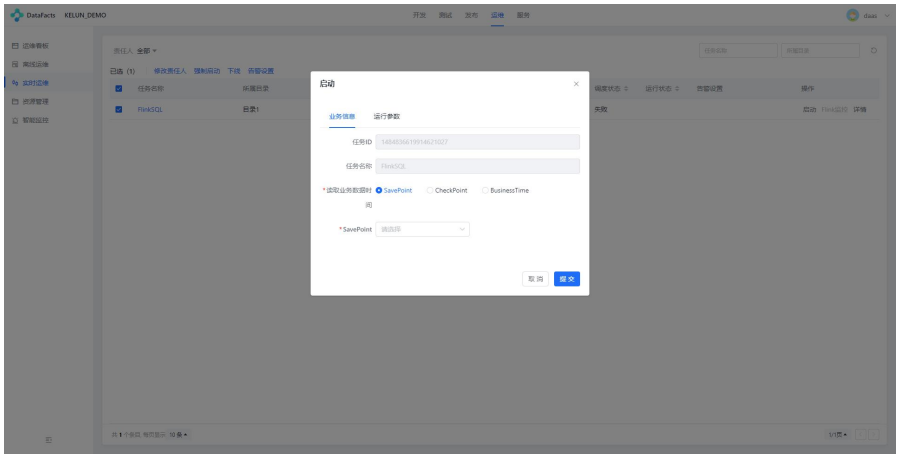


生产环境与开发环境实行严格的权限隔离及资源隔离，不同环境的任务运行所依赖的计算资源、数据资源、源码资源均相互独立，不受另一方影响。

3. 实时任务运维

   支持实时任务启动、停止、告警、查看日志等运维操作。支持通过保存点、检查点、业务时间等方式，来指定要读取的业务数据。

图 3-188 实时任务运维

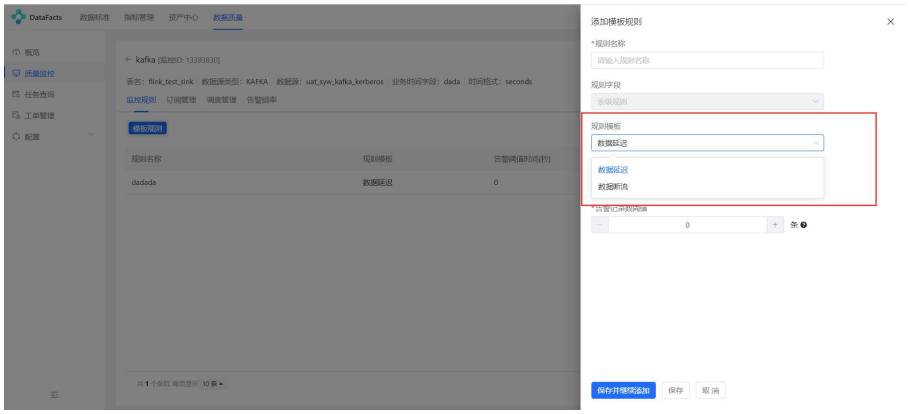


3.16.2.6 数据质量管理

1. 内置质量规则

   内置多种质量规则，如：空值校验、重复数据校验、时延校验、聚合计算校验等校验多种校验方式，支持从合理性、完整性、唯一性、准确性、规范性、一致性、及时性等质量评分指标对数据质量进行监控。

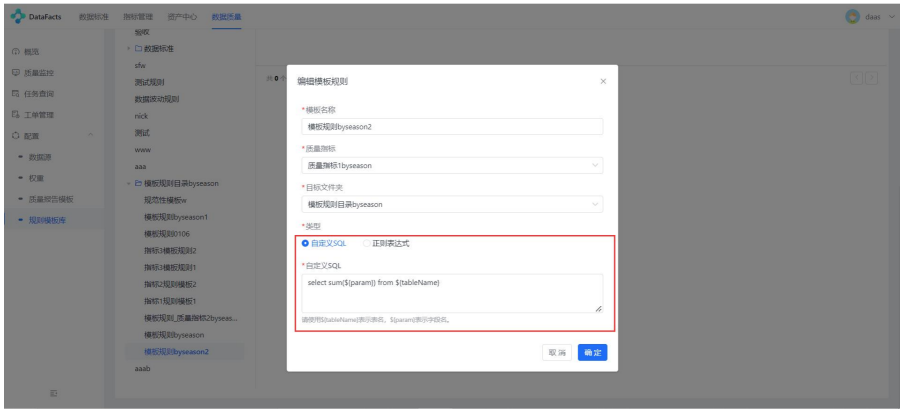
图 3-189 内置质量规则



2. 自定义质量规则

   支持用户通过自定义 SQL 校验、正则表达式校验等自定义的方式进行规则的扩展，实现如值域校验、精度校验等校验方式，形成规则模板以便重复使用。

图 3-190 自定义质量规则



3. 质量加权评分

支持用户自定义质量评分指标，为不同的质量规则关联指标并分配计分权重，从而根据企业的数据质量评估体系，有层次有重点地对数据质量进行评分。

图 3-191 质量加权评分 1

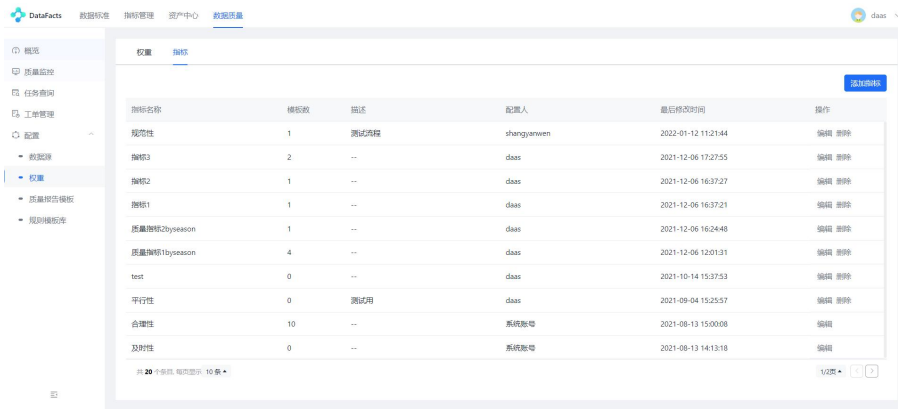
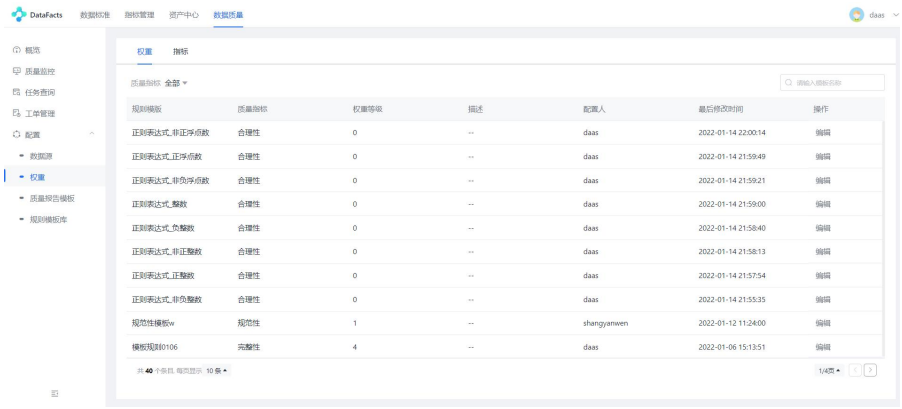


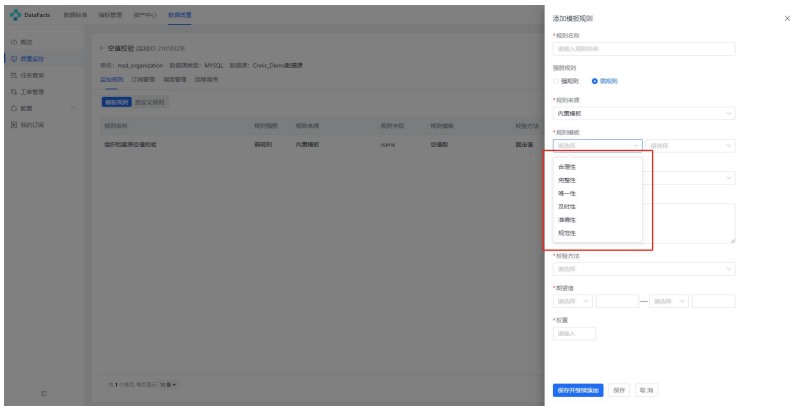
图 3-192 质量加权评分 2



4. 单表质量检测

支持针对 Hive、Kudu、Oracle、MySQL、SQLServer、impala 等数据源类型，从合理性、完整性、唯一性、及时性、准确性、规范性及企业自定义等数据质量指标集成多个质量规则，对指定的单张表进行质量检查。

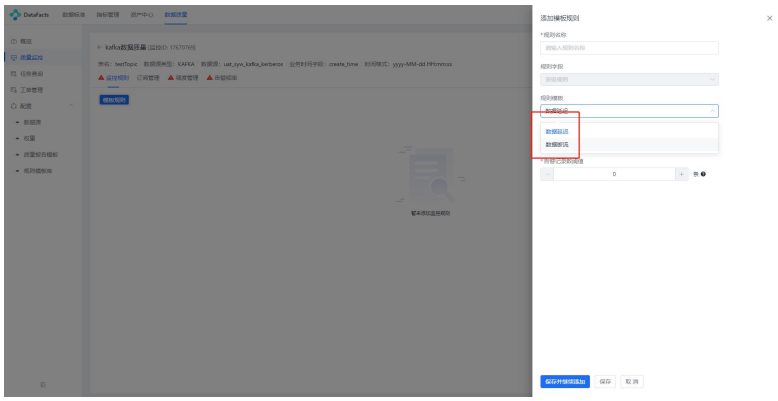
图 3-193 单表质量检测



5. 流式数据质量检测

   针对 Kafka 等流式实时数据的质量稽核，还内置数据延迟和数据断流等规则进行支持。

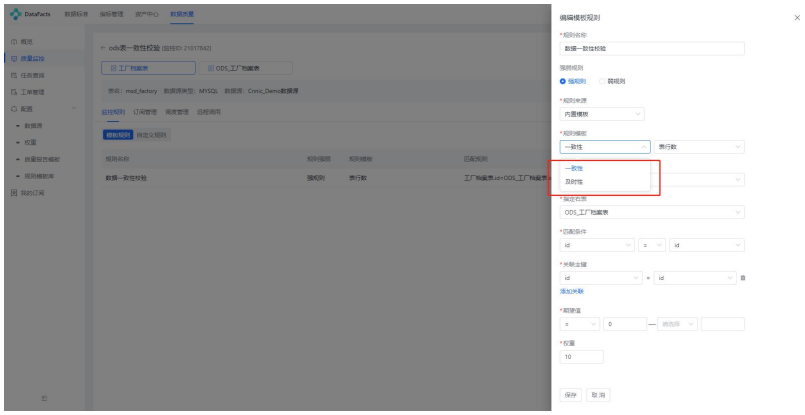
图 3-194 单表质量检测



6. 多表对比分析

   支持针对 Hive、Kudu、Oracle、MySQL、SQLServer、impala 等数据源类型，从一致性、及时性等数据质量指标集成多个质量规则，对指定的多张表进行比对分析，以判断，在数据同步或经 ETL 任务产出后，上下游的数据是否一致，所耗费的时间是否在预期范围内。

图 3-195 多表对比分析



7. 质量规则关联

支持关联内置模板规则、自定义模板库规则及自定义 SQL 规则。

图 3-196 质量规则关联 1

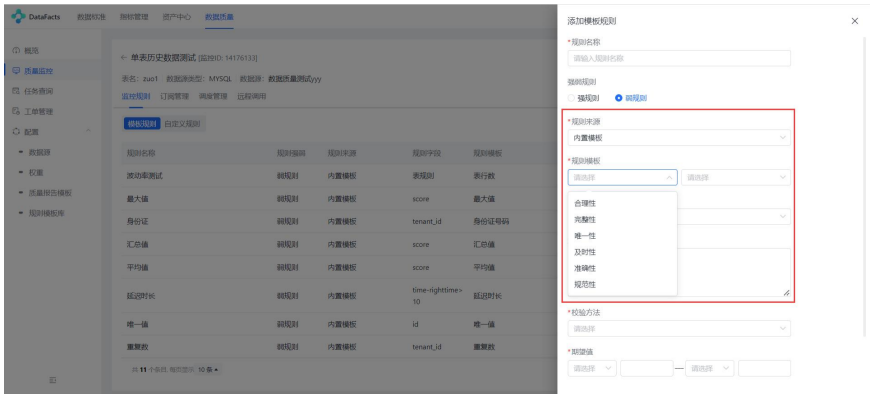


图 3-197 质量规则关联 2

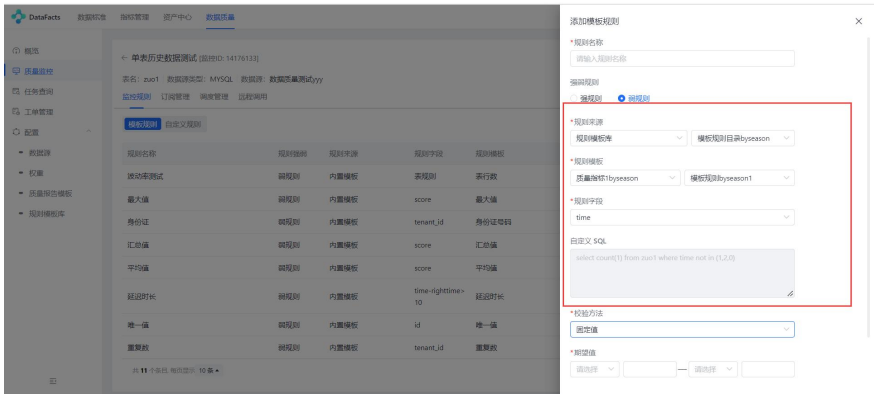
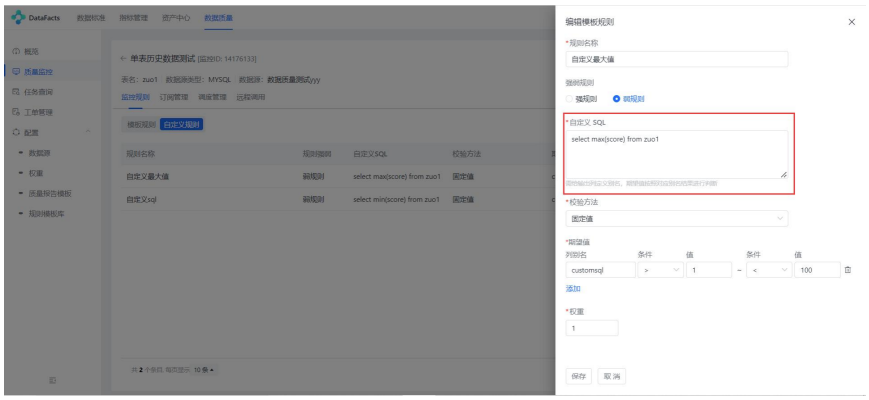


图 3-198 质量规则关联 3



8. 质量规则合规校验

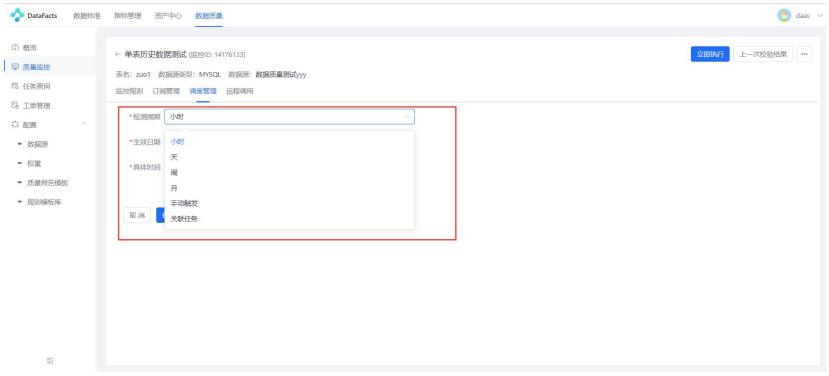
支持通过固定值域区间或周期内统计波动的方式来判定是监测目标是否合规。

图 3-199 质量规则关联 3

9. 质量检测任务调度

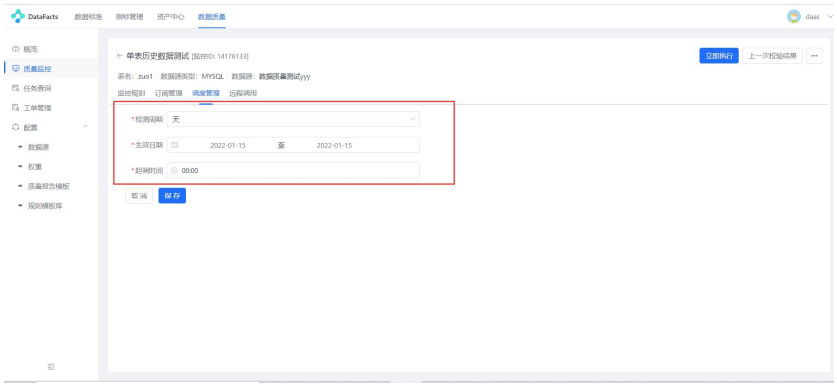
   支持周期性调度、手动触发、关联任务、远程触发等多种调度方式。

图 3-200 质量检测任务调度 1



周期性调度支持月、周、天、小时、分、秒等粒度的检测周期设置，并支持生效时间段、重复日期或时刻、具体起调时间点等设置。

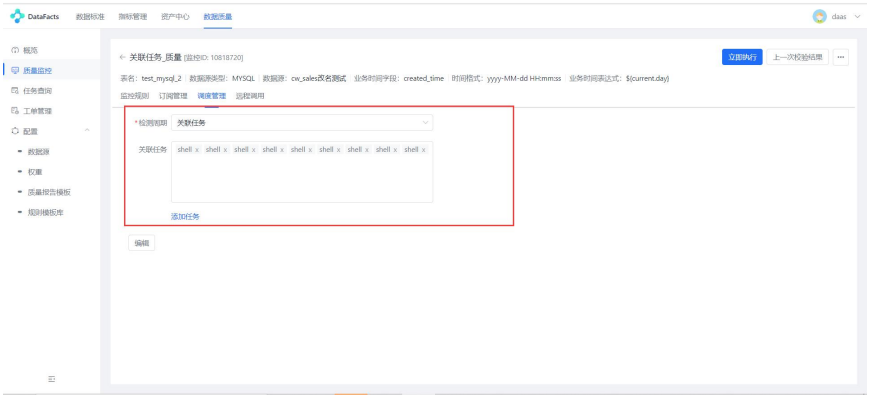
图 3-201 质量检测任务调度 2



10. 关联ETL任务

    关联任务调度支持数据质量任务关联多个数据开发任务。当数据开发任务运行一个周期后，数据质量任务会紧随其后对其产出的数据进行稽核，如稽核结果不达标，则会产生相应告警。如不达标的质量规则中包含强规则，则还会自动阻塞该数据开发任务的下游节点。

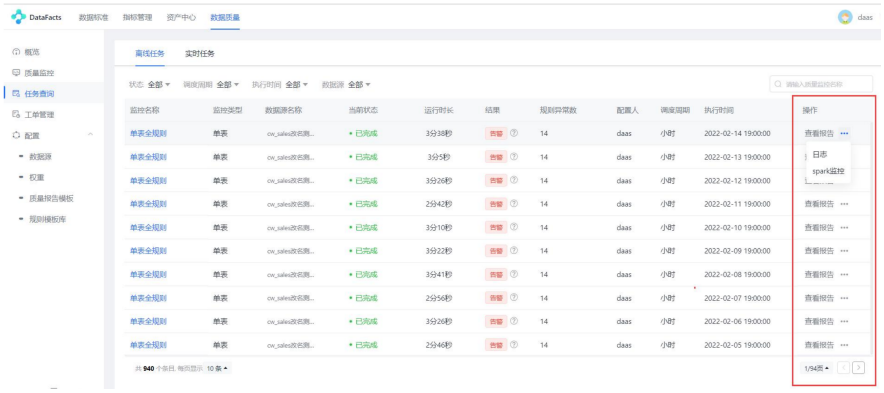
图 3-202 关联 ETL 任务



11. 质量监控报告

数据质量任务每次执行后会产生相应的实例。每个实例都包含对应的数据质量检测报告及执行日志。在任务运行实例监控列表中，可以直观的总览任务执行状态、运行时长、校验结果告警状态、错误规则数量等维度情况。

图 3-203 质量监控报告 1



数据质量报告以 Web 可视化的方式呈现。报告中，用户可以查看数据质量评分、总体规则数量及错误告警数量、任务时长详情等情况总览，查看触发错误告警的未通过规则列表、趋势、详情等信息。其中，数据质量规则列表清晰展示了各规则所检测的表字段、比对详情、校验结果等信息。

图 3-204 质量监控报告 2

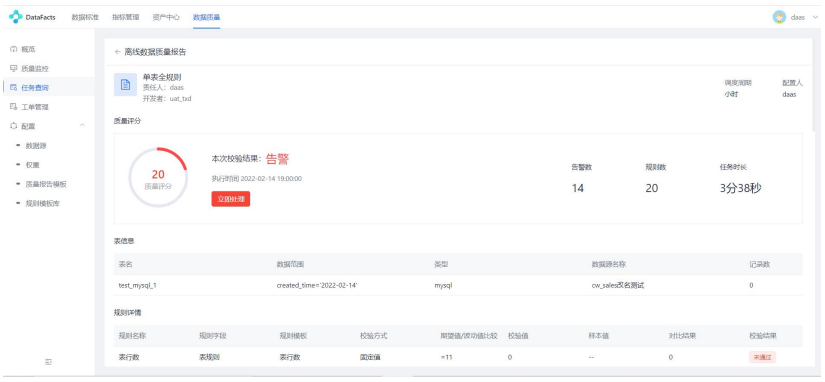
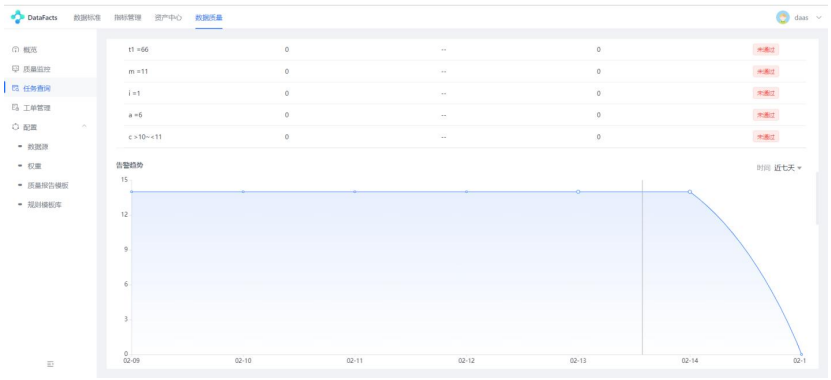
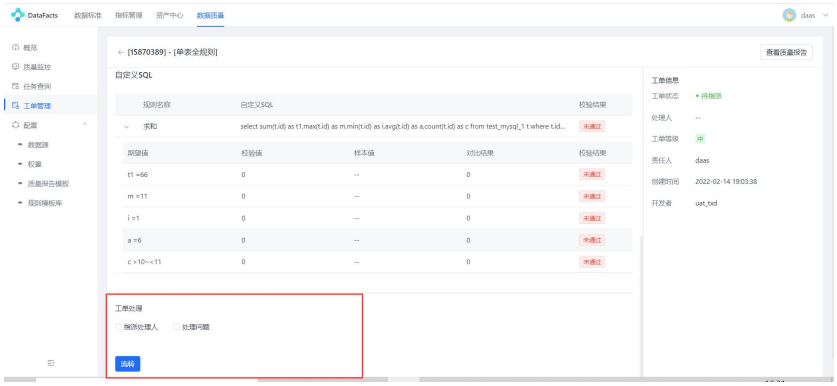


图 3-205 质量监控报告 3



对相应的数据质量问题以工单形式进行流转和处理。

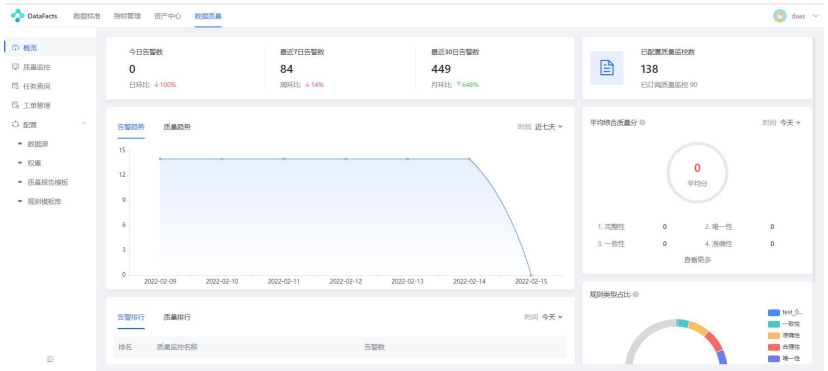
图 3-206 质量监控报告 4



12. 数据质量概览大屏

    数据质量概览大屏支持及时获悉不同统计周期内的错误告警数据量及环比、告警数量变换趋势及数据质量评分变化趋势、质量综合评分及不同质量指标的综合评分、按告警数量及指令分数对监控任务的排行等信息，从而整体地把握企业的数据质量现状及评估质量治理的重点。

图 3-207 数据质量概览大屏

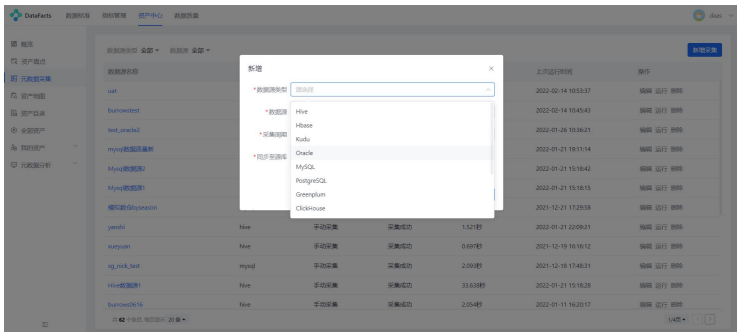


3.16.2.7 元数据管理

1. 元数据采集

- 支持 mysql、sqlserver、oracle、postgresql、db2 等主流关系型数据库；
- 支持 hive、greenplum、clickhouse 等分布式分析型数据库；
- 支持 hbase、kudu、mongodb 等 NoSQL 数据库；
- 支持 kafka、hdfs 等其他类型数据源；

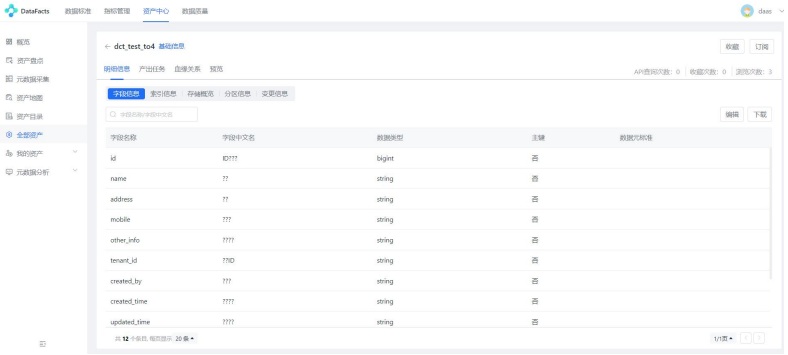
图 3-208 元数据采集



2. 元数据明细

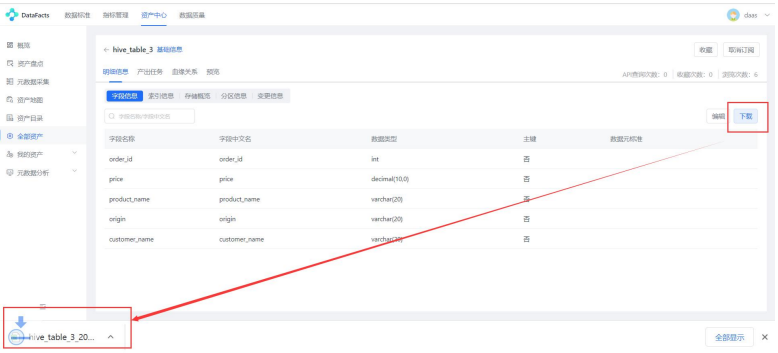
在具体的数据资产元数据明细里，可以查看字段信息、索引信息、存储情况概览、存储分区信息、以及变更信息。

图 3-209 元数据明细 1



支持以下载的方式导出数据表的元数据；

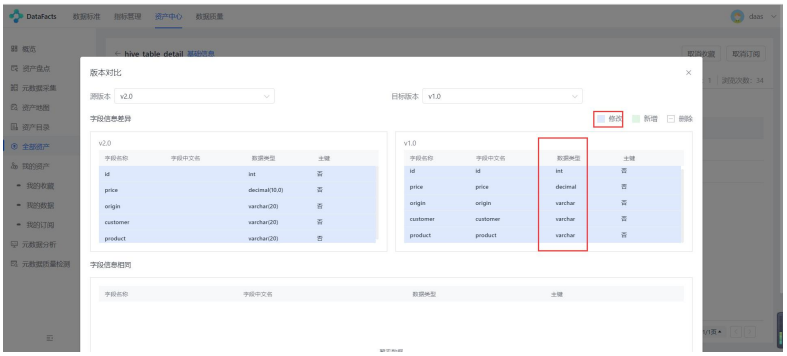
图 3-210 元数据明细 2



3. 元数据版本

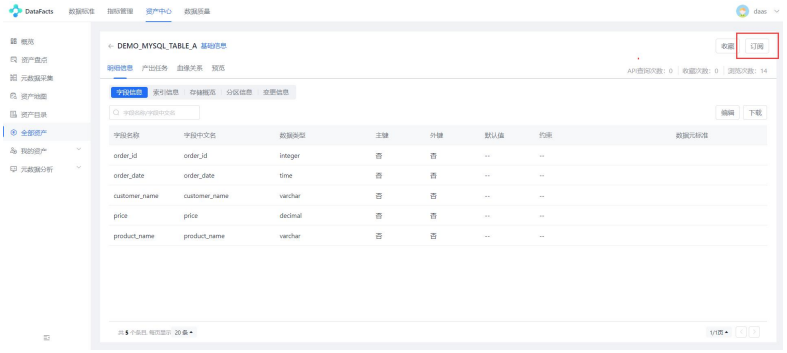
在变更信息中，可以查看元数据的版本列表，不同版本的元数据明细及差异比对。每次采集元数据后，如数据表的元数据与上次采集的结果有差异，则会形成新的元数据版本，并支持元数据差异比对。

图 3-211 元数据版本 1



如用户订阅了数据表，则该变更会通过用户邮件通知订阅者。订阅者可及时比对发现差异，并可进一步通过影响分析找到下游数据表及关联产出任务，及时进行相关调整。

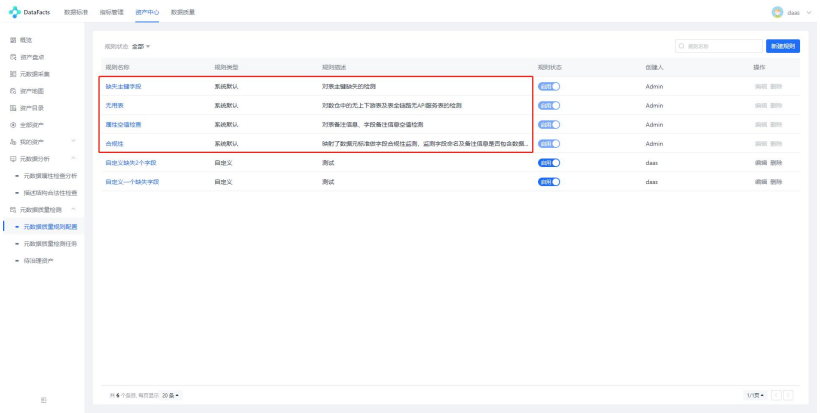
图 3-212 元数据版本 2



4. 元数据质量检测

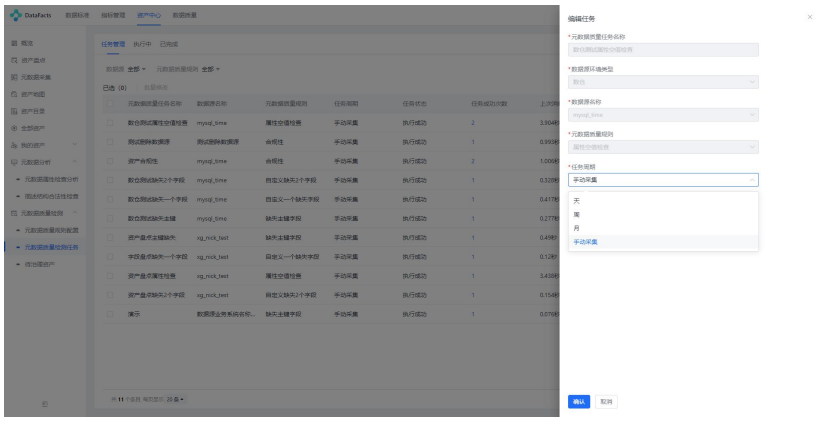
   内置主键缺失、无用表、属性空值及合规性检测等通用元数据质量规则，并支持用户自定义其他关键字段缺失规则，满足不同元数据质量检测场景的需要。其中，合理性规则用于检测各数据源中，字段的命名及备注等信息是否满足所关联的数据元标准约束，是感知标准落地情况的关键措施。

图 3-213 元数据质量检测 1



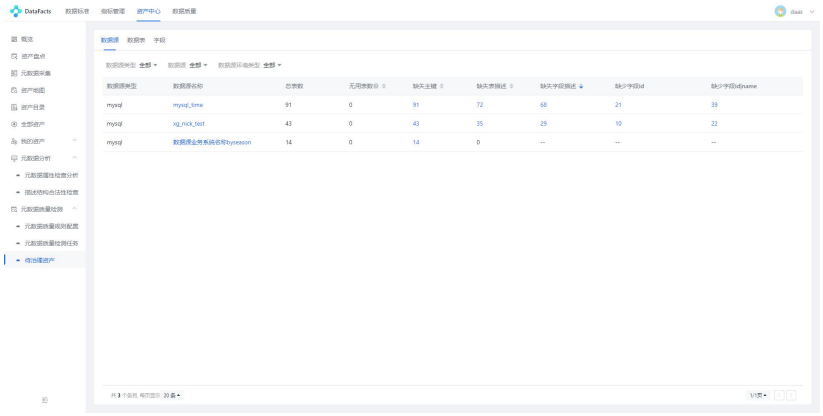
支持手动或周期性运行元数据质量检测任务。

图 3-214 元数据质量检测 2



任务运行后，所检测出有质量问题的元数据信息，会统一归档在待治理资产列表中。用户可以层层下钻，查看有质量问题的库、表及字段。

图 3-215 元数据质量检测 3

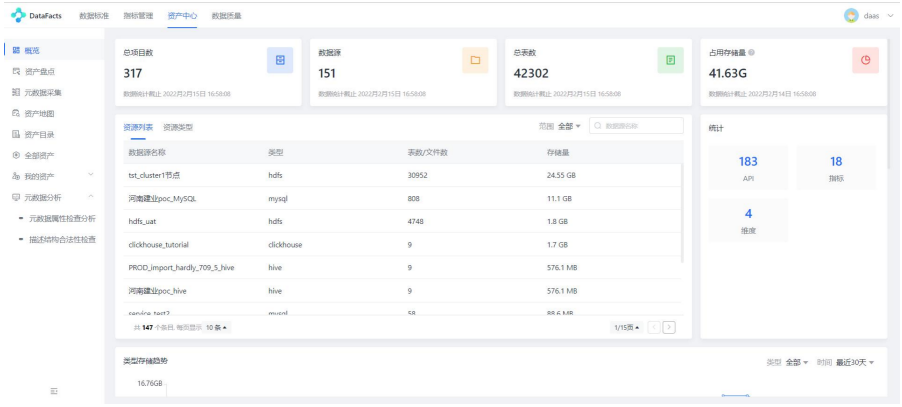


3.16.2.8 数据资源管理

1. 数据资源概览

   资源概览对数据平台内的数据项目、数据源、数据表、API、指标、维度等进行统计，并反映存储资源消耗情况、数据价值表排行、元数据质量缺陷统计等情况。

图 3-216 数据资源概览 1



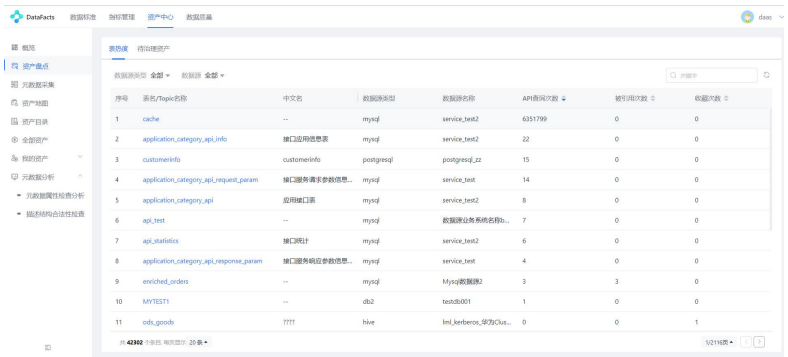
同时，可以查看不同数据源类型的数据源数量、表数量、数据量以及接入连通性等信息。

图 3-217 数据资源概览 2

2. 热点资源盘点

   根据 API 查询次数、被引用次数及收藏次数，对数据资产进行热点资源排行，及时发现闲置资源，以做好资源盘活工作。

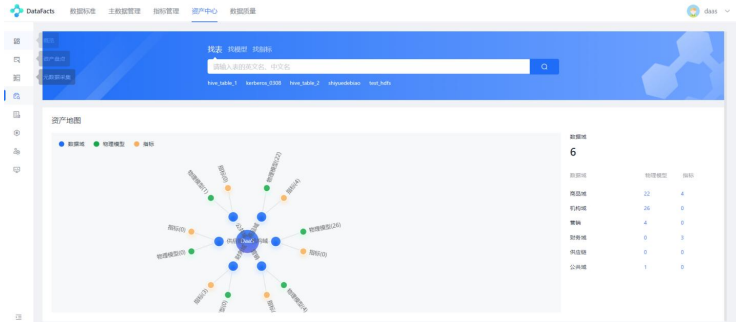
图 3-218 热点资源盘点



3. 数据地图

   通过数据地图，可以一目了然的总览数据模型及指标在不同业务主题域下的分布情况。

图 3-219 数据地图 1



用户可以依次展开想要探索的主题域、数据表，层层下钻，定位到目标表后，查看表详情。

图 3-220 数据地图 2

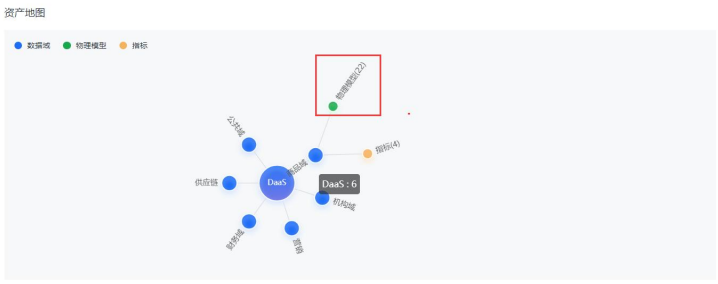


图 3-221 数据地图 3

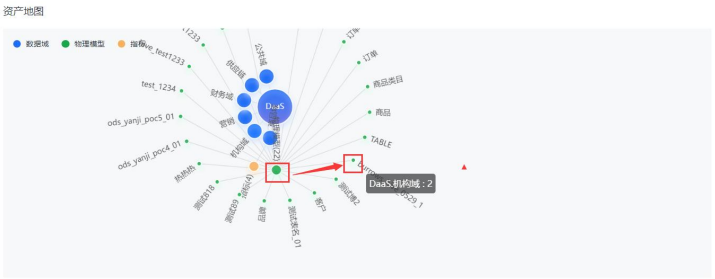
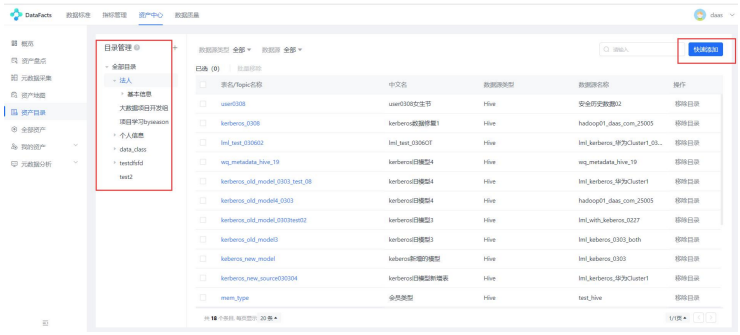


图 3-222 数据地图 4

4. 数据资源编目

   基于元数据采集获取的企业数据资源清单后，可以进行编目管理，使得数据资源更易被用户所检索、调用。

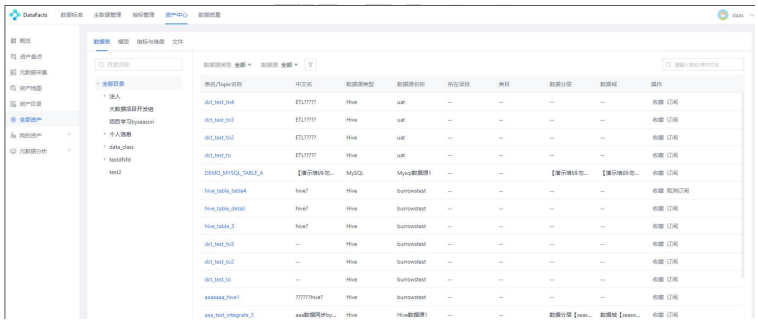
图 3-223 数据资源编目



5. 数据资源详情

   在全部资产中，可以查看所有数据表、数据模型、指标维度及 HDFS 上的文件。其中，数据表中可以进一步查看字段信息、索引信息、元数据变更信息、关联的数据产出任务、数据血缘关系等详情。

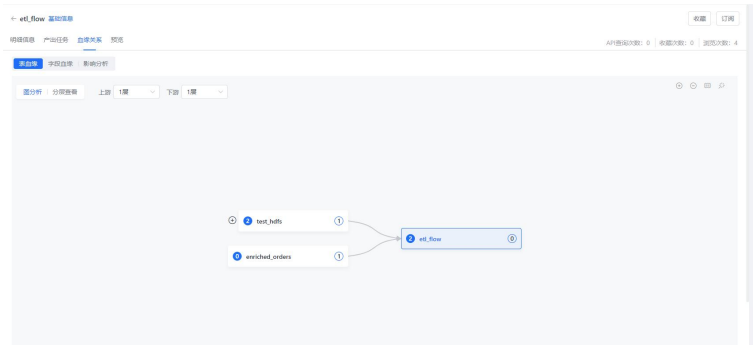
图 3-224 数据资源详情



6. 数据血缘分析

   通过数据血缘分析，可以分析问题表的数据质量问题来自于上游的哪些数据表和字段。

图 3-225 数据血缘分析

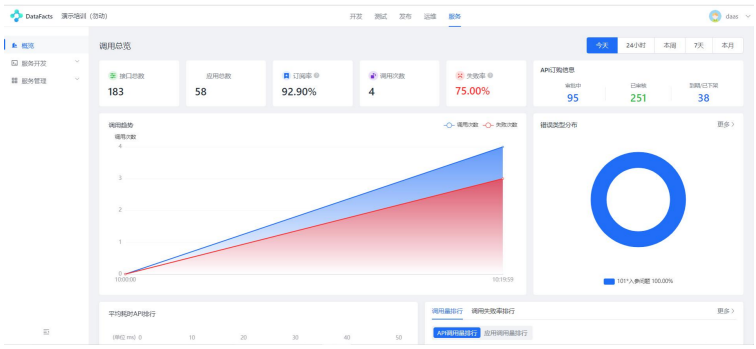


3.16.2.9 数据服务管理

1. 服务应用概览

   概览主要分为调用总览（接口总数、应用总数、订阅率、调用次数、失败率），API 订购信息，调用趋势,错误类型分布、平均耗时排行、速率监控、API 调用排行、应用调用排行、 API 调用失败率排行、应用调用失败率排行。以上维度，可以根据今天、24 小时、本周、近 7 天、本月进行筛选。

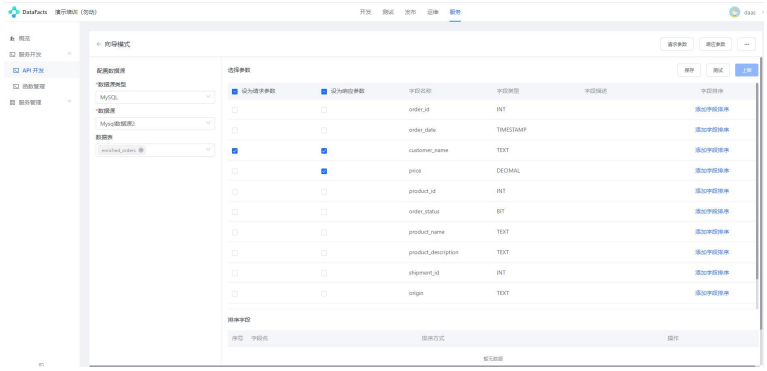
图 3-226 服务应用概览



2. API向导式生成

   支持通过可视化配置的向导模式，无需 SQL 即可快速对数据表配置 API 实现对表的数据获取。

图 3-227 API 向导式生成 1



配置好的 API 可自动生成 SQL 语句，用户可以方便的进行 API 测试后，发布上架。

图 3-228 API 向导式生成 2

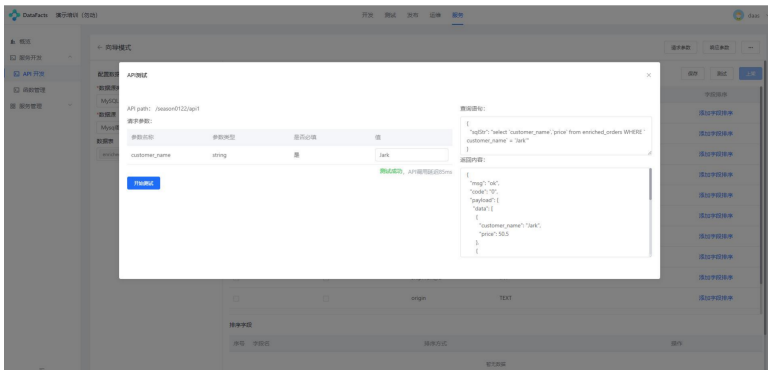
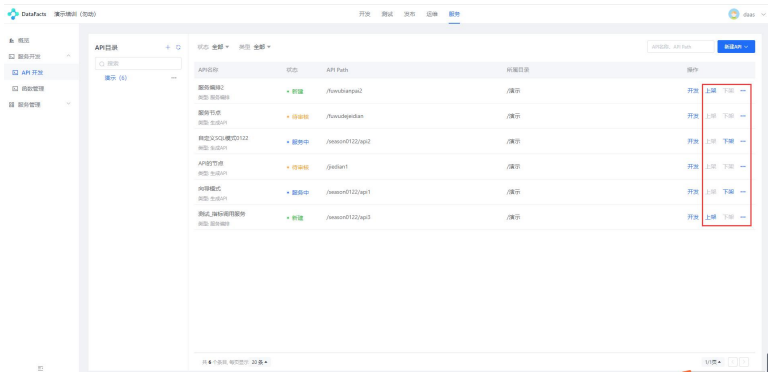


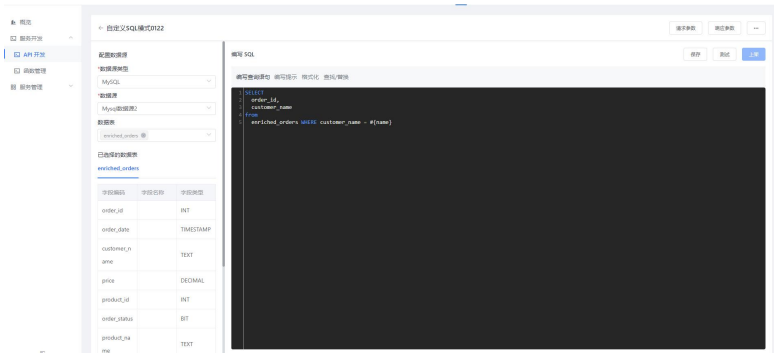
图 3-229 API 向导式生成 3



3. 自定义SQL开发

   针对稍复杂的查询分析场景，可通过自定义 SQL 的方式，生成对多个表进行关联查询的复杂 API。

图 3-230 自定义 SQL 开发



4. 服务编排

   支持用户通过服务编排的方式将多个 API 节点和 Python 函数进行串联处理的方式提供数据服务，以满足更复杂的服务开发需要。

图 3-231 服务编排 1

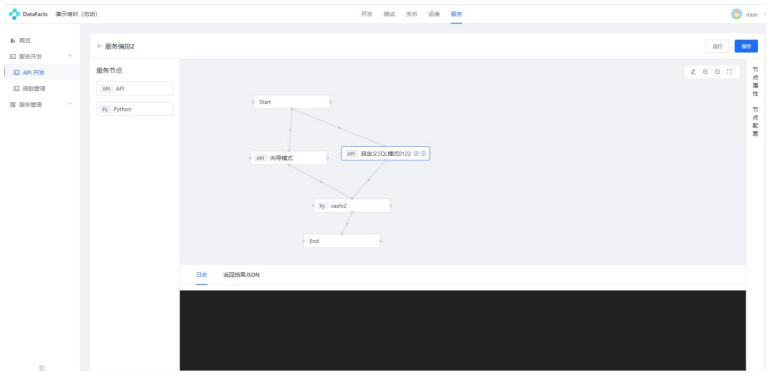
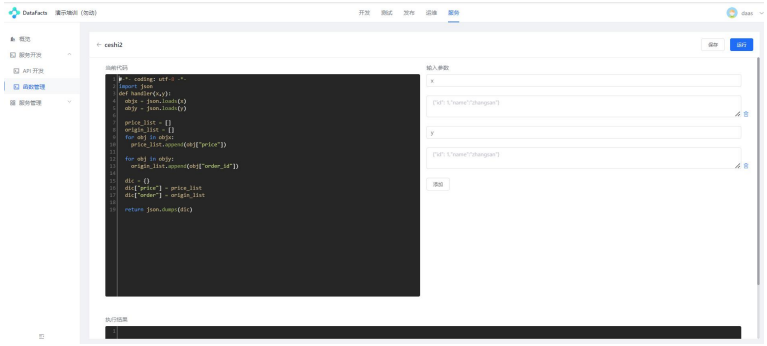


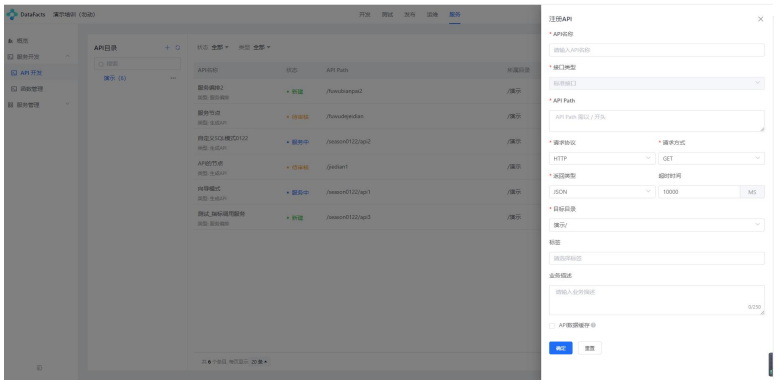
图 3-232 服务编排 2



5. API注册

   对于企业现有的 API，可以通过 API 注册的方式，与平台内开发的 API 一样，上架在统一的网关上。

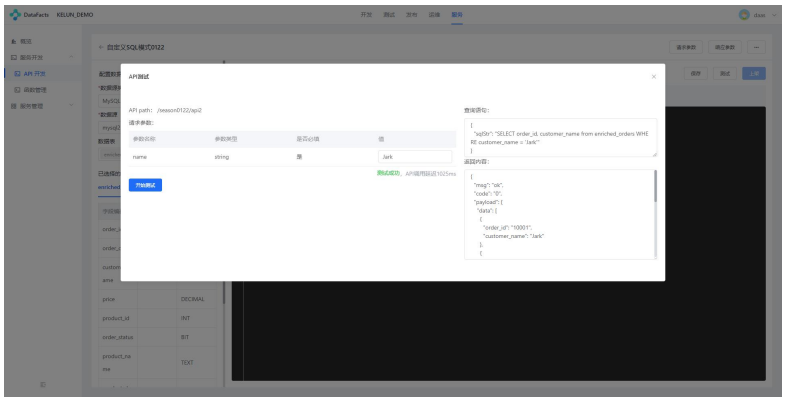
图 3-233 API 注册



6. API在线调试

   支持用户在线调试 API。

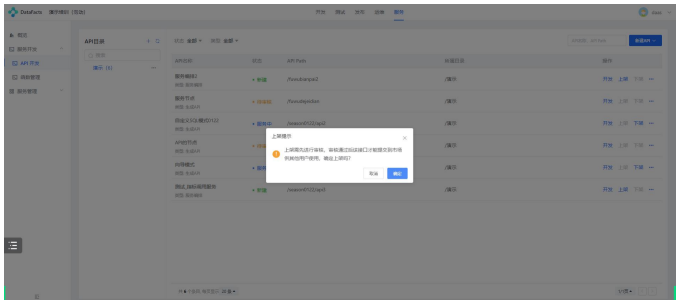
图 3-234 API 在线调试



7. API上架

   完成 API 的开发注册后，可以申请上架。API 上架后，外部应用可以通过 API 网关的统一鉴权获请求所订阅的数据 API。其中，DataFacts API 网关采用分布式架构设计，为平台服务层 提面向外部应用提供高并发的数据服务 API 调用能力。

图 3-235 API 上架



8. API安全

可配置黑白名单、调用次数限制等安全控制，实现企业数据服务的统一管理，保障企业数据的安全共享。

图 3-236 API 安全 1

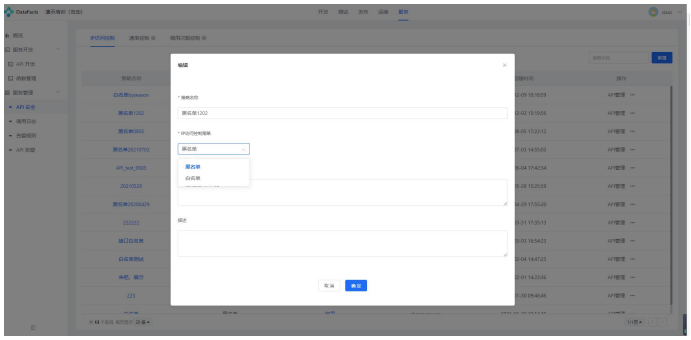


图 3-237 API 安全 2

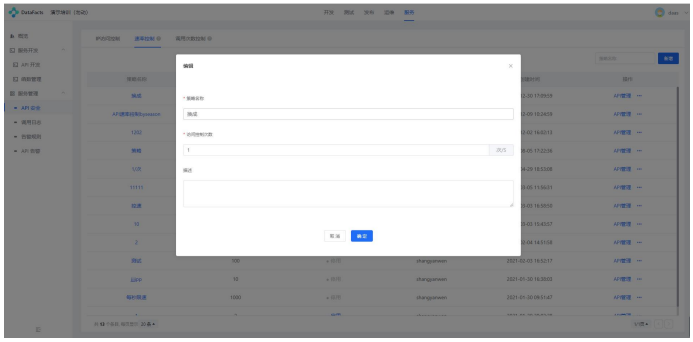


图 3-238 API 安全 3

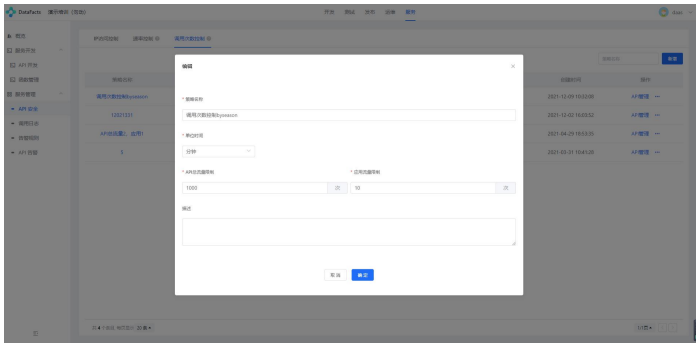
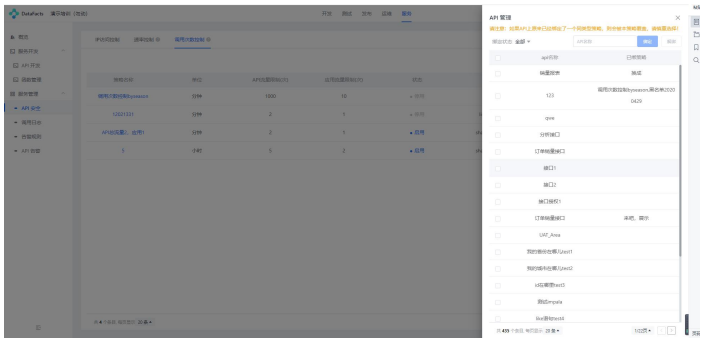


图 3-239 API 安全 4



9. API告警

   完成服务的注册或开发后，可以对各个 API 进行服务报错、耗时过长、超出限流等告警配置，从而实现对数据服务调用的统一、便捷的运维管理；

图 3-240 API 告警 1

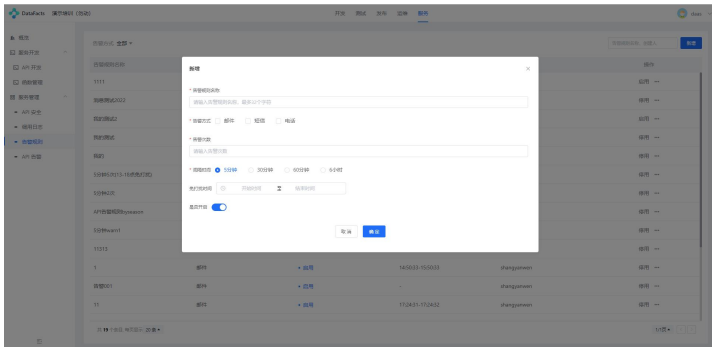
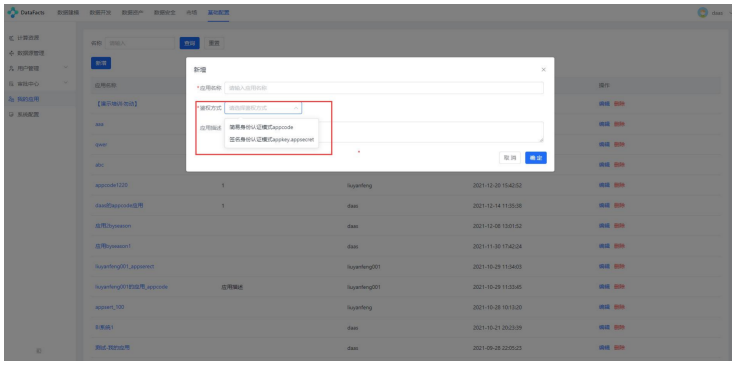


图 3-241 API 告警 2

10. 应用鉴权

    获取服务需要通过应用来订阅。应用支持 appcode 简单身份认证模式和 AK/AS 签名身份认证模式两种鉴权方式，来访问 API。

图 3-242 应用鉴权



11. API市场

    API市场面向应用开发者提供了正式发布在 API 网关上的统一服务目录。通过对 API 进行订阅申请，并配置所需服务的应用及期限，在申请通过后，即可获得跨部门的数据服务。通过数据服务开发及 API 市场，实现数据资源在组织内跨部门的流通，从而将数据价值最大化释放。

图 3-243 API 市场 1

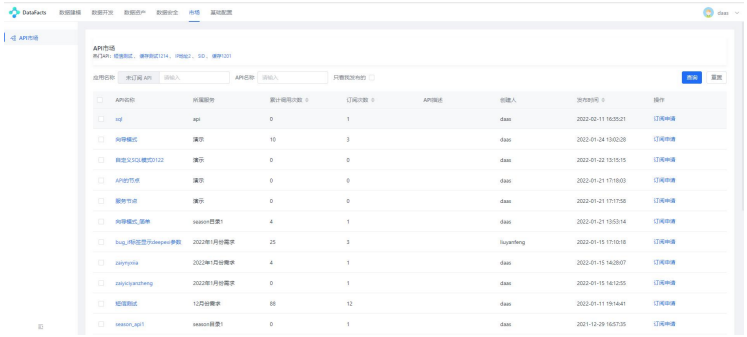
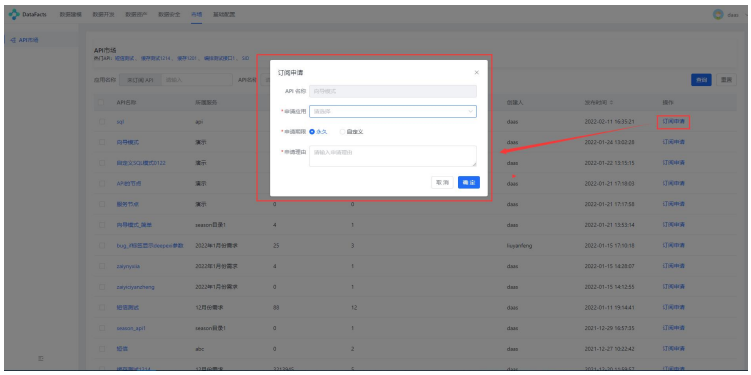


图 3-244 API 市场 2



3.16.2.10 数据安全

1. 数据安全等级

   支持数据安全等级自定义，设置密级名称。诸如设置机密、公开、绝秘等密级。设置完密级后，可以用于后续关联敏感数据扫描等安全策略。

图 3-245 数据安全等级 1

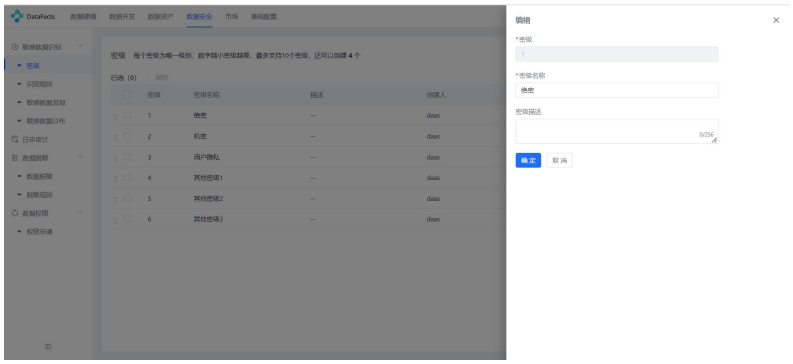
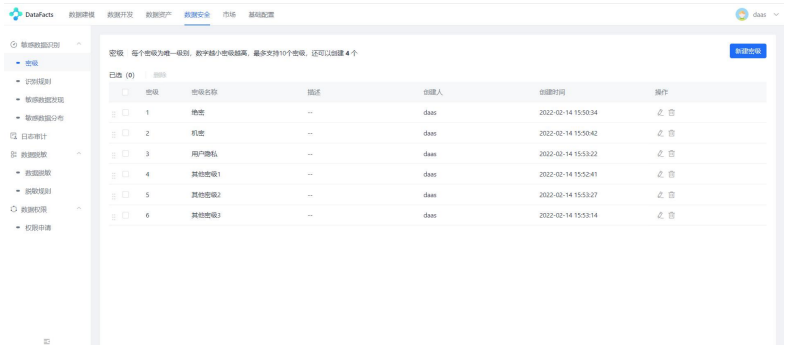


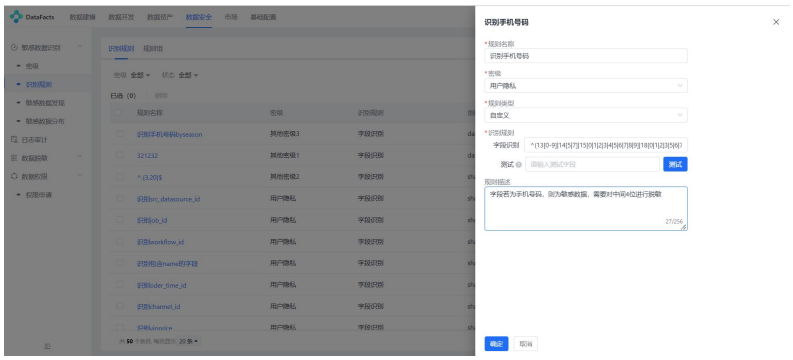
图 3-246 数据安全等级 2



2. 敏感数据发现

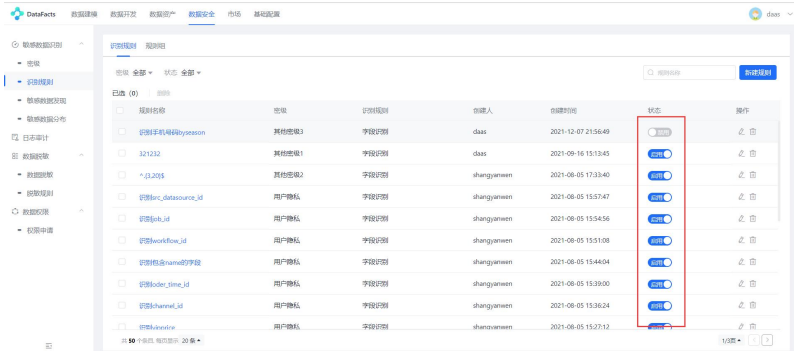
   支持基于不同数据安全等级的敏感数据扫描任务，以周期自动化或手动方式发现敏感数据并定位其来源。用户可以通过统一的视图总览敏感数据的分布状况。在新建规则中，可以根据企业安全策略，选择相应的密级，并通过正则表达式设置识别规则。

图 3-247 敏感数据发现



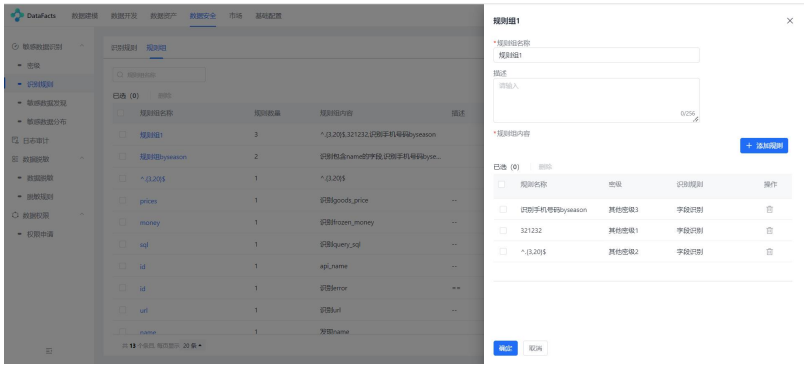
对创建好的规则，可以对其进行启用、停用等操作。

图 3-248 敏感数据发现 2



敏感数据的扫描任务可以同时按照多个规则进行扫描识别，因此，需要进一步新建规则组，来将多个规则关联进来。

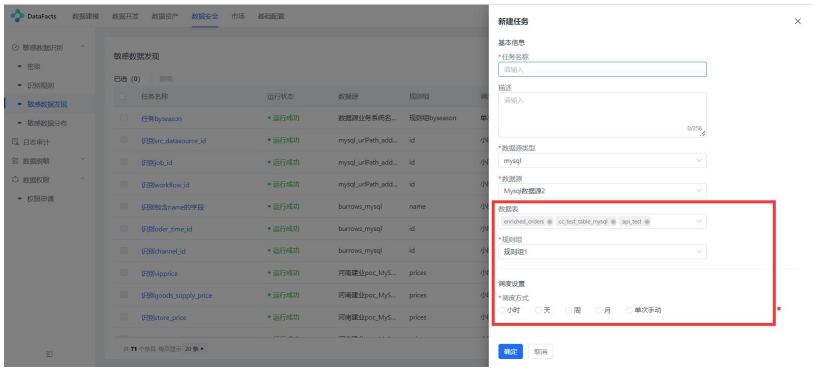
图 3-249 敏感数据发现 3



3. 敏感发现任务

   扫描任务支持支持对多个数据表同时按照多个规则构成的规则组进行敏感数据识别，其调度方式支持小时、天、周、月等周期性触发及手动触发。每次扫描后，都会更新敏感数据分布情况，并支持明细查看。

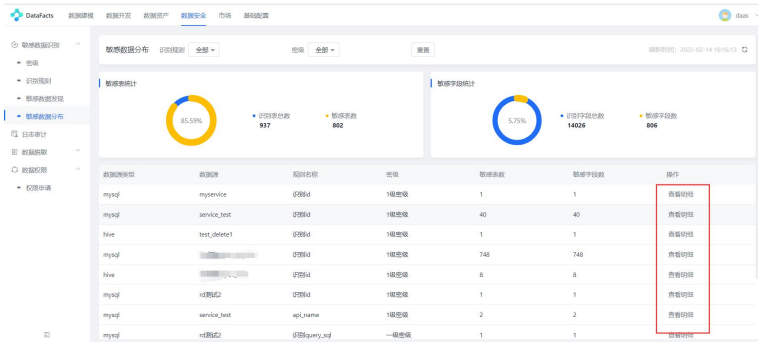
图 3-250 敏感发现任务



4. 敏感数据分布

   在敏感数据分布界面中，可以查看涉敏表、涉敏字段的统计结果，及各个数据源的敏感表数、字段数、涉敏明细情况等；

图 3-251 敏感数据分布 1



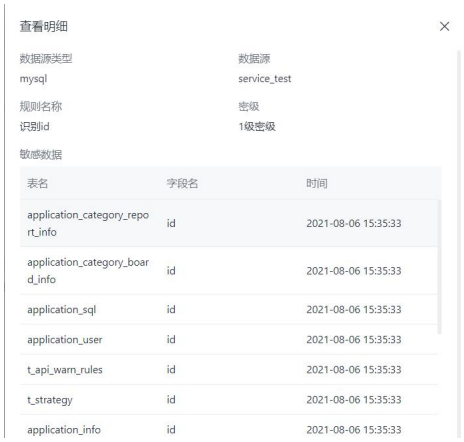
用户可以筛选查看不同密级、不同识别规则下的敏感数据分布情况。

图 3-252 敏感数据分布 2

图 3-253 敏感数据分布 3

数据源敏感情况明细表中清晰展示了哪些表的哪些字段涉敏，以供后续实施脱敏策略时作为目标。

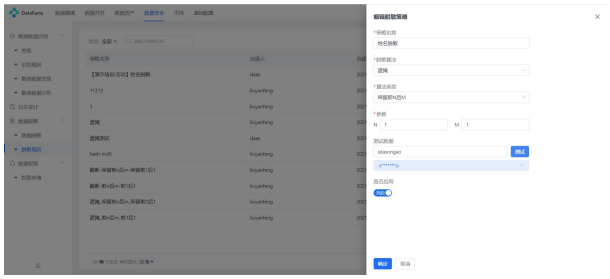
图 3-254 敏感数据分布 4



5. 脱敏策略

   内置遮蔽、截断、Hash 等多种脱敏算法，供用户配置脱敏策略并进行测试。

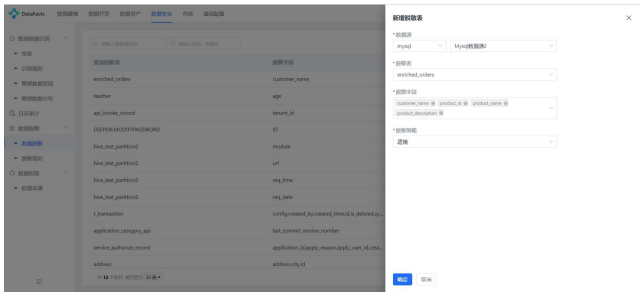
图 3-255 脱敏策略



6. 脱敏表

   完成脱敏策略的配置后，可根据敏感数据扫描发现结果，构建脱敏表，选择目标表及其需脱敏的字段，关联相应的脱敏策略，实现动态脱敏。保障用户顺畅访问数据的情况下，避免敏感数据发生泄露。

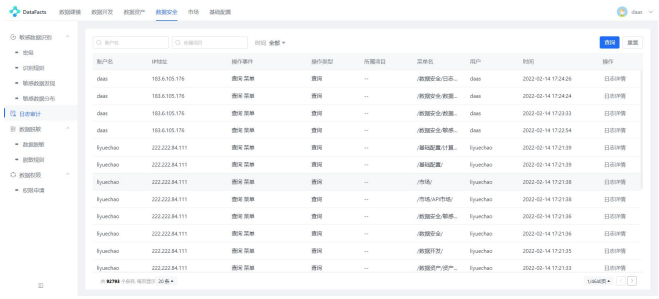
图 3-256 脱敏表



7. 操作日志审计

   日志审计提供详细的访问记录，包括事件类型、访问时间、日志详情等信息，帮助企业及时发现不合规的操作行为，保障数据安全。

图 3-257 操作日志审计



8. 数据权限控制

   支持为数据开发、服务开发、数据共享，基于不同密级提供库级、表级、字段级等颗粒度的权限管控，保障数据安全。

图 3-258 数据权限控制 1

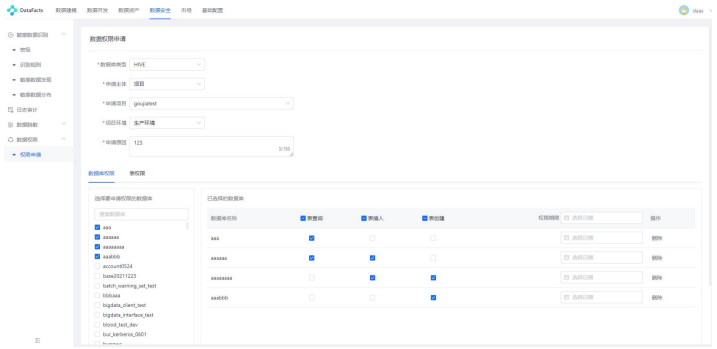
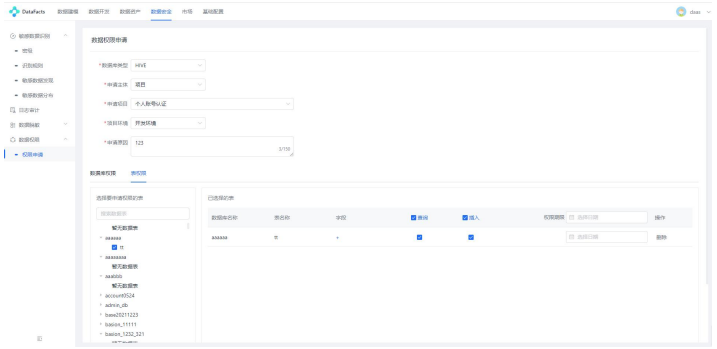


图 3-259 数据权限控制 2



4 修订记录

表 4-1 修订记录

发布日期	修订记录
2024-04-23	规范词、敏感词专项处理，章节优化
2023-02-15	第一次正式发布。