Version: 3.5.4

赛意工业物联网平台

「SIIOT」产品部署文档

目录

1.产品概述

0.1. 产品简介

赛意 SIIOT 工业物联网平台着力于强化工业互联网平台纵深，融合 SMDC、SMOM、EMS、SRM、ERP、SWMS、SQMS、STMS 等工业 APP，致力于帮助工业企业打造从设备接入，物联呈现到行业应用的端到端深度价值解决方案的运营平台，为工业企业落地工业互联网或数字化转型提供端到端成本最低、效率最优、可靠性最强的解决方案。

赛意 SIIOT 工业物联网平台是一个集成物联网接入与建模、数据安全通信、消息订阅、工业数据分析与管理、身份识别与访问管理、资产管理台账、轻量级工业 APP 等能力的一体化平台。向下支持连接海量设备，采集设备数据上云；向上提供丰富的云端 API 赋能工业 APP，助力快速搭建工业业务场景的解决方案。

0.2. 联系方式

您可以通过下列方式联系赛意信息公司，我们在全国各主要城市设置有分支机构，以便进行快捷的本地服务，分支机构的联系方式见公司网站。

公司网站：http://www.chinasie.com/

技术热线：0757-26339186

全国热线：400-968-5633

1. 准备工作

1.1. IIDP 基础平台

阅读该文档前，请确保已经安装 IIDP 基础平台。假若还没有安装好 IIDP 基础平台，请参考 IIDP 部署文档（请联系 IIDP 平台负责人获取）。

IIDP 默认账号及密码信息：

- 平台管理员账号：implementer/Implementer_2023
- 租户管理员账号：admin_000001/Admin_000001

1.2. IIOT 业务平台

1.2.1. 应用服务

IIOT 应用服务以 App 形式进行组织（PS：后端安装包 xxx.jar，前端安装包 xxx.zip，部分 App 没有前端安装包），安装包请在 SMDC 官网获取，访问路径：【IIOT】->【迭代更新详细内容】->【App 归档下载】。各 App 信息参见表 1-1。

表 1-1 App 信息表

一级菜单	二级菜单	APP	说明	属性	备注
基础公共能力（必装）		iiot_access	基础套餐包	基础	
		iiot_system_store		基础	
		iiot_alarm		基础	
		iiot_base		基础	
		redis		基础	
		iiot_task_scheduler		基础	
		net		基础	
		iiot_store/iiot_tdstore（分别是 InfluxDB 和 TDengine 时序数据库二选一安装即可）		基础	

		iiot_access_opcua、iiot_access_coap、iiot_access_http	基础
		iiot_apiAdapter	基础
项目管理	–	iiot_project	基础
接入与建模	物资源总览	iiot_thing、iiot_edge_config	基础
	物模型		基础
	物实体		基础
	工况查询		基础
	原始报文		基础
行业模板库	工厂实体	iiot_factory	标准
	聚合规则	iiot_aggregation	标准
	–	iiot_industry_template_common	标准
云边协同	工程协同	iiot_edge_project	标准
	固件升级	iiot_edge_firmware_app、iiot_edge_sync	标准
	驱动管理	iiot_edge_drive_app	标准
数据调度	用户编程	iiot_program	标准
	规则引擎	iiot_ruleengine	标准
数据转发	协议配置	iiot_dataservice、iiot_dataservice_opcua（分布式部署，需单独部署 opcua 容器）、iiot_dataservice_api、	标准
	转发配置		标准
	同步配置		标准

		iiot_messagetransfer、rabbitmq			
边缘节点	边缘管理	iiot_edge_node		标准	
数据存储管理	存储监控	iiot_store_monitor		基础	
	数据库管理	iiot_db_management		基础	
	数据查询	iiot_data_query		基础	
	存储管理	iiot_store_management		基础	
系统运维	系统配置	iiot_base		基础	
	关键日志	iiot_keylog		基础	
	集群监控	iiot_cluster_manager、iiot_cluster_data_transfer、iiot_cluster_rpc		基础	分布式部署必备
	设备节点管理	iiot_device_scheduler		基础	分布式部署必备
	工况回补	iiot_thing		基础	
数据编织	数据建模	iiot_data_fabric		高阶	
	数据管理			高阶	
	HMI (IOT)	iiot_hmi、iiot_hmi_2D、iiot_hmi_3D、iiot_hmi_layout		高阶	
数据仿真	MI 设计器	iiot_datasource	选配套餐包	高阶	IIOT 数据源扩展 APP 需要与 MI 安装在同一容器（需安装 MI-Base/IDA）

消息推送	消息渠道配置	iiot_messagepush、iiot_messagepolicy、iiot_messagepolicy_alarm、iiot_messagepolicy_ruleengine（暂不推荐）、iiot_messagepolicy_store_management		高阶	
	消息模板配置			高阶	
	消息接收组			高阶	
	发送策略			高阶	
	消息统计			高阶	
	请求记录			高阶	
数据存储	数据归档	iiot_data_archive		高阶	需要安装 powerjob-server 中间件
	数据备份	iiot_data_backup		高阶	需要安装 powerjob-server 中间件
主数据扩展	IOT 扩展 MBM 主数据	iiot-thing-mbm-mdm		高阶	

1.2.2. 依赖组件

IIOT 应用服务的依赖组件参见表 1-2，请在部署前点击链接提前下载相关依赖服务的 docker 镜像（注意：请连接公司内网下载相关镜像及附件）。

表 1-2 依赖组件表

服务名称	文件	镜像版本	是否必装	备注
docker 安装包	docker.zip	/	是	单机版的安装基础；分布式版、高可用版需用来安装 tdengine、influxdb
docker 安装脚本	docker-install.sh	/	是	

高可用代理服务	haproxy.tar	V2.3.1	是	
TDengine 数据库	tdengine.tar	V3.4.0.2	二选一	请咨询项目经理，结合服务器资源清单选择
InfluxDB 数据库	influxdb.tar	V1.8.0		
TDengineProxy 服务	tdengine-proxy.tar	V1.3.32	否	数据库安装 TDengine，需配套安装
InfluxProxy 服务	influx-proxy.tar	V2.5.10	否	数据库安装 InfluxDB，需配套安装
PowerjobServer 服务	powerjob-server.tar	V4.3.9	否	安装数据归档、数据备份 APP，则配置套安装

1.2.3. 部署模式

IIOT 应用服务的部署模式参见表 1-3，请在部署前根据服务器资源清单选择相应的部署模式。

表 1-3 部署模式表

部署模式	容器化技术	备注
单机版	docker	基于 DOCKER 部署 1 个 IIOT 应用服务
分布式版	k8s	基于 K8S 部署 1 个 IIOT 应用服务
高可用版	k8s	基于 K8S 部署 2 个 IIOT 应用服务（iiot1、iiot2）

1.2.4. 端口清单

请根据部署模式开通以下端口，注意服务数量和副本数量（通常由客户 IT 进行处理）。

注意：若服务器启用了防火墙，请开启相关端口的防火墙策略，否则，请关闭防火墙。

1.2.4.1. 单机版

表 1-4 单机版部署端口清单

应用服务	端口	备注
系统入口	30666	浏览器访问平台端口

实时数据订阅	8888	iiot 服务的 websocket 端口
时序数据库存储服务	31003	供 influxdb 或 TD 时序数据库使用
Redis	30097	
时钟同步	123	
mqtt 数采接入	1883	数采上报到 iiot 服务的 MQTT 端口
coap 数采接入	5683	
PowerjobServer	37700	若安装数据归档、数据备份

1.2.4.2. 分布式版

表 1-5 分布式版部署端口清单

应用服务	端口	备注
系统入口	30666	浏览器访问平台端口
K8S 运维平台	30880	
实时数据订阅	30788	iiot1 服务的 websocket 端口
实时数据订阅	30888	iiot2 服务的 websocket 端口
时序数据库存储服务	31003	供 influxdb 或 TD 时序数据库使用
Redis	30097	
时钟同步	123	
mqtt 数采接入	31881	数采上报到 iiot1 服务的 MQTT 端口
mqtt 数采接入	31882	数采上报到 iiot2 服务的 MQTT 端口
http 数采接入	30134	http 接入
http 数采接入	30135	http 接入

coap 数采接入	30683	coap 接入
coap 数采接入	30684	coap 接入
PowerjobServer	37700	若安装数据归档、数据备份

1.2.4.3. 高可用版

表 1-6 高可用版部署端口清单

应用服务	端口	备注
系统入口	30666	浏览器访问平台端口
K8S 运维平台	30880	
实时数据订阅	30788	iiot1 服务的 websocket 端口
实时数据订阅	30888	iiot2 服务的 websocket 端口
时序数据库存储服务	31003	供 influxdb 或 TD 时序数据库使用
Redis	30097	
时钟同步	123	
mqtt 数采接入	31881	数采上报到 iiot1 服务的 MQTT 端口
mqtt 数采接入	31882	数采上报到 iiot2 服务的 MQTT 端口
http 数采接入	30134	http 接入
http 数采接入	30135	http 接入
coap 数采接入	30683	coap 接入
coap 数采接入	30684	coap 接入
PowerjobServer	37700	若安装数据归档、数据备份

1.3. 相关中间件安装

1.3.1. 安装 Docker

1.3.1.1. 安装 docker、docker-compose 组件

单机版的安装基础；分布式版、高可用版需用来安装 tdengine、influxdb

注意：若服务器已安装 docker 及 docker-compose 基础软件，请忽略该步骤，否则请参考该步骤执行 docker 及 docker-compose 基础软件安装（假设数据盘挂载点是：/var，请根据实际情况修改）。

- 参考表 1-2 依赖组件表下载依赖组件 docker.zip、docker-install.sh ，再从安装人员本机上传依赖组件到服务器的 /var 目录。
- 登录服务器，解压 docker.zip 安装包。

```
cd /var && unzip docker.zip
```

- 赋予 docker-install.sh 可执行权限。

```
cd /var && chmod +x docker-install.sh
```

- 执行 docker-install.sh 脚本安装 docker 及 docker-compose 组件。

```
cd /var && ./docker-install.sh
```

1.3.2. 安装 PowerJobServer

数据归档、数据备份 APP 依赖任务调度组件 PowerjobServer，其部署方式有单机版（基于 docker 安装）、分布式版（基于 K8S 安装），部署人员请根据服务器资源情况选择部署方式。

1.3.2.1. 单机版安装

使用该部署方式前，请确保已正确安装 docker、docker-compose 组件（参考 2.3.1 章节安装 docker、docker-compose），（假设数据盘挂载点是：/var，请根据实际情况修改）。

- 参考表 2-2 依赖组件表下载 powerjob-server 镜像到目标服务器的/var 目录。
- 登录目标服务器，装载 powerjob-server 镜像到 docker 镜像库。

```
cd /var && docker load < powerjob-server-4.3.9.tar
```

- 登录目标服务器，创建 tdengine 安装及配置目录。

```
mkdir -p /var/powerjob-server
mkdir -p /var/powerjob-server/powerjob/server
```

- 下载 powerjob-server 部署脚本 powerjob-server-compose.yml，上传至目标服务器的 /var/powerjob-server 目录，根据以下截图修改配置参数。

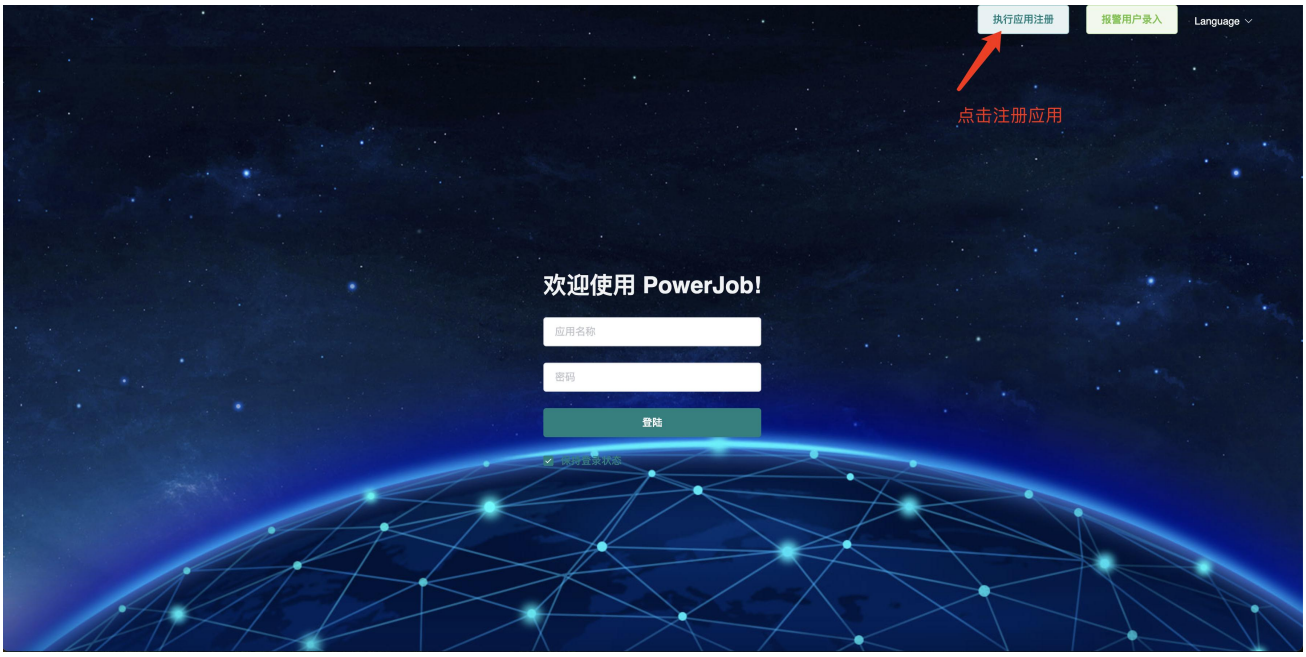
- 登录目标服务器，启动 powerjob-server 服务。

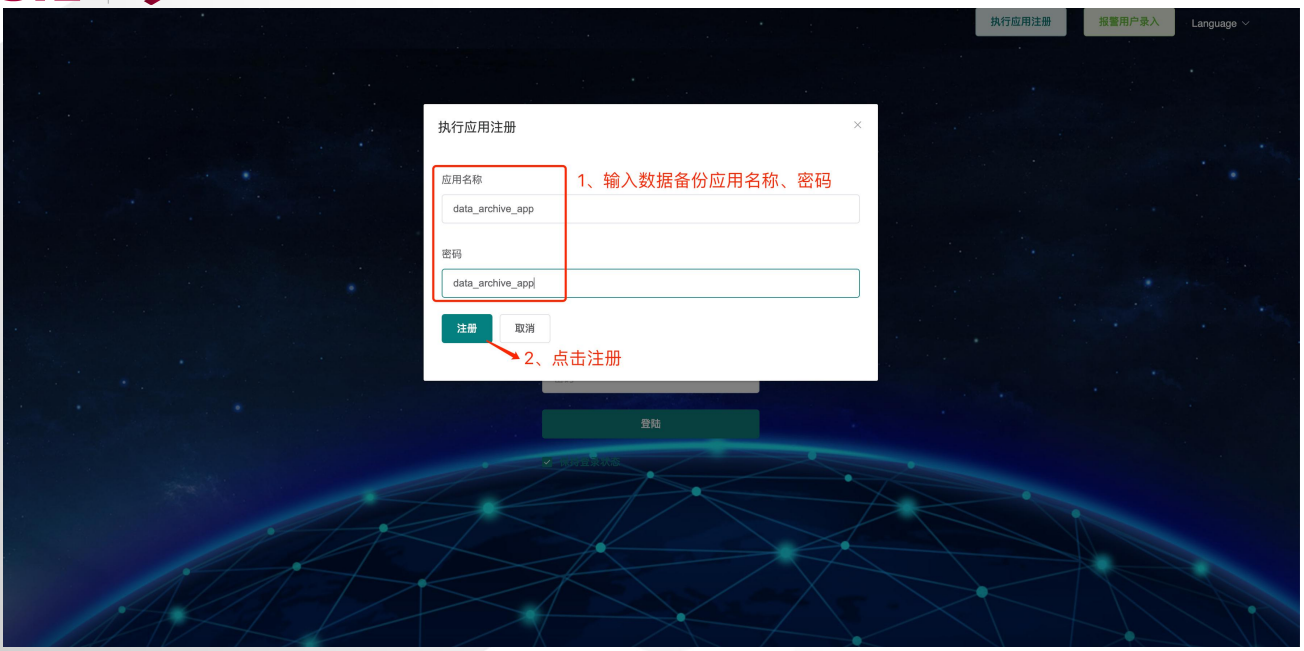
```
docker-compose -f /var/powerjob-server/powerjob-server-compose.yml up -d
```

- 登录目标服务器，验证 powerjob-server 服务是否正常。

```
docker ps | grep powerjob-server
```

- 若数据库是高斯数据库（GaussDB），需要在 powerjob_server 数据库（根据实际情况修改）执行兼容脚本 PowerjobServer 高斯数据库兼容 DDL 脚本。若是非高斯数据库，则忽略该步骤。
- 若版本号低于 SIE IIOT v4.3.0107，或数据归档、数据备份 App 版本低于 4.3.x.20251201，登录 powerjob-server 服务（http://部署服务器 IP:37700/#/welcome），注册 data_archive_app、data_backup_app。





- 若版本号低于 SIE IIOT v4.3.0107，或数据归档、数据备份 App 版本低于 4.3.x.20251201，登录 IIOT 部署服务器，编辑配置文件：/snest/config/application-dev.properties（默认该路径，项目具体路径根据部署情况修改）配置文件新增以下配置信息。（增加完配置信息保存后重启 snest 容器服务）。


```
# powerjob
powerjob.worker.data-archive-app-name=data_archive_app
powerjob.worker.data-archive-app-password=data_archive_app
powerjob.worker.data-archive-app-port=28777
powerjob.worker.data-backup-app-name=data_backup_app
powerjob.worker.data-backup-app-password=data_backup_app
powerjob.worker.data-backup-app-port=28778
powerjob.worker.protocol=http
powerjob.worker.server-address=powerjob-server:7700      # 根据实际修改
powerjob.worker.store-strategy=DISK
powerjob.worker.max-result-length=4096
powerjob.worker.max-appended-wf-context-length=4096
powerjob.worker.max-lightweight-task-num=1024
powerjob.worker.max-heavyweight-task-num=64
powerjob.worker.health-report-interval=10
powerjob.worker.enabled=true
```

1.3.2.2. 分布式版安装

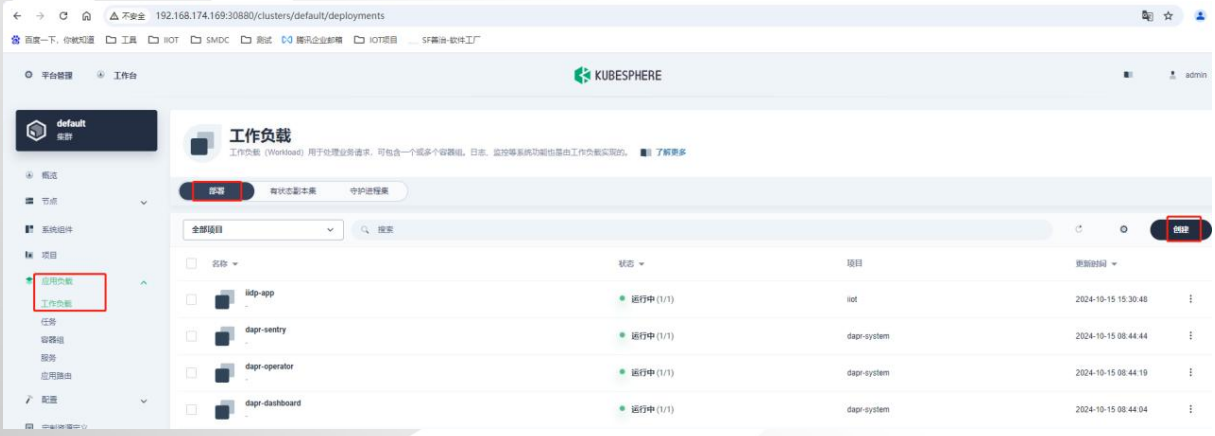
使用该部署方式前，请确保已正确安装 K8S 服务。

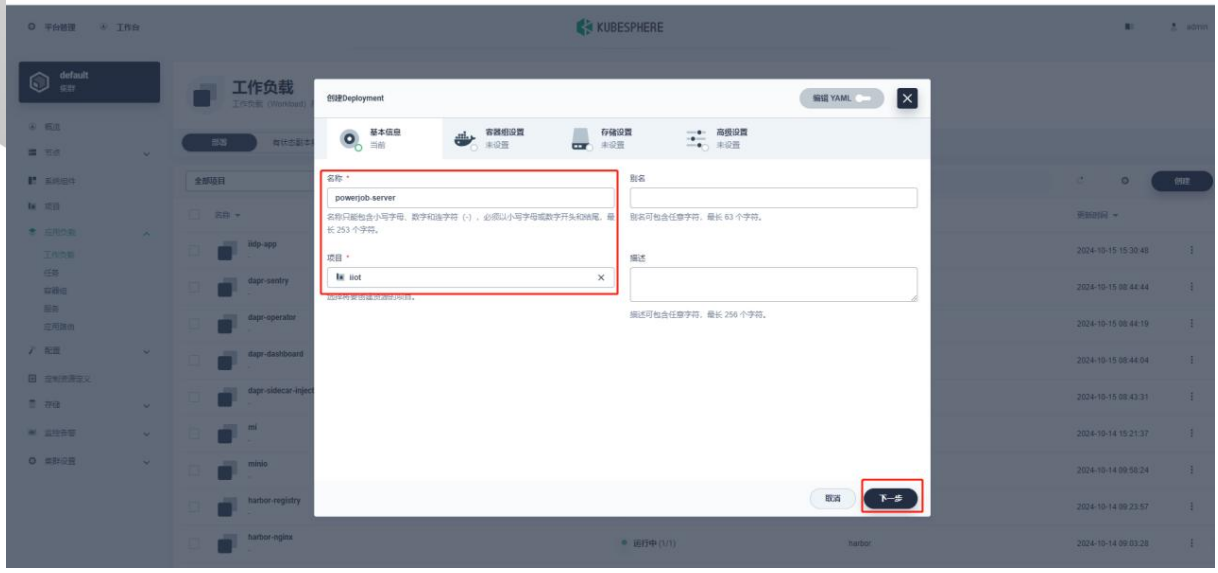
- 参考表 2-2 依赖组件表下载 powerjob-server 镜像到 K8S 集群任意服务器的/var 目录（假设数据盘挂载点是：/var，请根据实际情况修改）。
- 登录目标服务器，装载 powerjob-server 镜像到 docker 镜像库，推送到 harbor 镜像仓库。

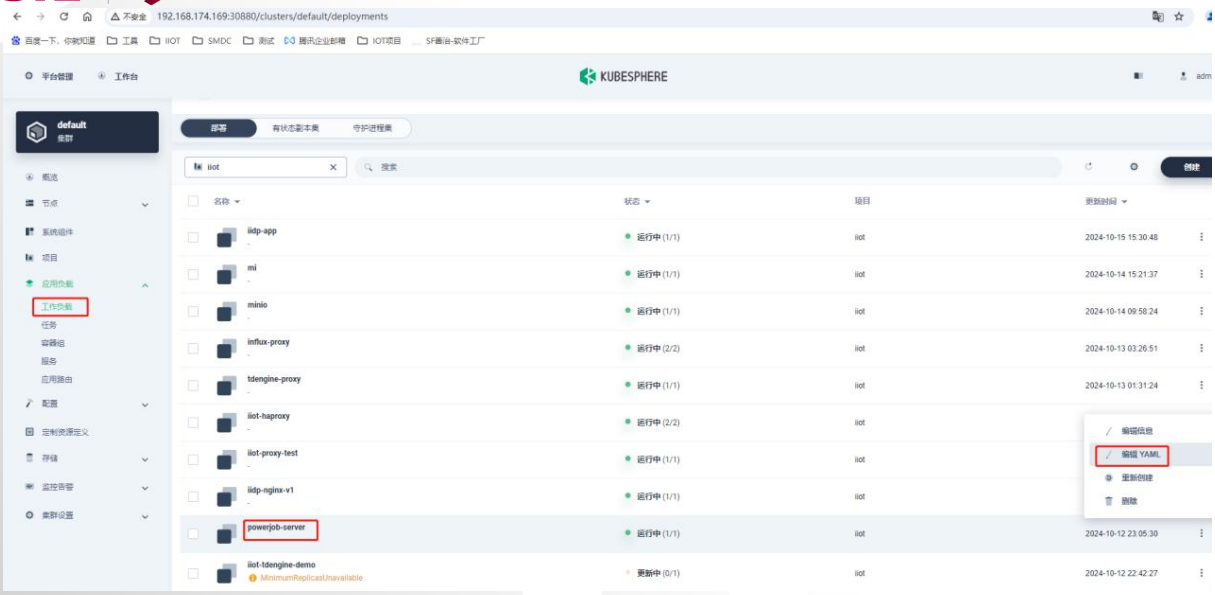
```
cd /var && docker load < powerjob-server-4.3.9.tar
```

```
# 注意：若 harbor 镜像仓库地址（harbor.devcenter.gushen.sieiot.com）名称与实际不一致，请先把镜像重新打 tag，再推送到 harbor 仓库。
docker push harbor.devcenter.gushen.sieiot.com/iidp-library/powerjob-server:4.3.9
```

- 登录 K8S 管理页面，在【应用负载】->【工作负载】菜单页面下选择【部署】，点击创建，输入服务名称：powerjob-server，选择所属项目。







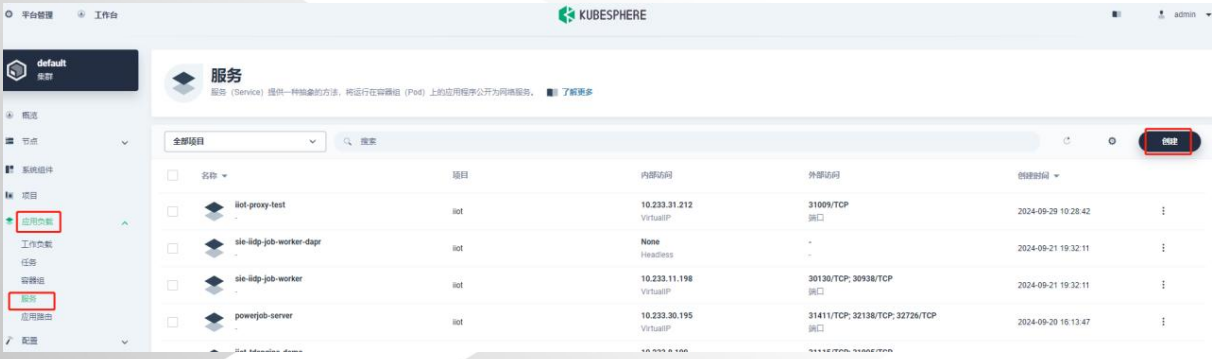
【部署】创建成功，编辑 YAML 配置文件内容（注意点：修改名称空间以及 mysql 相数据库的 IP、端口、账号、密码以及创建 powerjob_server 数据库）。

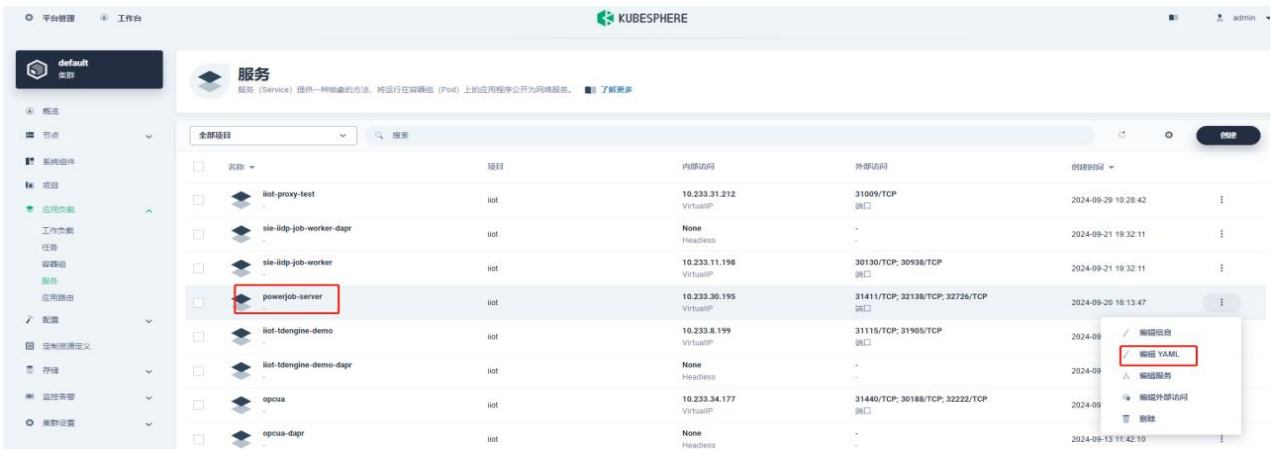
```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: powerjob-server
  namespace: iiot                    # 根据实际修改名称空间
  labels:
    app: powerjob-servery
  annotations:
    deployment.kubernetes.io/revision: '9'
    kubesphere.io/creator: admin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: powerjob-server
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: powerjob-server
      annotations:
        kubesphere.io/creator: admin
        kubesphere.io/restartedAt: '2025-03-07T08:49:40.604Z'
        logging.kubesphere.io/logsidecar-config: '{}'
    spec:
      volumes:
        - name: volume-01
```

```
            persistentVolumeClaim:
              claimName: powerjob-server
        containers:
          - name: container-yxsa3k
            image: >-
              harbor.devcenter.gushen.sieiot.com/iidp-library/powerjob-server:4.3.9
            ports:
              - name: tcp-7700
                containerPort: 7700
                protocol: TCP
              - name: tcp-10086
                containerPort: 10086
                protocol: TCP
              - name: tcp-10010
                containerPort: 10010
                protocol: TCP
            env:
              - name: JVMOPTIONS
                value: >-
                  -Xmx512m -Dpowerjob.network.external.port.http=10010
                  -Dpowerjob.network.external.port.akka=10086
              - name: PARAMS
                value: >-
                  --spring.profiles.active=product
                  --spring.datasource.core.jdbc-
url=jdbc:mysql://192.168.168.82:3306/powerjob_server?useUnicod=true&characterEncoding=UTF-
8&zeroDateTimeBehavior=convertToNull&useSSL=false      # 根据实际修改 mysql 信息
                  --spring.datasource.core.username=snest_powerjob
                  --spring.datasource.core.password=SnestPowerjob1472i
                  --oms.mongodb.enable=false
            resources: {}
            volumeMounts:
              - name: volume-01
                mountPath: /root/powerjob/server/
            terminationMessagePath: /dev/termination-log
            terminationMessagePolicy: File
            imagePullPolicy: IfNotPresent
        restartPolicy: Always
        terminationGracePeriodSeconds: 30
        dnsPolicy: ClusterFirst
        serviceAccountName: default
        serviceAccount: default
        securityContext: {}
        schedulerName: default-scheduler
```

```
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 25%
  revisionHistoryLimit: 10
  progressDeadlineSeconds: 600
```

- 登录 K8S 管理页面，在【应用负载】->【服务】菜单页面下，点击创建，输入服务名称：powerjob-server，选择所属项目。



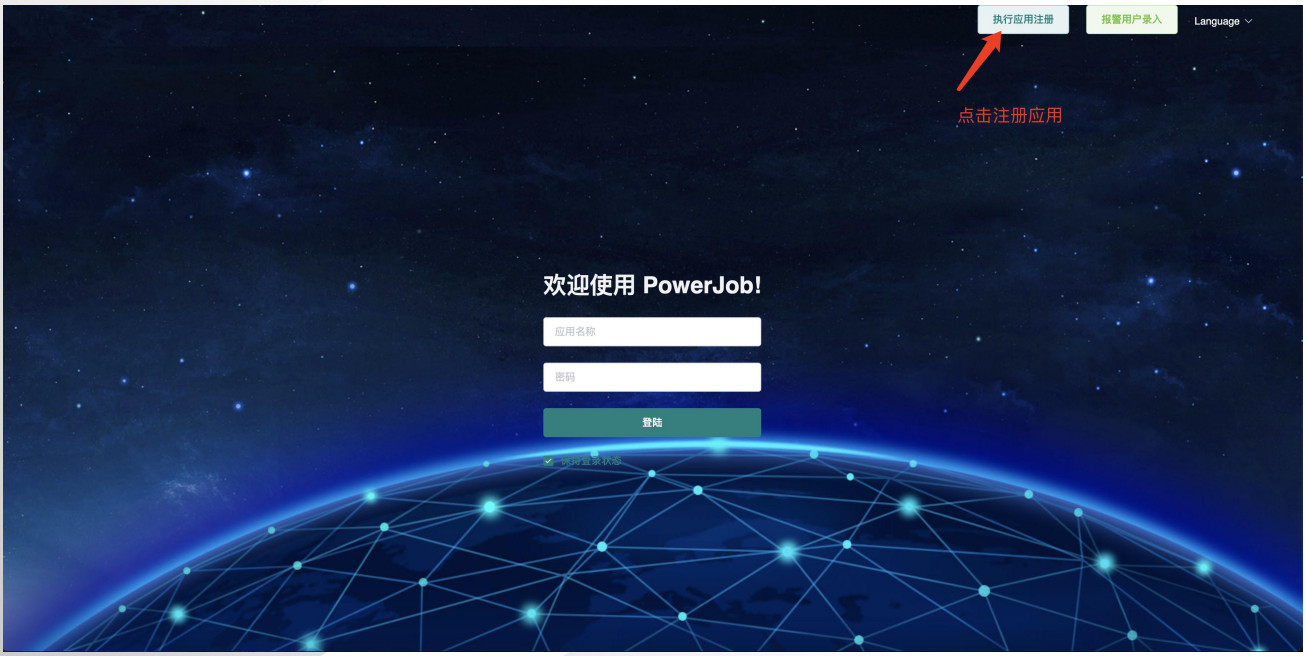


【服务】创建成功，找到 powerjob-server 服务，编辑 YAML 配置文件内容(注意点：根据实际修改名称空间)

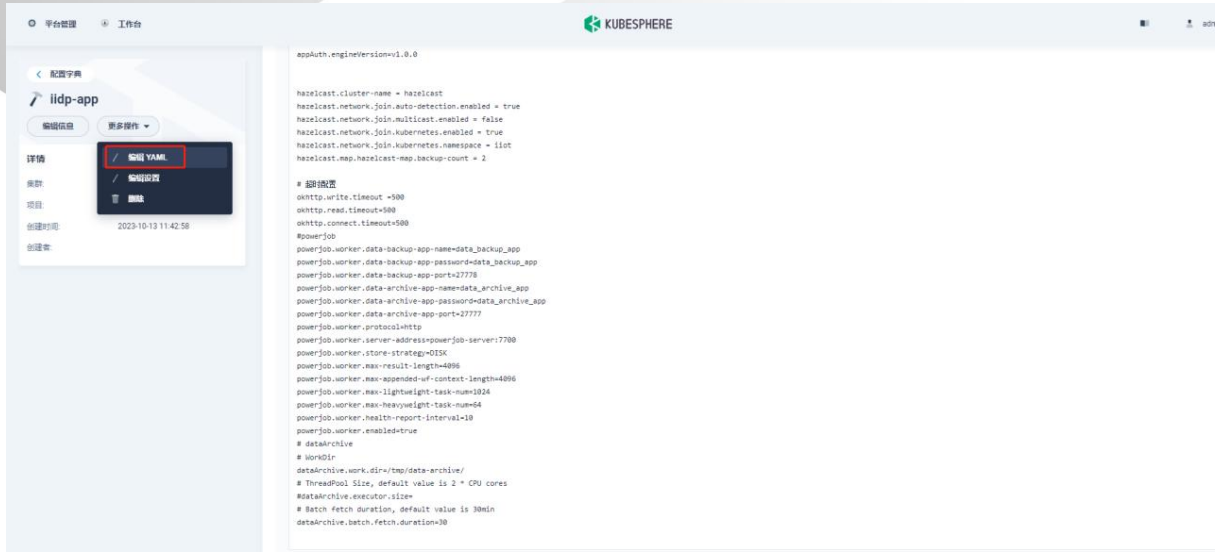
```
kind: Service
apiVersion: v1
metadata:
  name: powerjob-server
  namespace: iiot                    # 根据实际修改名称空间
  labels:
    app: powerjob-server
  annotations:
    kubesphere.io/creator: admin
spec:
  ports:
    - name: tcp-7700
      protocol: TCP
      port: 7700
      targetPort: 7700
      nodePort: 37700
    - name: tcp-10086
      protocol: TCP
      port: 10086
      targetPort: 10086
      nodePort: 30086
    - name: tcp-10010
      protocol: TCP
      port: 10010
      targetPort: 10010
      nodePort: 30010
  selector:
    app: powerjob-server
  type: NodePort
  sessionAffinity: None
  externalTrafficPolicy: Cluster
  ipFamilies:
    - IPv4
  ipFamilyPolicy: SingleStack
  internalTrafficPolicy: Cluster
```

- 若数据库是高斯数据库（GaussDB），需要在 powerjob_server 数据库（根据实际情况修改）执行兼容脚本 PowerjobServer 高斯数据库兼容 DDL 脚本。若是非高斯数据库，则忽略该步骤。

- 登录 powerjob-server 服务（http://部署服务器 IP:37700/#/welcome），注册 data_archive_app、data_backup_app。



➢ 登录 K8S 管理页面，在【配置】->【配置字典】菜单页面下找到 IIDP 平台的配置字典，点击编辑 YAML，在 application-dev.properties 配置文件新增以下配置信息。




```
# powerjob
powerjob.worker.data-archive-app-name=data_archive_app
powerjob.worker.data-archive-app-password=data_archive_app
powerjob.worker.data-archive-app-port=28777
powerjob.worker.data-backup-app-name=data_backup_app
powerjob.worker.data-backup-app-password=data_backup_app
powerjob.worker.data-backup-app-port=28778
powerjob.worker.protocol=http
powerjob.worker.server-address=powerjob-server:7700
powerjob.worker.store-strategy=DISK
powerjob.worker.max-result-length=4096
powerjob.worker.max-appended-wf-context-length=4096
```

```
powerjob.worker.max-lightweight-task-num=1024
powerjob.worker.max-heavyweight-task-num=64
powerjob.worker.health-report-interval=10
powerjob.worker.enabled=true
```

2. 部署时序数据库

IIOT 依赖时序数据库存储设备采集数据，其存储介质有 2 种，即 TDengine 数据库、InfluxDB 数据库。请根据项目情况选择数据库类型以及部署模式（单机版/集群版）。

2.1. 部署 TDengine 数据库

2.1.1. 单机版

单机版 TDengine 数据库即仅使用一台服务器部署 TDengine 数据库。

2.1.1.1. 安装 docker 及 docker-compose 组件

参考 1.3.1 章节安装 docker、docker-compose 组件。

2.1.1.2. 安装 tdengine 数据库

- 参考表 1-2 依赖组件表下载依赖组件 tdengine.tar，再从安装人员本机上传依赖组件到服务器的 /var 目录（假设数据盘挂载点是：/var，请根据实际情况修改）。
- 登录服务器，装载 tdengine 镜像到 docker 镜像库。

```
docker load < tdengine-3.4.0.2.tar
```

- 登录服务器，创建 tdengine 安装及配置目录。

```
mkdir -p /var/tdengine
mkdir -p /var/tdengine/{data,log}
```

- 下载 tdengine 配置文件及部署脚本 standalone.tar，上传至服务器目录：/var/tdengine/，解压压缩文件。
-

```
cd /var/tdengine
tar -zxvf standalone.tar
mv standalone/* .
rm -rf standalone
```

- 登录服务器，启动 tdengine 服务。

```
docker-compose -f /var/tdengine/td-standalone-compose.yml up -d
```

- 登录服务器，验证 tdengine 服务是否正常。

```
docker ps | grep td-standalone
```


```
docker exec -it td-standalone ps -ef
```

```
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 Mar05 ?        00:00:15 /tini -- /usr/bin/entrypoint.sh taosd
root           7     1  0 Mar05 ?        00:00:00 /bin/sh /usr/bin/entrypoint.sh taosd
root          38     7 14 Mar05 ?        22:28:04 taosd
root          39     7  2 Mar05 ?        03:34:44 taosadapter
root          66    38  0 Mar05 ?        00:08:42 /usr/bin/udfd -c /etc/taos/
root         188     7  0 Mar05 ?        00:07:09 taoskeeper
root         210     7  0 Mar05 ?        00:13:15 taos-explorer
root        1080     0  0 07:42 pts/0    00:02:50 taos
root        1160     0  1 14:49 pts/1    00:00:00 ps -ef
```

2.1.2. 集群版

集群版 TDengine 数据库即使用多台（通常是 3 台）服务器部署 TDengine 数据库。

注意：若服务器已安装 docker 及 docker-compose 基础软件，请忽略该步骤，否则请参考该步骤执行 docker 及 docker-compose 基础软件安装（假设数据盘是：/var，服务器 IP 依次：192.168.168.26、192.168.168.27、192.168.168.28）。

2.1.2.1. 安装 docker、docker-compose 组件

参考 1.3.1 章节安装 docker、docker-compose 组件。

2.1.2.2. 安装 TDengine 数据库

- 参考表 2-2 依赖组件表下载依赖组件 tdengine.tar，再从安装人员本机分别上传依赖组件到【3 台】服务器的 /var 目录（假设数据盘挂载点是：/var，请根据实际情况修改）。
- 分别登录【3 台】服务器，装载 tdengine 镜像到 docker 镜像库。

```
cd /var && docker load < tdengine-3.3.3.0.tar
```

- 分别登录【3 台】服务器，创建 tdengine 安装及配置目录。

```
mkdir -p /var/tdengine
mkdir -p /var/tdengine/{data,log}
```

- 下载 tdengine 配置文件 td1.tar，上传到第一台服务器（假设命名是 td1）的目录：/var/tdengine/，执行解压命令。

```
cd /var/tdengine
tar -zxvf td1.tar
```

- 下载 tdengine 配置文件 td2.tar，上传到第二台服务器（假设命名是 td2）的目录：/var/tdengine/，执行解压命令。

```
cd /var/tdengine
tar -zxvf td2.tar
```

- 下载 tdengine 配置文件 td3.tar，上传到第一台服务器（假设命名是 td3）的目录：/var/tdengine/，执行解压命令。

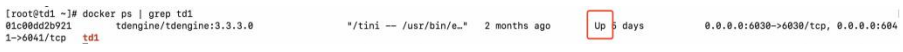
```
cd /var/tdengine
tar -zxvf td3.tar
```

- 分别登陆 3 台服务器，修改目录 /var/tdengine/ 下文件名后缀是 -compose.yml 的部署脚本文件。**请根据实际服务器信息修改集群节点 IP 信息(此处使用：192.168.168.26、192.168.168.27、192.168.168.28)。**
- 分别登录【3 台】服务器，启动 tdengine 服务。

```
docker-compose -f /var/tdengine/td-cluster-td1-compose.yml up -d
docker-compose -f /var/tdengine/td-cluster-td2-compose.yml up -d
docker-compose -f /var/tdengine/td-cluster-td3-compose.yml up -d
```

- 分别登录【3 台】服务器，验证 tdengine 服务是否正常（此处以 td1 为例）。

```
docker ps | grep td1
```




```
docker exec -it td1 ps -ef
```

```
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 Mar05 ?        00:00:15 /tini -- /usr/bin/entrypoint.sh taosd
root           7     1  0 Mar05 ?        00:00:00 /bin/sh /usr/bin/entrypoint.sh taosd
root          38     7 14 Mar05 ?        22:28:04 taosd
root          39     7  2 Mar05 ?        03:34:44 taosadapter
root          66    38  0 Mar05 ?        00:08:42 /usr/bin/udfd -c /etc/taos/
root         188     7  0 Mar05 ?        00:07:09 taoskeeper
root         210     7  0 Mar05 ?        00:13:15 taos-explorer
root        1080     0  0 07:42 pts/0    00:02:50 taos
root        1160     0  1 14:49 pts/1    00:00:00 ps -ef
```

2.2. 部署 InfluxDB 数据库

2.2.1. 单机版

单机版 InfluxDB 数据库即仅使用一台服务器部署 InfluxDB 数据库。

2.2.1.1. 安装 docker 及 docker-compose 组件

参考 1.3.1 章节安装 docker、docker-compose 组件。

2.2.1.2. 安装 InfluxDB 数据库

- 参考表 2-2 依赖组件表下载依赖组件 influxdb.tar，再从安装人员本机上传依赖组件到服务器的 /var 目录（假设数据盘挂载点是：/var，请根据实际情况修改）。
- 登录服务器，装载 influxdb 镜像到 docker 镜像库。

```
cd /var && docker load < influxdb-1.8.0-alpine.tar
```

- 登录服务器，创建 influxdb 安装及配置目录。

```
mkdir -p /var/influxdb
mkdir -p /var/influxdb/{conf,data,log}
```

- 登录服务器，配置 influxdb 配置文件 influxdb.conf。

```
echo '
[http]
  enabled = true
  auth-enabled = false
[meta]
  dir = "/var/lib/influxdb/meta"
[data]
  dir = "/var/lib/influxdb/data"
```

```
    engine = "tsm1"
    wal-dir = "/var/lib/influxdb/wal"
    max-values-per-tag = 0
    max-series-per-database = 0

' > /var/influxdb/conf/influxdb.conf
```

- 登录服务器，配置 influxdb 部署脚本（influxdb-standalone-compose.yml）。

```
echo '
version: "3.6"

services:
  influxdb-standalone:
    image: "harbor.devcenter.gushen.sieiot.com/iiot/influxdb:1.8.0-alpine"
    hostname: influxdb-standalone
    container_name: influxdb-standalone
    restart: always
    privileged: true
    ports:
      - "8083:8083"
      - "8086:8086"
    volumes:
      - "./data:/var/lib/influxdb"
      - "./conf:/etc/influxdb"
    environment:
      - TZ=Asia/Shanghai
      - INFLUXDB_ADMIN_USER=admin
      - INFLUXDB_ADMIN_PASSWORD=admin123
      - GODEBUG=madvdontneed=1
    logging:
      driver: "json-file"
      options:
        max-size: "200m"
' > /var/influxdb/influxdb-standalone-compose.yml
```

- 登录服务器，启动 influxdb 服务。

```
docker-compose -f /var/influxdb/influxdb-standalone-compose.yml up -d
```

- 登录服务器，验证 influxdb 服务是否正常。

2.2.2. 集群版

集群版 InfluxDB 数据库即使用多台（通常是 3 台）服务器部署 InfluxDB 数据库。

注意：若服务器已安装 docker 及 docker-compose 基础软件，请忽略该步骤，否则请参考该步骤执行 docker 及 docker-compose 基础软件安装（假设数据盘是：/data，服务器 IP 依次：192.168.168.26、192.168.168.27、192.168.168.28）。

2.2.2.1. 安装 docker、docker-compose 组件

参考 1.3.1 章节安装 docker、docker-compose 组件。

2.2.2.2. 安装 InfluxDB 数据库

- 参考表 2-2 依赖组件表下载依赖组件 influxdb.tar、influx-proxy.tar，再从安装人员本机分别上传依赖组件到【3 台】服务器的 /var 目录（假设数据盘挂载点是：/var，请根据实际情况修改）。
- 分别登录【3 台】服务器，装载 influxdb 镜像到 docker 镜像库。

```
cd /var && docker load < influxdb-1.8.0-alpine.tar
cd /var && docker load < influx-proxy-2.5.10.tar
```

- 分别登录【3 台】服务器，创建 influxdb 安装及配置目录。

```
mkdir -p /var/influxdb
mkdir -p /var/influxdb/{conf,data,log}
```

- 分别登录【3 台】服务器，配置 influxdb 配置文件 influxdb.conf。

```
echo '
[http]
  enabled = true
  auth-enabled = false
[meta]
  dir = "/var/lib/influxdb/meta"
[data]
  dir = "/var/lib/influxdb/data"
  engine = "tsm1"
  wal-dir = "/var/lib/influxdb/wal"
  max-values-per-tag = 0
  max-series-per-database = 0

' > /var/influxdb/conf/influxdb.conf
```

- 分别登录【3 台】服务器，配置 influxdb 部署脚本 influxdb-cluster-idb1-compose.yml、influxdb-cluster-idb2-compose.yml、influxdb-cluster-idb3-compose.yml。

```
echo '
version: "3.6"

services:
  influxdb-cluster-idb1:
    image: "harbor.devcenter.gushen.sieiot.com/iiot/influxdb:1.8.0-alpine"
    hostname: influxdb-cluster-idb1
    container_name: influxdb-cluster-idb1
    restart: always
    privileged: true
    ports:
      - "8083:8083"
      - "8086:8086"
    volumes:
      - "./data:/var/lib/influxdb"
      - "./conf:/etc/influxdb"
    environment:
      - TZ=Asia/Shanghai
      - INFLUXDB_ADMIN_USER=admin
      - INFLUXDB_ADMIN_PASSWORD=admin123
      - GODEBUG=madvdontneed=1
    logging:
      driver: "json-file"
      options:
        max-size: "200m"
' > /var/influxdb/influxdb-cluster-idb1-compose.yml
echo '
version: "3.6"

services:
  influxdb-cluster-idb2:
    image: "harbor.devcenter.gushen.sieiot.com/iiot/influxdb:1.8.0-alpine"
    hostname: influxdb-cluster-idb2
    container_name: influxdb-cluster-idb2
    restart: always
    privileged: true
    ports:
      - "8083:8083"
      - "8086:8086"
    volumes:
      - "./data:/var/lib/influxdb"
```

```
      - "./conf:/etc/influxdb"
    environment:
      - TZ=Asia/Shanghai
      - INFLUXDB_ADMIN_USER=admin
      - INFLUXDB_ADMIN_PASSWORD=admin123
      - GODEBUG=madvdontneed=1
    logging:
      driver: "json-file"
      options:
        max-size: "200m"
' > /var/influxdb/influxdb-cluster-idb2-compose.yml
echo '
version: "3.6"

services:
  influxdb-cluster-idb3:
    image: "harbor.devcenter.gushen.sieiot.com/iiot/influxdb:1.8.0-alpine"
    hostname: influxdb-cluster-idb3
    container_name: influxdb-cluster-idb3
    restart: always
    privileged: true
    ports:
      - "8083:8083"
      - "8086:8086"
    volumes:
      - "./data:/var/lib/influxdb"
      - "./conf:/etc/influxdb"
    environment:
      - TZ=Asia/Shanghai
      - INFLUXDB_ADMIN_USER=admin
      - INFLUXDB_ADMIN_PASSWORD=admin123
      - GODEBUG=madvdontneed=1
    logging:
      driver: "json-file"
      options:
        max-size: "200m"
' > /var/influxdb/influxdb-cluster-idb3-compose.yml
```

- 分别登录【3 台】服务器，启动 influxdb 服务。

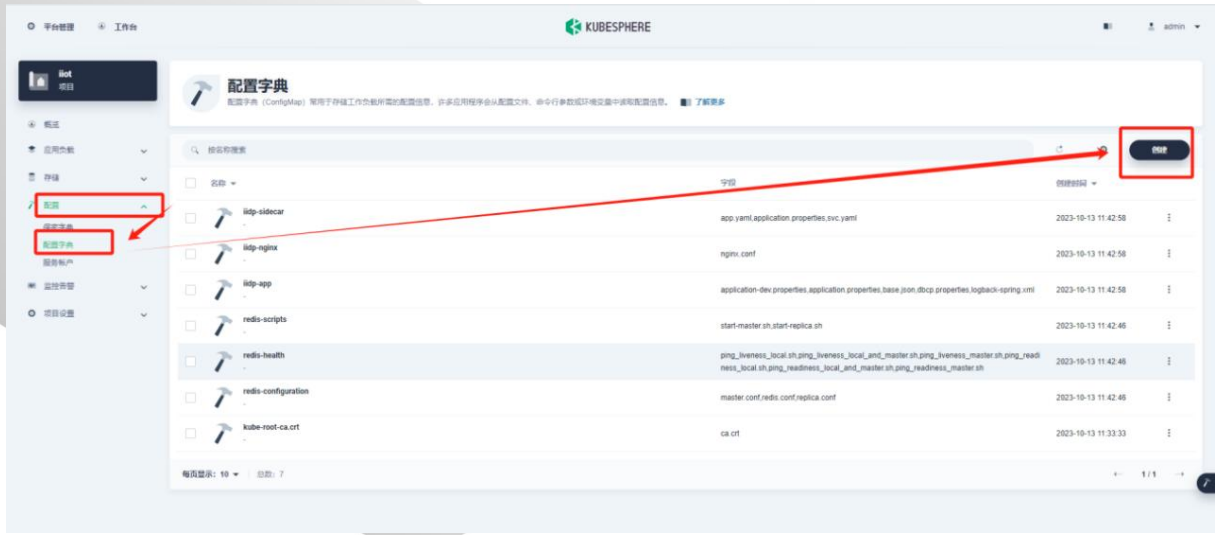
```
docker-compose -f /var/influxdb/influxdb-cluster-idb1-compose.yml up -d
docker-compose -f /var/influxdb/influxdb-cluster-idb2-compose.yml up -d
docker-compose -f /var/influxdb/influxdb-cluster-idb3-compose.yml up -d
```

- 分别登录【3 台】服务器，验证 influxdb 服务是否正常。

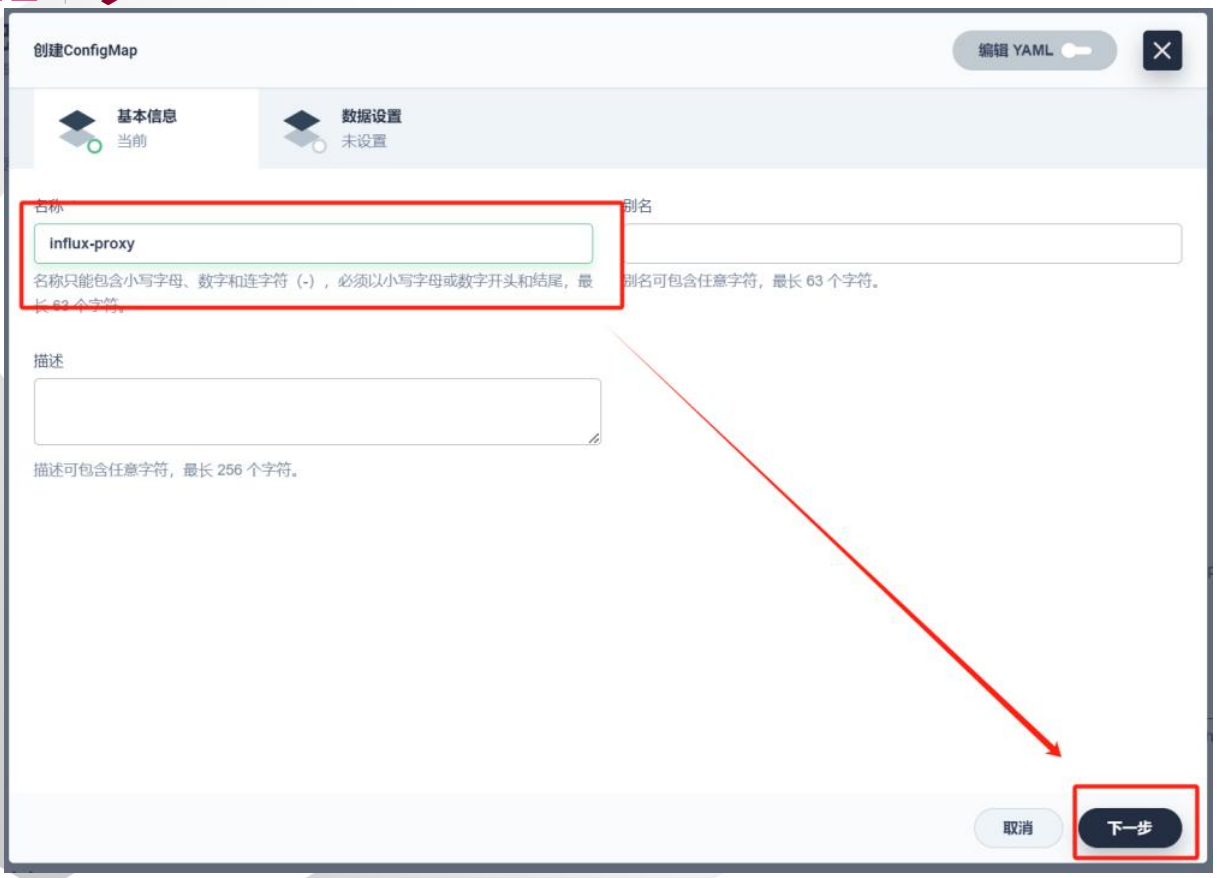
```
[root@td-1 snest]# docker ps | grep influxdb-cluster-idb1
95bb1536efa3        influxdb:1.8.0-alpine                "/entrypoint.sh infl…"   18 seconds ago      Up 17 seconds
      0.0.0.0:8083->8083/tcp, 0.0.0.0:8086->8086/tcp
 influxdb-cluster-idb1
[root@td-1 snest]# docker ps | grep influxdb-cluster-idb2
bbbfec5522e1        influxdb:1.8.0-alpine                "/entrypoint.sh infl…"   4 seconds ago       Up 3 seconds
      0.0.0.0:8083->8083/tcp, 0.0.0.0:8086->8086/tcp
 influxdb-cluster-idb2
[root@td-1 snest]# docker ps | grep influxdb-cluster-idb3
e937dd4d5ae0        influxdb:1.8.0-alpine                "/entrypoint.sh infl…"   4 seconds ago       Up 3 seconds
      0.0.0.0:8083->8083/tcp, 0.0.0.0:8086->8086/tcp
 influxdb-cluster-idb3
```

2.2.2.3. 安装 InfluxDB 代理服务

- 确保已安装 K8S 集群，登录集群管理平台（kubesphere），在相应的项目里，点击【配置】-->【配置字典】，点击【创建】。



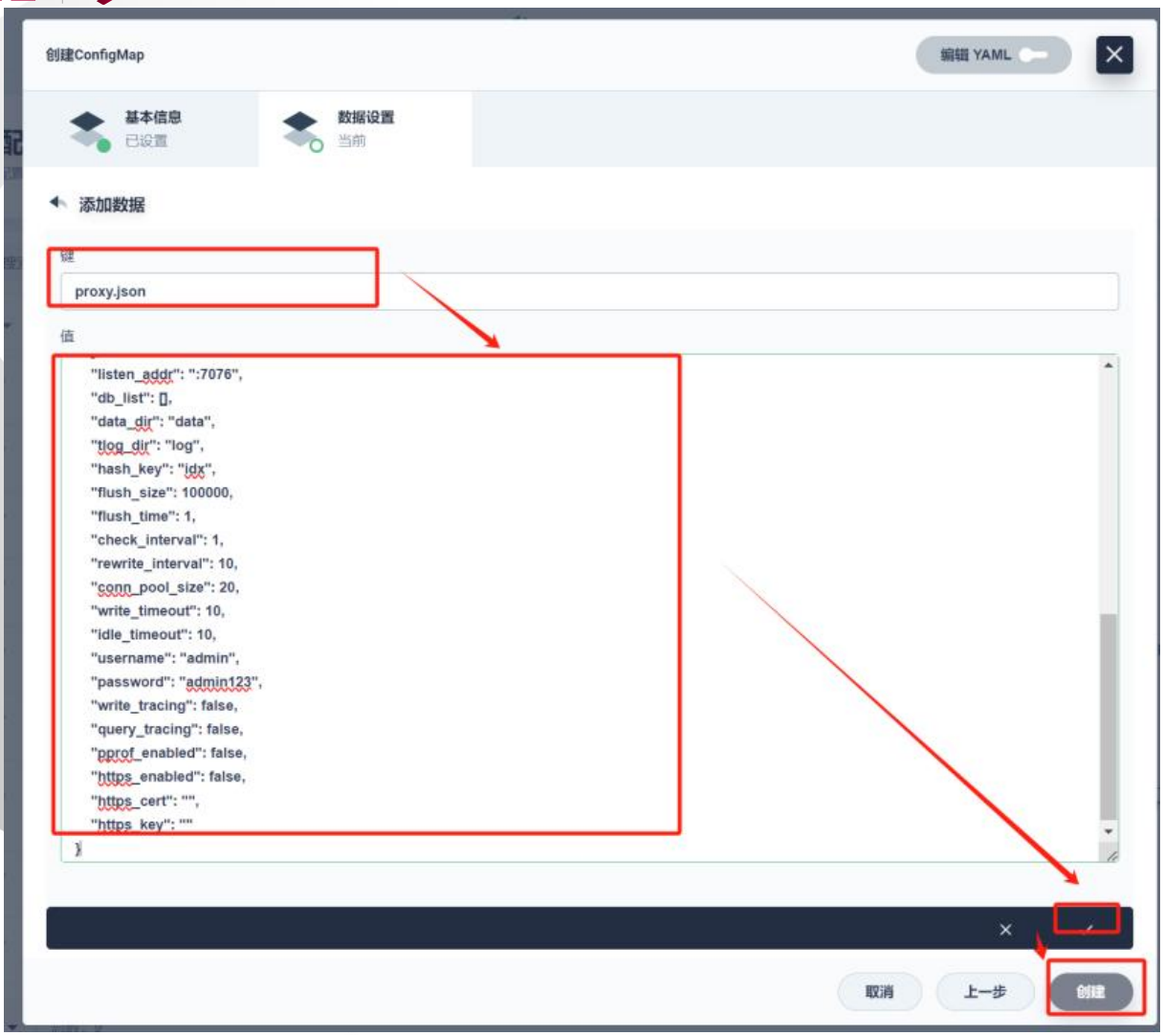
- 弹框界面填写名称“influx-proxy”，点击【下一步】。



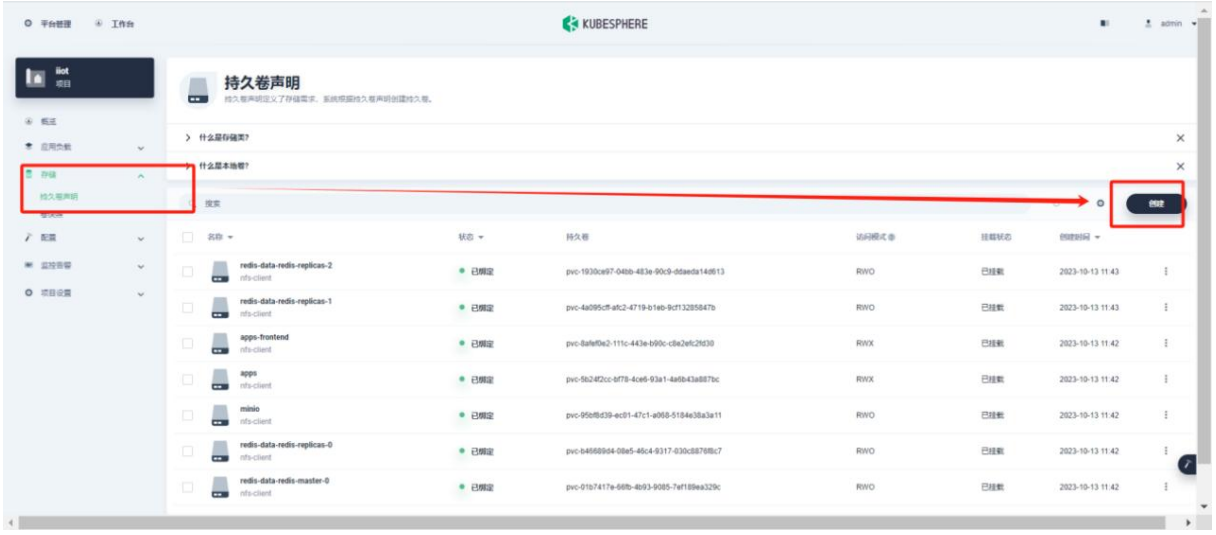
➢ 点击【添加数据】，键填写“proxy.json”，值填写

```
{
    "circles": [
        {
            "name": "circle-1",
            "backends": [
                {
                    "name": "influxdb-cluster-idb1",
                    "url": "http://192.168.168.26:8086",
                    "username": "admin",
                    "password": "admin123"
                }
            ]
        },
        {
            "name": "circle-2",
            "backends": [
                {
                    "name": "influxdb-cluster-idb2",
                    "url": "http://192.168.168.27:8086",
                    "username": "admin",
                    "password": "admin123"
```

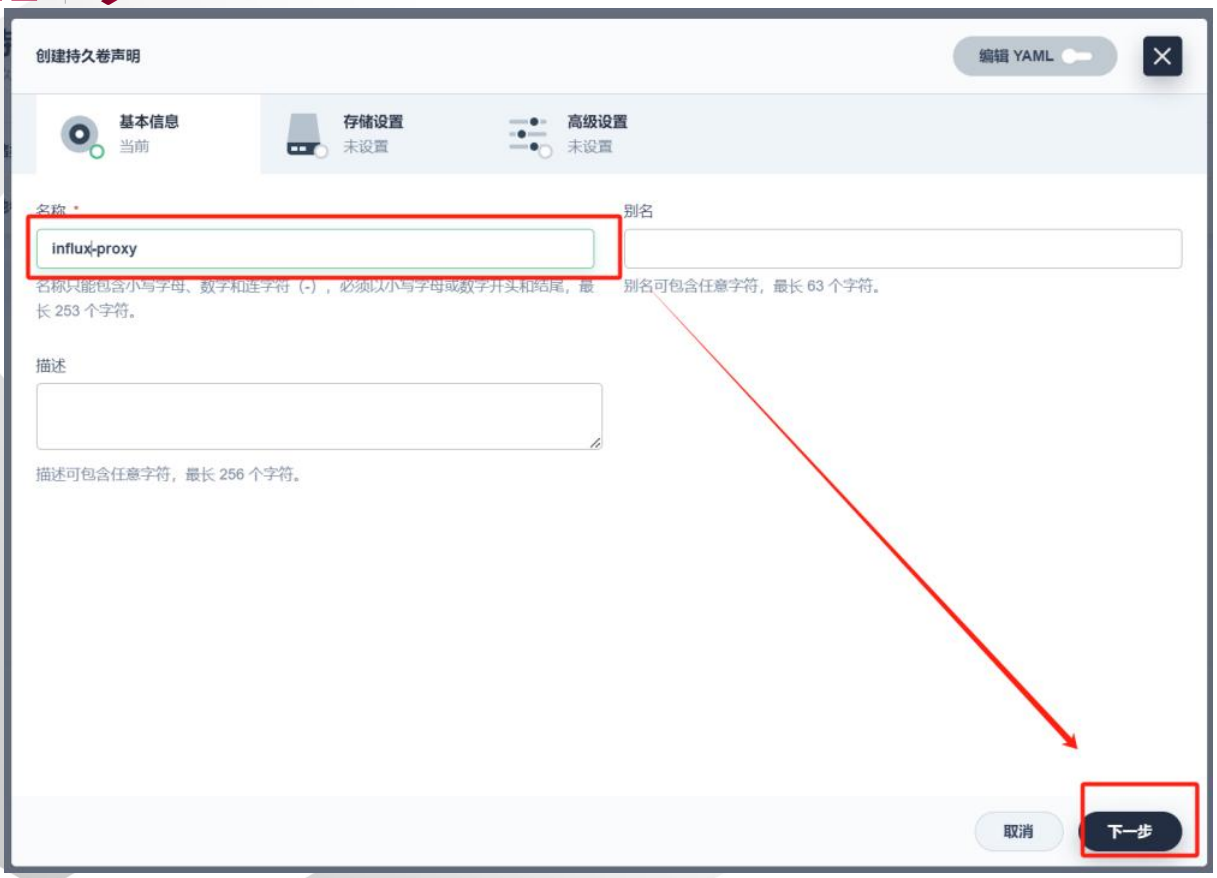
```
                }
            ]
        },
        {
            "name": "circle-3",
            "backends": [
                {
                    "name": "influxdb-cluster-idb3",
                    "url": "http://192.168.168.28:8086",
                    "username": "admin",
                    "password": "admin123"
                }
            ]
        }
    ],
    "listen_addr": ":7076",
    "db_list": [],
    "data_dir": "data",
    "tlog_dir": "log",
    "hash_key": "idx",
    "flush_size": 100000,
    "flush_time": 1,
    "check_interval": 1,
    "rewrite_interval": 10,
    "conn_pool_size": 20,
    "write_timeout": 10,
    "idle_timeout": 10,
    "username": "admin",
    "password": "admin123",
    "write_tracing": false,
    "query_tracing": false,
    "pprof_enabled": false,
    "https_enabled": false,
    "https_cert": "",
    "https_key": ""
}
```



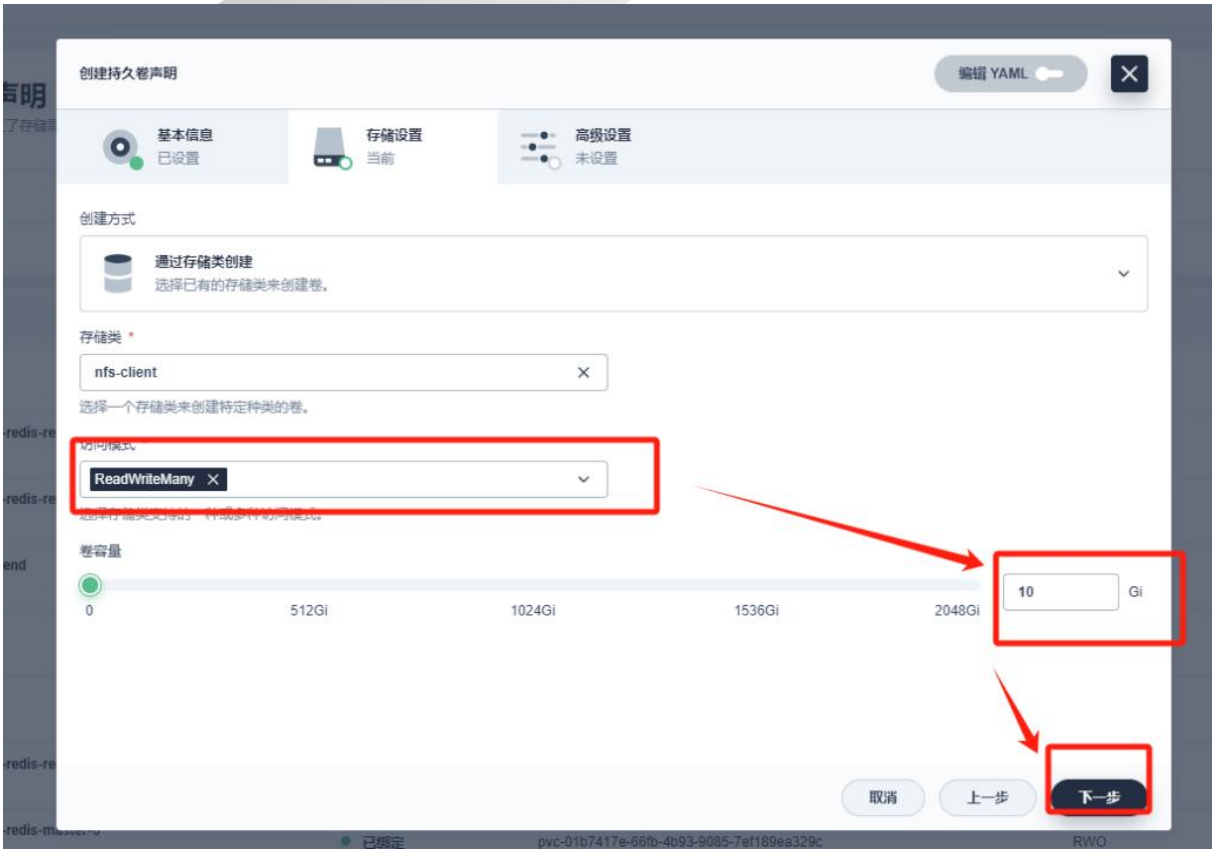
➢ 登录集群管理平台（kubesphere），在相应的项目里，点击【存储】-->【持久卷声明】，点击【创建】。



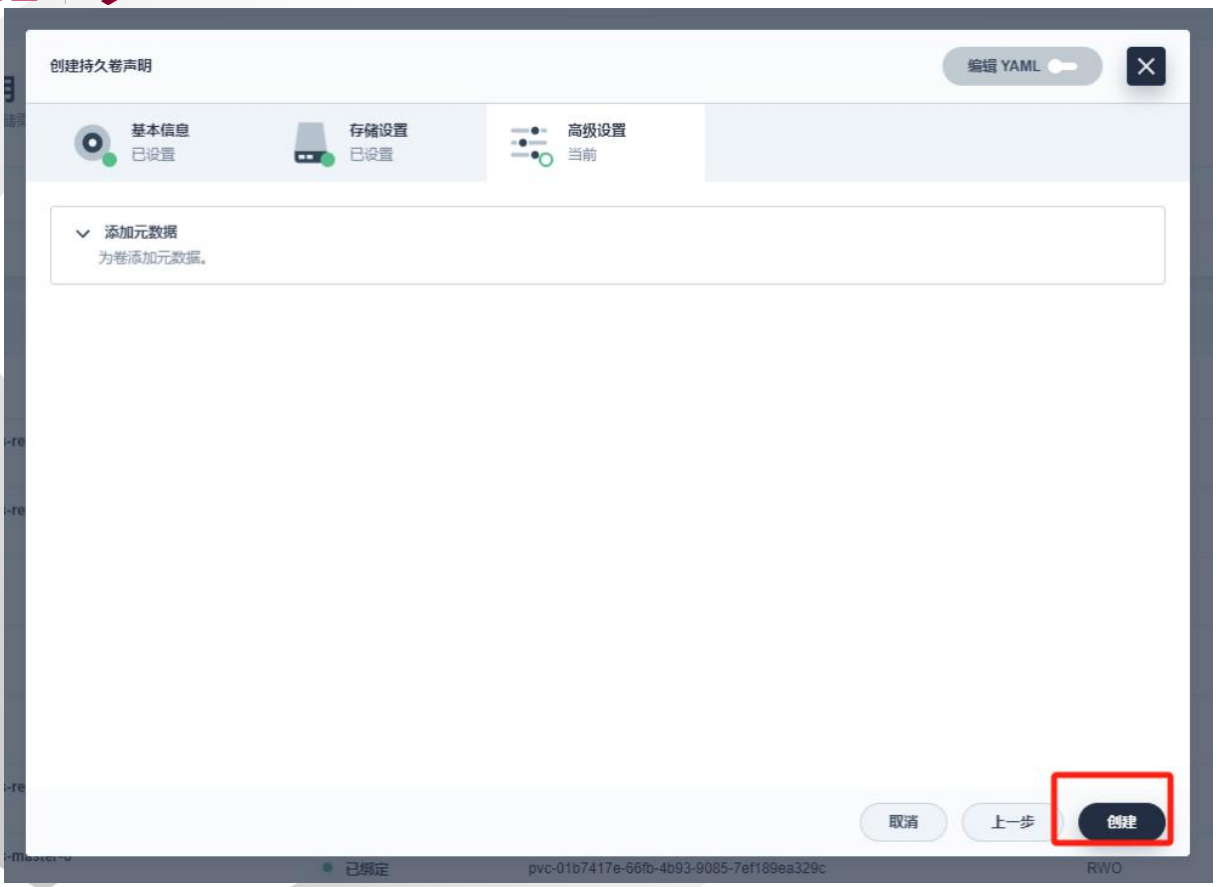
➢ 弹框界面填写名称“influx-proxy”，点击【下一步】。



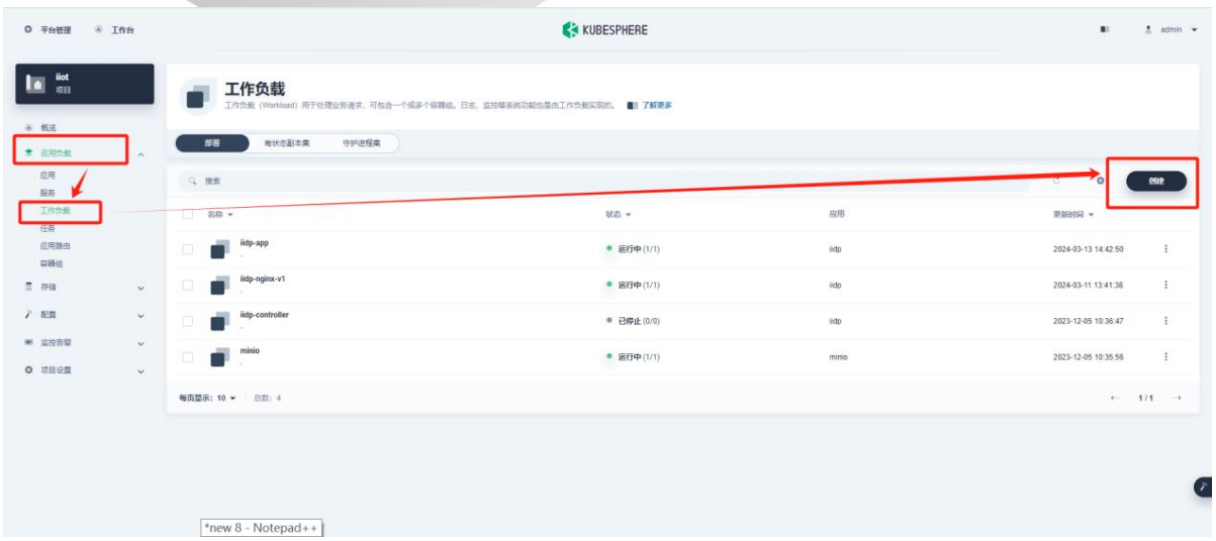
➢ 弹框界面访问模式选择“ReadWriteMany”，卷容量填写“10”，点击【下一步】。



➢ 弹框界面点击【创建】完成。

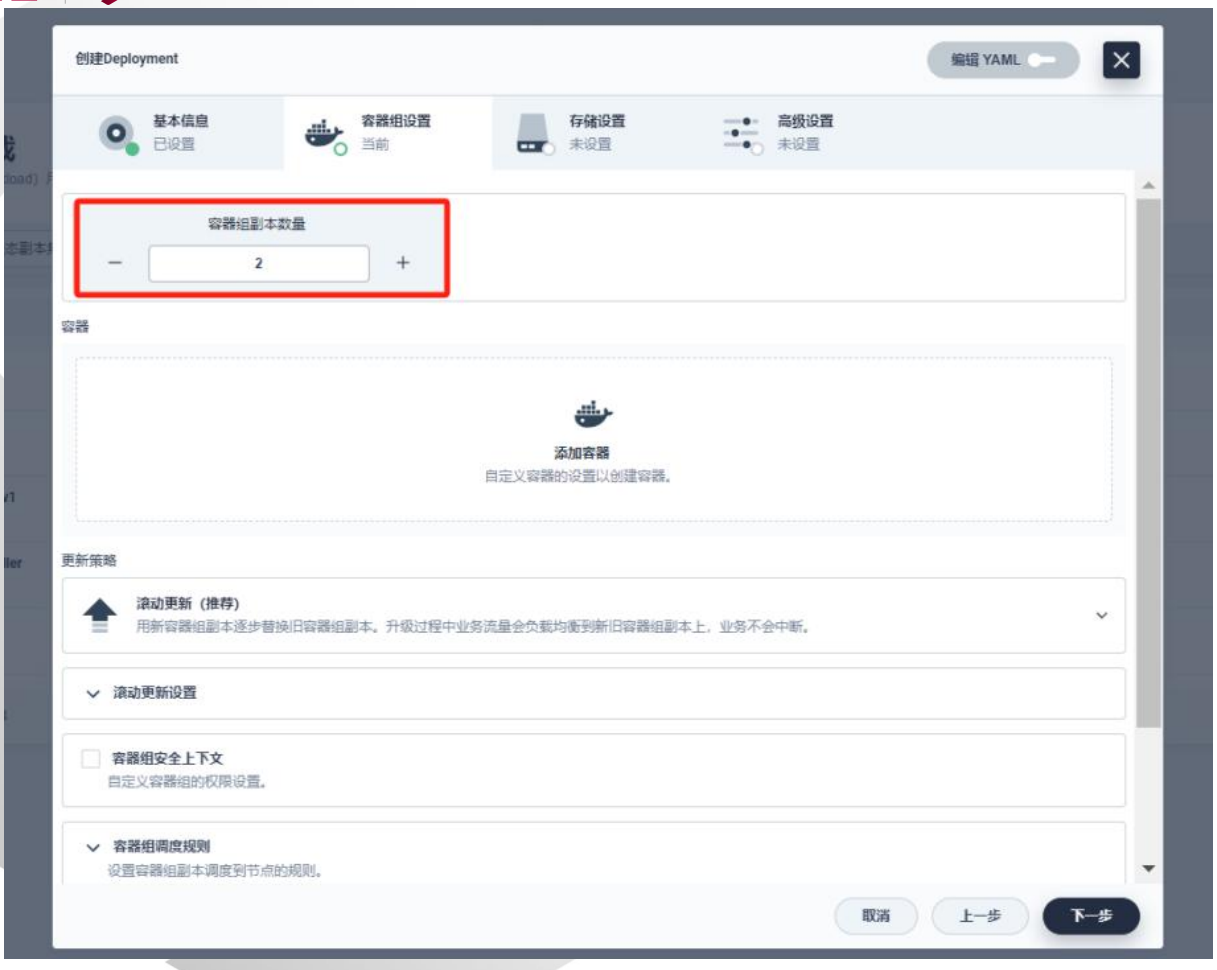


- 登录集群管理平台（kubesphere），在相应的项目里，点击【应用负载】-->【工作负载】，点击【创建】。

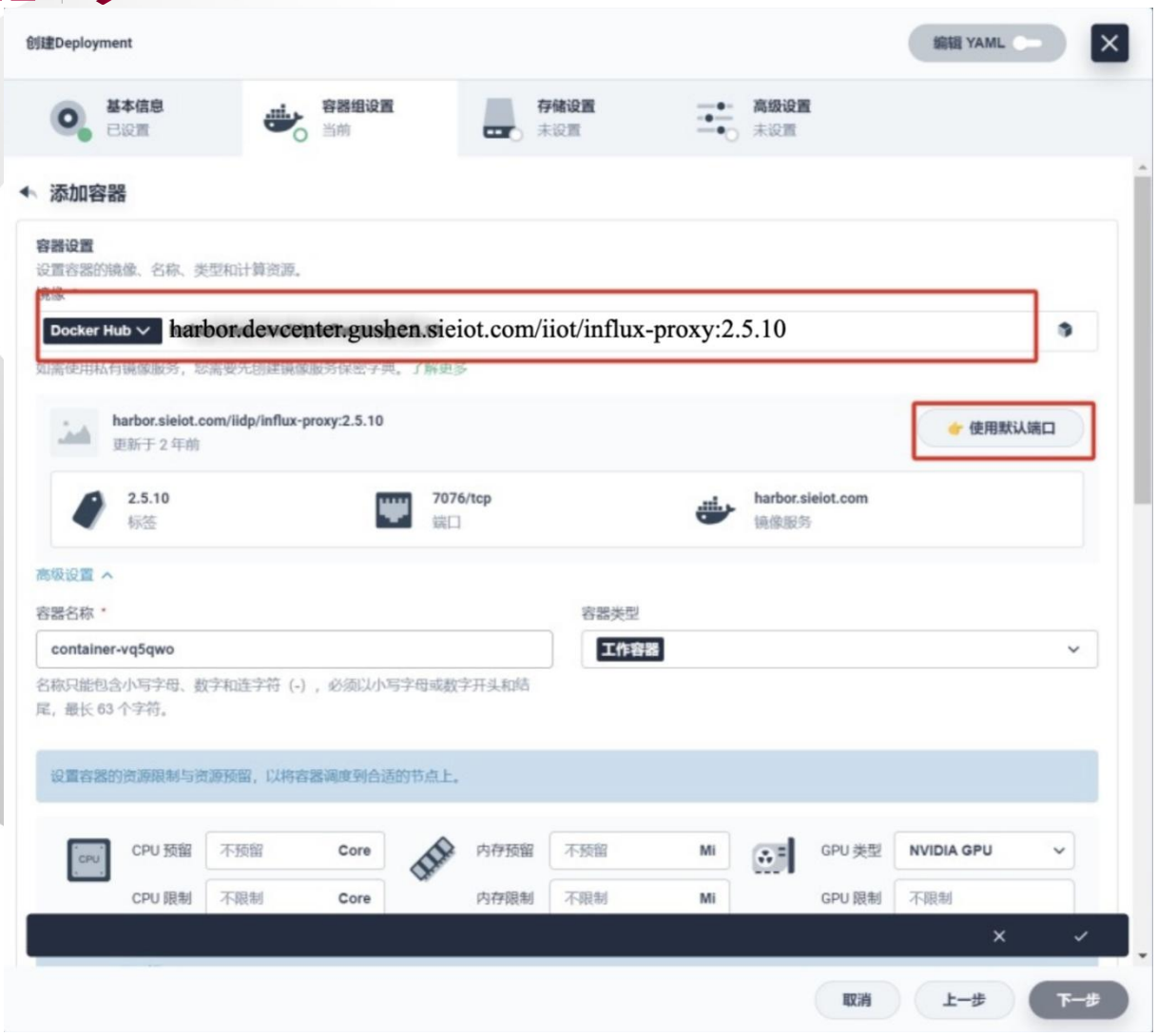


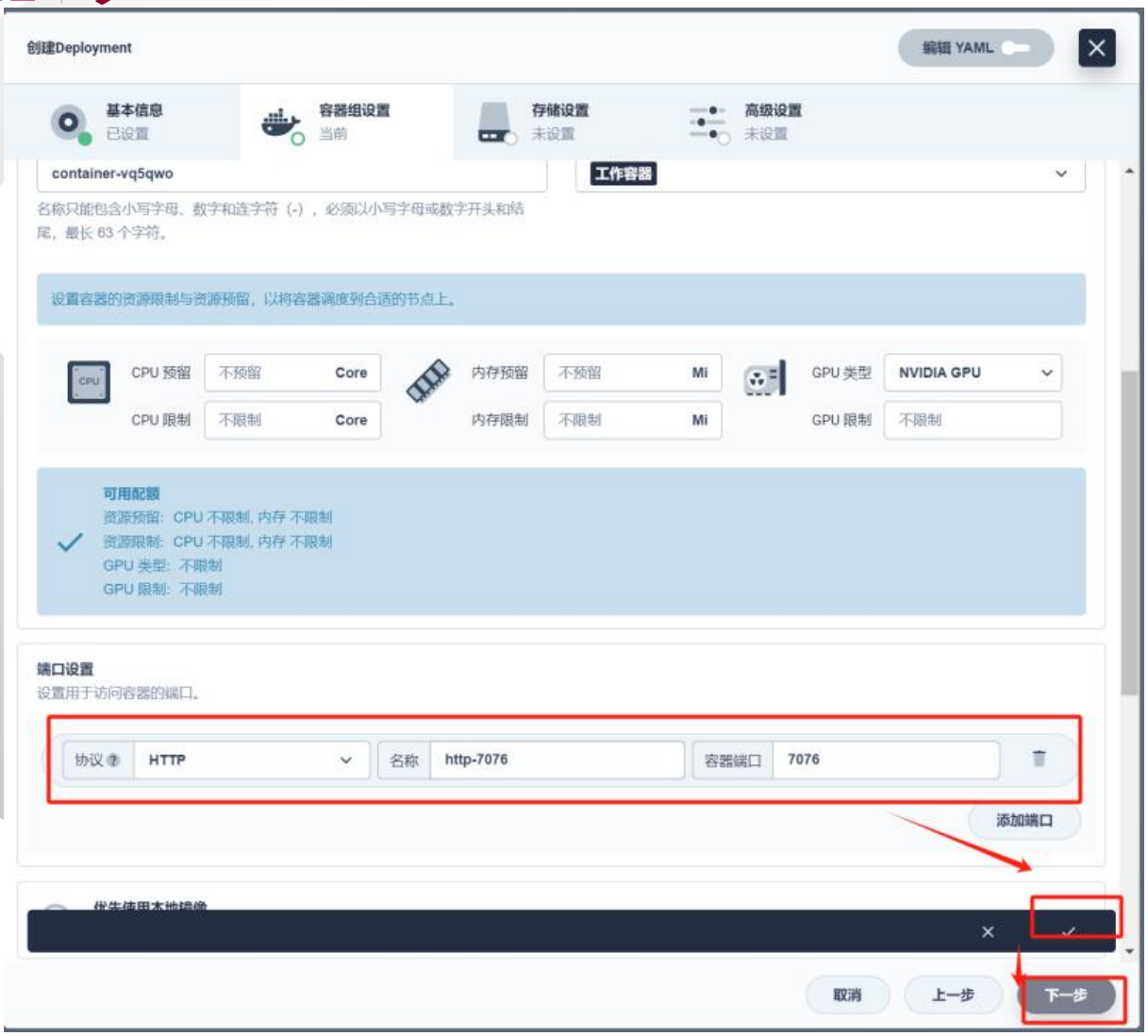
- 弹框界面填写名称“influx-proxy”，点击【下一步】。

- 弹框界面容器组副本数量填写“2” (分布式部署填写 1，高可用部署填写 2 或者以上）。

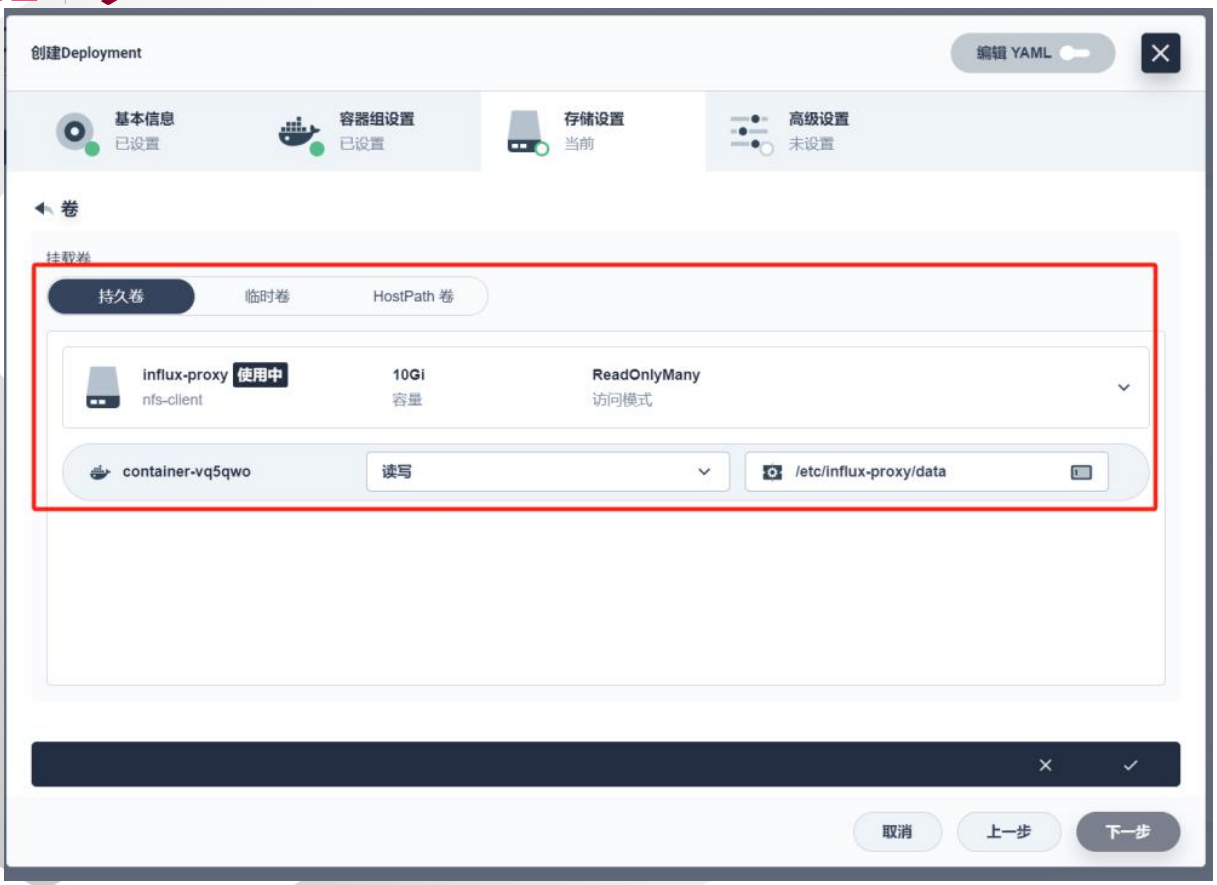


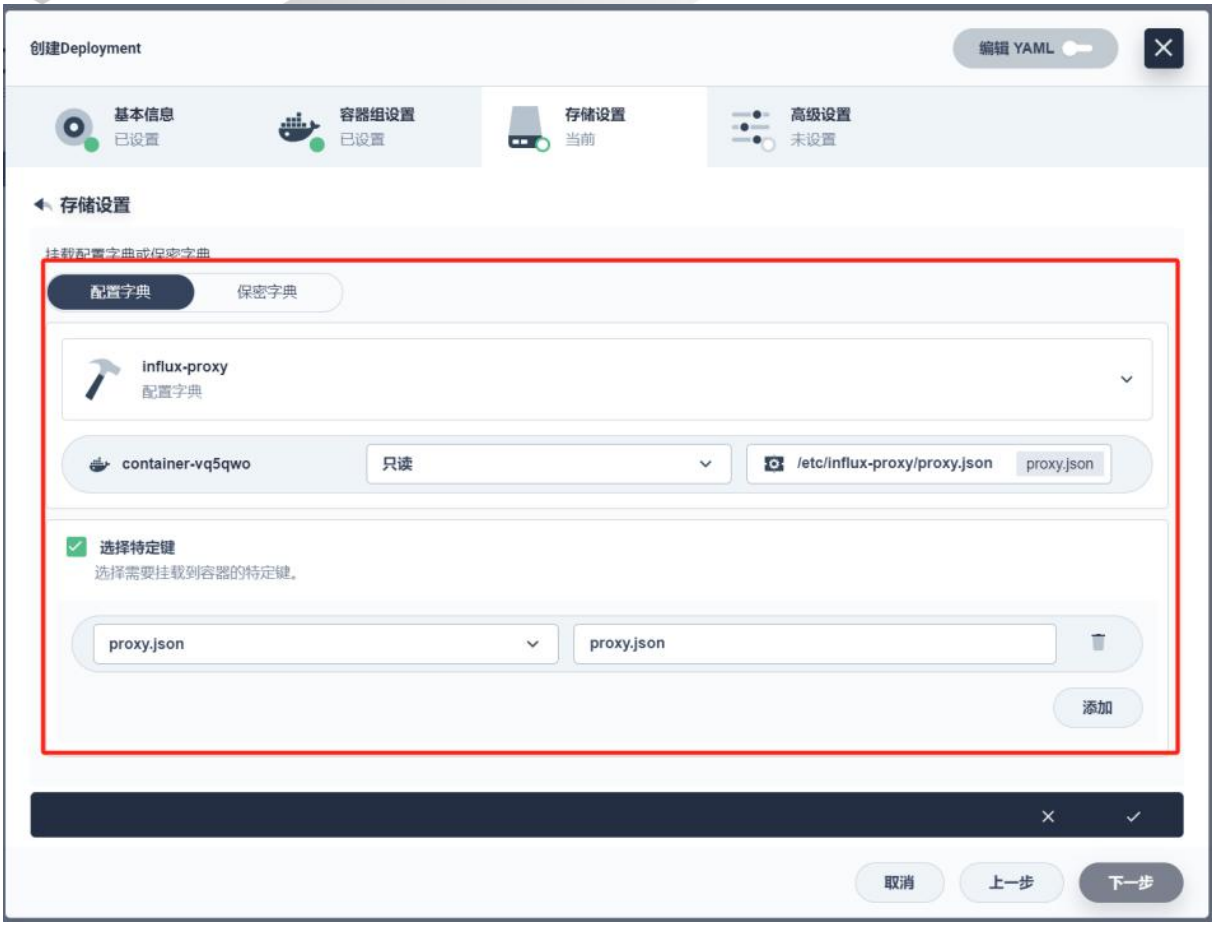
- 点击添加容器，填写容器 url，点击“使用默认端口”（如果没有，则端口设置为 http 7076），点击“√”，容器组调度规则选择分散调度，点击【下一步】。

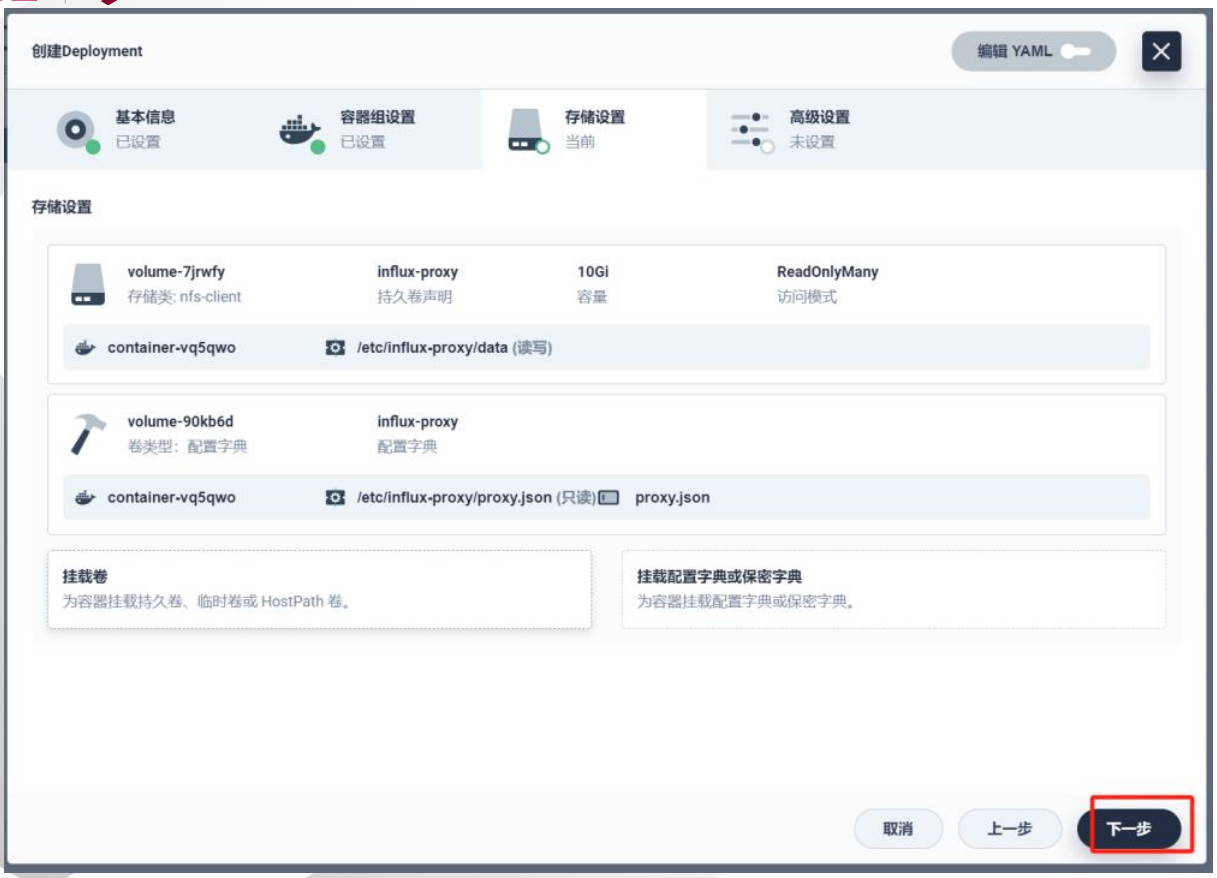

创建Deployment
编辑 YAML
基本信息
已设置
容器组设置
当前
存储设置
未设置
高级设置
未设置
添加容器
容器设置
设置容器的镜像、名称、类型和计算资源。
Docker Hub
harbor.devcenter.gushen.sieiot.com/iiot/influx-proxy:2.5.10
harbor.sieiot.com/iidp/influx-proxy:2.5.10
更新于 2 年前
使用默认端口
2.5.10
标签
7076/tcp
端口
harbor.sieiot.com
镜像服务
高级设置
容器名称 *
container-vq5qwo
容器类型
工作容器
名称只能包含小写字母、数字和连字符（-），必须以小写字母或数字开头和结尾，最长 63 个字符。
设置容器的资源限制与资源预留，以将容器调度到合适的节点上。
CPU 预留
不预留
Core
内存预留
不预留
Mi
GPU 类型
NVIDIA GPU
CPU 限制
不限制
Core
内存限制
不限制
Mi
GPU 限制
不限制
取消
上一步
下一步

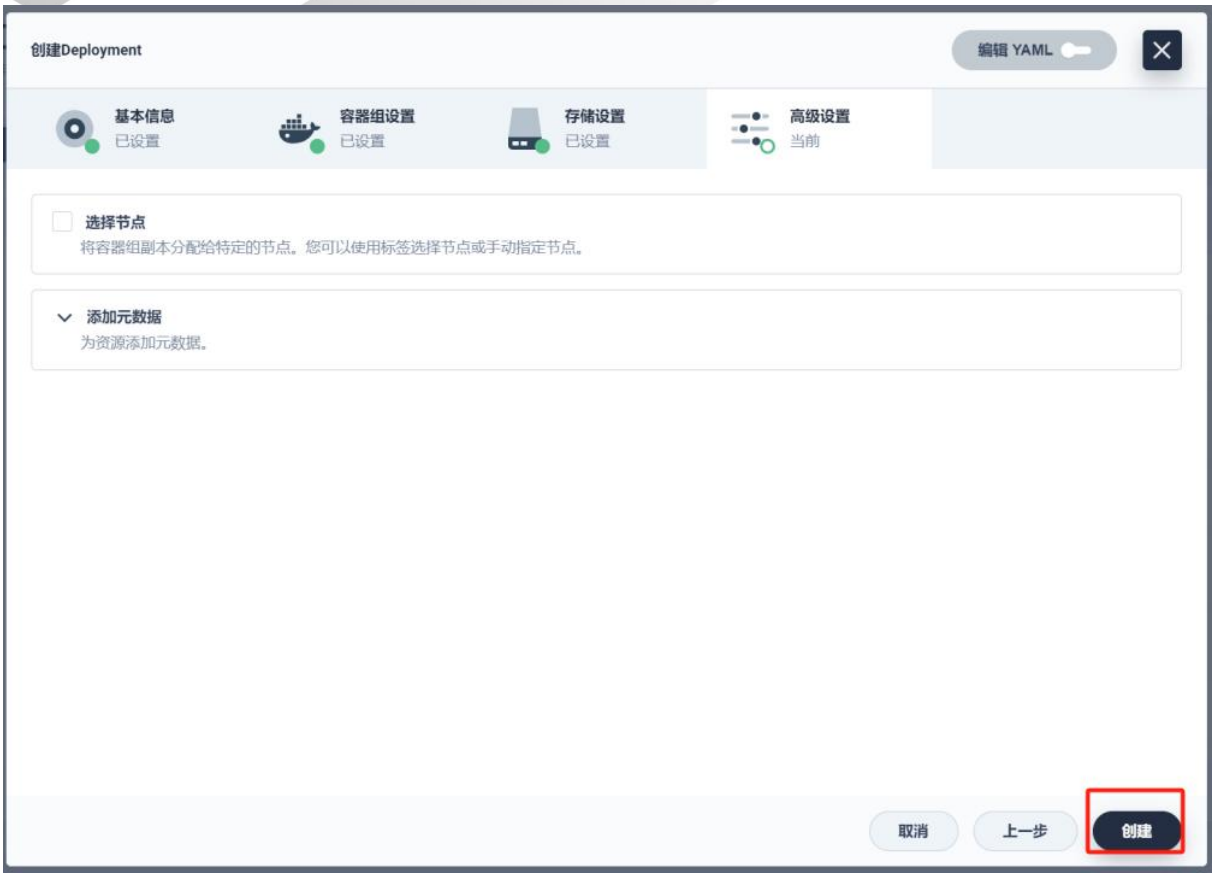


- 挂载卷（选择持久卷 influx-proxy）和配置字典（选择配置字典 influx-proxy），点击【下一步】，最后点击【创建】。

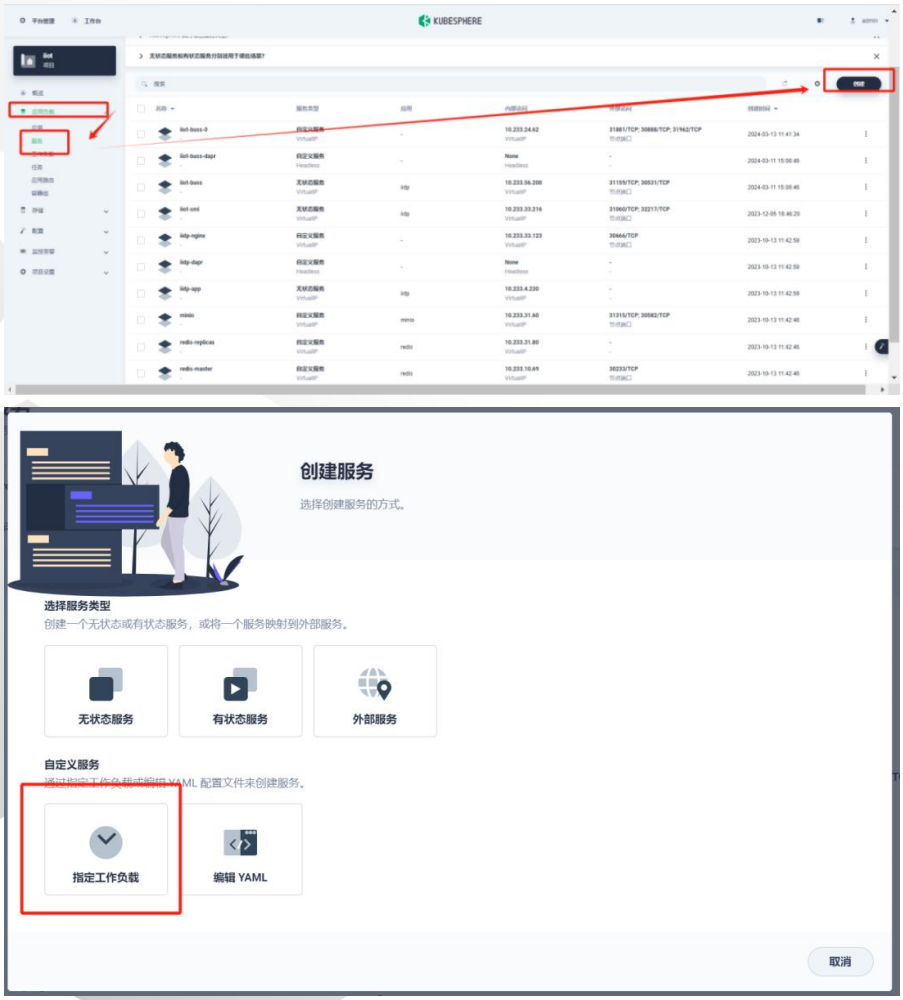

创建Deployment
编辑 YAML
基本信息
已设置
容器组设置
已设置
存储设置
当前
高级设置
未设置
卷
挂载卷
持久卷
临时卷
HostPath 卷
influx-proxy 使用中
nfs-client
10Gi
容量
ReadOnlyMany
访问模式
container-vq5qwo
读写
/etc/influx-proxy/data
取消
上一步
下一步


创建Deployment
编辑 YAML
基本信息
已设置
容器组设置
已设置
存储设置
当前
高级设置
未设置
存储设置
挂载配置字典或保密字典
配置字典
保密字典
influx-proxy
配置字典
container-vq5qwo
只读
/etc/influx-proxy/proxy.json
proxy.json
选择特定键
选择需要挂载到容器的特定键。
proxy.json
proxy.json
添加
取消
上一步
下一步

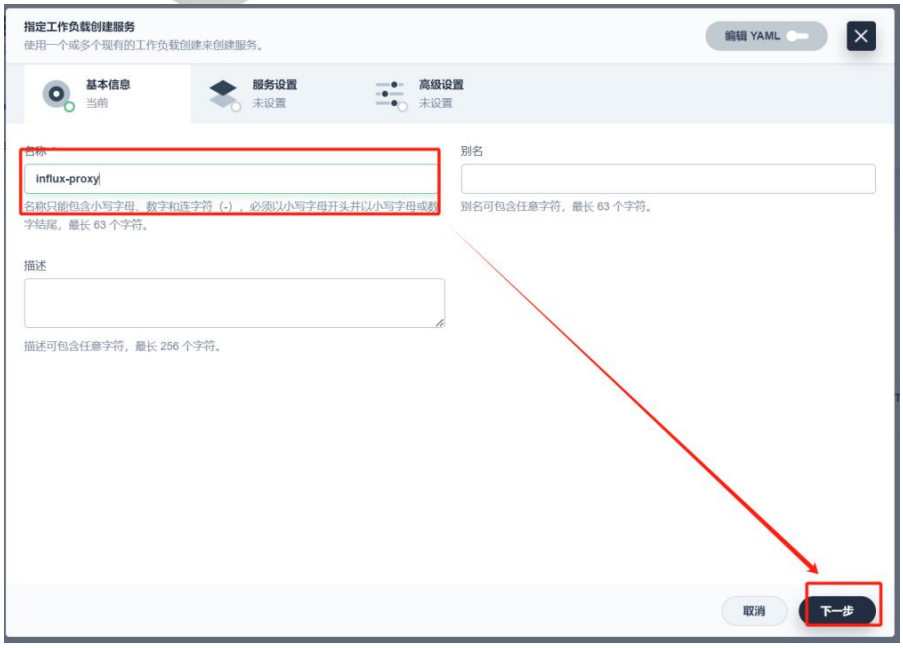




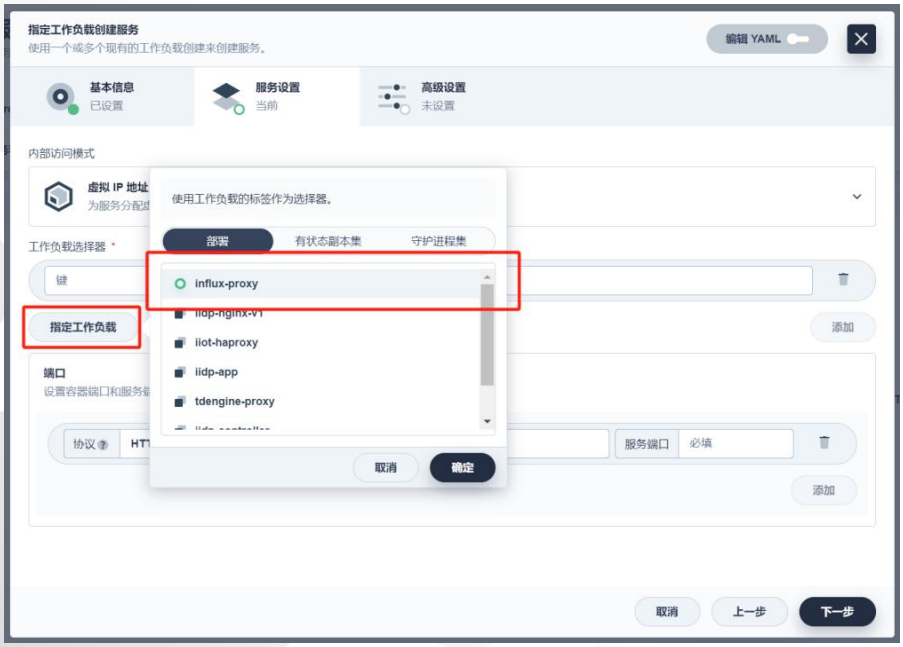
- 登录集群管理平台（kubesphere），在相应的项目里，点击【应用负载】-->【服务】，点击【创建】，再点击【指定工作负载】。

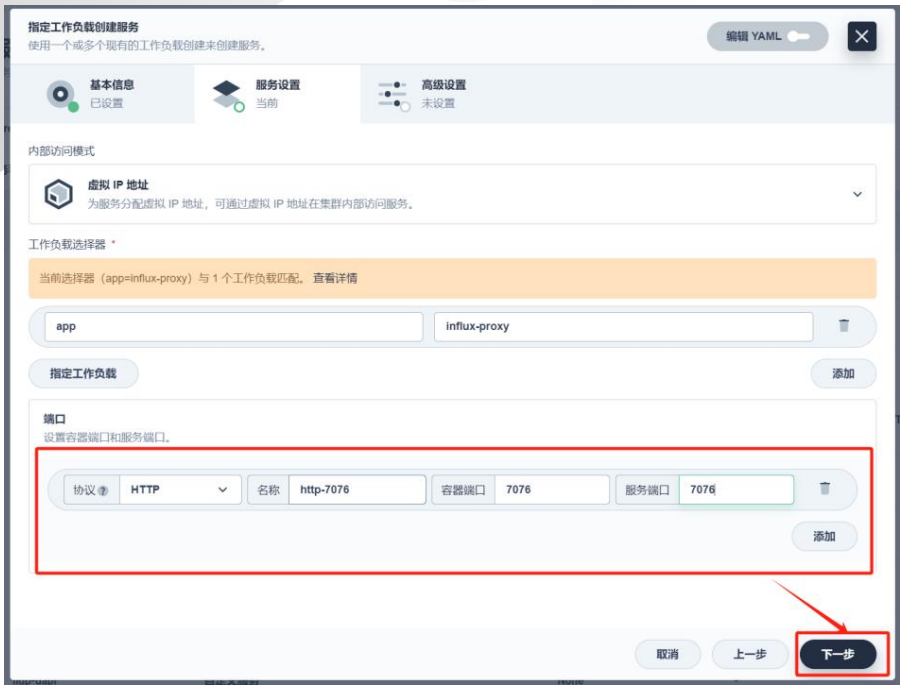


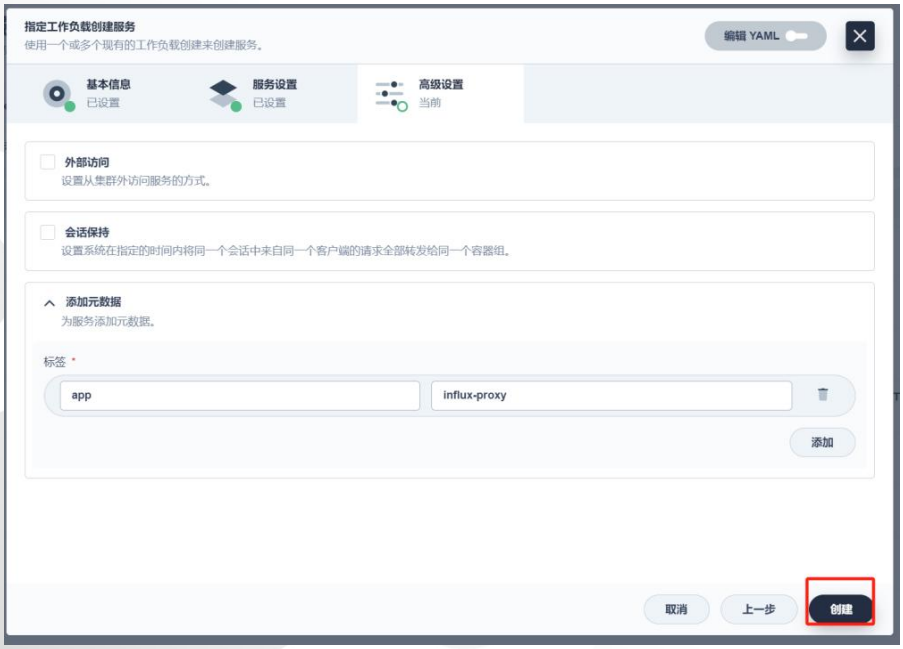
➢ 弹框界面填写名称“influx-proxy”，点击【下一步】。



➢ 点击【指定工作负载】，选择“influx-proxy”，按以下填写端口，点击【创建】。


指定工作负载创建服务
使用一个或多个现有的工作负载创建来创建服务。
编辑 YAML
基本信息
已设置
服务设置
当前
高级设置
未设置
内部访问模式
虚拟 IP 地址
为服务分配虚
使用工作负载的标签作为选择器。
部署
有状态副本集
守护进程集
工作负载选择器 *
键
influx-proxy
iiot-haproxy
iidp-app
tdengine-proxy
指定工作负载
添加
端口
协议
服务端口
必填
添加
取消
确定
取消
上一步
下一步


指定工作负载创建服务
使用一个或多个现有的工作负载创建来创建服务。
编辑 YAML
基本信息
已设置
服务设置
当前
高级设置
未设置
内部访问模式
虚拟 IP 地址
为服务分配虚拟 IP 地址，可通过虚拟 IP 地址在集群内部访问服务。
工作负载选择器 *
当前选择器（app=influx-proxy）与 1 个工作负载匹配。查看详情
app
influx-proxy
指定工作负载
添加
端口
设置容器端口和服务端口。
协议
HTTP
名称
http-7076
容器端口
7076
服务端口
7076
添加
取消
上一步
下一步



3. 部署 IIOT 服务

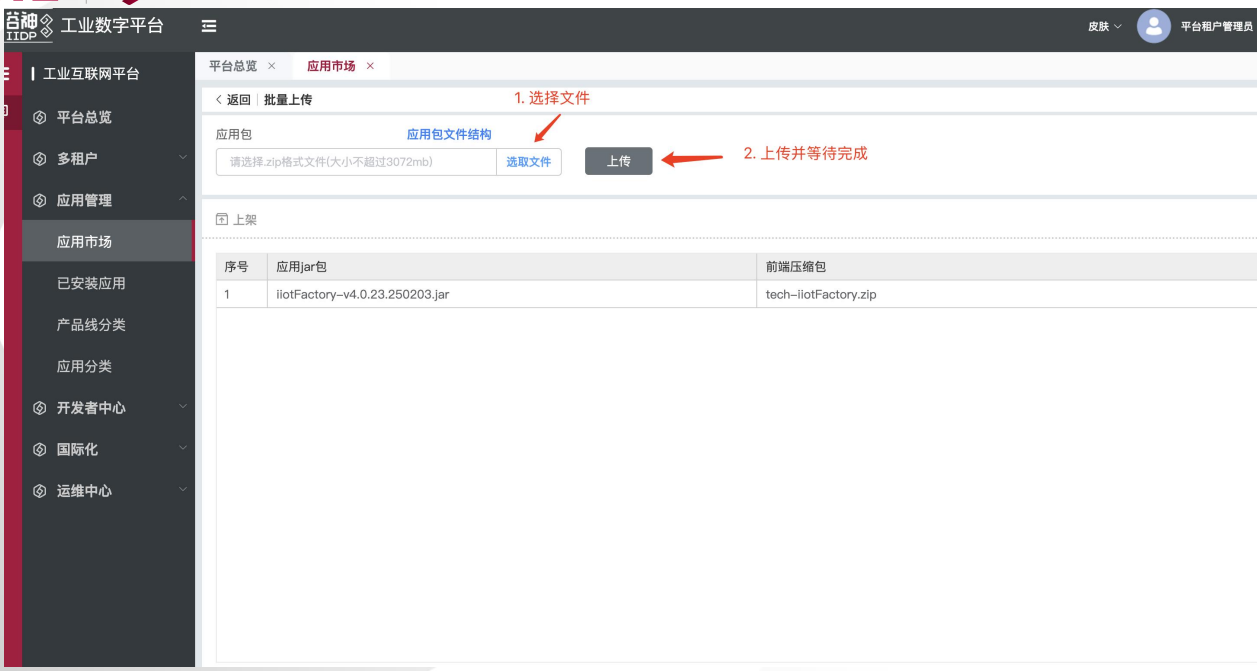
3.1. 单机版

3.1.1. 上传 App

使用平台租户管理员账号登录 IIDP 系统，菜单选择**【应用管理】 -->【应用市场】**，点击**【批量上传】**上传项目所需 App（第 2 章已经在 SMDC 官网下载的 zip 包文件）。

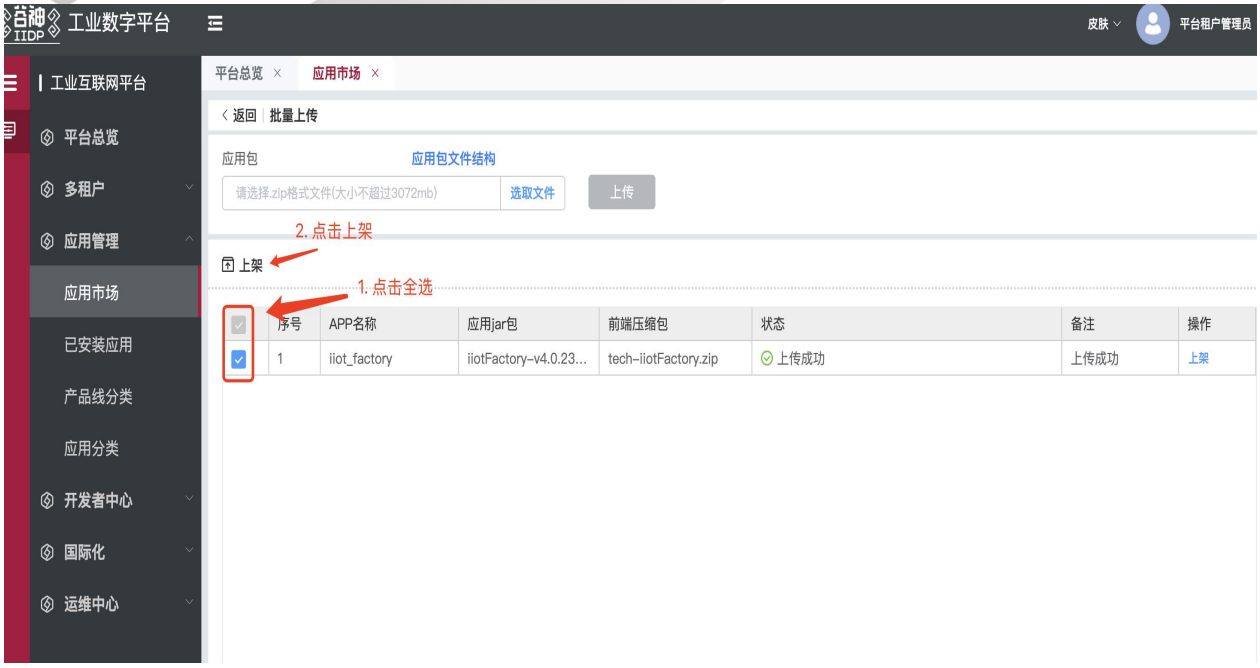
注意：在部署前，请针对项目需要规划需要安装的 App。需注意每个 App 是单独文件夹，App 文件夹若存在 zip 前端包，则需在上架当前 App 时，同步上传 zip 前端包。针对所部署的时序数据库，选择相应的时序数据库 App，具体信息如下：

- InfluxDB：需安装 iiot_store。
- TDengine：需安装 iiot_tdstore。



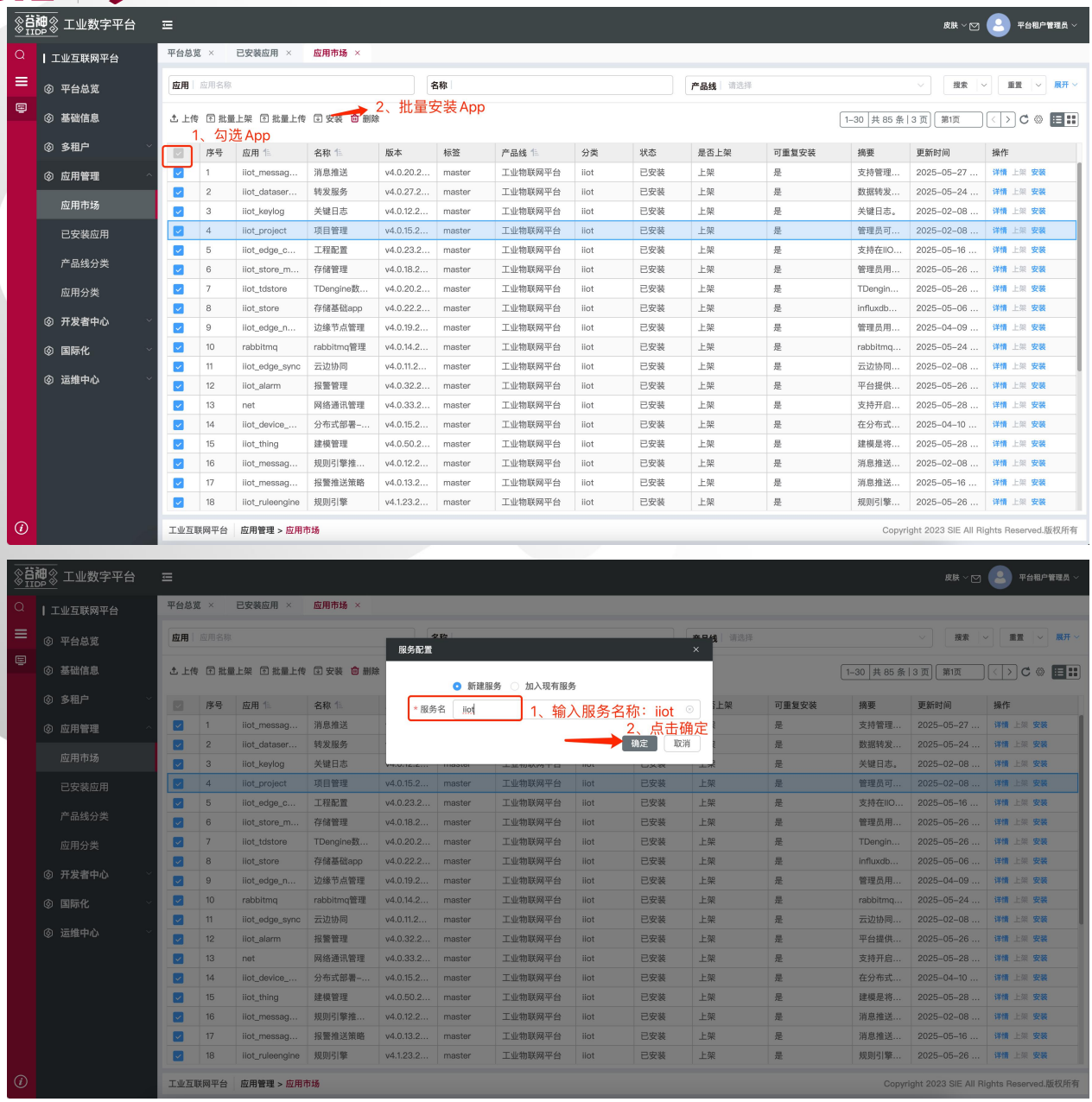
3.1.2. 上架 App

等待 App 上传结束，在页面勾选相应的 App，点击【上架】上架所选 App。



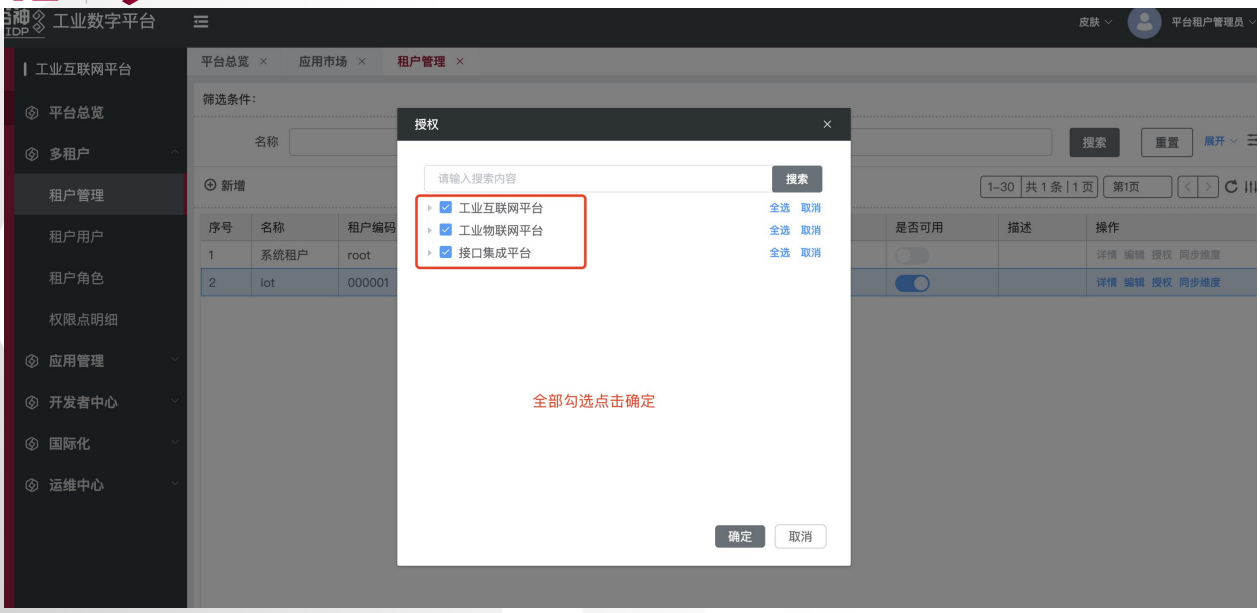
3.1.3. 安装 App

在【应用管理】-->【应用市场】页面，勾选需要安装的 App，点击【安装】，将 App 安装至 iiot 服务。



3.1.4. 授权租户管理账号

在【多租户】-->【租户管理】页面，在租户管理员账号行，点击【授权】，勾选工业物联网全部 App，点击【确定】完成授权。



3.1.5. 安装 TDengineProxy 代理

- 参考 2.2.2 依赖组件表，下载 TDengineProxy 服务镜像文件 tdengine-proxy.tar 并上传到目标服务器目录 /root。
- 登录目标服务器，执行以下命令将 TDengineProxy 服务镜像导入 docker 镜像仓库（version 表示实际版本号），创建安装目录（**假设数据盘挂载目录是：/var**）。

```
docker load < /root/tdengine-proxy-{version}.tar
mkdir -p /var/tdengine-proxy
mkdir -p /var/tdengine-proxy/conf
```

下载 TDengineProxy 服务配置文件，上传到目标服务器目录/var/tdengine-proxy/conf，根据实际服务器 IP 修改配置文件 DataSource 条目。其中，Host 参数值必须是 tdengine 容器的宿主机 IP。

```
{
  "DataSource":
  [
    {
      "Host": "172.20.194.35",
      "Port": 6041,
      "Database": "sie_iiot",
      "Username": "root",
      "Password": "taosdata"
    }
  ],
  "BatchSize": 5000,
  "IsDebug": false
}
```

- 下载 TDengineProxy 服务启动脚本，上传到目标服务器目录/var/tdengine-proxy/，根据实际镜像版本号修改启动脚本版本号。

```
version: "3"

services:
  td-proxy-standalone:
    image: harbor.devcenter.gushen.sieiot.com/iiot/tdengine-proxy:1.3.15
    hostname: td-proxy-standalone
    container_name: td-proxy-standalone
    restart: always
    privileged: true
    volumes:
      - ./conf/appsettings.json:/app/appsettings.json
    ports:
      - 8080:80
```

- 登录目标服务器，执行以下命令启动 TDengineProxy 服务。

```
docker-compose -f /var/tdengine-proxy/tdproxy-standalone-compose.yml up -d
```

- 验证 TDengineProxy 服务启动。

```
[root@td-3 tdengine-proxy]# docker ps | grep td-proxy-standalone
c369a1f55065   harbor.devcenter.gushen.sieiot.com/iiot/tdengine-proxy:1.3.15   "dotnet SIE.IIOT.TDe…"   26 seconds ago   Up 25 seconds   0.0.0.0:8080->80/tcp, :::
8080->80/tcp                                                                   td-proxy-standalone
[root@td-3 tdengine-proxy]# docker logs td-proxy-standalone
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app/
```

3.1.6. 系统配置

使用租户管理员账号登录 IIOT 系统，在【系统运维】-->【系统配置】页面，修改以下配置项。

表 3-1 系统配置表

配置键	说明	示例

server.ip	可以访问平台的 ip 地址或域名地址	192.168.175.192
server.api	可以访问平台 api 接口的 url 地址	http://192.168.175.192:30666/api/root/master
server.websocket	对外开放的 websocket 地址	ws://192.168.175.192:8888/ws
server.mqtt	对外开放的 mqtt 地址	192.168.175.192:1883
server.http	服务 iiot 向外暴露的 http 地址（按需）	http://192.168.174.228:30134
server.coap	服务 iiot 向外暴露的 coap 地址（按需）	coap://192.168.174.228:30809

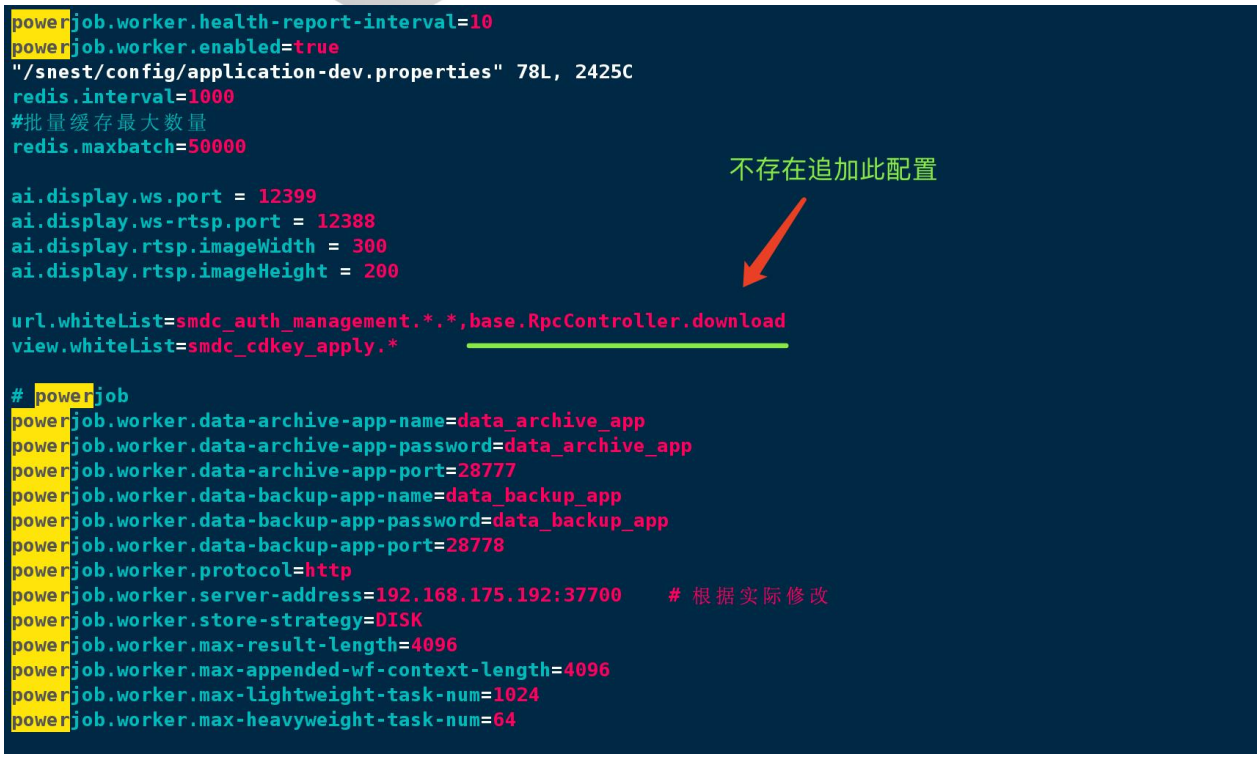
以上配置中涉及的 ip 修改为部署服务器的 ip，端口取章节【端口开放清单】准备的信息。

3.1.7. 环境配置

3.1.7.1. 白名单配置

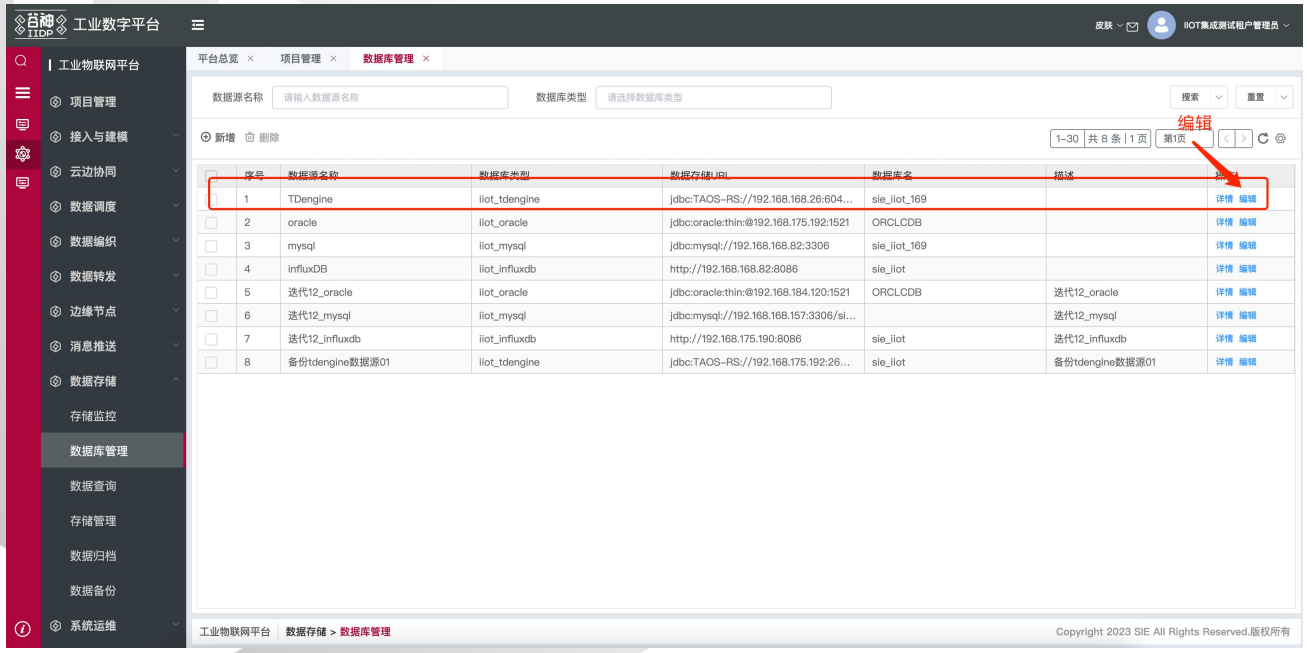
登录 IIOT 部署服务器，检查平台配置文件: /snest/config/application-dev.properties（默认该路径，项目具体路径根据部署情况修改）中是否存在如下配置，如果不存在，需添加并重启 snest 容器。

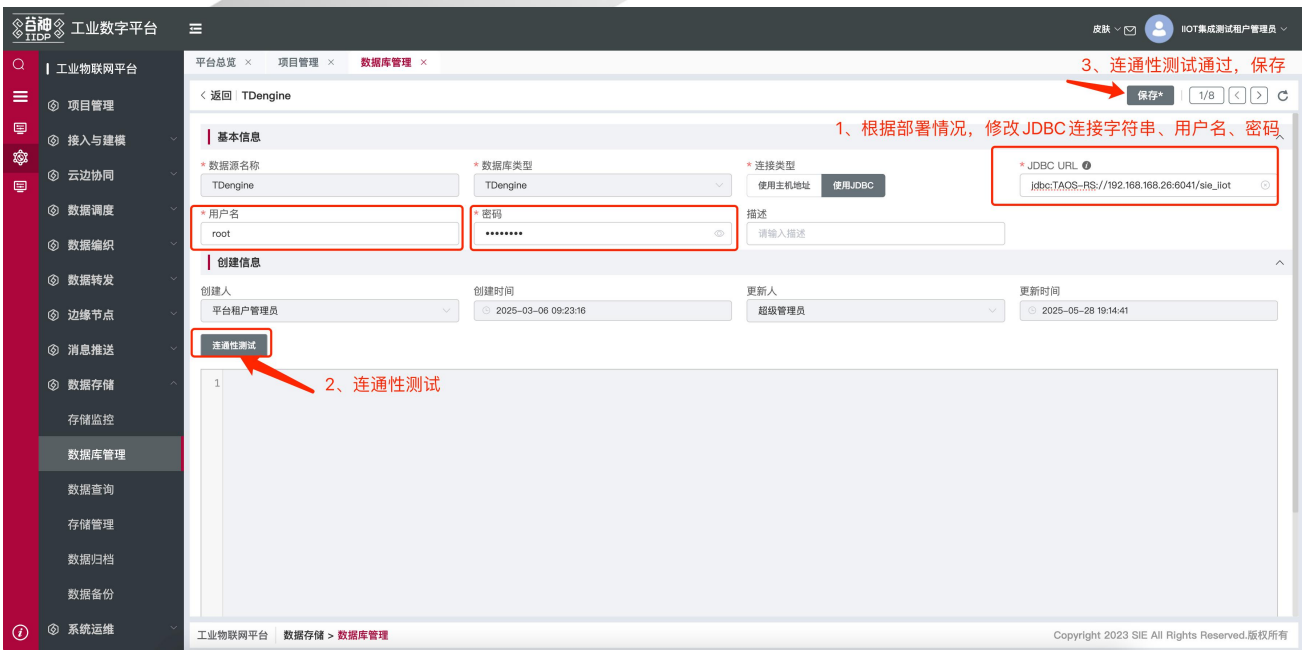
```
# 白名单
url.whiteList=base.RpcController.download
```

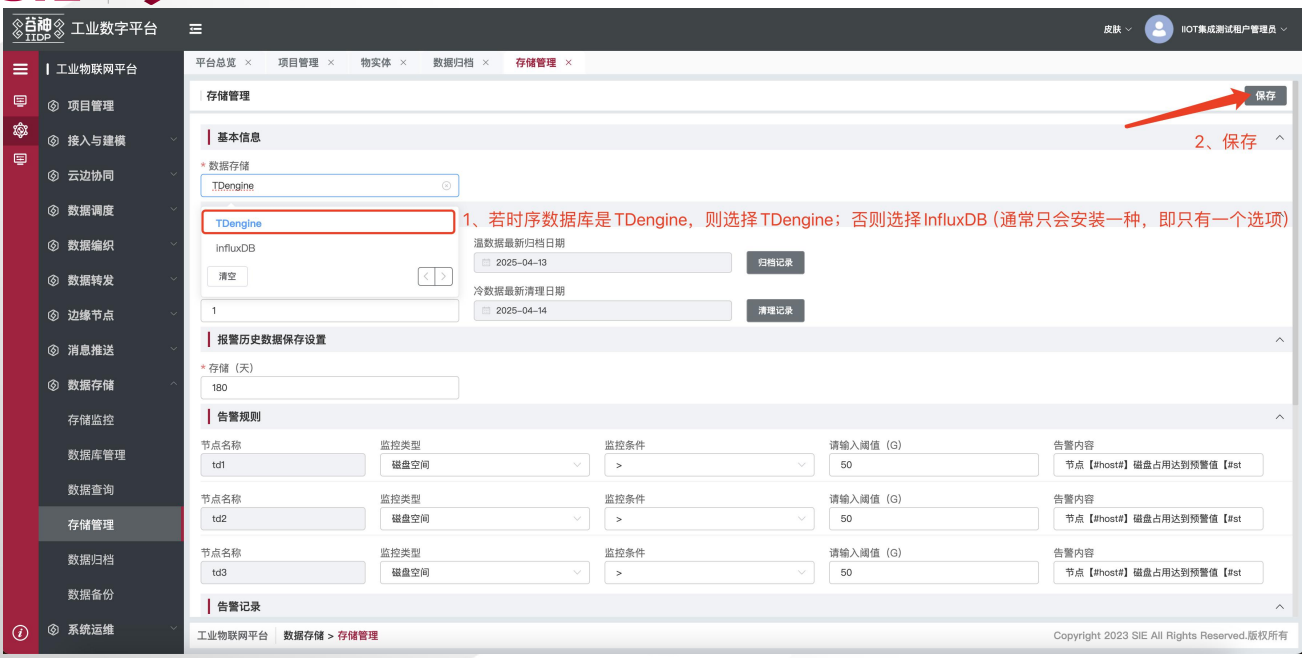


3.1.8. 存储配置

- 使用租户管理员账号登录 IIOT 系统，在【数据存储】-->【数据库管理】页面，修改时序数据库的信息。







3.2. 分布式版

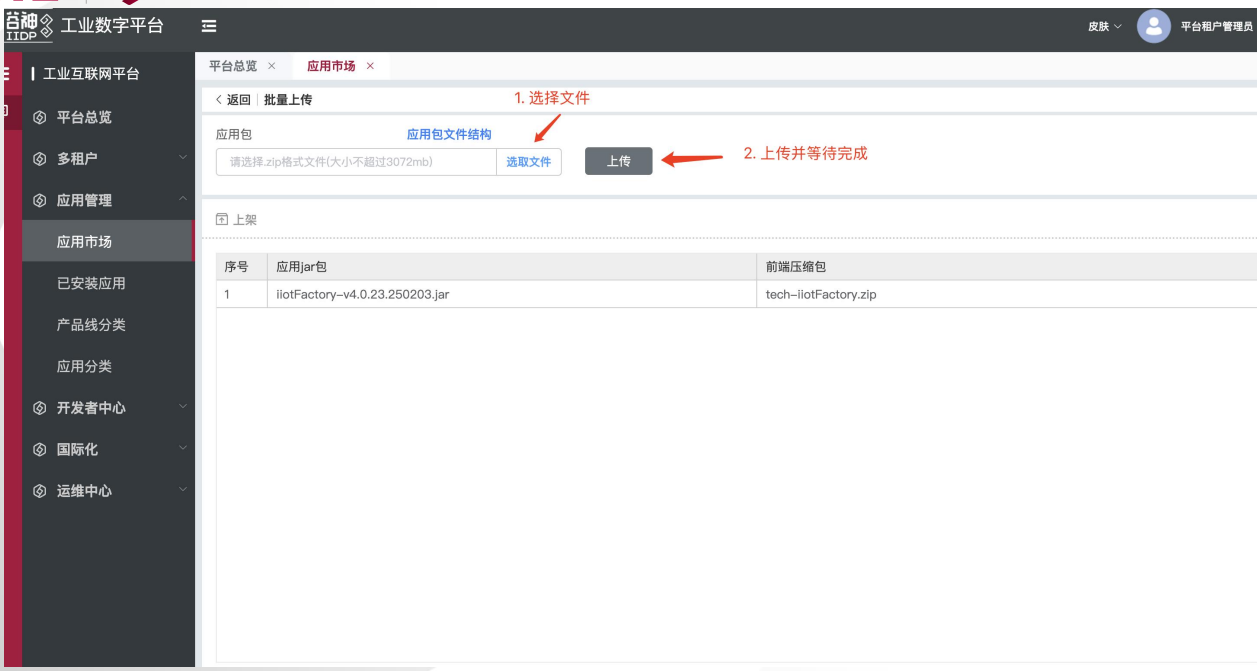
3.2.1. 上传 App

使用平台超级管理员账号（默认账号：implementer）登录 IIDP 系统，菜单选择**【应用管理】→【应用市场】**，点击**【批量上传】**上传项目所需 App（第 2 章已经在 SMDC 官网下载的 zip 包文件）。

注意：部署前请针对项目需要规划需要安装的 App。需注意每个 App 是单独文件夹，App 文件夹若存在 zip 前端包，则需在上架当前 App 时，同步上传 zip 前端包。

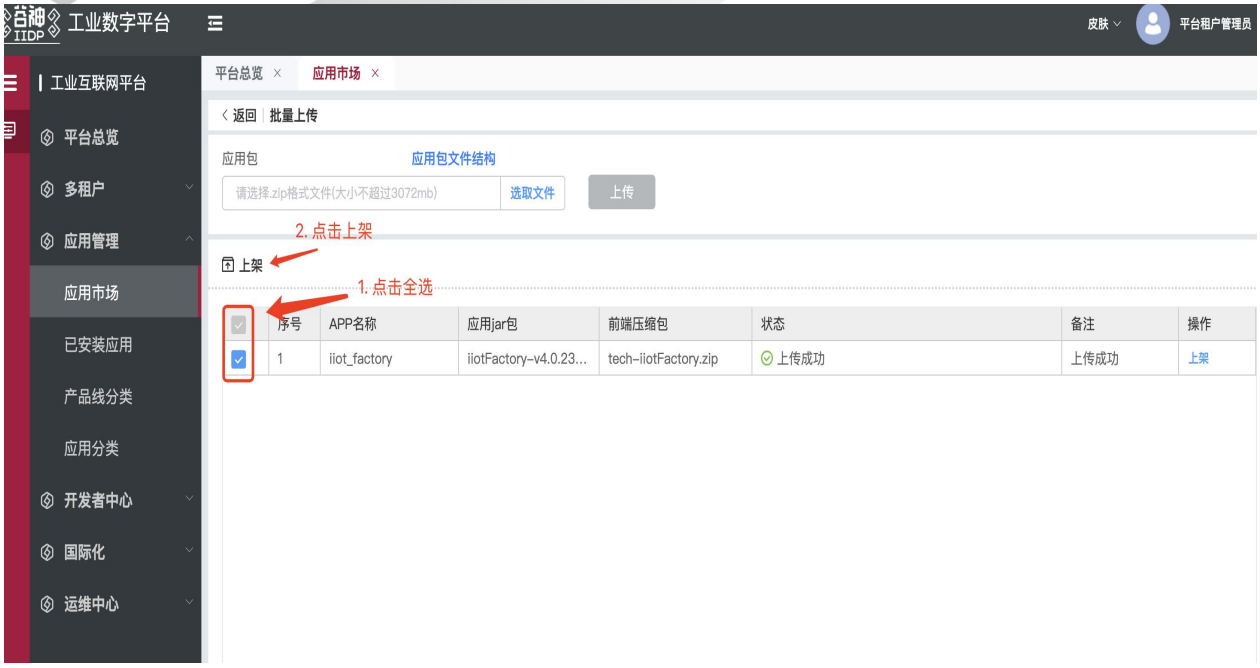
针对所部署的时序数据库，选择相应的时序数据库 App，具体信息如下：

- InfluxDB：对应安装 iiot_store。
- TDengine：对应安装 iiot_tdstore。



3.2.2. 上架 App

等待 App 上传完成，在页面勾选相应的 App，点击【上架】上架所选 App。



3.2.3. 安装 App

强制单独安装服务的 App 有：

- iiot_cluster_manager

- iiot_dataservice_opcua、iiot_dataservice_api（共同安装至同一服务）

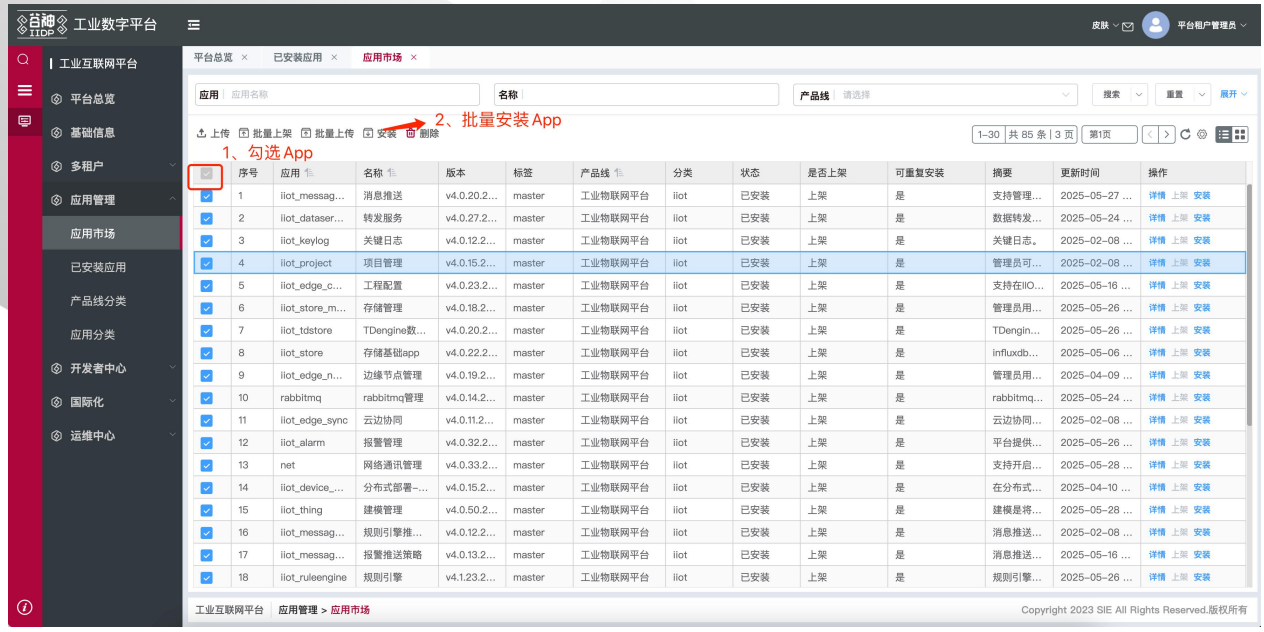
建议单独安装服务的 App 有：

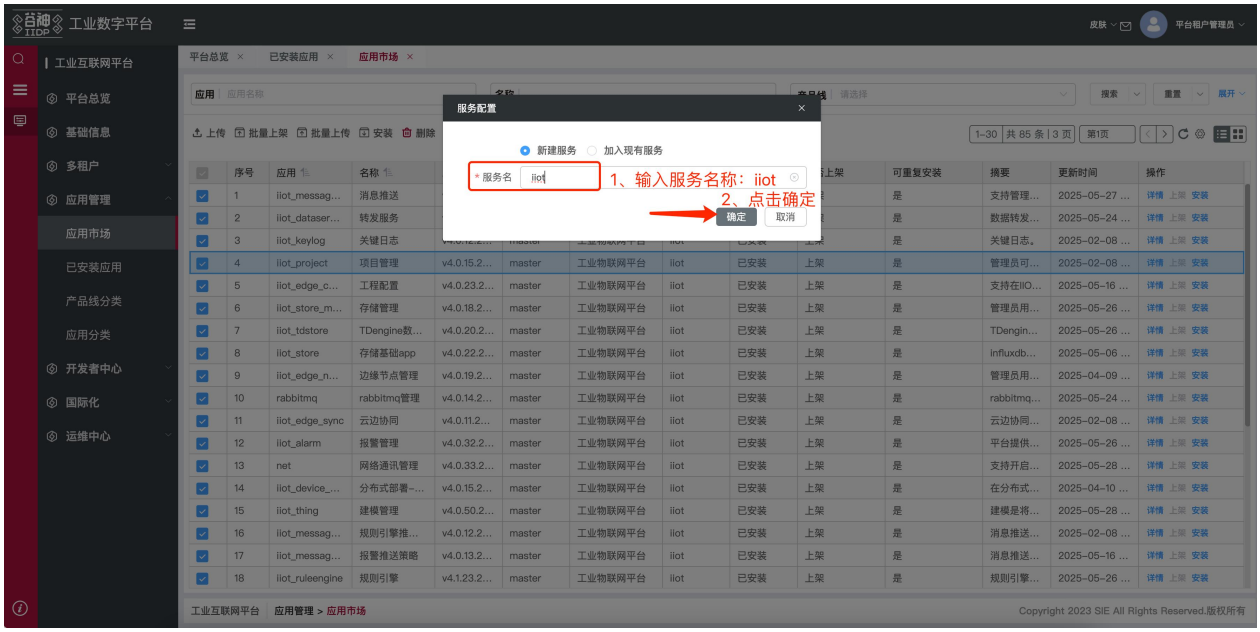
- iiot_data_archive
- iiot_data_backup

其它 App 安装至 iot 服务。

3.2.3.1. 安装 iot 服务

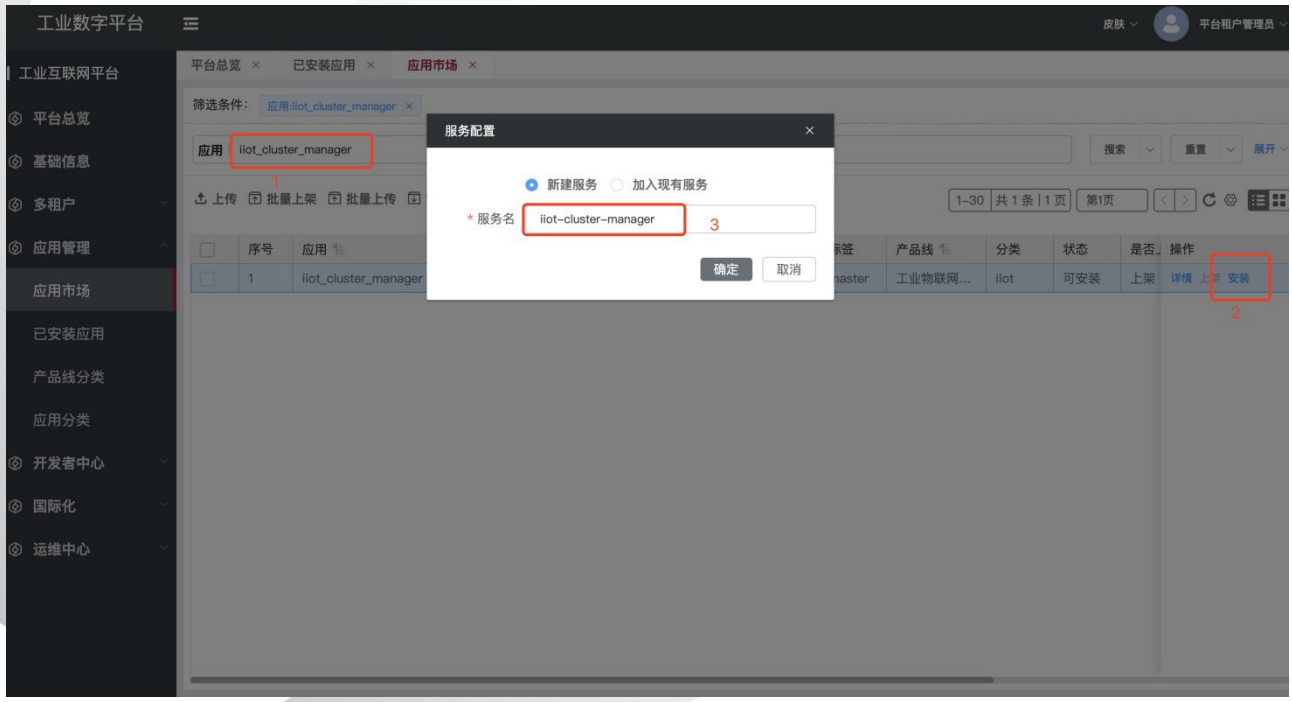
在【应用管理】-->【应用市场】页面，勾选 iot 服务需要安装的 App，点击【安装】，将 App 安装至 iiot1 服务（若需安装多个 iot 服务，则依次重复该步骤，服务名称：iiot2、iiot3、...）。





3.2.3.2. 安装 iiot_cluster_manager 服务

在【应用管理】→【应用市场】页面，勾选【iiot_cluster_manager】App，点击【安装】，将 App 安装至 iiot_cluster_manager 服务）。

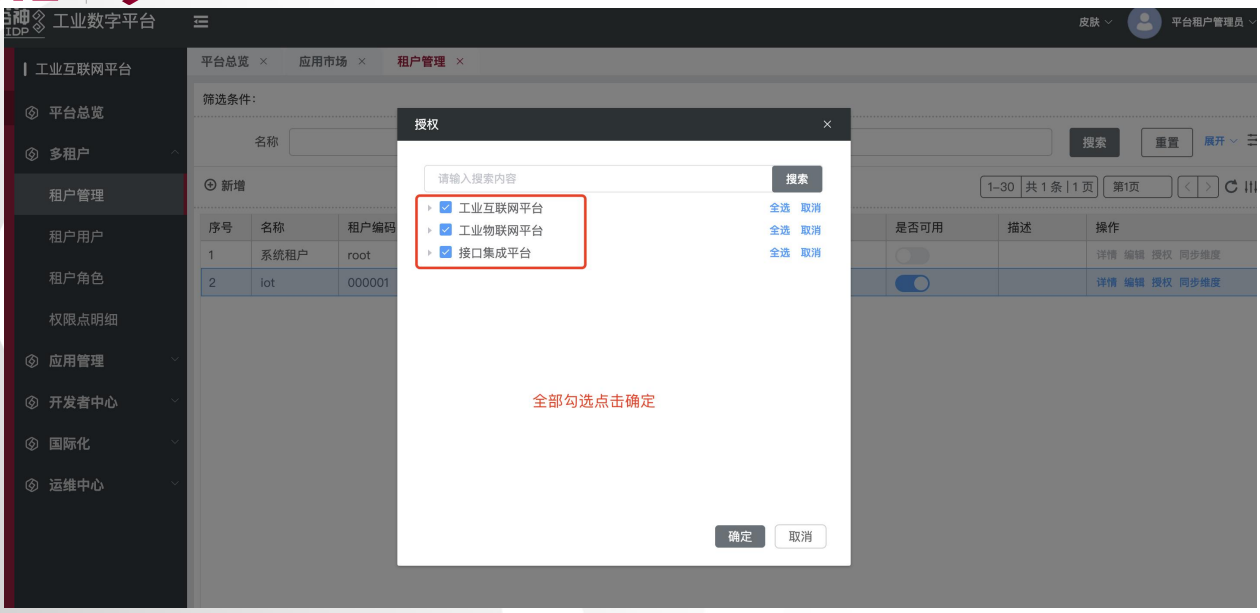


3.2.3.3. 安装 opcua 服务

在【应用管理】→【应用市场】页面，勾选【iiot_dataservice_opcua、iiot_dataservice_api】App，点击【安装】，将 App 安装至 opcua 服务。

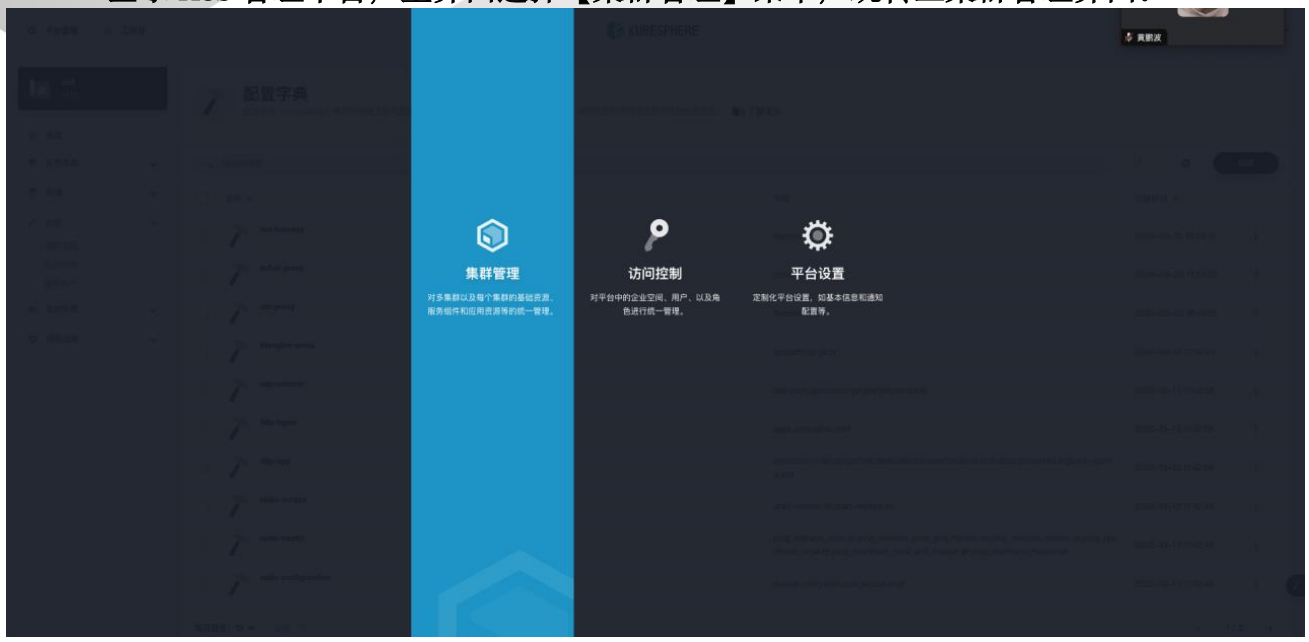
3.2.4. 授权租户管理账号

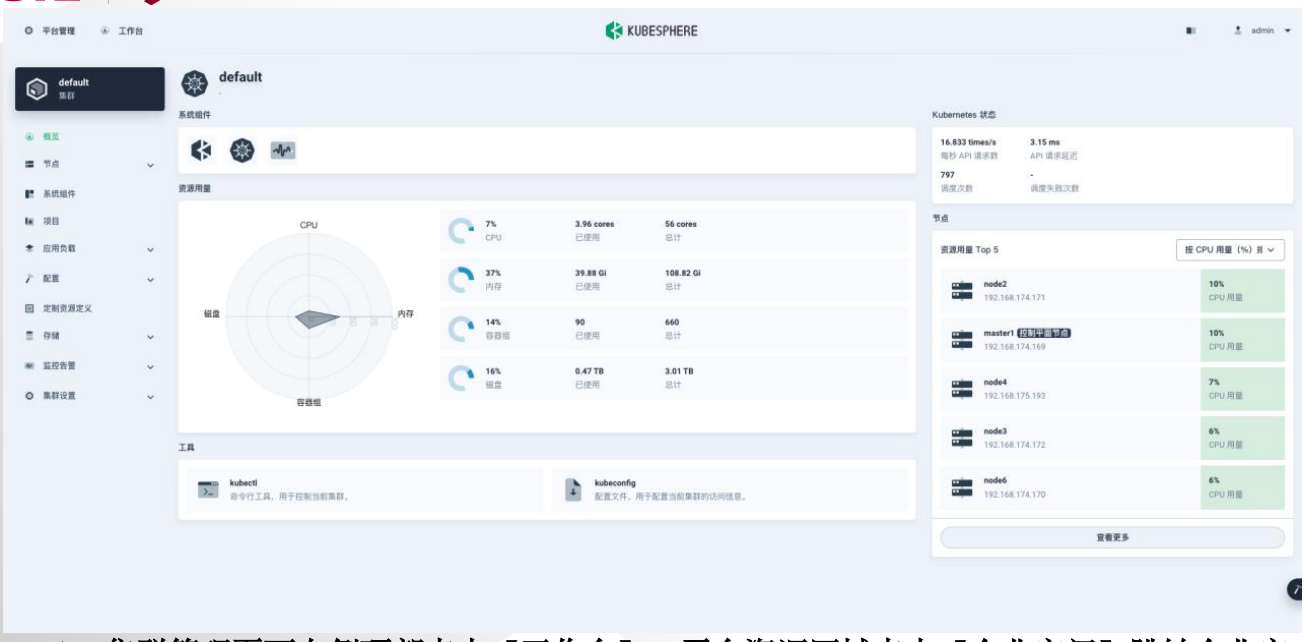
在【多租户】-->【租户管理】页面，在租户管理员账号行，点击【授权】，勾选工业物联网全部 App，点击【确定】完成授权。



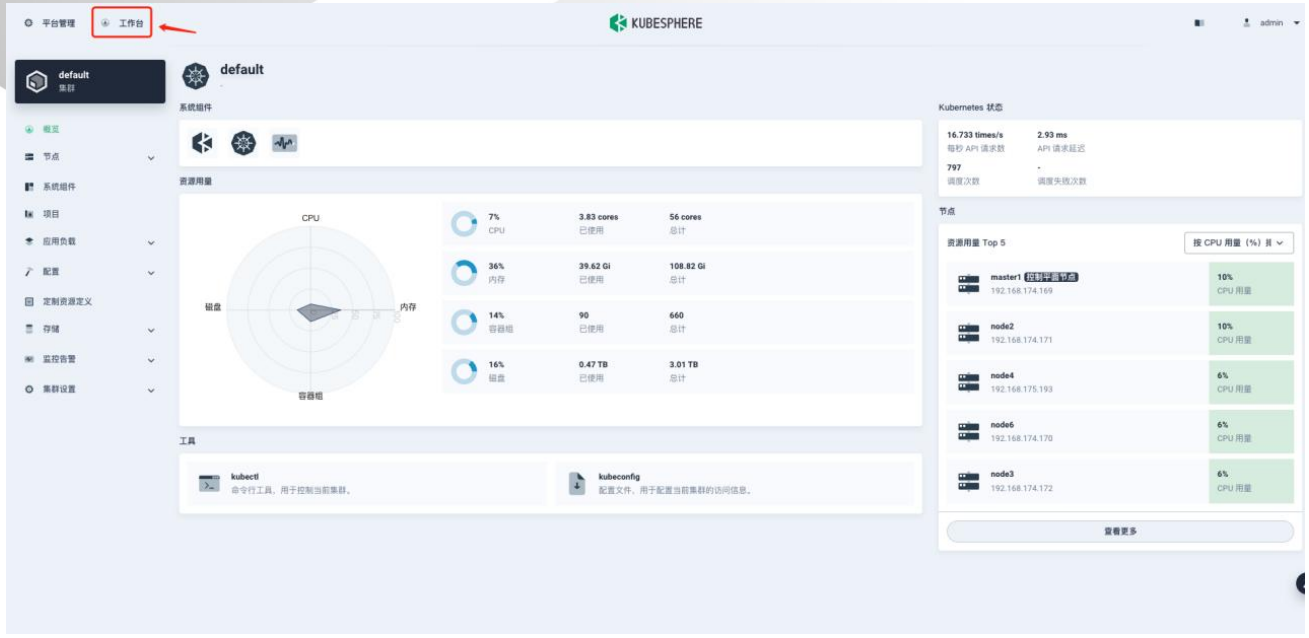
3.2.5. 安装 TDengineProxy 代理

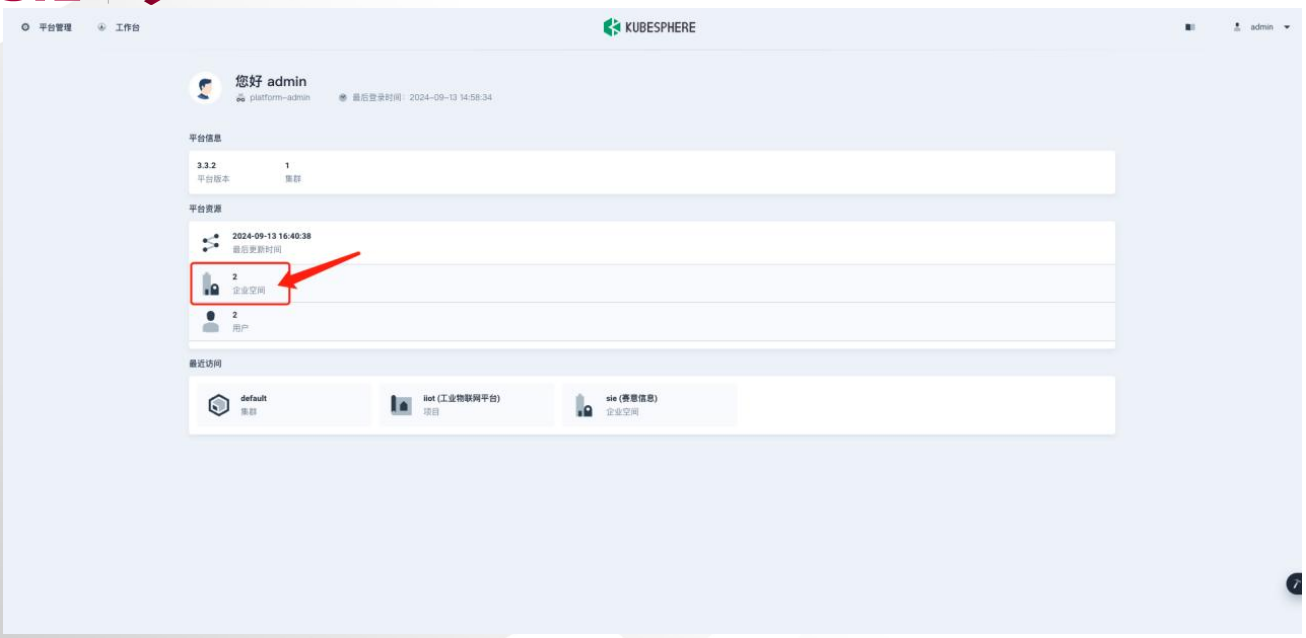
- 登录 K8S 管理平台，主界面选择【集群管理】菜单，跳转至集群管理界面。

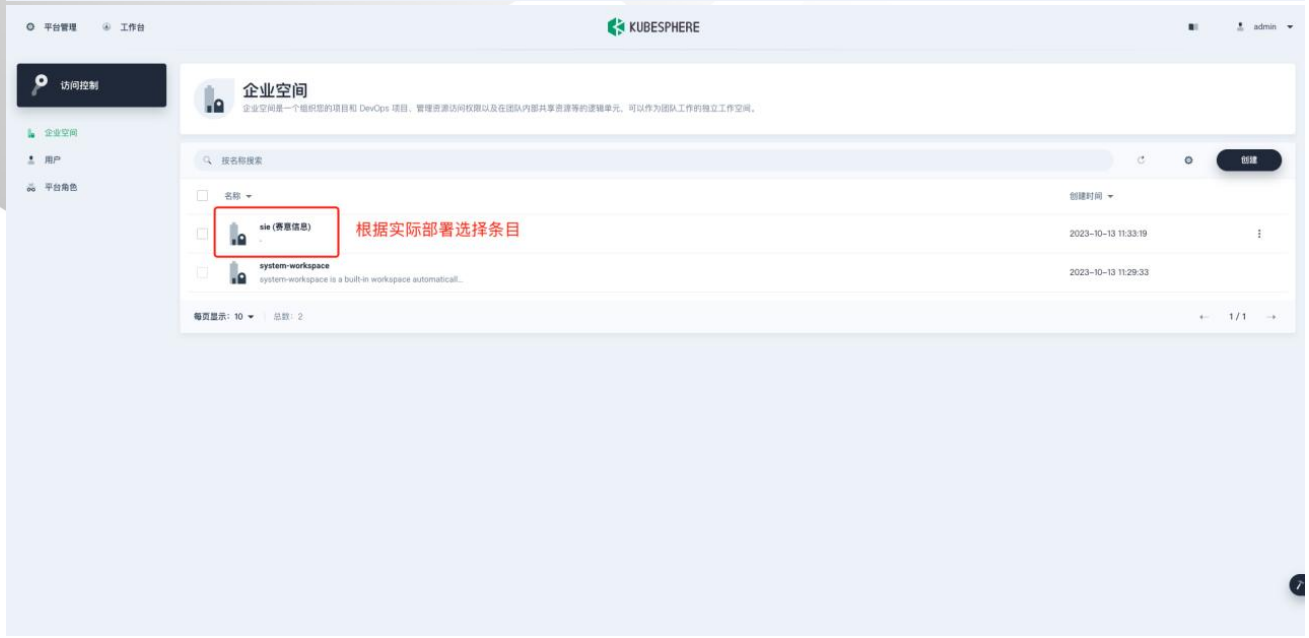


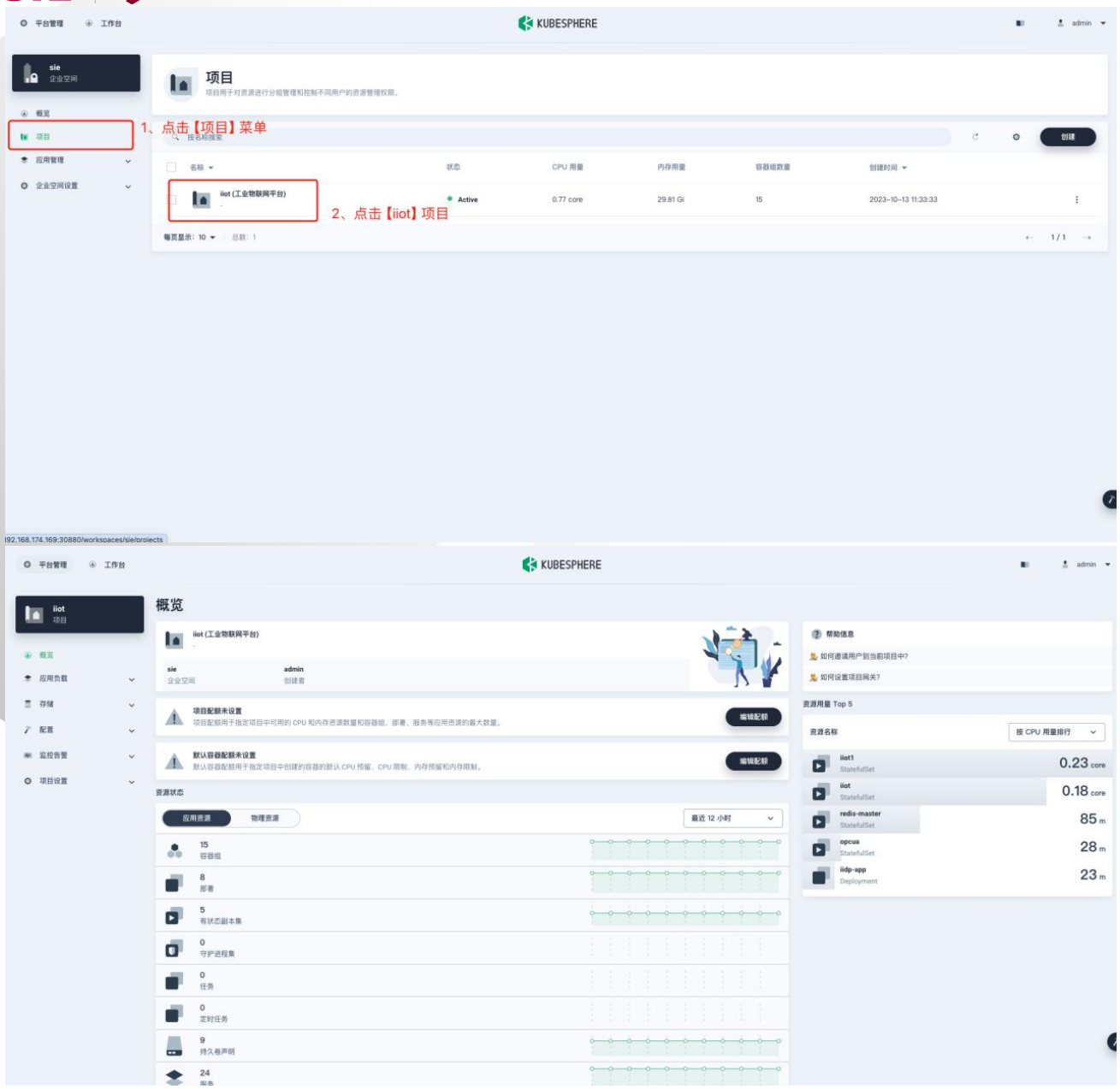


- 集群管理页面左侧顶部点击【工作台】，平台资源区域点击【企业空间】跳转企业空间页面，然后在中间区域选择【相应的企业空间】，接着在相应企业空间页面左侧选择【项目】菜单，再在中间区域选择【iiot】项目，跳转至项目管理页面。

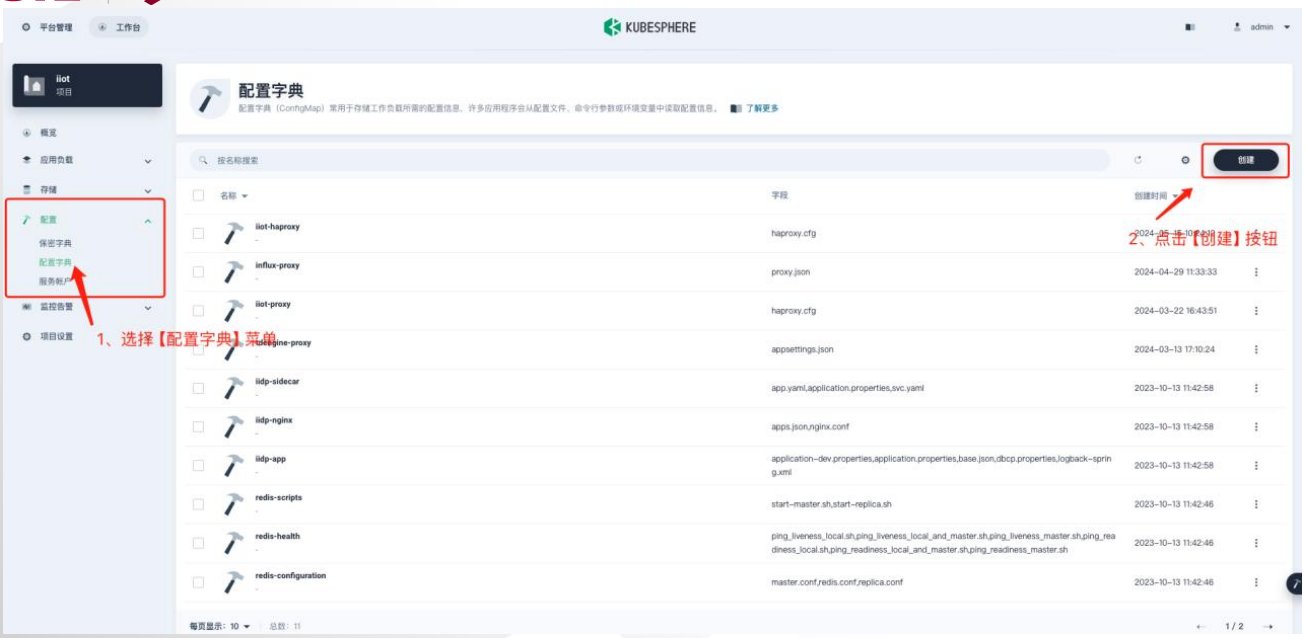


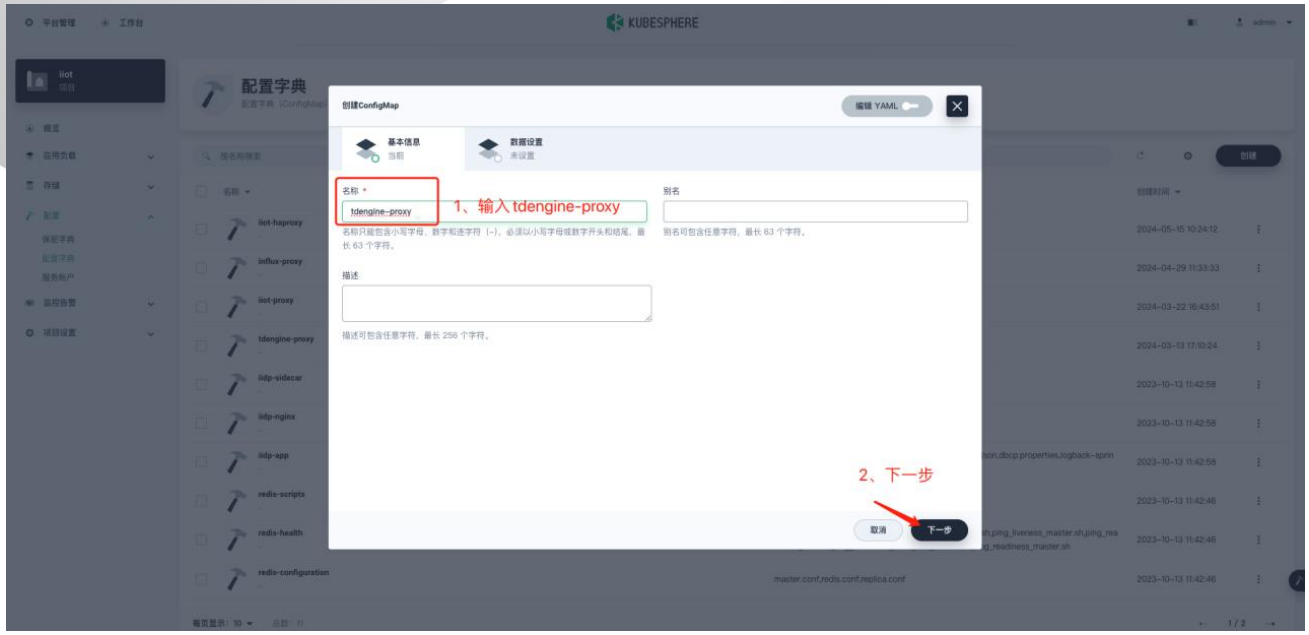

KUBESPHERE
admin
您好 admin
platform-admin
平台信息
3.3.2
平台版本
1
集群
平台资源
2024-09-13 16:40:38
最后更新时间
2
企业空间
2
用户
最近访问
default
集群
iiot (工业物联网平台)
项目
sie (赛意信息)
企业空间

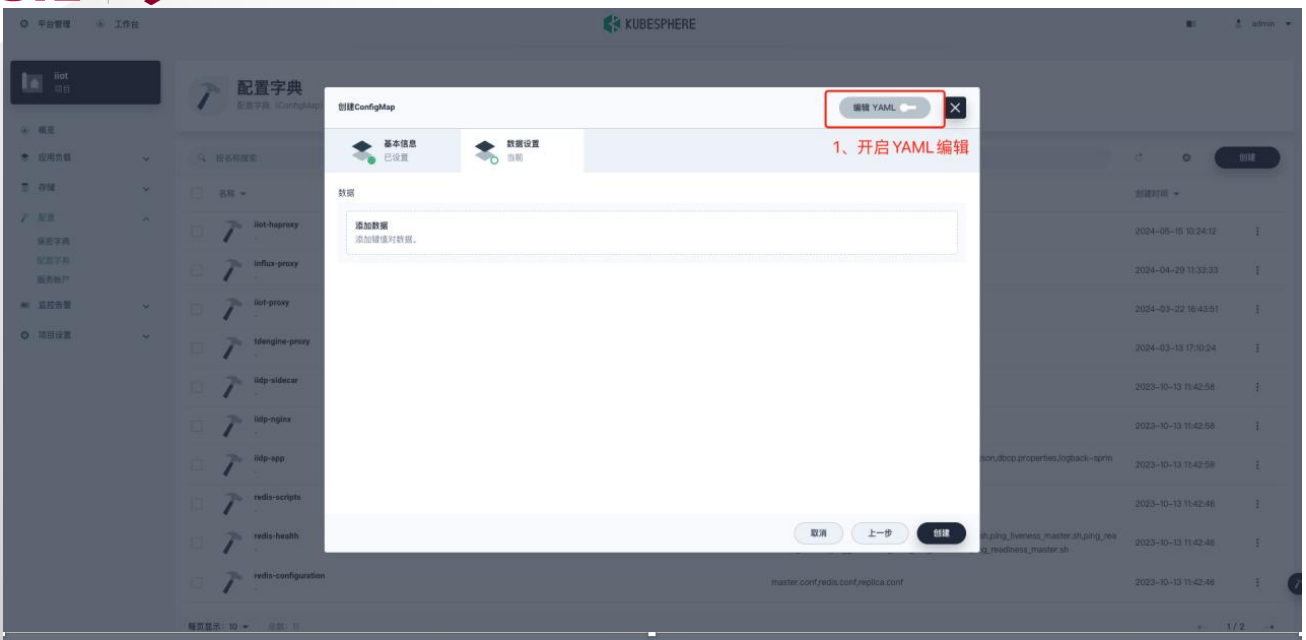

KUBESPHERE
admin
访问控制
企业空间
用户
平台角色
企业空间
sie (赛意信息)
system-workspace
根据实际部署选择条目
2023-10-13 11:33:19
2023-10-13 11:29:33
1 / 1

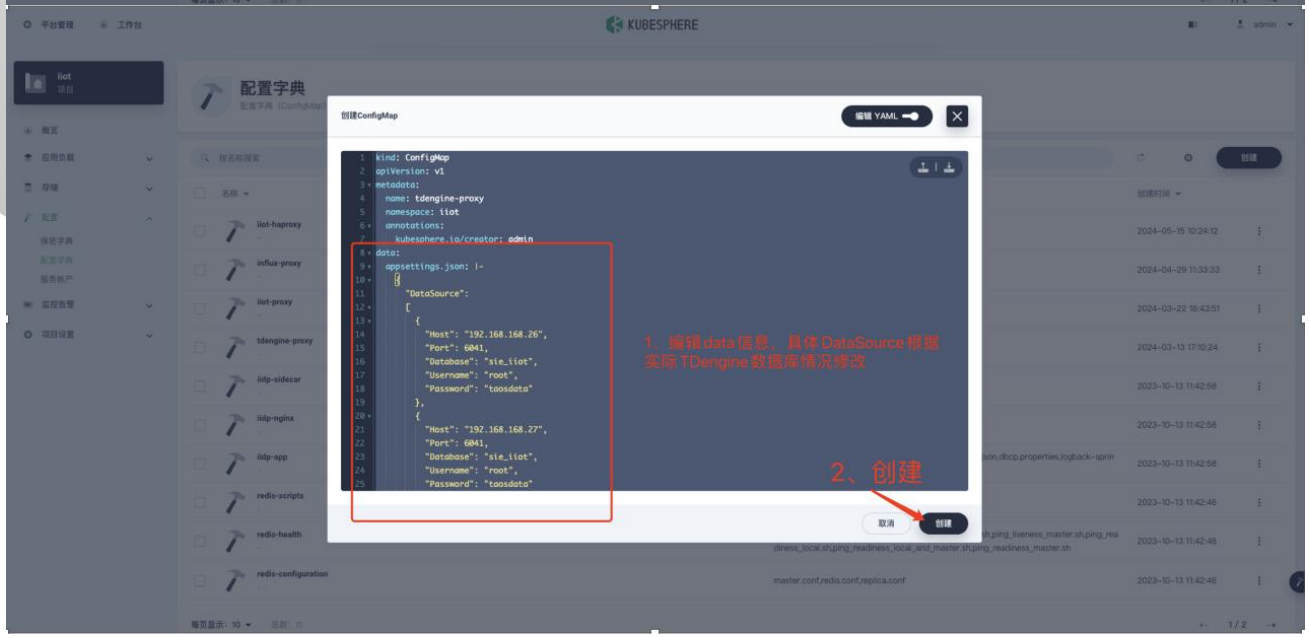


➢ 项目页面左侧选择【配置】->【配置字典】菜单，右上角点击【创建】按钮创建 tdengine-proxy 字典。


KUBESPHERE
配置字典
创建
1、选择【配置字典】菜单
2、点击【创建】按钮


KUBESPHERE
配置字典
创建ConfigMap
编辑 YAML
基本信息
数据设置
名称 *
tdengine-proxy
1、输入 tdengine-proxy
别名
描述
2、下一步
取消
下一步





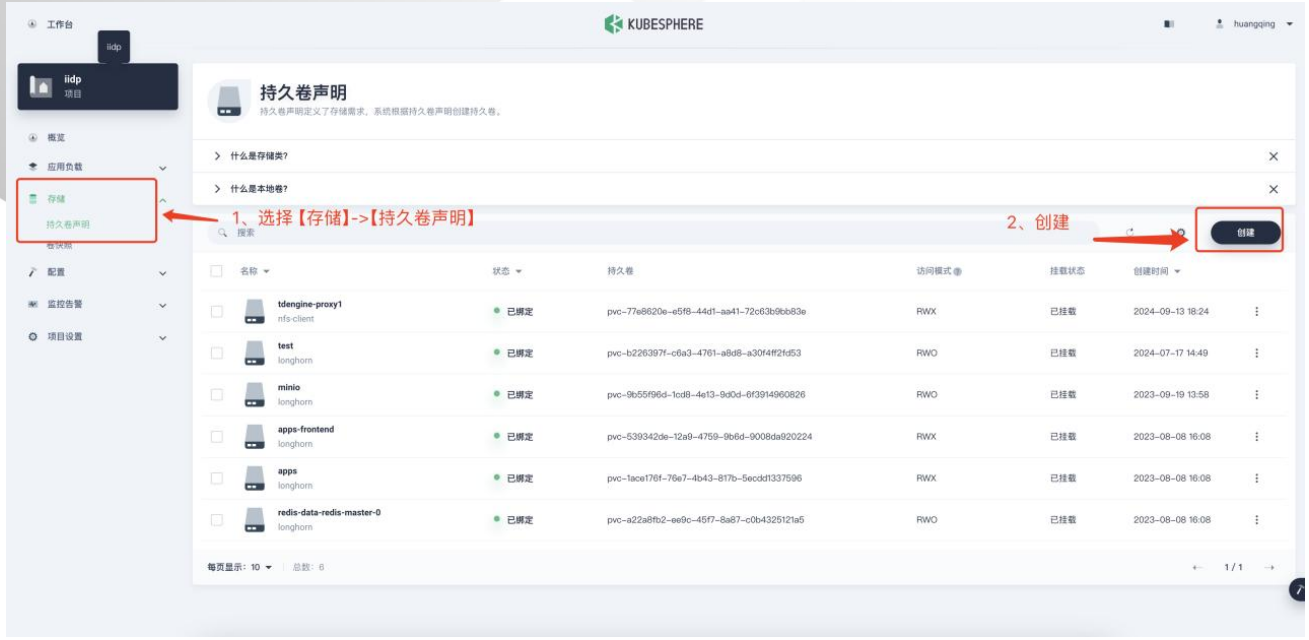
配置文件内容（DataSource 部分按照实际 TDengine 数据库情况修改，其中：Host 参数值必须是各 tdengine 容器的宿主机 IP，BatchSize，IsDebug 必须配置）：

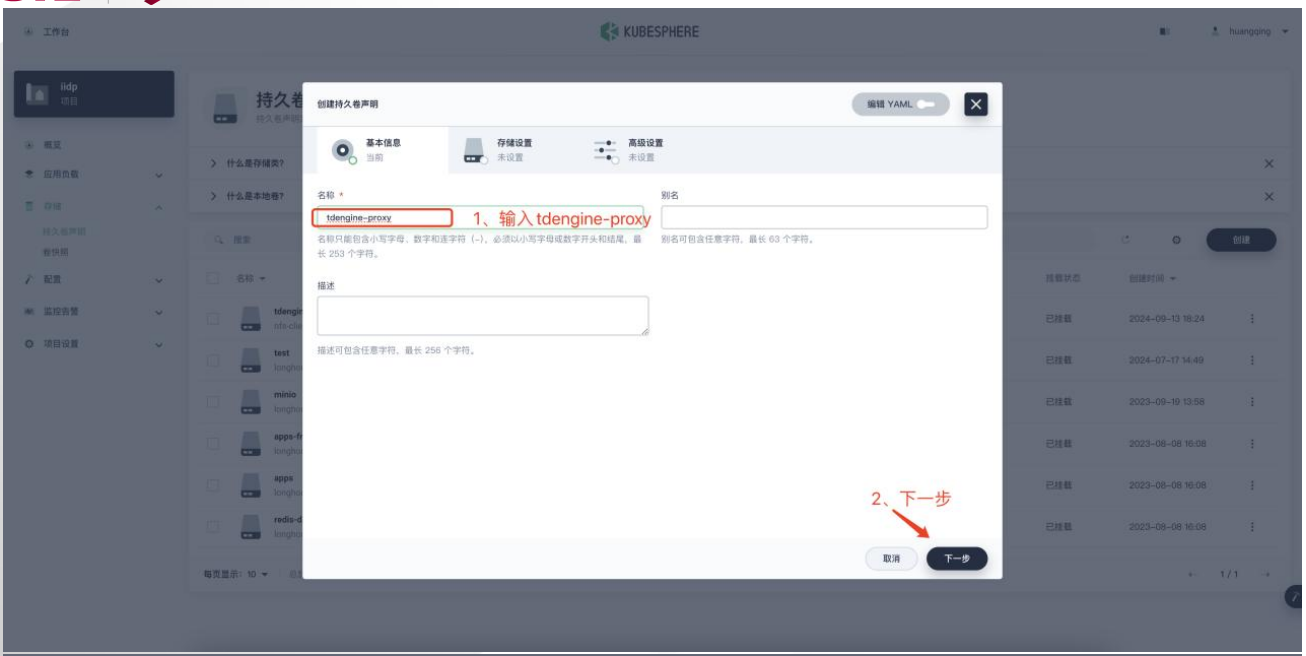
```
kind: ConfigMap
apiVersion: v1
metadata:
  name: tdengine-proxy
  namespace: iiot
  annotations:
    kubesphere.io/creator: admin
data:
  appsettings.json: |-
    {
      "DataSource":
      [
```

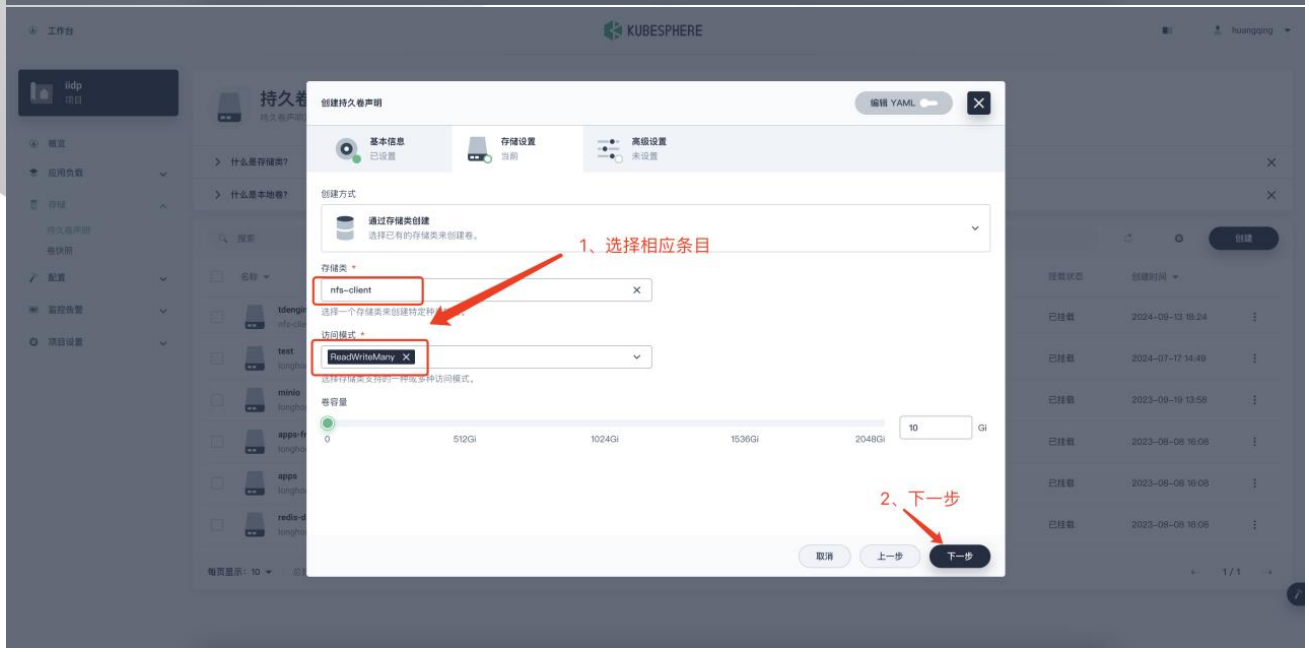
```
        {
          "Host": "192.168.168.26",
          "Port": 6041,
          "Database": "sie_iiot",
          "Username": "root",
          "Password": "taosdata"
        },
        {
          "Host": "192.168.168.27",
          "Port": 6041,
          "Database": "sie_iiot",
          "Username": "root",
          "Password": "taosdata"
        },
        {
          "Host": "192.168.168.28",
          "Port": 6041,
          "Database": "sie_iiot",
          "Username": "root",
          "Password": "taosdata"
        }
      ],
      "BatchSize": 5000,
      "IsDebug": false,
      "Serilog": {
      "MinimumLevel": {
        "Default": "Information",
        "Override": {
          "Microsoft": "Warning",
          "Microsoft.AspNetCore": "Warning",
          "System.Net.Http.HttpClient": "Warning"
        }
      },
      "WriteTo": [
        {
          "Name": "Console"
        },
        {
          "Name": "File",
          "Args": {
            "path": "/app/logs/tdengine-proxy-.log",
            "rollingInterval": "Day",
```

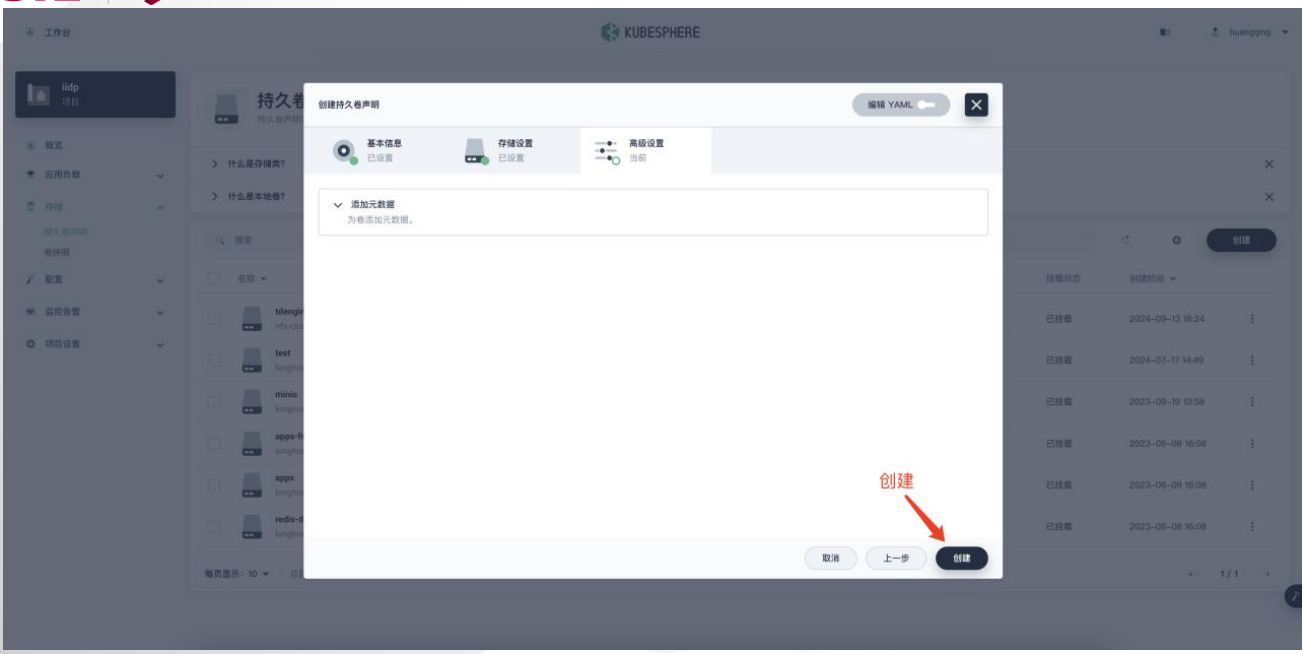
```
            "fileSizeLimitBytes": 104857600,
            "rollOnFileSizeLimit": true,
            "retainedFileCountLimit": 30,
            "shared": true,
            "flushToDiskInterval": "00:00:01"
          }
        }
      ],
      "Enrich": [
        "FromLogContext"
      ]
    }
```

- 项目页面左侧选择【存储】->【持久卷声明】菜单，右上角点击【创建】按钮创建 tdengine-proxy 存储。

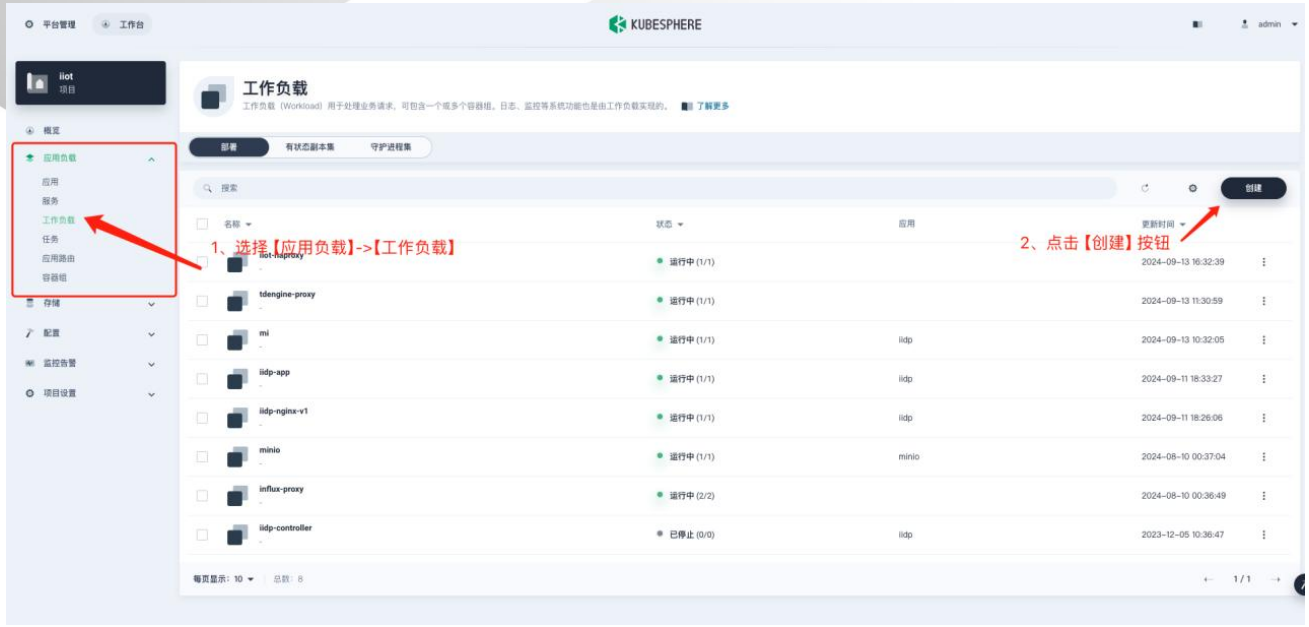


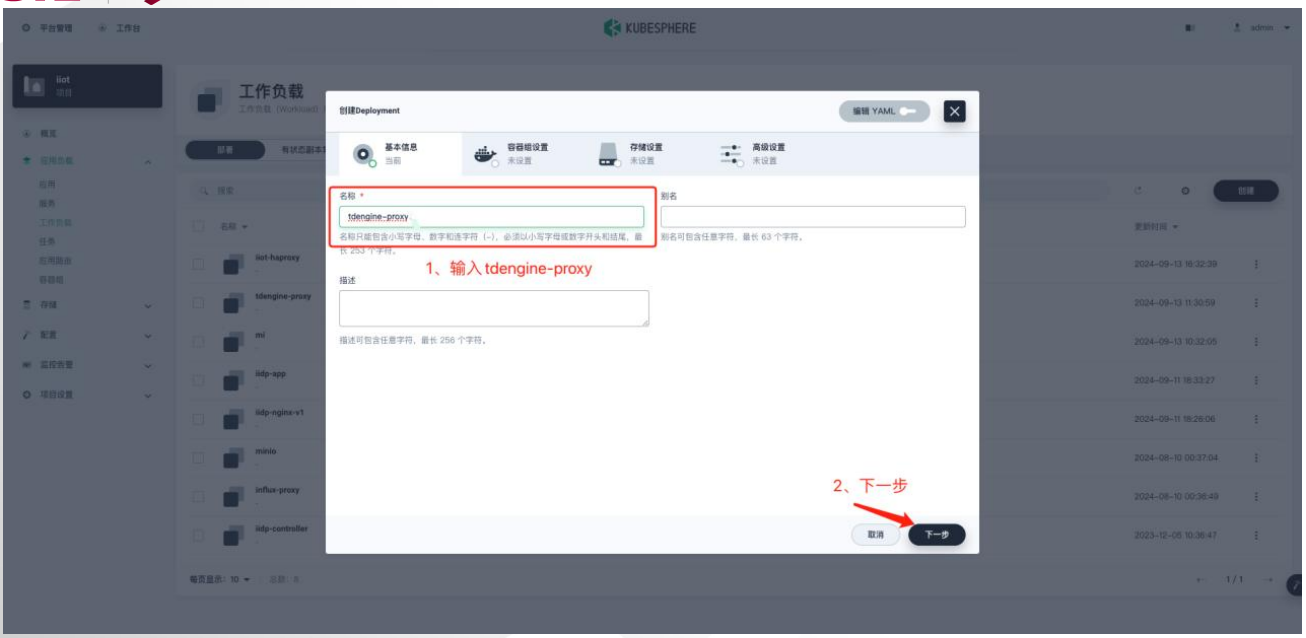

KUBESPHERE
创建持久卷声明
编辑 YAML
基本信息
当前
存储设置
未设置
高级设置
未设置
名称 *
tdengine-proxy
1、输入tdengine-proxy
别名
描述
2、下一步
取消
下一步

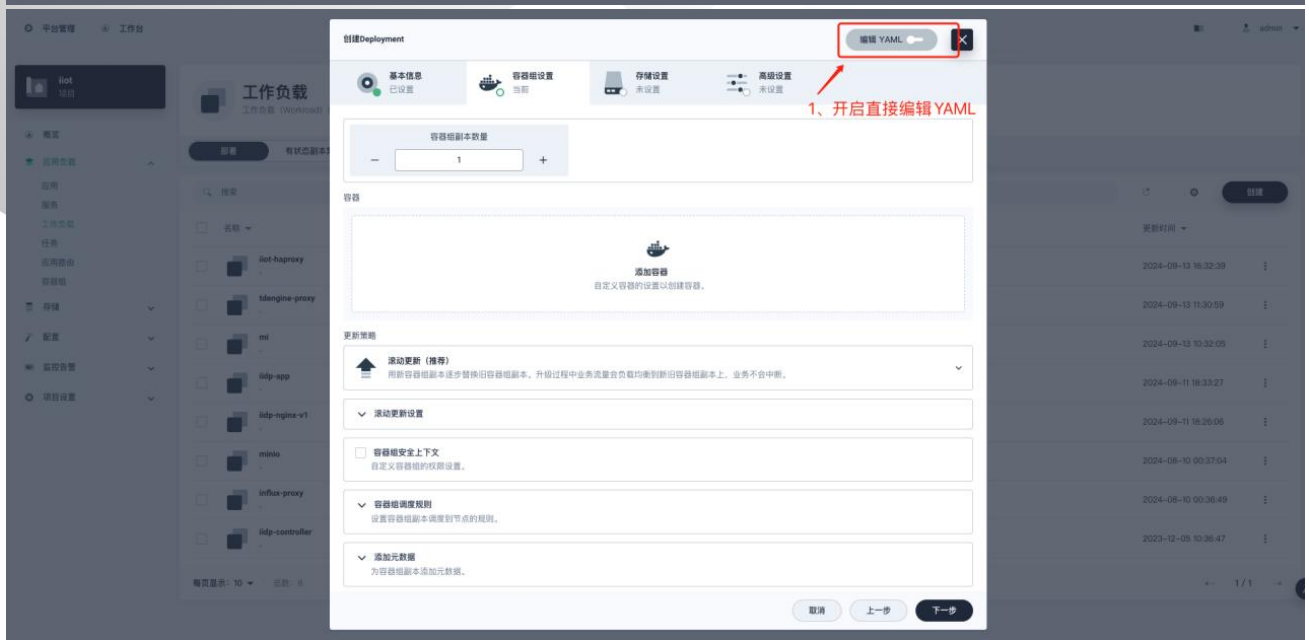

KUBESPHERE
创建持久卷声明
编辑 YAML
基本信息
已设置
存储设置
当前
高级设置
未设置
创建方式
通过存储类创建
选择已有的存储类来创建卷。
1、选择相应条目
存储类 *
nfs-client
访问模式 *
ReadWriteMany
卷容量
0
512Gi
1024Gi
1536Gi
2048Gi
10
Gi
2、下一步
取消
上一步
下一步

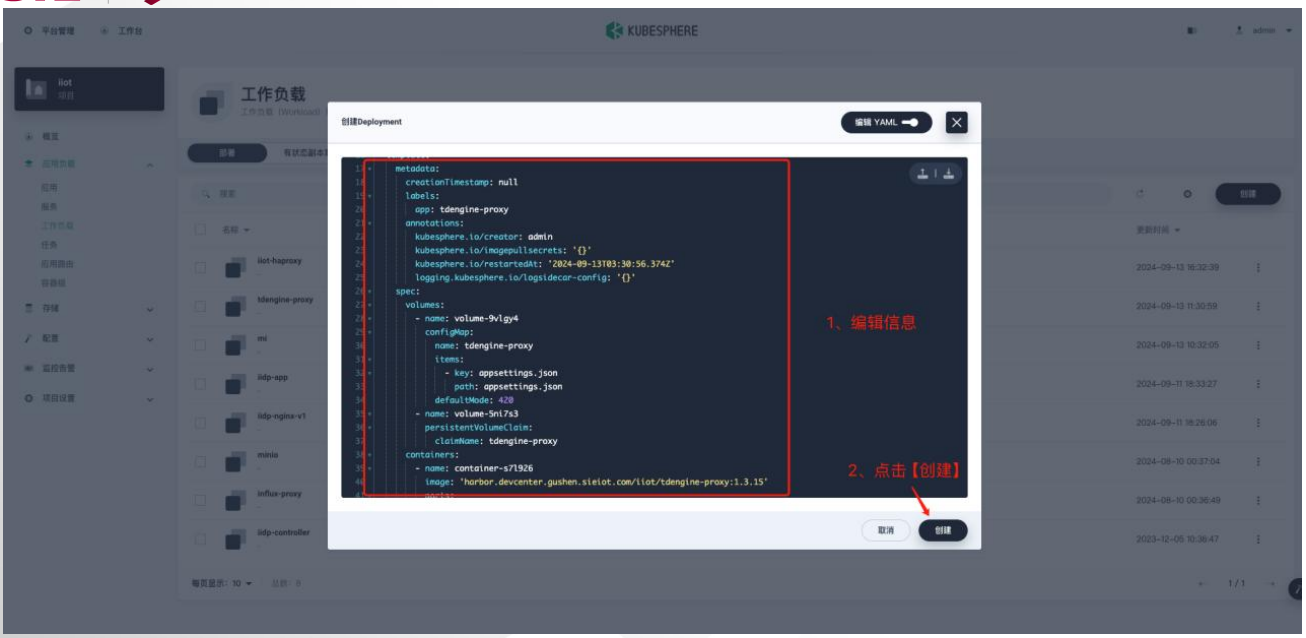


- 项目页面左侧选择【应用负载】->【工作负载】菜单，右上角点击【创建】按钮创建 tdengine-proxy 负载。




KUBESPHERE
工作负载
创建Deployment
编辑 YAML
基本信息
容器组设置
存储设置
高级设置
名称 *
tdengine-proxy
别名
1、输入tdengine-proxy
描述
2、下一步
取消
下一步


创建Deployment
编辑 YAML
基本信息
容器组设置
存储设置
高级设置
1、开启直接编辑 YAML
容器组副本数量
容器
添加容器
更新策略
滚动更新（推荐）
滚动更新设置
容器组安全上下文
容器组调度规则
添加元数据
取消
上一步
下一步



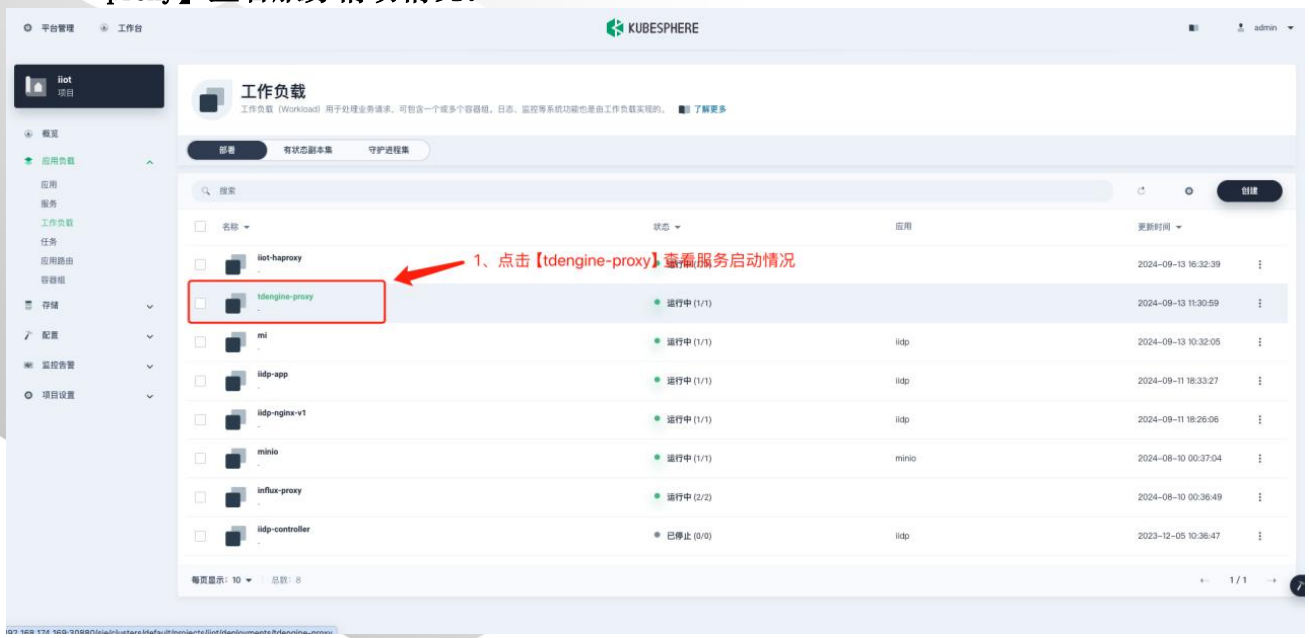
配置文件内容（TDengineProxy 镜像版本号根据实际修改，此处使用 1.3.15 版本）：

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: tdengine-proxy
  namespace: iiot
  labels:
    app: tdengine-proxy
  annotations:
    deployment.kubernetes.io/revision: '42'
    kubesphere.io/creator: admin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tdengine-proxy
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: tdengine-proxy
      annotations:
        kubesphere.io/creator: admin
        kubesphere.io/imagepullsecrets: '{}'
        kubesphere.io/restartedAt: '2024-09-13T03:30:56.374Z'
        logging.kubesphere.io/logsidecar-config: '{}'
    spec:
      volumes:
```

```
      - name: volume-9vlgy4
        configMap:
          name: tdengine-proxy
          items:
            - key: appsettings.json
              path: appsettings.json
          defaultMode: 420
      - name: volume-5ni7s3
        persistentVolumeClaim:
          claimName: tdengine-proxy
  containers:
    - name: container-s71926
      image: 'harbor.devcenter.gushen.sieiot.com/iiot/tdengine-proxy:1.3.32'
      ports:
        - name: http-80
          containerPort: 80
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: volume-9vlgy4
          readOnly: true
          mountPath: /app/appsettings.json
          subPath: appsettings.json
        - name: volume-5ni7s3
          mountPath: /app/logs
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      imagePullPolicy: IfNotPresent
  restartPolicy: Always
  terminationGracePeriodSeconds: 30
  dnsPolicy: ClusterFirst
  serviceAccountName: default
  serviceAccount: default
  securityContext: {}
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchLabels:
                app: tdengine-proxy
            topologyKey: kubernetes.io/hostname
  schedulerName: default-scheduler
```

```
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 25%
  revisionHistoryLimit: 10
  progressDeadlineSeconds: 600
```

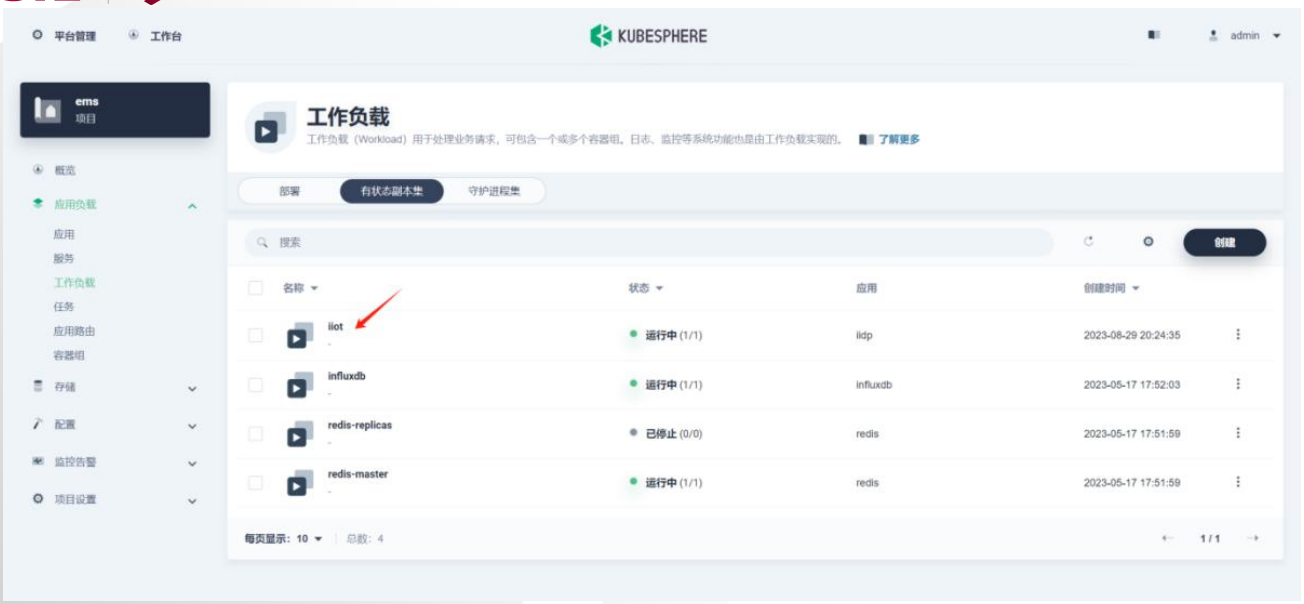
- 项目管理页面左侧选择【应用负载】->【工作负载】菜单，中间区域选择【tdengine-proxy】查看服务情动情况。



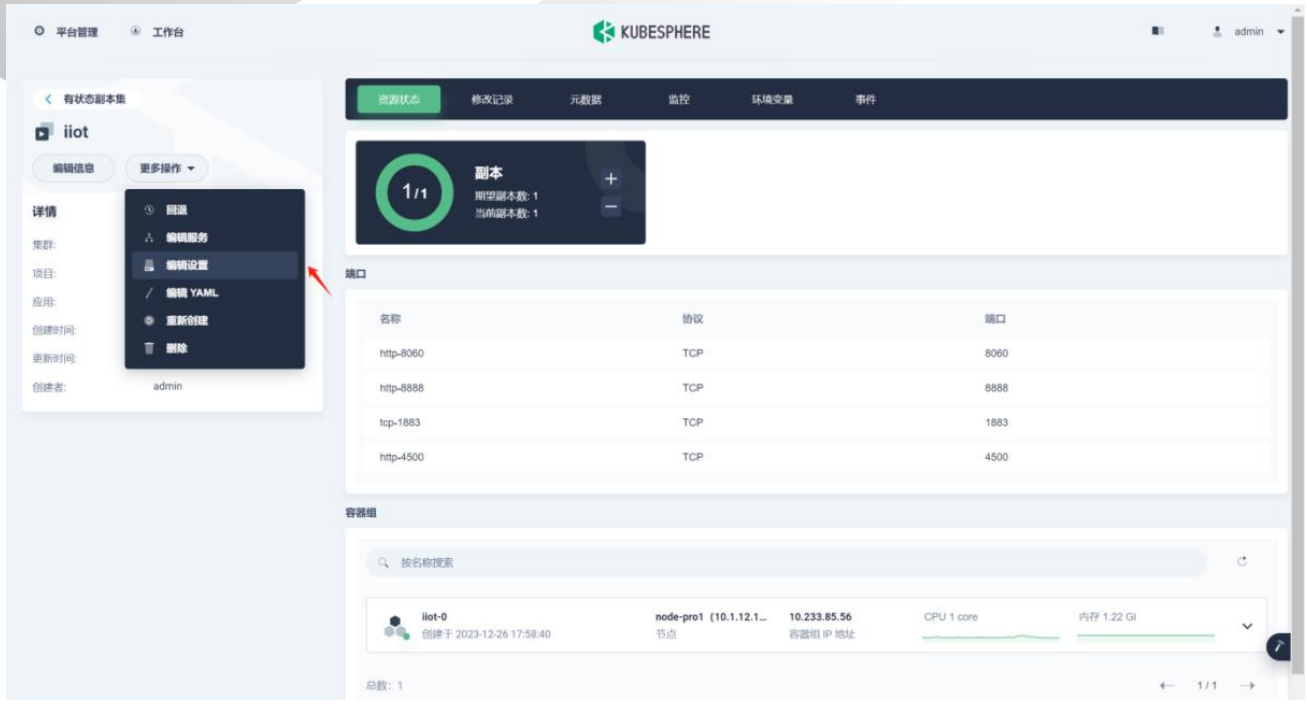
3.2.6. 环境配置

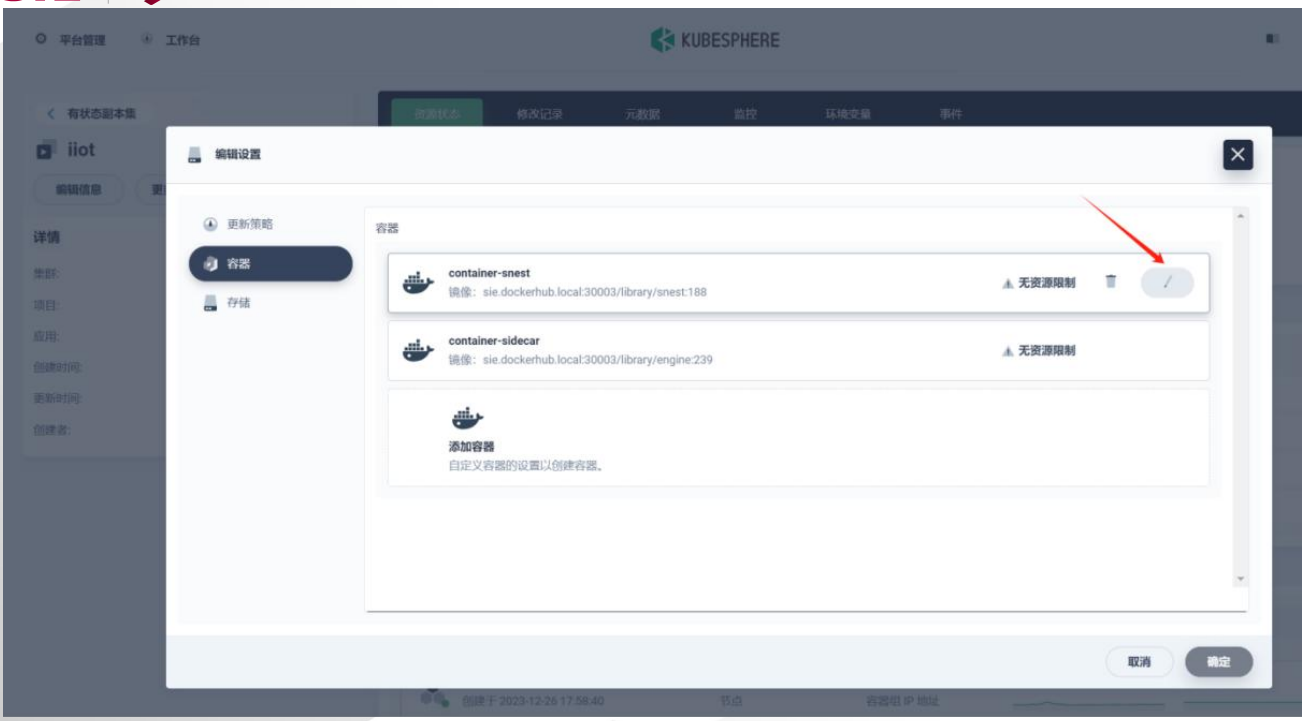
3.2.6.1. 工作负载配置

- 登录 kubesphere 管理平台，进入项目所属集群页面，选择【应用负载】→【工作负载】→【有状态副本集】，依次 iiot 副本集。



- 页面左上角选择【更多操作】→【编辑设置】，左侧选择【容器】，点击 container-snest 容器右侧【编辑】按钮，在【端口设置】部分参考截图新增 TCP-1883 和 HTTP-8888 配置，点击“√”，再点击【确定】按钮保存。





- 紧接上一步操作，在【环境变量】部分参考截图新增/修改 SW_OPTS 参数，参数值：-XX:NewRatio=1 -XX:InitialHeapSize=4g -XX:MaxHeapSize=16g -XX:ActiveProcessorCount=8 -XX:ParallelGCThreads=8 -XX:+UseStringDeduplication -XX:+UseCompressedOops -XX:MaxMetaspaceSize=512m -XX:NativeMemoryTracking=summary -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/apps/iiot/

- 页面左上角选择【更多操作】→【编辑 YAML】，在【securityContext】同级新增反亲和配置，点击“√”，再点击【确定】按钮保存（以下代码中的“iiot1”为当前工作负载名称）。

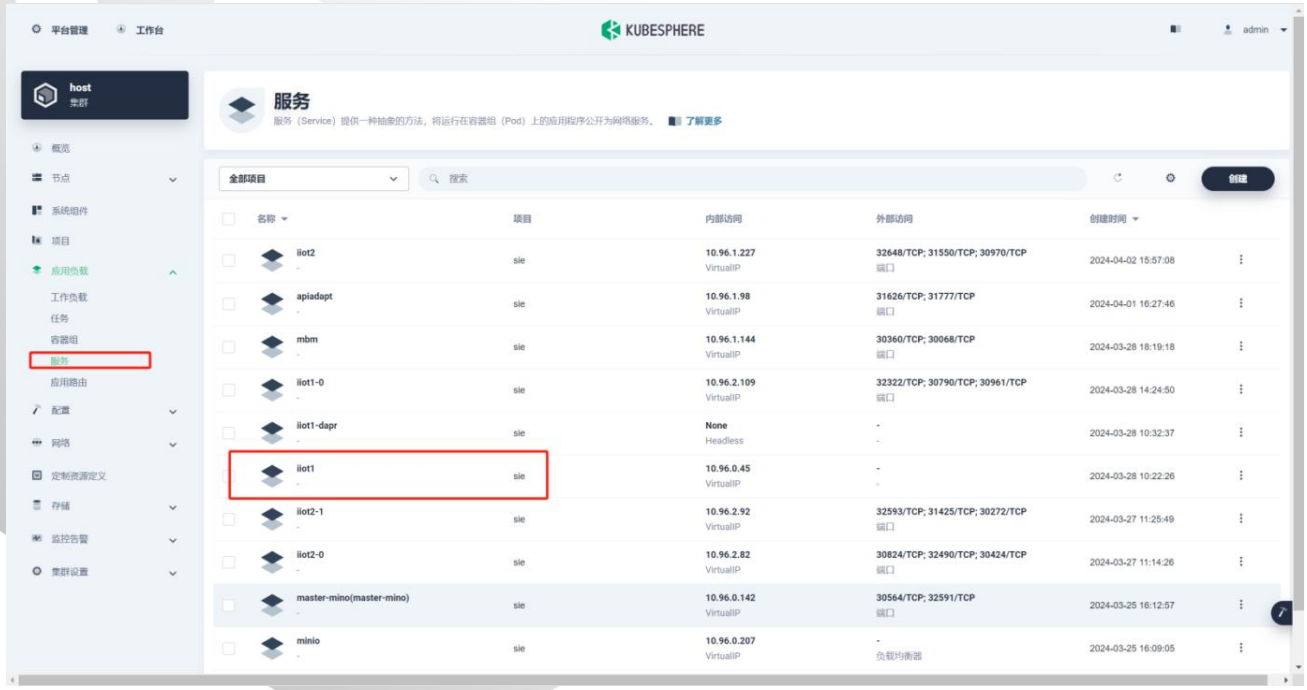
```
affinity:
    podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                    - key: app
                      operator: In
                      values:
                        - iiot1
              topologyKey: kubernetes.io/hostname
```

注意：如果安装了多个 iiot 服务，则按以上步骤重复查找相应工作负载并执行。

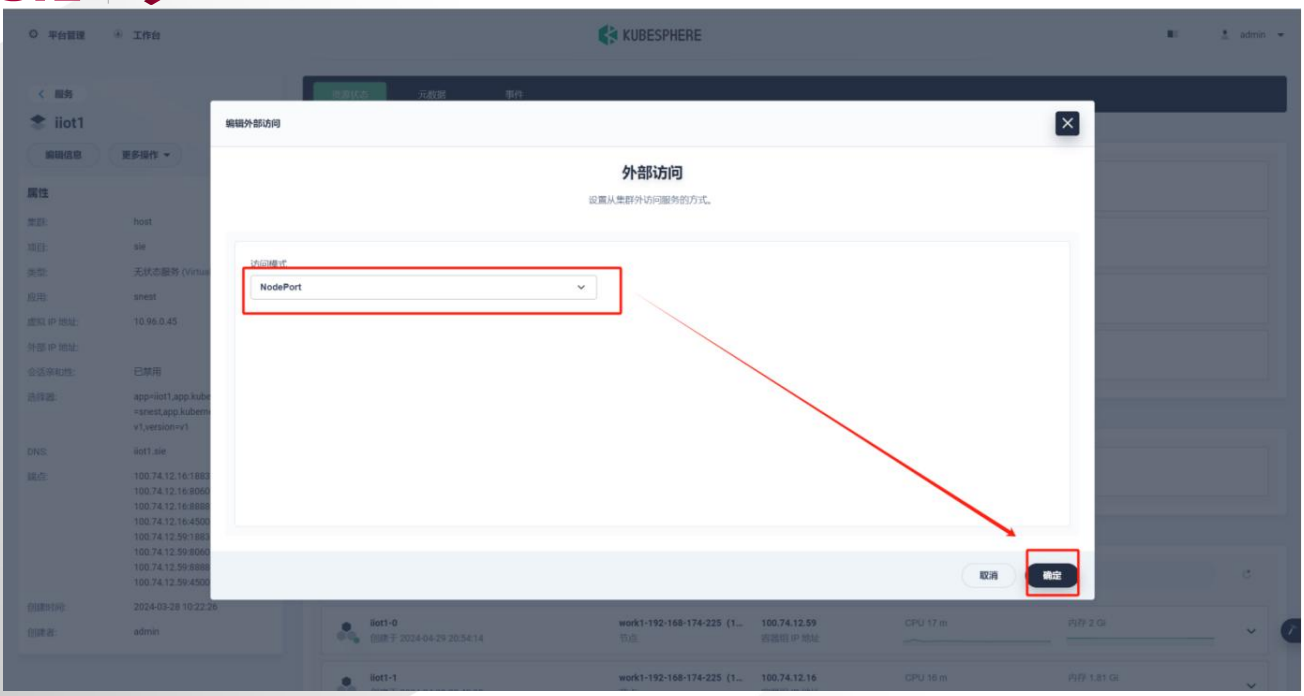
3.2.6.2. 服务配置

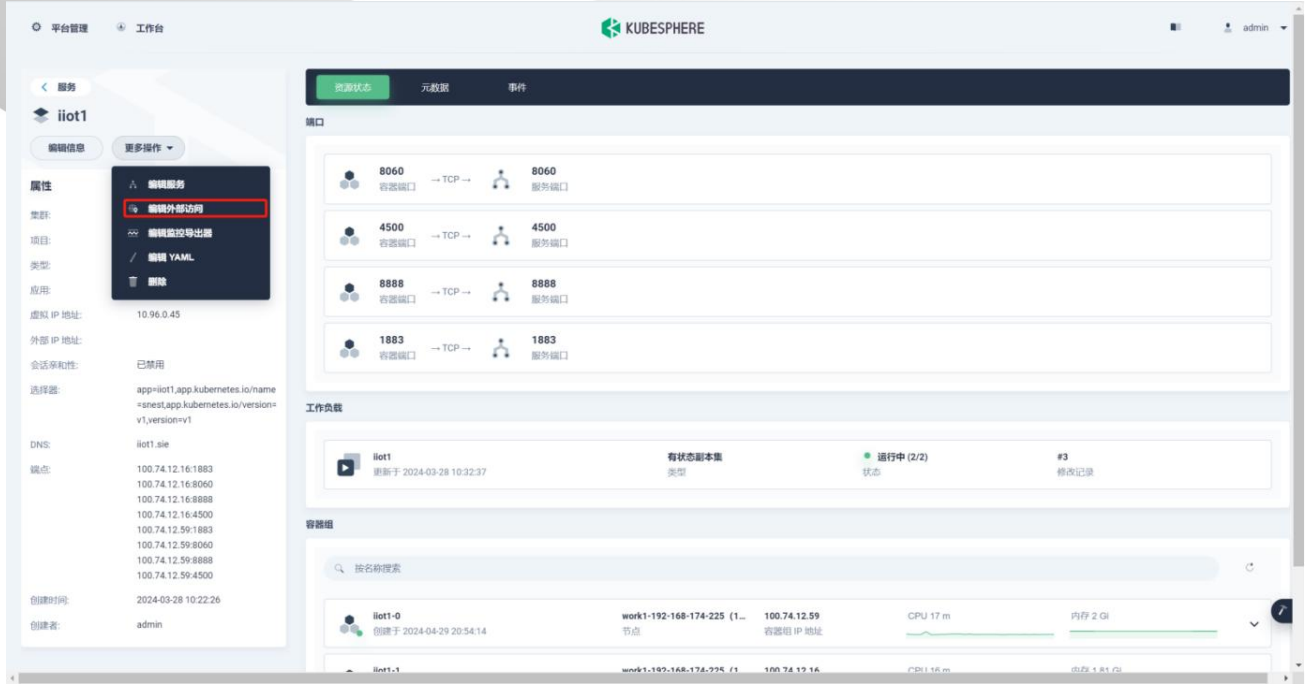
3.2.6.2.1. 配置负载服务

- 登录 kubesphere 管理平台，进入项目所属集群页面，选择【应用负载】→【服务】，查找 iiot1 服务（IIDP 部署 iiot1 服务自动生成），点击进入服务详情页面。

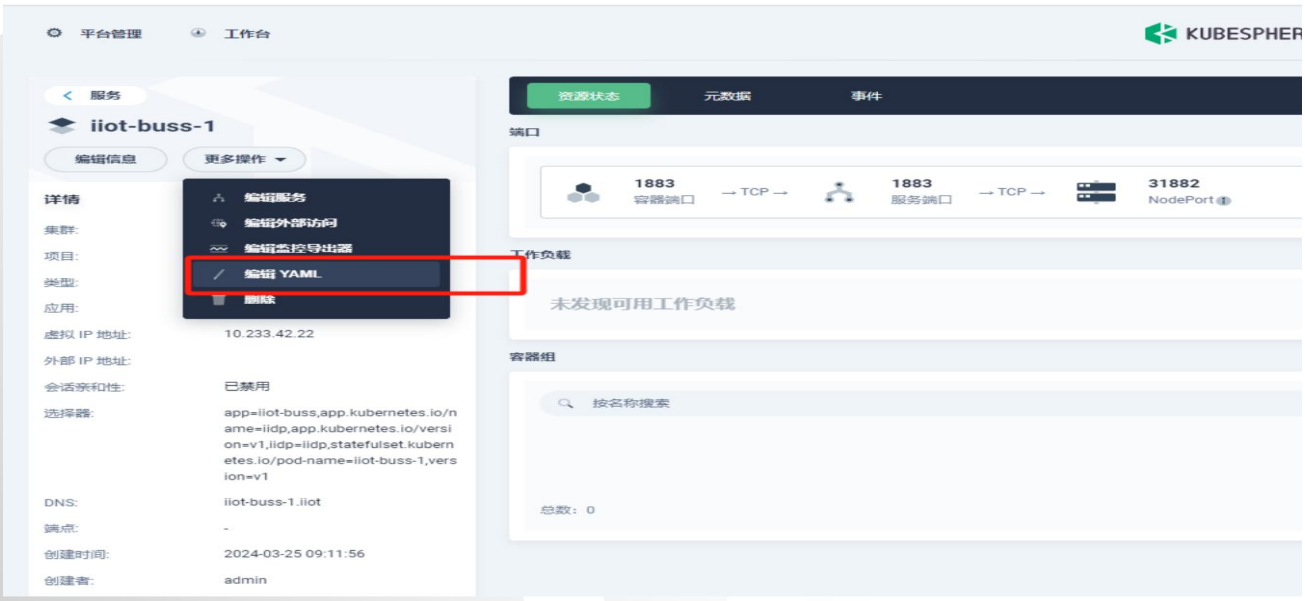


- 页面左上角选择【更多操作】→【编辑外部访问】，选择“NodePort”并点击【确定】按钮保存。





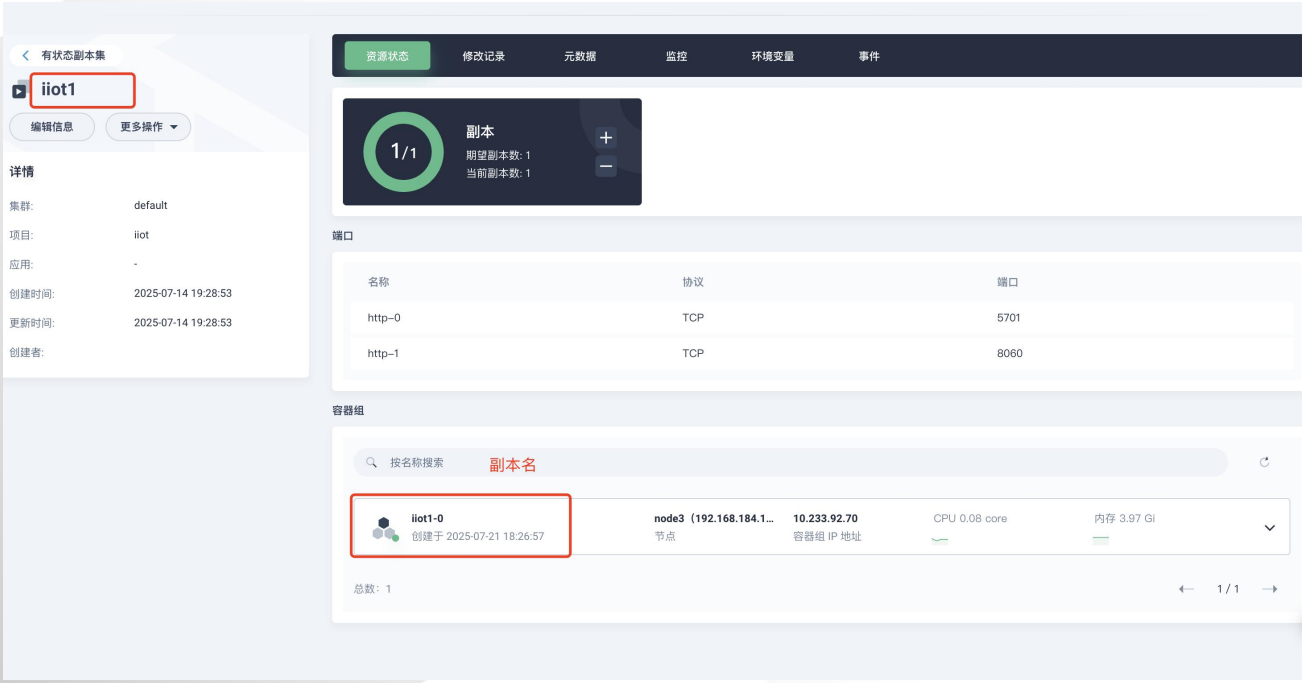
- 页面左上角选择【更多操作】→【编辑 YAML】，按端口开放清单，修改其中的对外开放端口号，并点击【确认】按钮保存。



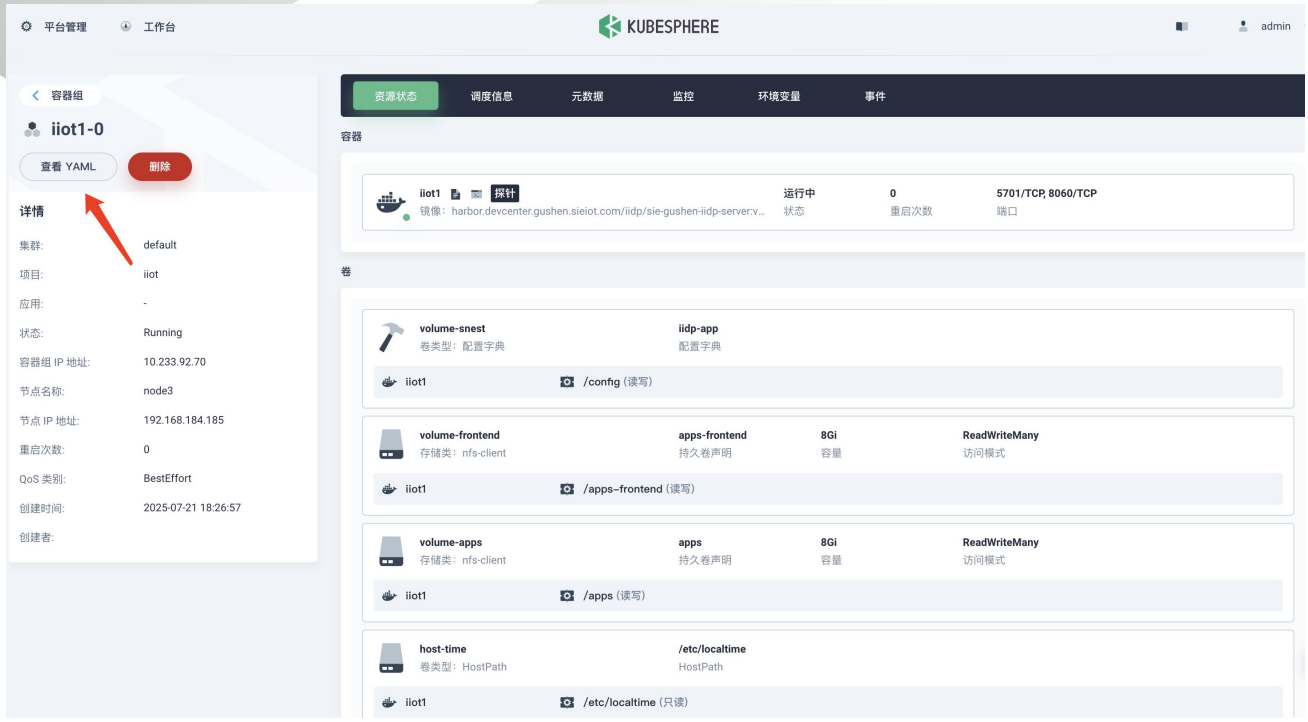
注意：如果在 IIDP 应用市场安装了多个 iiot 服务，则按以上步骤重复查找相应服务并执行。

3.2.6.2.2. 配置副本服务

- 登录 kubesphere 管理平台，进入项目所属集群页面，选择【应用负载】→【服务】给 iiot 副本（假设 IIDP 安装了 iiot1 服务，则副本名称类似：iiot1-0）创建一个服务。



➢ 点击副本名跳转到副本容器组页面，点击查看 YAML。



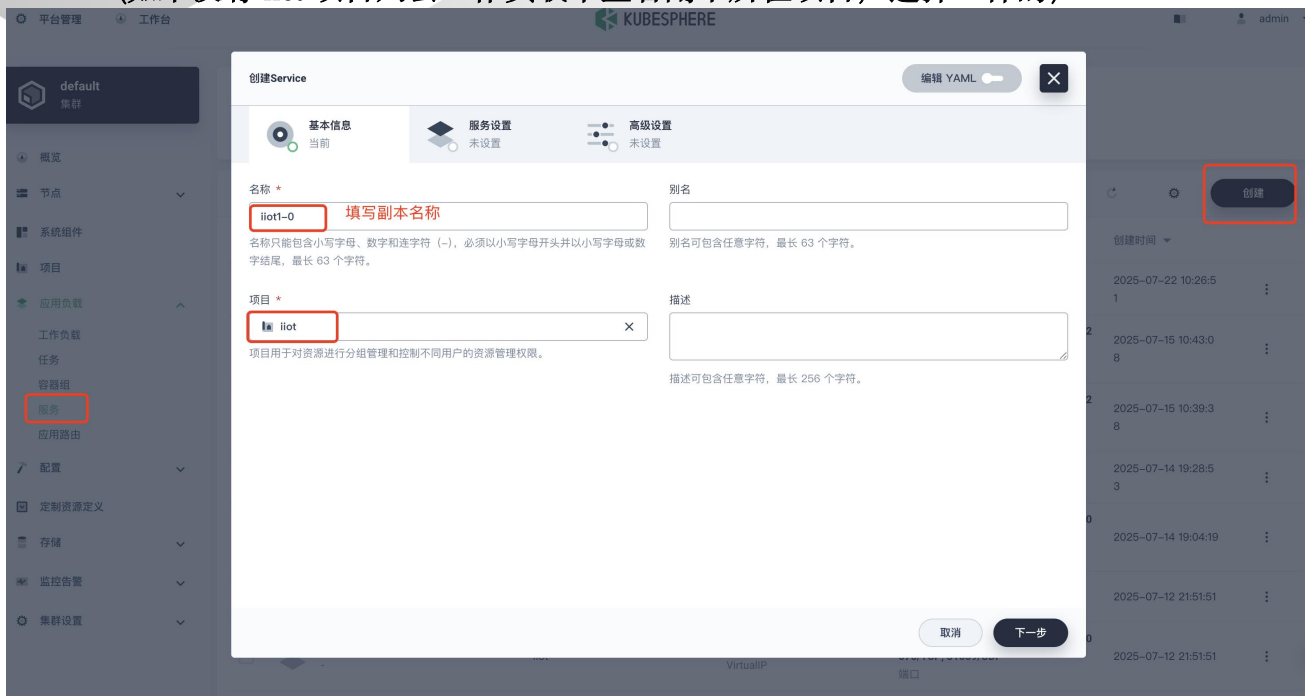
➢ 复制 yaml 文件中这三行之后要用

查看 YAML

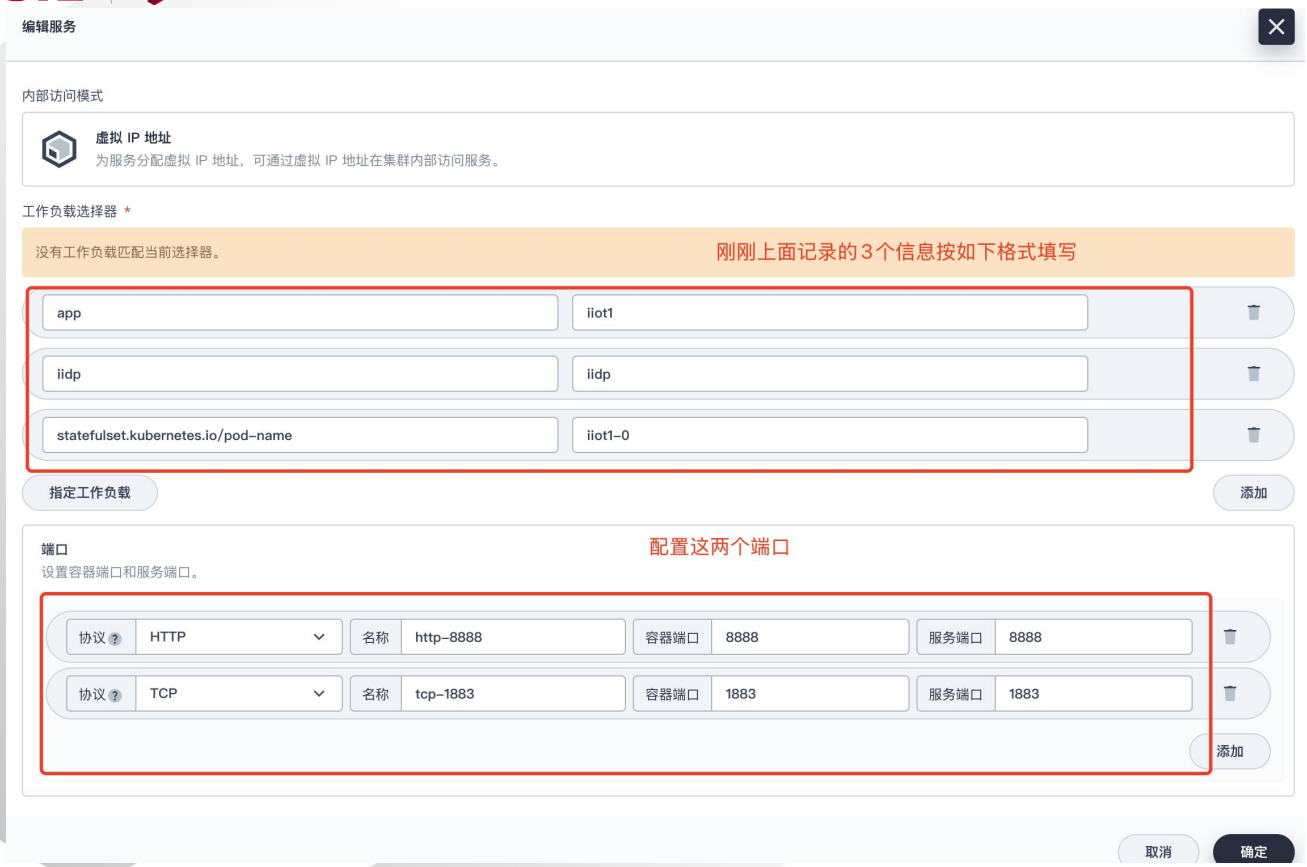
```yaml
kind: Pod
apiVersion: v1
metadata:
  name: iiot1-0
  generateName: iiot1-
  namespace: iiot
  labels:
    app: iiot1
    controller-revision-hash: iiot1-795d8d5555
    iidp: iidp
    statefulset.kubernetes.io/pod-name: iiot1-0
  annotations:
    cni.projectcalico.org/containerID: 4bc17144986846cbe95e2ab4d48735504b4e1f93406ad0282643f1a3e257316f
    cni.projectcalico.org/podIP: 10.233.92.70/32
    cni.projectcalico.org/podIPs: 10.233.92.70/32
    iidp.com/restartedAt: '2025-07-21T10:21:28.024Z'
    kubesphere.io/imagepullsecrets: '{}'
spec:
  volumes:
    - name: volume-snest
      configMap:
        name: iidp-app
        defaultMode: 420
    - name: volume-frontend
      persistentVolumeClaim:
        claimName: apps-frontend
    - name: volume-apps
      persistentVolumeClaim:
        claimName: apps
    - name: host-time
      hostPath:
        path: /etc/localtime
        type: ''
    - name: kube-api-access-xb57n
```

复制这三行，之后要用

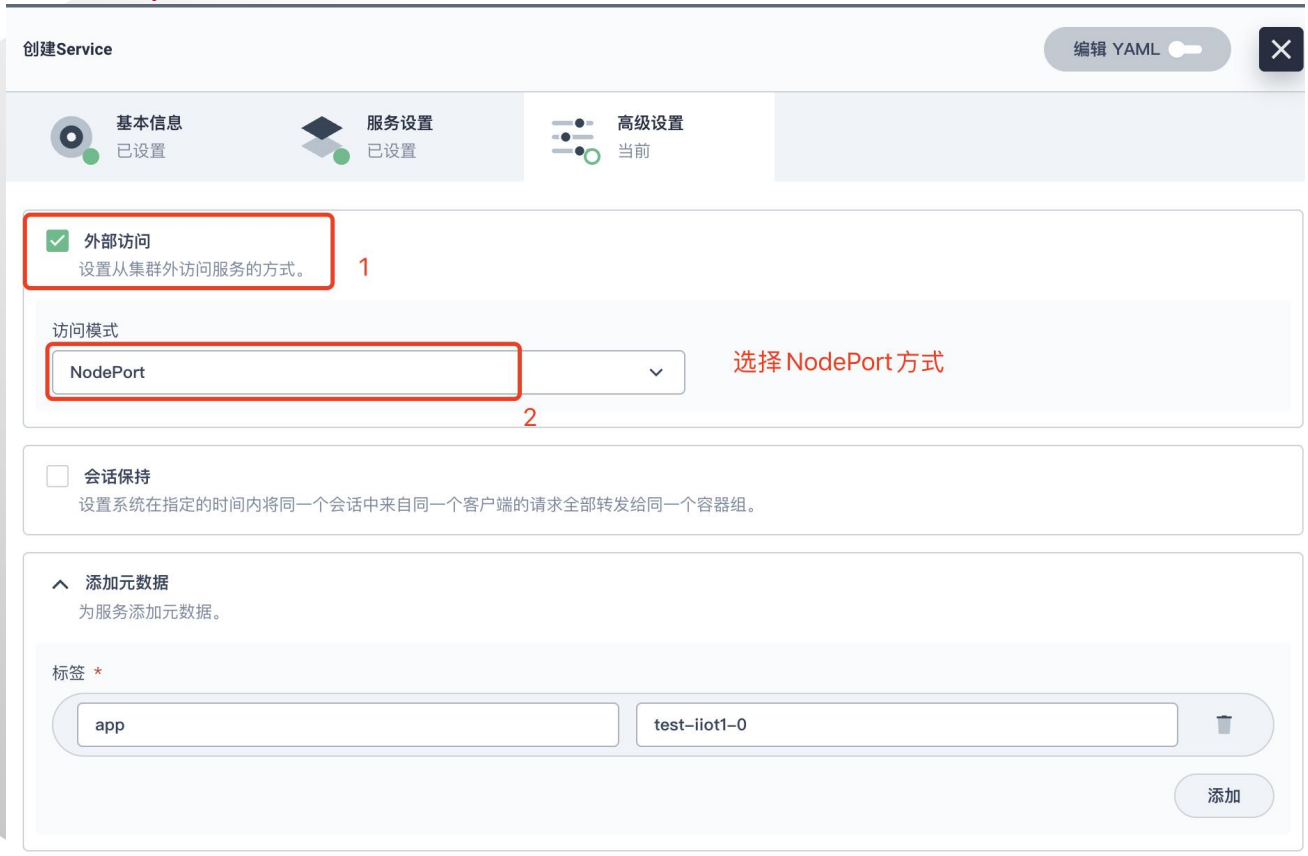
- 进入当前项目的服务列表，点击“创建”，填写和副本名一样的名称，选择 iiot 项目(如果没有 iiot 项目则去工作负载中查看副本所在项目，选择一样的)

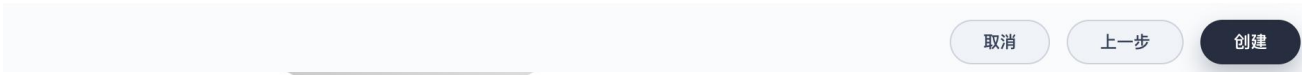


- 下一步，按照图片方式填写

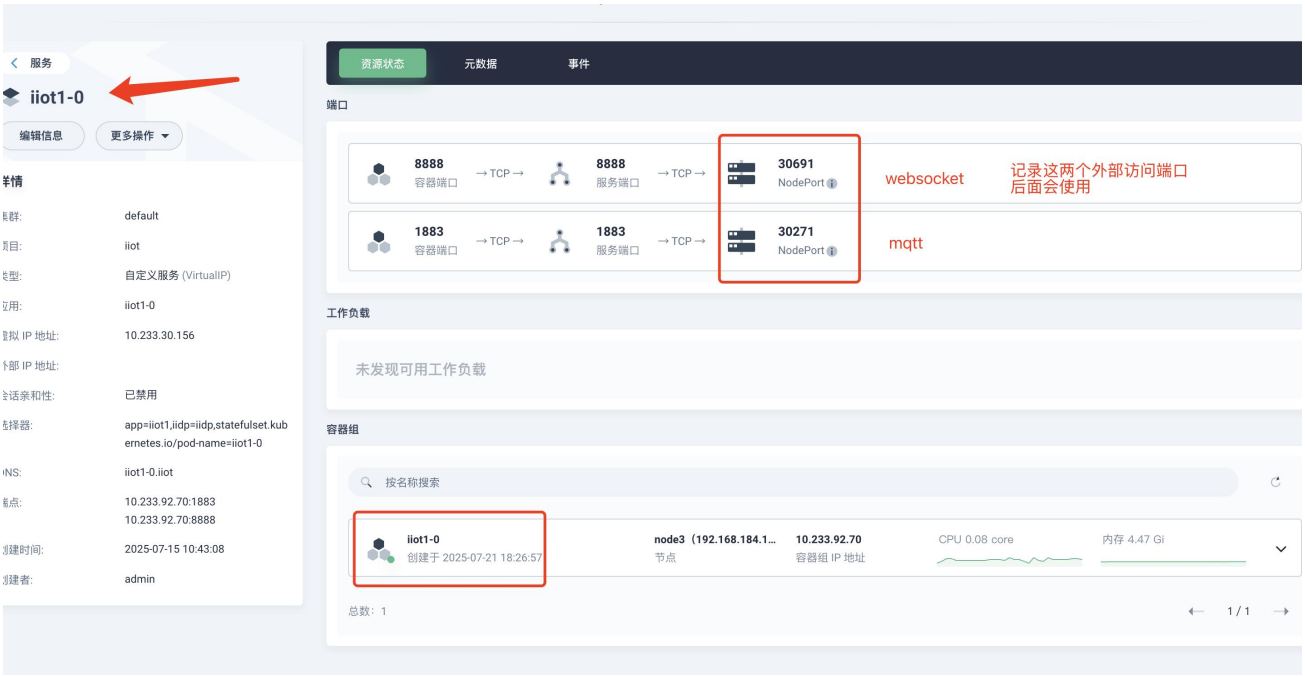


- 下一步，选择 NodePort 访问模式，点击创建





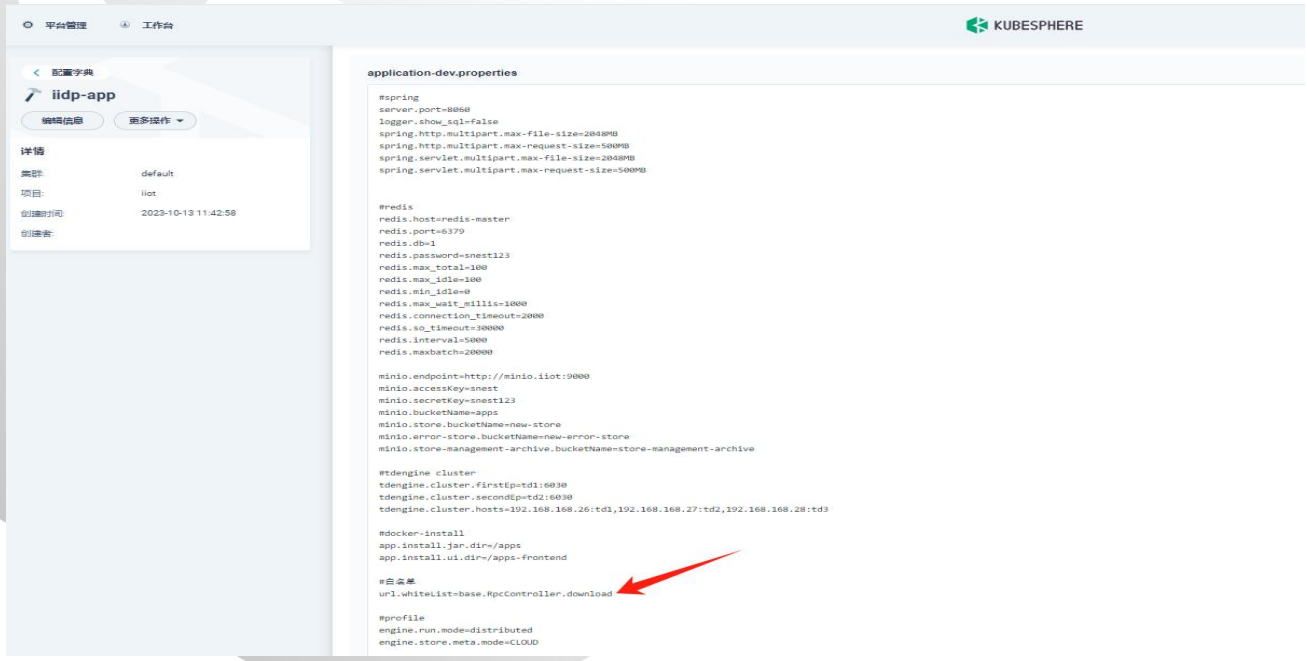
- 再次点击进入 iiot1-0 服务，记录服务分配的 2 个类型的外部访问端口，之后在系统配置时要使用



- 到此一个副本已配置完成，若有多个 iiot 服务和副本需要对不同服务、副本重复以上配置。

3.2.6.3. 白名单配置

- 检查平台配置文件 application-dev.properties 中是否存在如下配置，如果不存在，需添加并重启容器（配置路径：【配置】→【配置字典】→【iidp-app】）



配置内容：

```
# 白名单
url.whiteList=base.RpcController.download
```

3.2.7. 系统配置

使用租户管理员账号登录 IIOT 系统，在【系统运维】-->【系统配置】页面，修改以下配置项。

表 3-1 系统配置表

配置键	说明	示例
server.ip	可以访问平台的 ip 地址或域名地址	192.168.175.192
server.api	可以访问平台 api 接口的 url 地址	http://192.168.175.192:30666/api/root/master
iiot1.websocket	服务 iiot1 对外开放的 websocket 地址	ws://192.168.175.192:30788/ws

iiot1.mqtt	服务 iiot1 对外开放的 mqtt 地址	192.168.175.192:31881
iiot2.websocket	服务 iiot1 对外开放的 websocket 地址	ws://192.168.175.192:30888/ws
iiot2.mqtt	服务 iiot1 对外开放的 mqtt 地址	192.168.175.192:31882
iiot1.http	服务 iiot1 向外暴露的 http 地址（按需）	http://192.168.174.228:30134
iiot2.http	服务 iiot2 向外暴露的 http 地址（按需）	http://192.168.174.228:30135
iiot1.coap	服务 iiot1 向外暴露的 coap 地址（按需）	coap://192.168.174.228:30809
iiot2.coap	服务 iiot2 向外暴露的 coap 地址（按需）	coap://192.168.174.228:30810
iiot1.iiot1-0.mqtt	服务 iio1t 的 iiot1-0 副本向外暴露的 mqtt 地址 对应 4.2.3.3 章节配置副本服务记录的 mqtt 端口，不同副本的服务端口不一样	192.168.174.169:30460
iiot2.iiot2-0.mqtt	服务 iiot2 的 iiot2-0 副本向外暴露的 mqtt 地址 对应 4.2.3.3 章节配置副本服务记录的 mqtt 端口，不同副本的服务端口不一样	192.168.174.169:30560
iiot1.iiot1-0.websocket	服务 iiot1 的 iiot1-0 副本向外暴露的 websocket 地址 对应 4.2.3.3 章节配置副本服务记录的 websocket 端口，不同副本的服务端口不一样	ws://192.168.174.169:32248/ws
iiot2.iiot2-0.websocket	服务 iiot2 的 iiot2-0 副本向外暴露的 websocket 地址 对应 4.2.3.3 章节配置副本服务记录的 websocket 端口，不同副本的服务端口不一样	ws://192.168.174.169:32348/ws

以上配置中涉及的 ip 修改为部署服务器的 ip，端口取章节【端口开放清单】准备的信息。

- 参数调优

分析数据上报的测点数据类型，针对性进行调优。例如上报测点的数据类型 double 较多，可以适量增加 double 类型的存储配置，对应配置为 store.double.thread 和 store.double.interval，分别为 double 类型数据存储任务的线程数量和存储间隔（毫秒），线程数可以调整为 2-4，存储间隔可以适当缩短，最小不要低于 1000（毫秒）

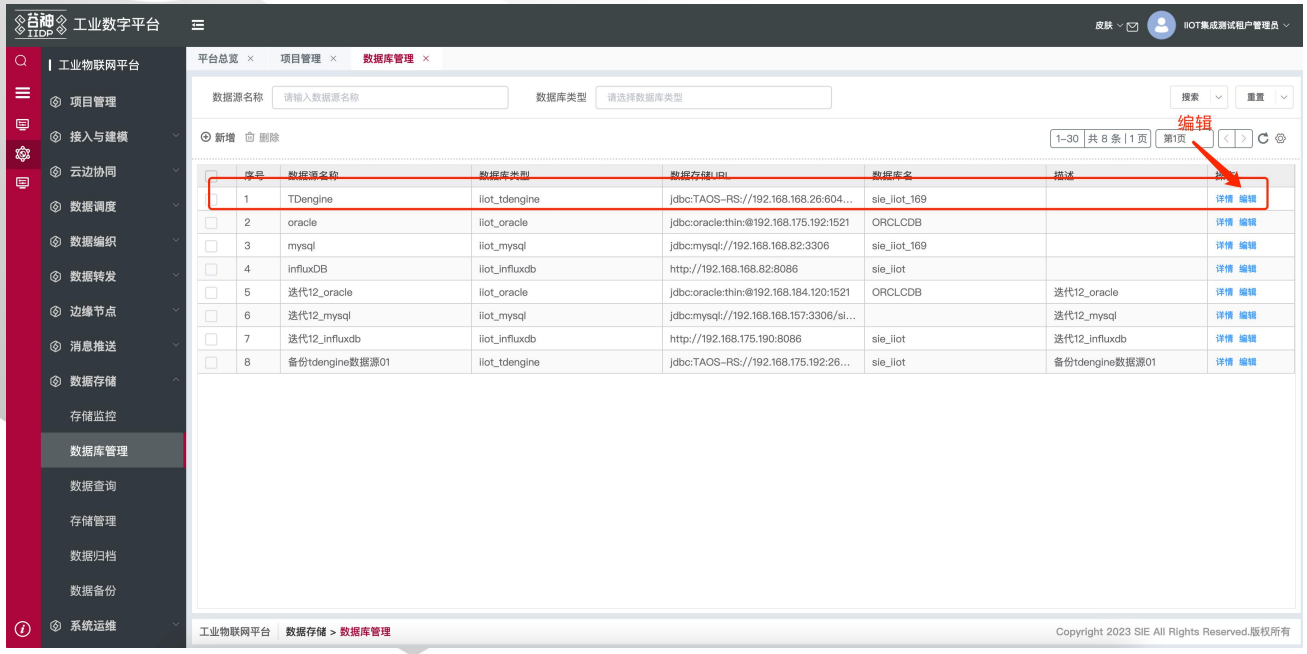
注意：参数调优配置修改后需要手动重启服务。

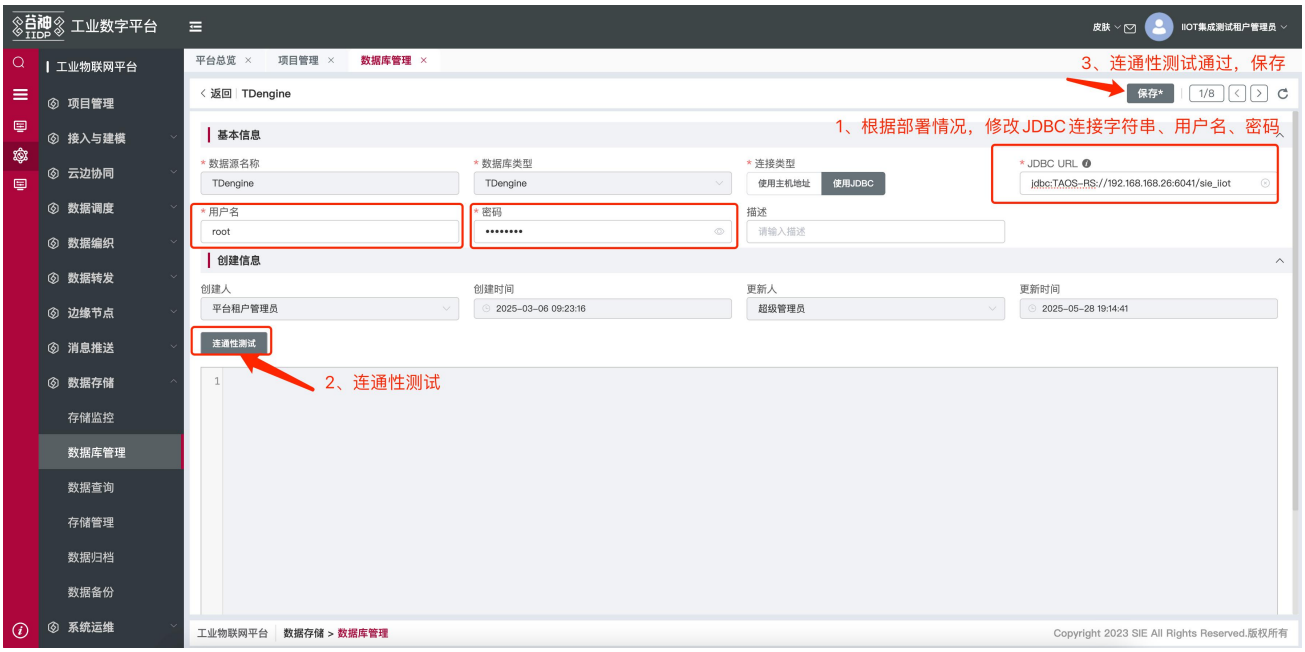
配置键	说明	示例

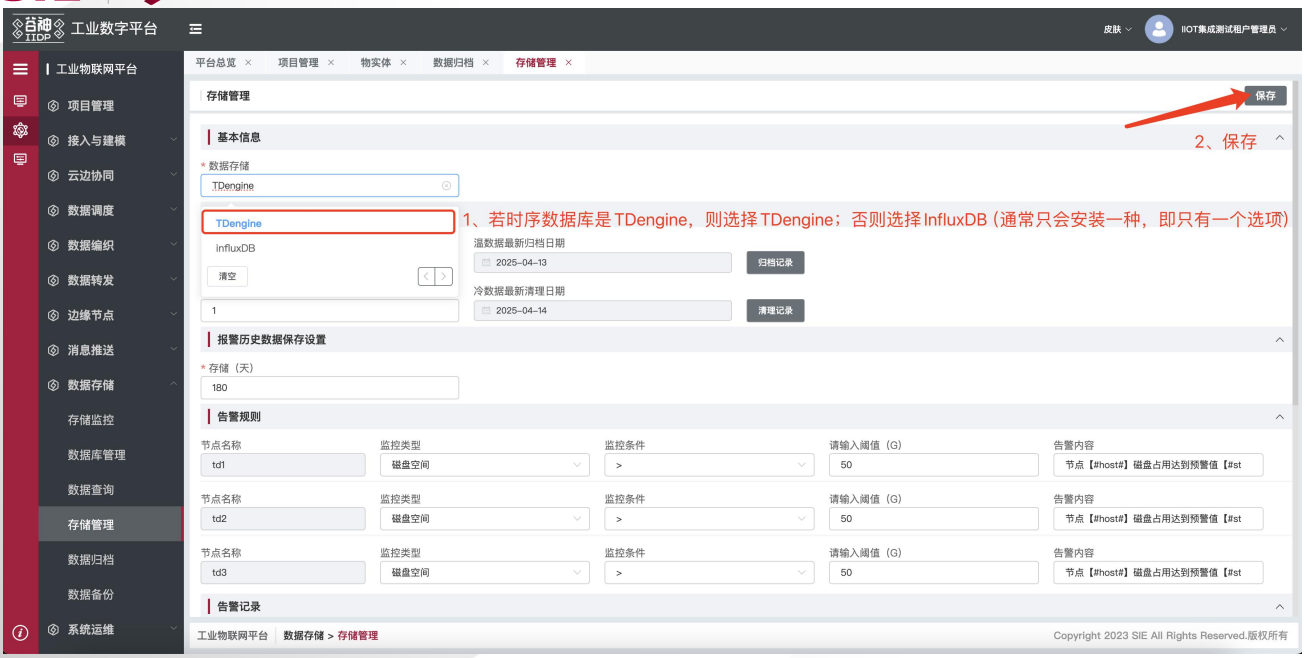
store.double.thread	定时执行时序数据存储-double 类型任务的线程数量，默认为 1，修改后需要重启服务	4
store.double.interval	定时执行时序数据存储-double 类型任务的存储间隔（毫秒），默认值为 2000，修改后需要重启服务	1000

3.2.8. 存储配置

- 使用租户管理员账号登录 IIOT 系统，在【数据存储】-->【数据库管理】页面，修改时序数据库的信息。







3.3. 高可用版

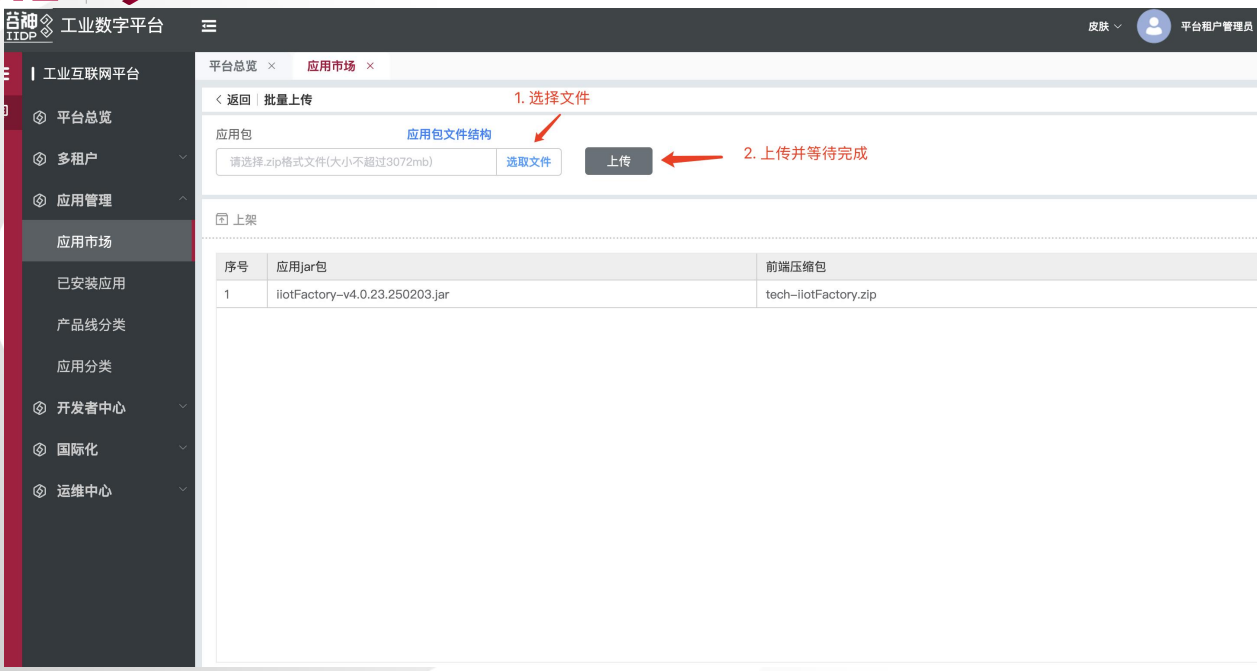
3.3.1. 上传 App

使用平台超级管理员账号（默认账号：implementer）登录 IIDP 系统，菜单选择**【应用管理】→【应用市场】**，点击**【批量上传】**上传项目所需 App（第 2 章已经在 SMDC 官网下载的 zip 包文件）。

注意：部署前请针对项目需要规划需要安装的 App。需注意每个 App 是单独文件夹，App 文件夹若存在 zip 前端包，则需在上架当前 App 时，同步上传 zip 前端包。

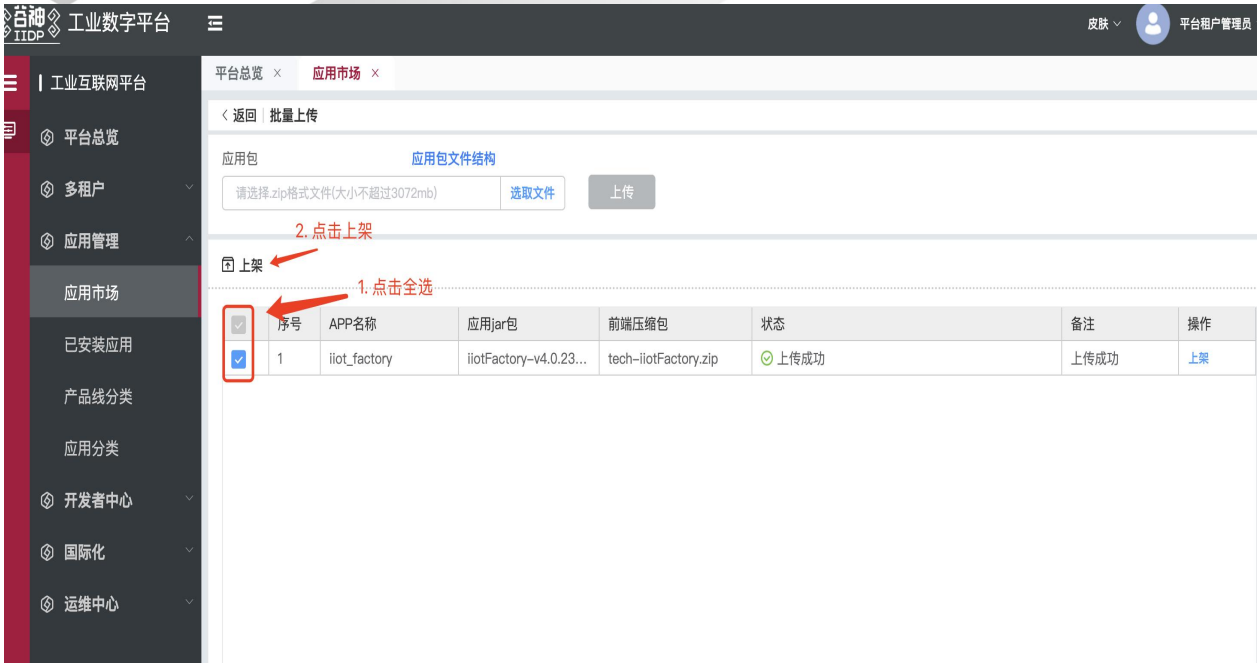
针对所部署的时序数据库，选择相应的时序数据库 App，具体信息如下：

- InfluxDB：对应安装 iiot_store。
- TDengine：对应安装 iiot_tdstore。



3.3.2. 上架 App

等待 App 上传完成，在页面勾选相应的 App，点击【上架】上架所选 App。



3.3.3. 安装 App

强制单独安装服务的 App 有：

- iiot_cluster_manager

- iiot_dataservice_opcua、iiot_dataservice_api（共同安装至同一服务）

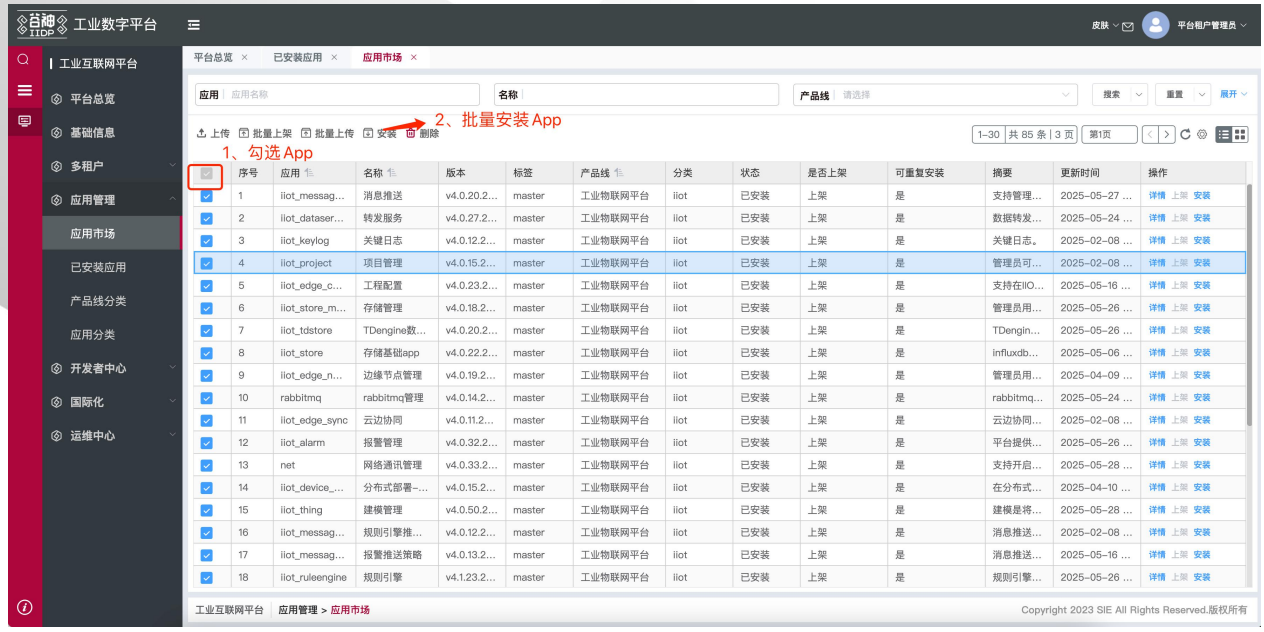
建议单独安装服务的 App 有：

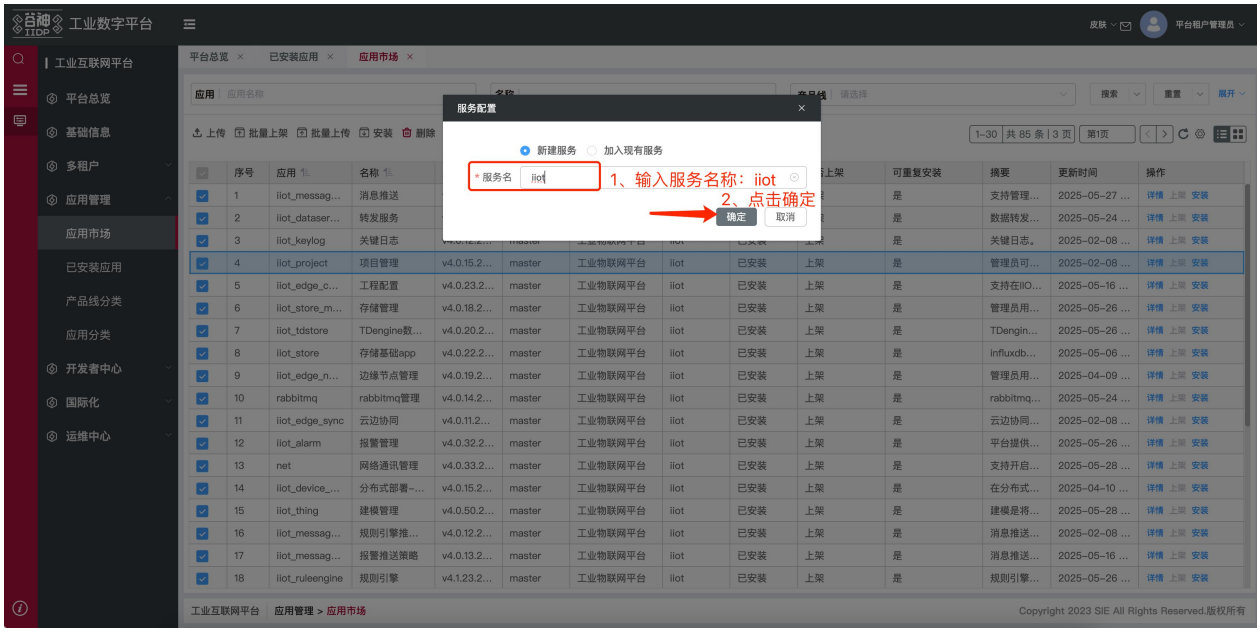
- iiot_data_archive
- iiot_data_backup

其它 App 安装至 iot 服务。

3.3.3.1. 安装 iot 服务

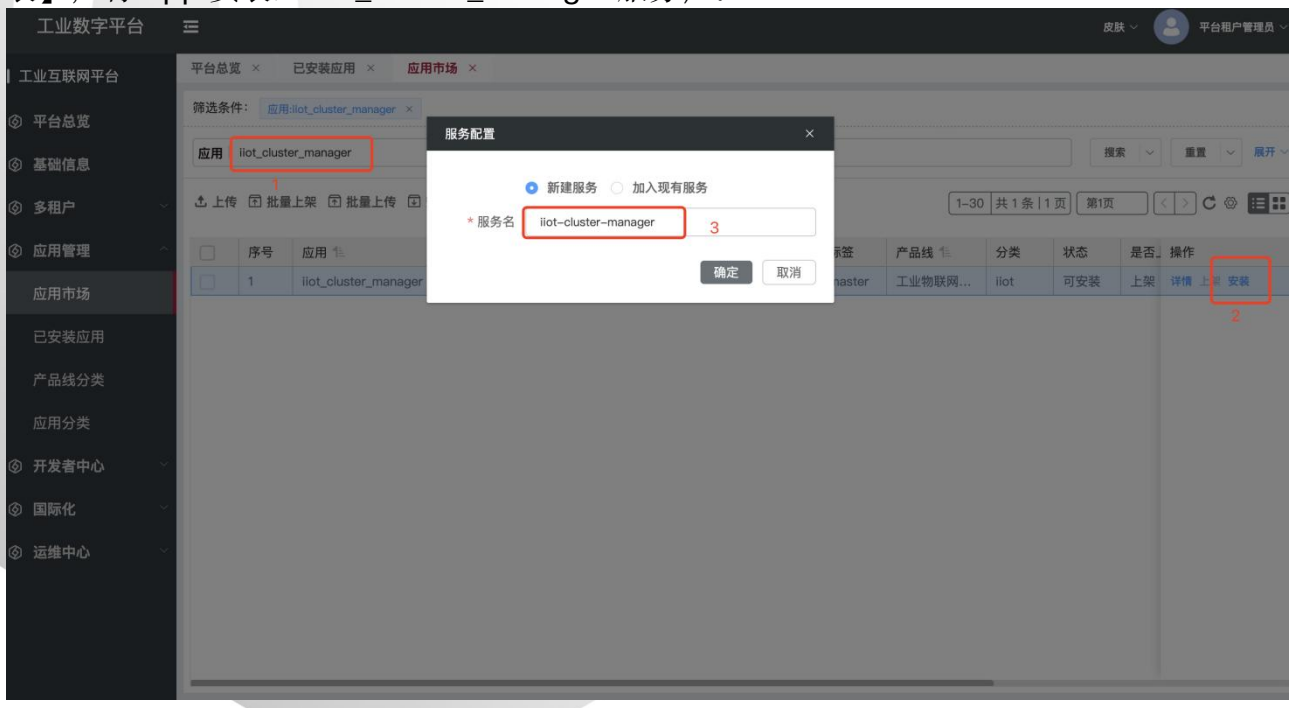
在【应用管理】→【应用市场】页面，勾选 iot 服务需要安装的 App，点击【安装】，将 App 安装至 iiot1 服务（若需安装多个 iot 服务，则依次重复该步骤，服务名称：iiot2、iiot3、...）。





3.3.3.2. 安装 iiot_cluster_manager 服务

在【应用管理】→【应用市场】页面，勾选【iiot_cluster_manager】App，点击【安装】，将 App 安装至 iiot_cluster_manager 服务）。

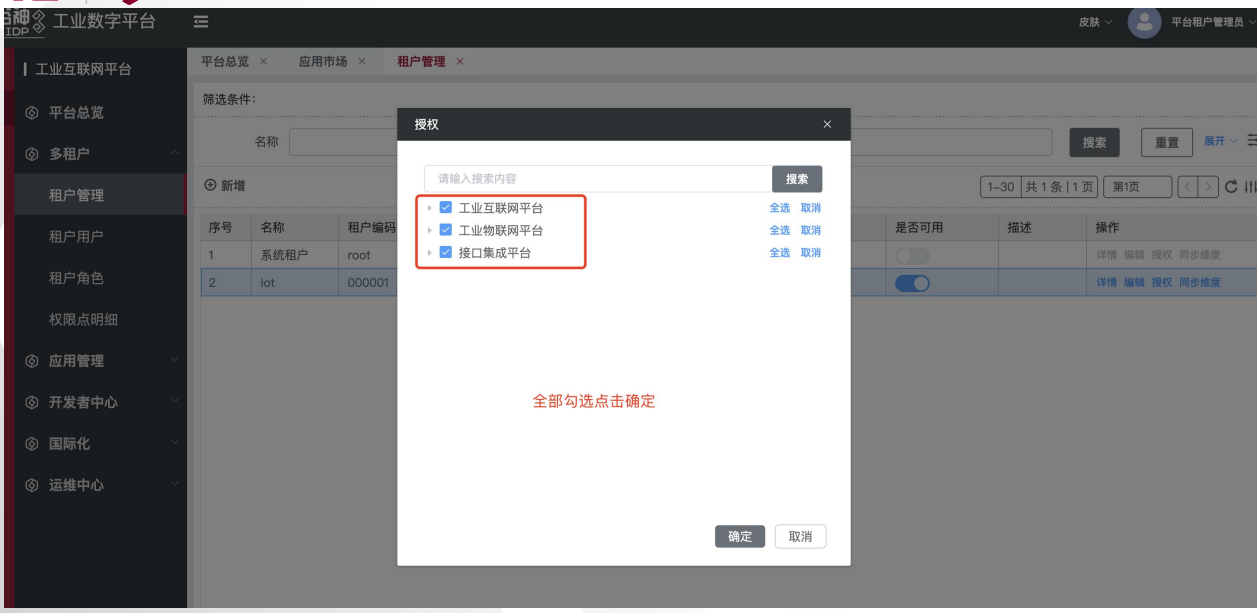


3.3.3.3. 安装 opcua 服务

在【应用管理】→【应用市场】页面，勾选【iiot_dataservice_opcua、iiot_dataservice_api】App，点击【安装】，将 App 安装至 opcua 服务。

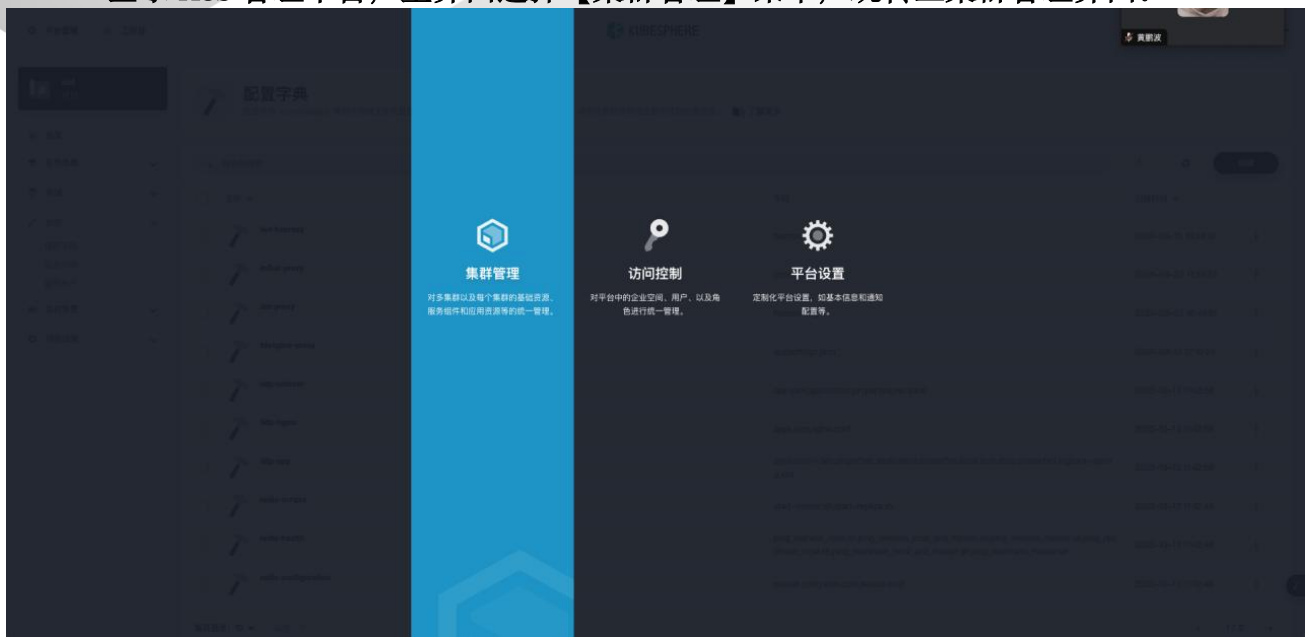
3.3.4. 授权租户管理账号

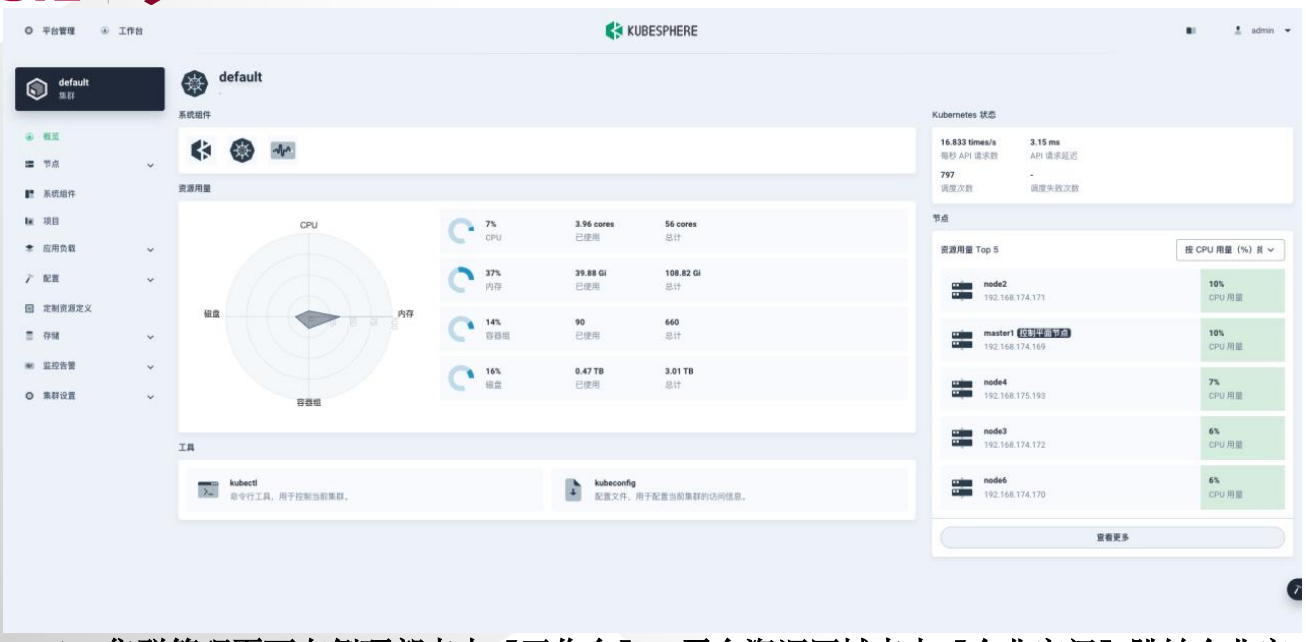
在【多租户】-->【租户管理】页面，在租户管理员账号行，点击【授权】，勾选工业物联网全部 App，点击【确定】完成授权。



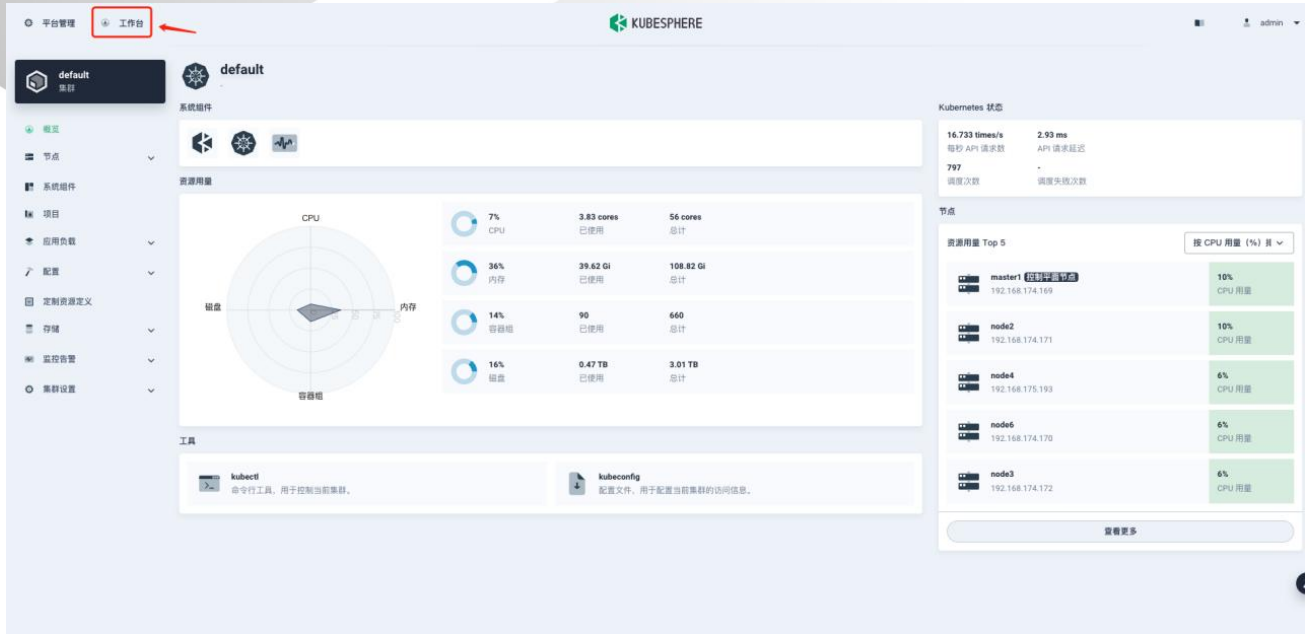
3.3.5. 安装 TDengineProxy 代理

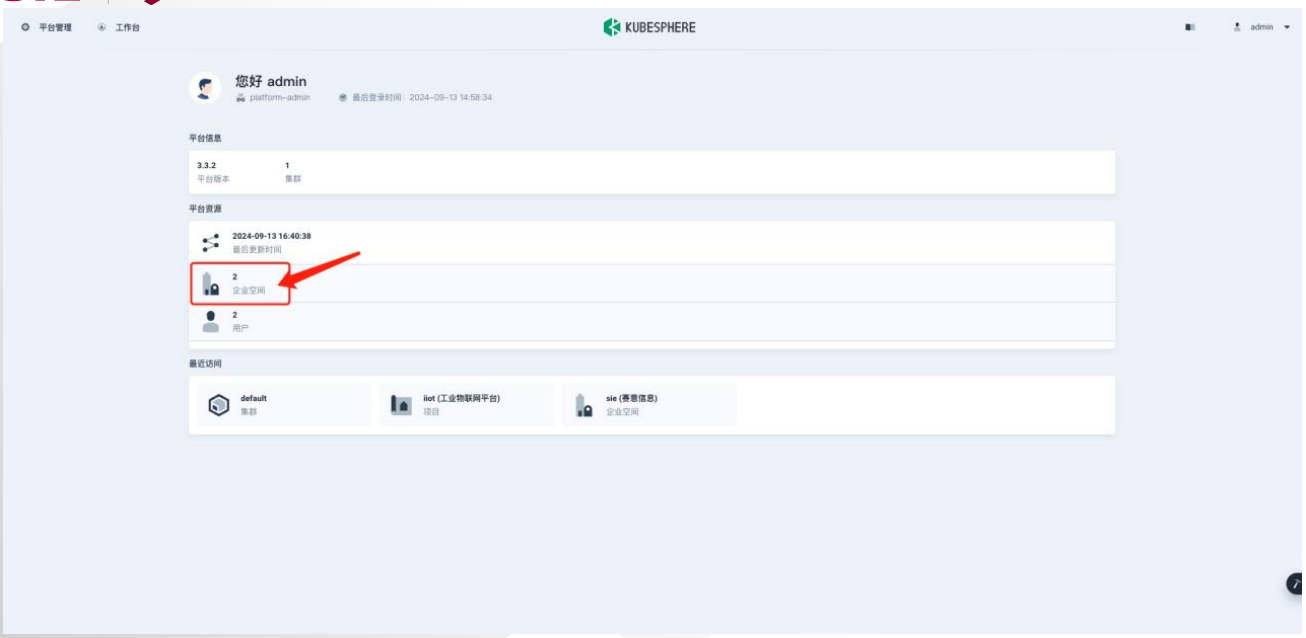
- 登录 K8S 管理平台，主界面选择【集群管理】菜单，跳转至集群管理界面。

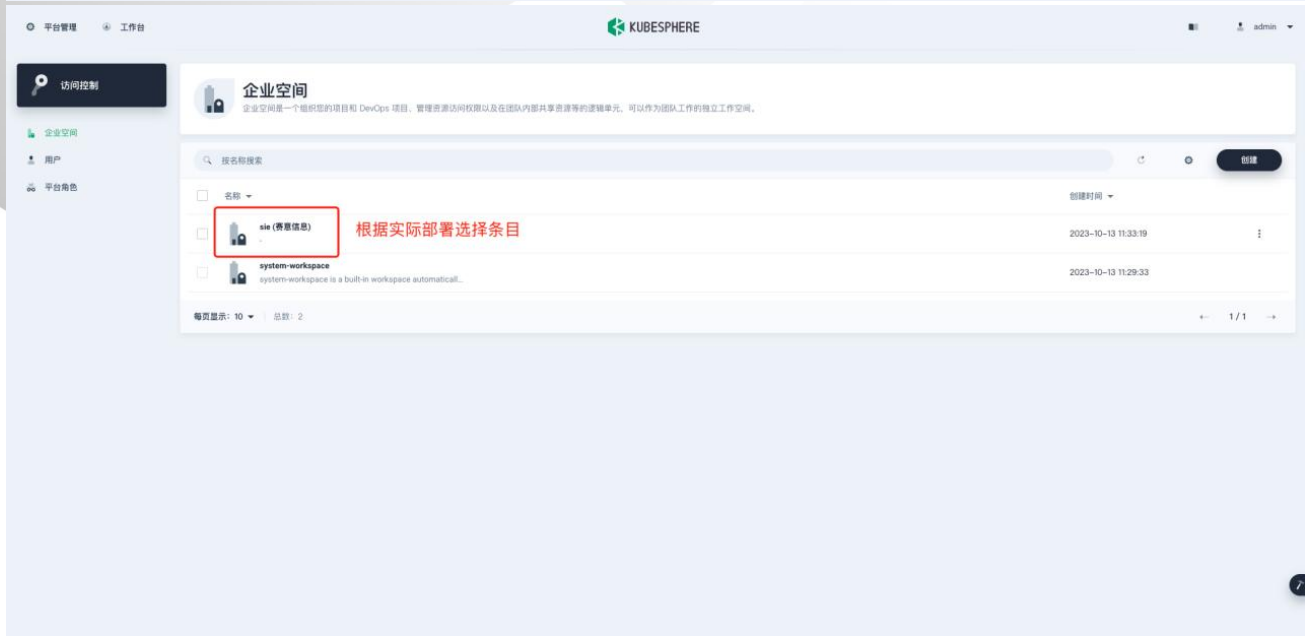


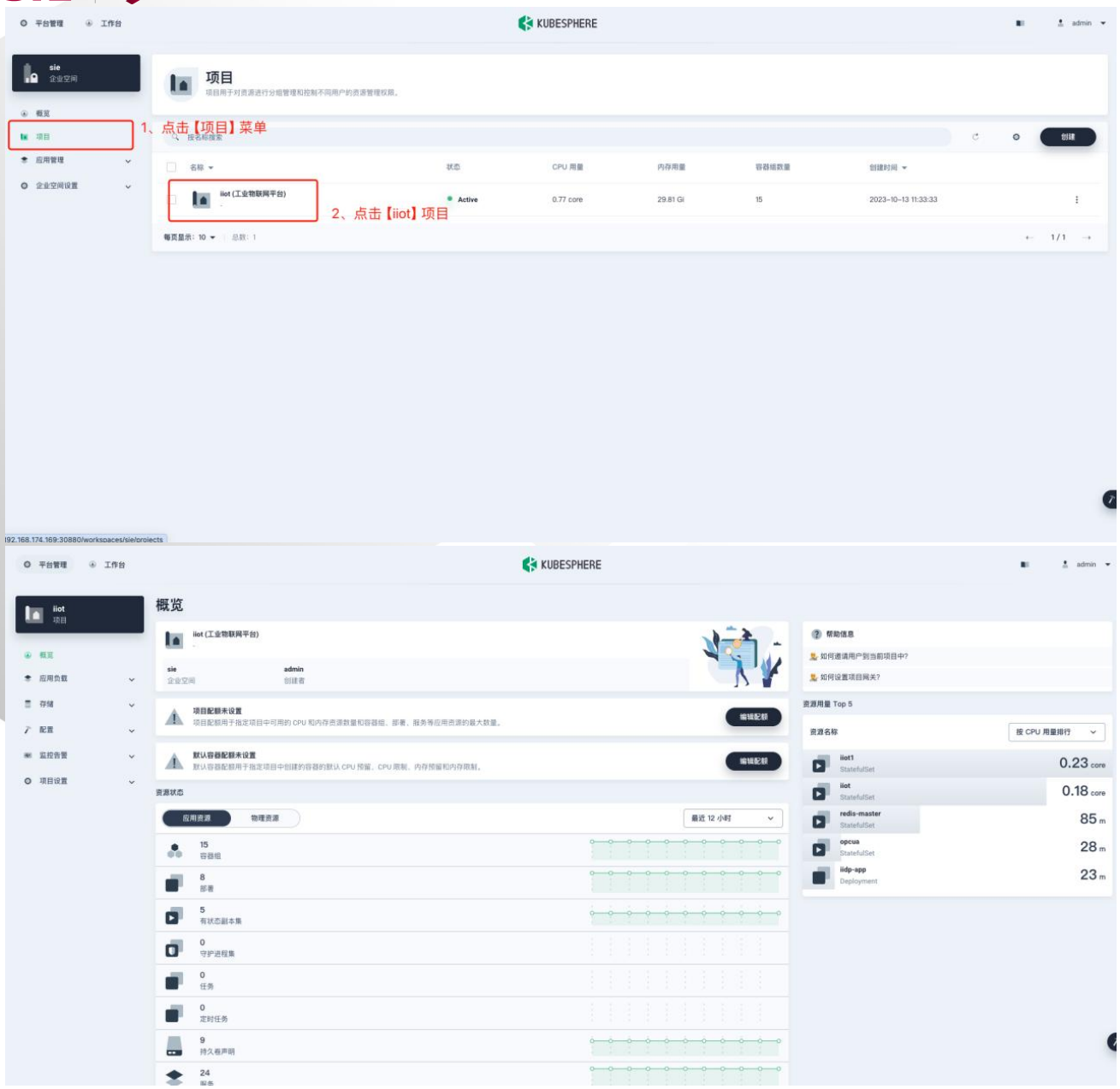


- 集群管理页面左侧顶部点击【工作台】，平台资源区域点击【企业空间】跳转企业空间页面，然后在中间区域选择【相应的企业空间】，接着在相应企业空间页面左侧选择【项目】菜单，再在中间区域选择【iiot】项目，跳转至项目管理页面。

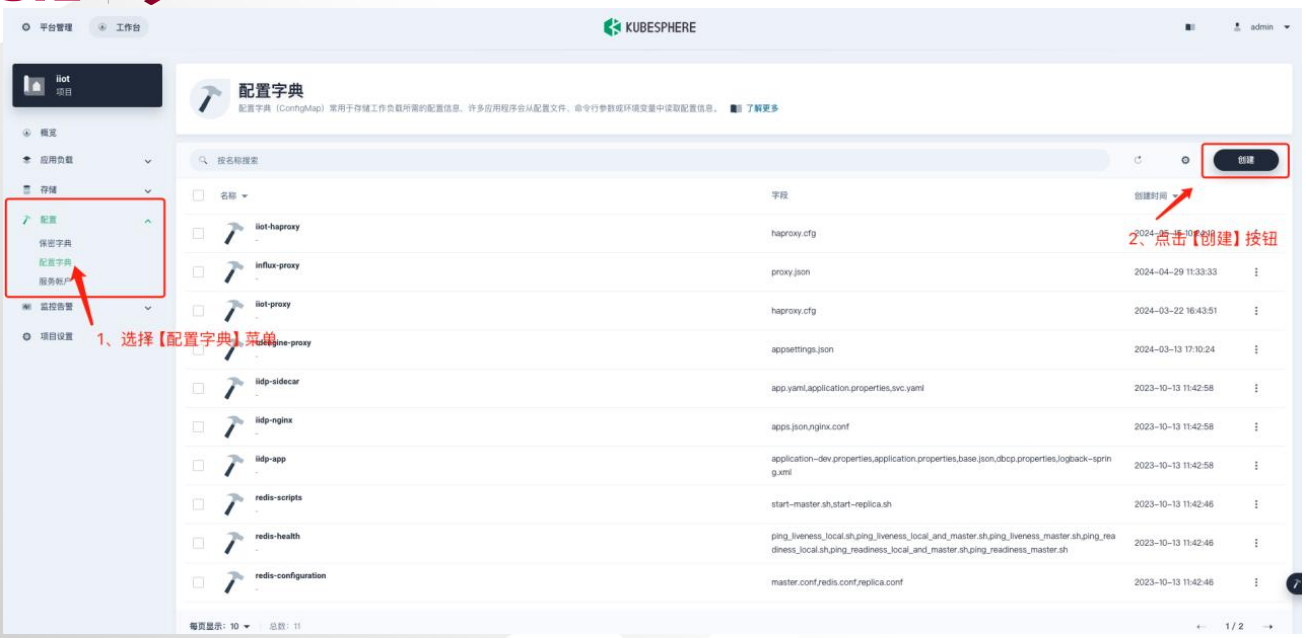


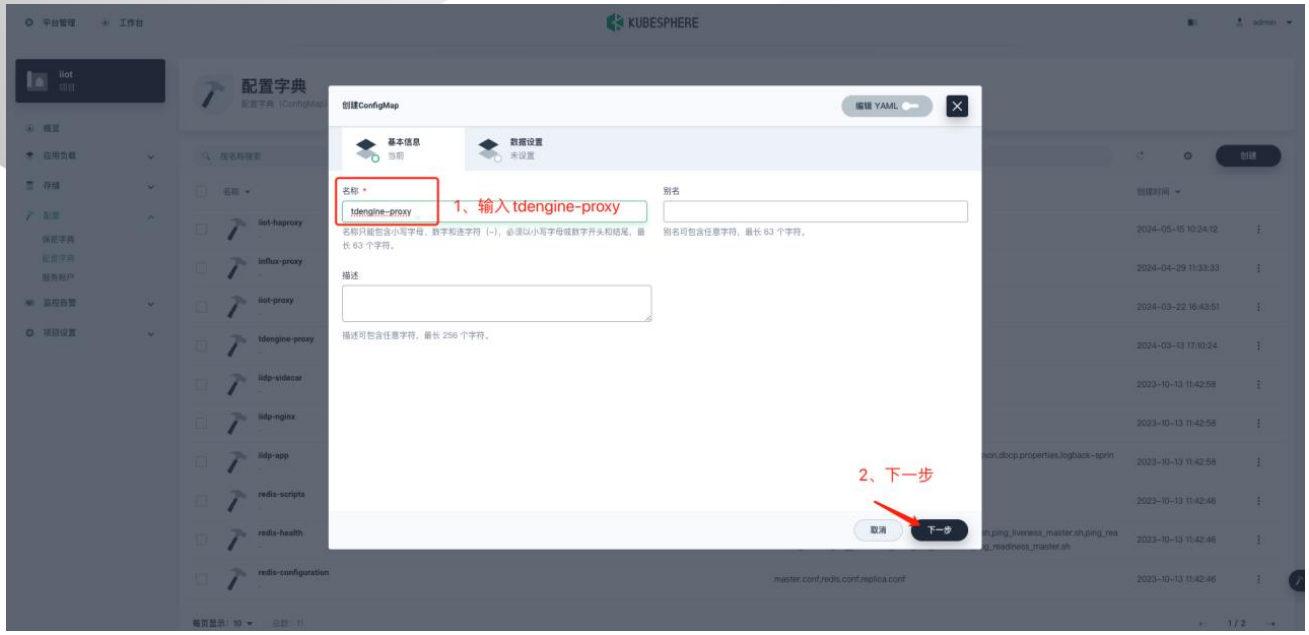

平台管理
工作台
KUBESPHERE
admin
您好 admin
platform-admin
平台信息
3.3.2
平台版本
1
集群
平台资源
2024-09-13 16:40:38
最后更新时间
2
企业空间
2
用户
最近访问
default
集群
iiot (工业物联网平台)
项目
sie (赛意信息)
企业空间

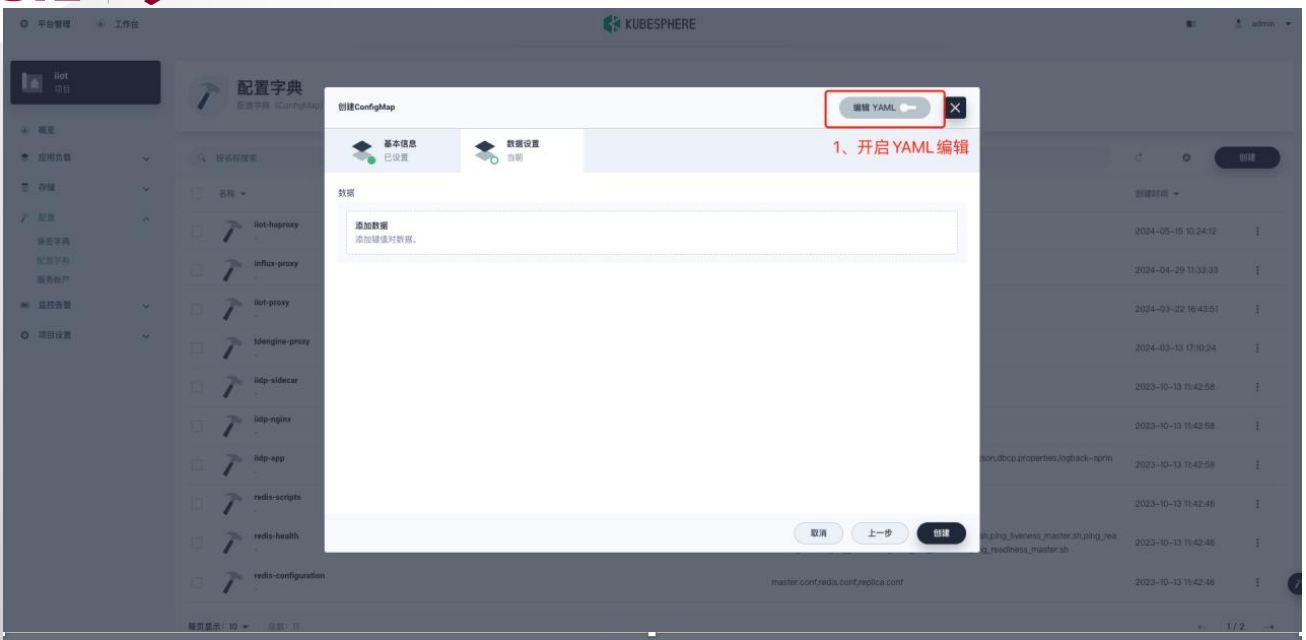

平台管理
工作台
KUBESPHERE
admin
访问控制
企业空间
用户
平台角色
企业空间
创建
名称
创建时间
sie (赛意信息)
根据实际部署选择条目
2023-10-13 11:33:19
system-workspace
2023-10-13 11:29:33
每页显示: 10
总数: 2
1 / 1

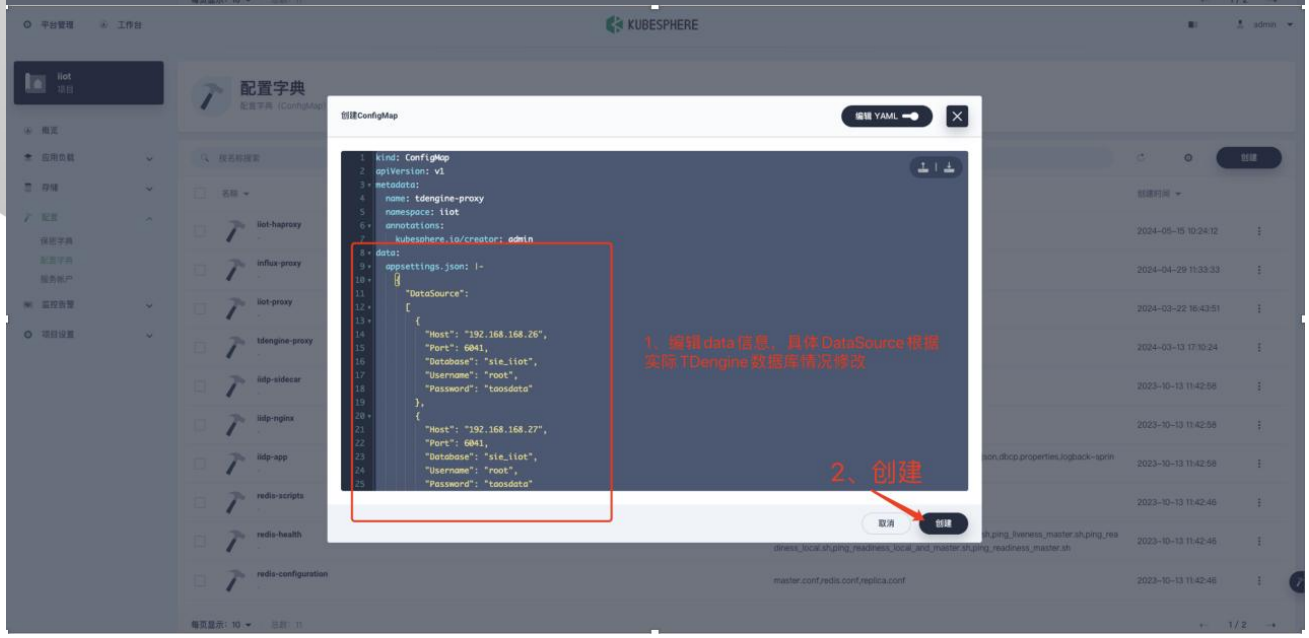


- 项目页面左侧选择【配置】->【配置字典】菜单，右上角点击【创建】按钮创建 tdengine-proxy 字典。


KUBESPHERE
配置字典
1、选择【配置字典】菜单
2、点击【创建】按钮


KUBESPHERE
配置字典
创建ConfigMap
编辑 YAML
基本信息
数据设置
名称 *
tdengine-proxy
1、输入 tdengine-proxy
别名
描述
2、下一步
取消
下一步



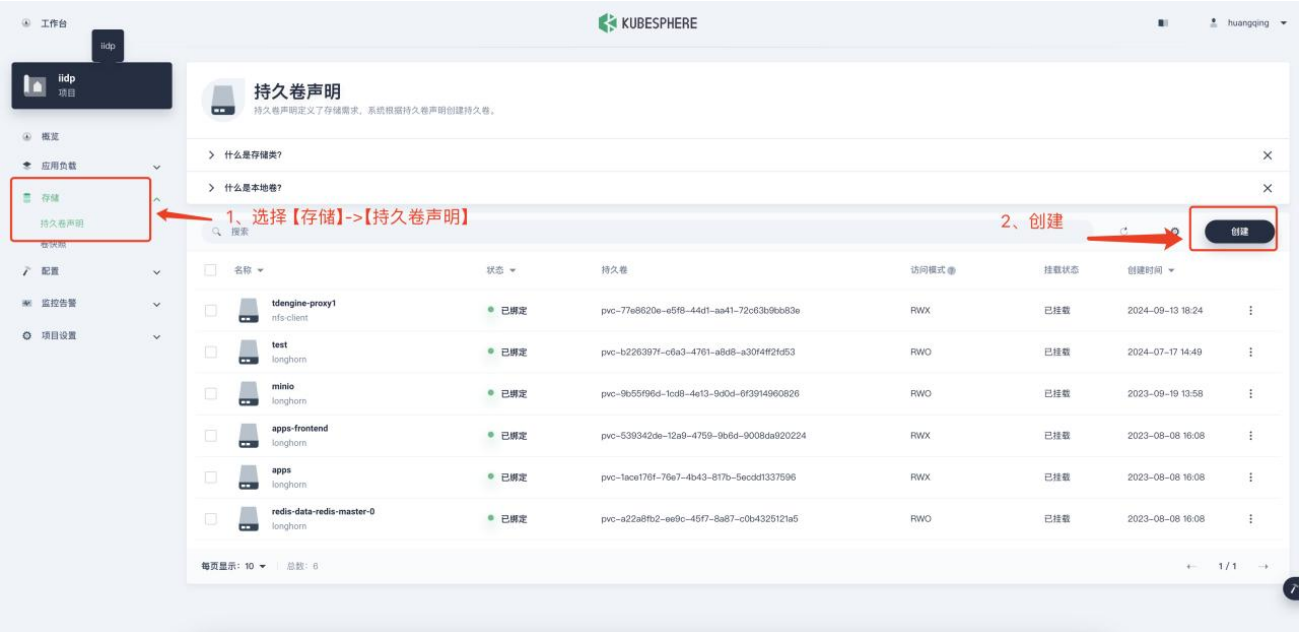


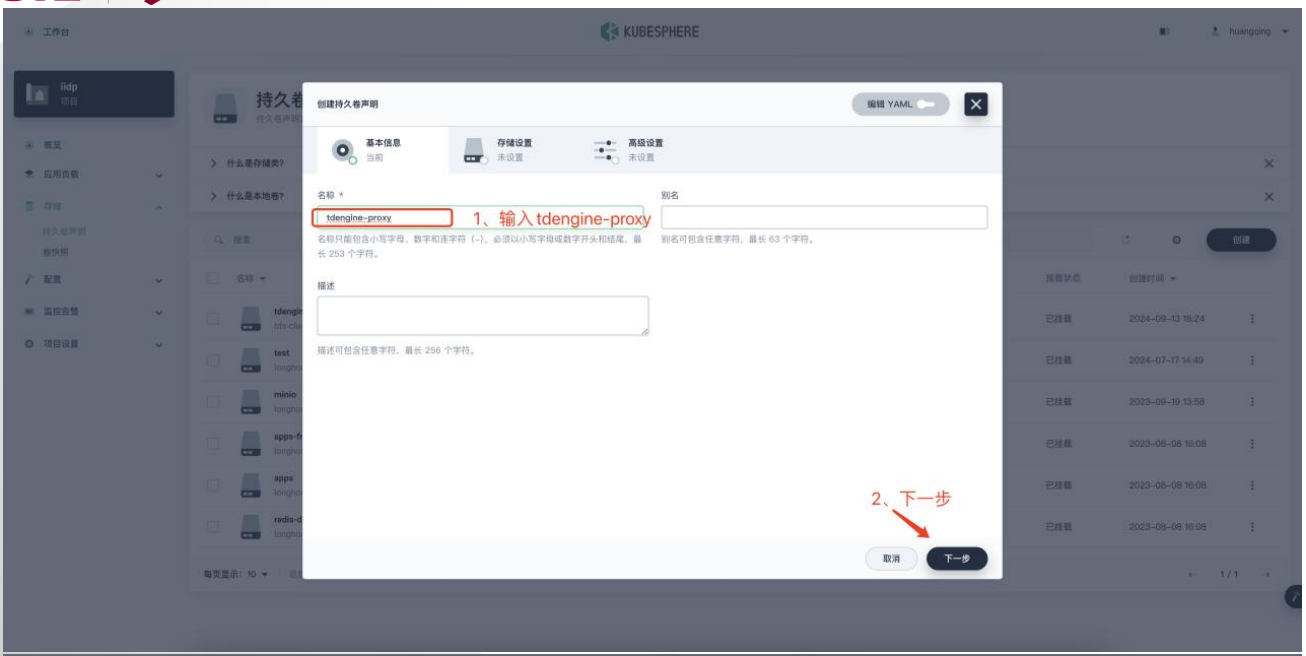
配置文件内容（DataSource 部分按照实际 TDengine 数据库情况修改）：

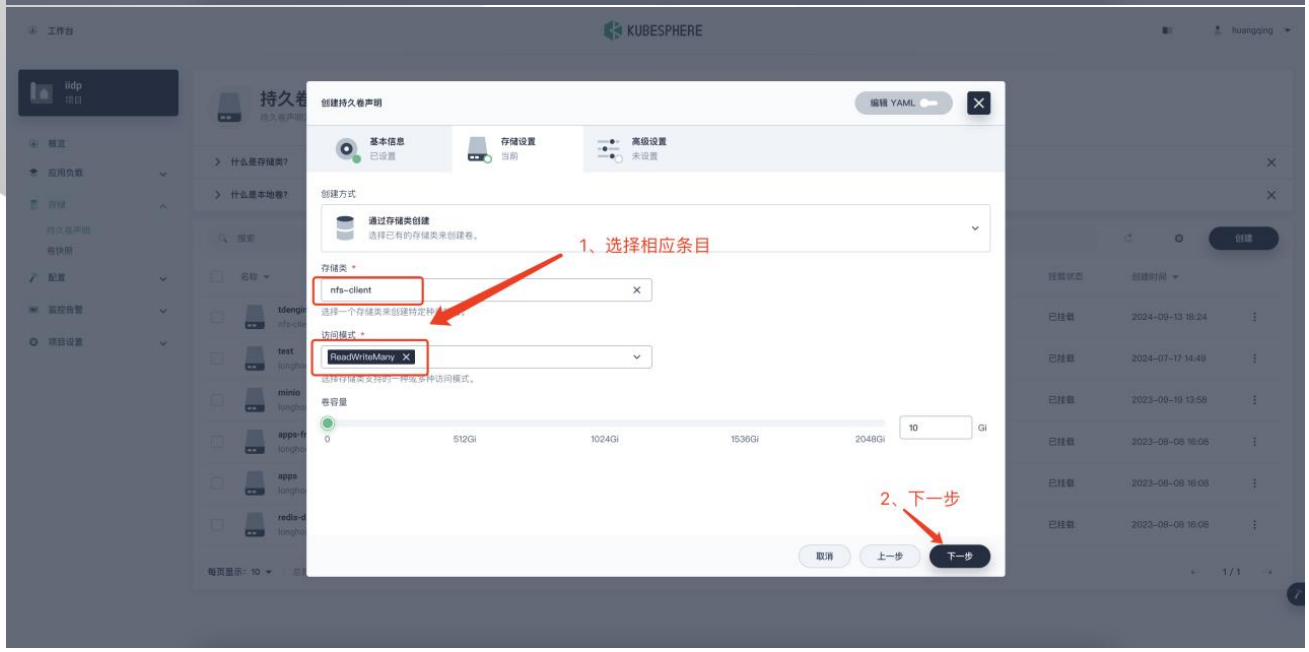
```
kind: ConfigMap
apiVersion: v1
metadata:
    name: tdengine-proxy
    namespace: iiot
    annotations:
        kubesphere.io/creator: admin
data:
    appsettings.json: |-
        {
            "DataSource":
            [
```

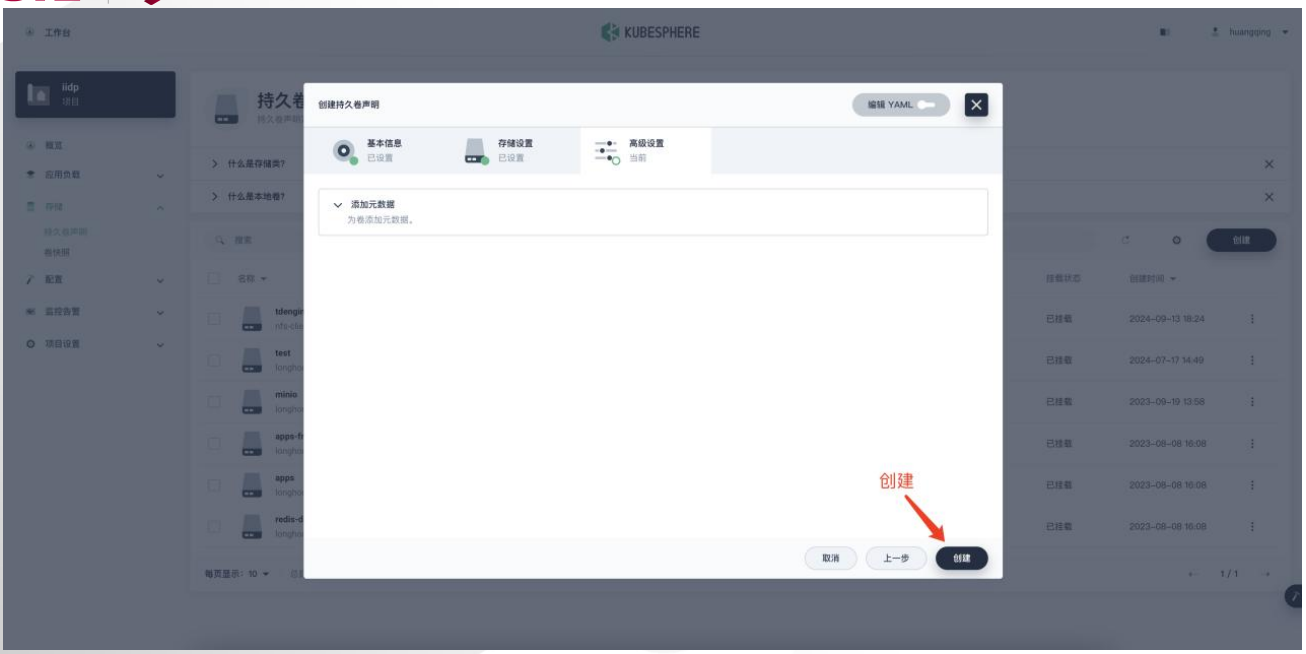
```
        {
            "Host": "192.168.168.26",
            "Port": 6041,
            "Database": "sie_iiot",
            "Username": "root",
            "Password": "taosdata"
        },
        {
            "Host": "192.168.168.27",
            "Port": 6041,
            "Database": "sie_iiot",
            "Username": "root",
            "Password": "taosdata"
        },
        {
            "Host": "192.168.168.28",
            "Port": 6041,
            "Database": "sie_iiot",
            "Username": "root",
            "Password": "taosdata"
        }
    ],
    "BatchSize": 5000,
    "IsDebug": false
}
```

- 项目页面左侧选择【存储】->【持久卷声明】菜单，右上角点击【创建】按钮创建 tdengine-proxy 存储。

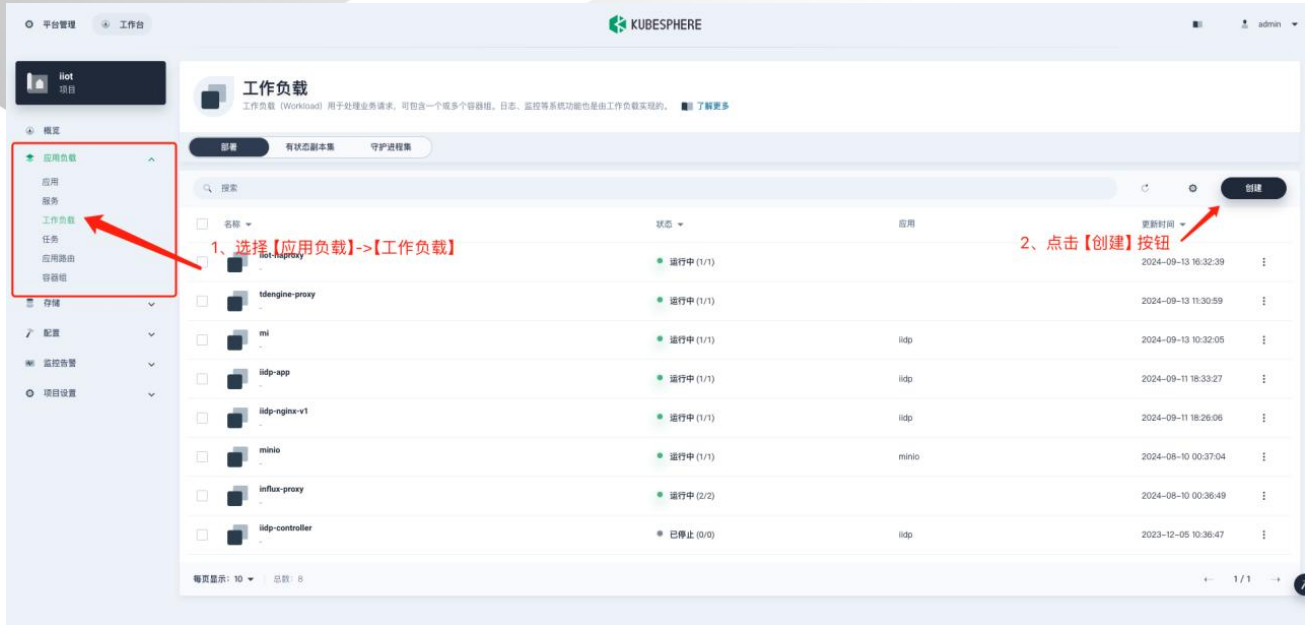


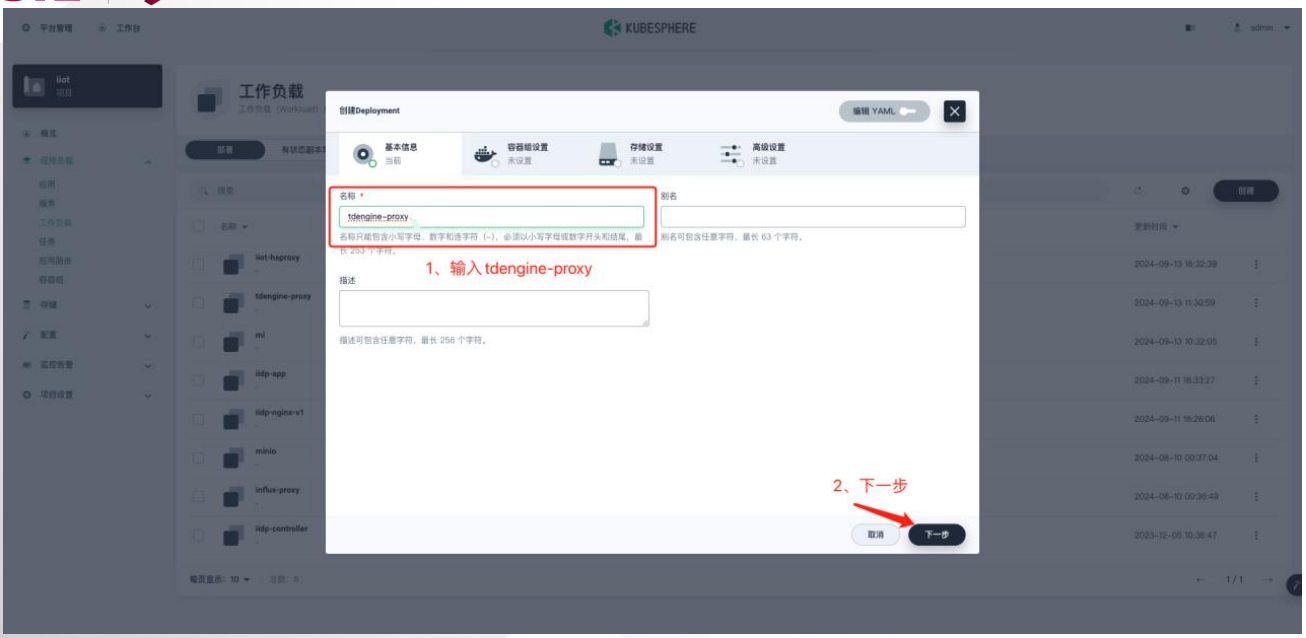

KUBESPHERE
创建持久卷声明
编辑 YAML
基本信息
当前
存储设置
未设置
高级设置
未设置
名称 *
tdengine-proxy
1、输入tdengine-proxy
别名
描述
2、下一步
取消
下一步

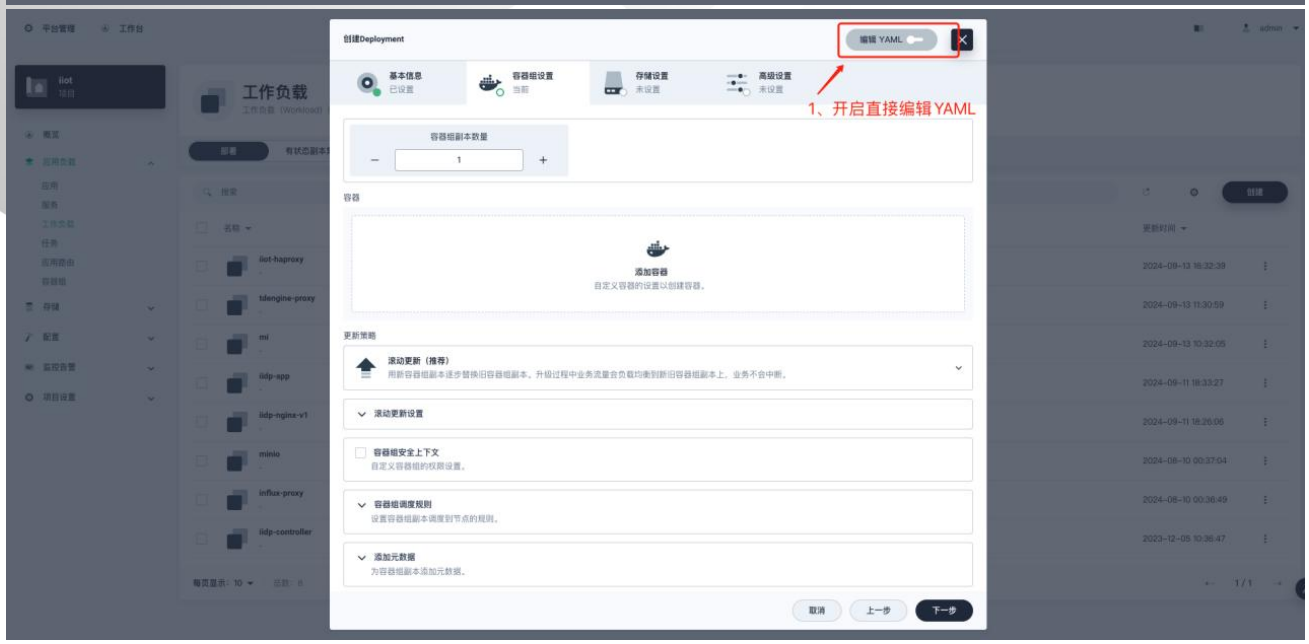

KUBESPHERE
创建持久卷声明
编辑 YAML
基本信息
已设置
存储设置
当前
高级设置
未设置
创建方式
通过存储类创建
选择已有的存储类来创建卷。
1、选择相应条目
存储类 *
nfs-client
访问模式 *
ReadWriteMany
卷容量
0
512Gi
1024Gi
1536Gi
2048Gi
10
Gi
2、下一步
取消
上一步
下一步

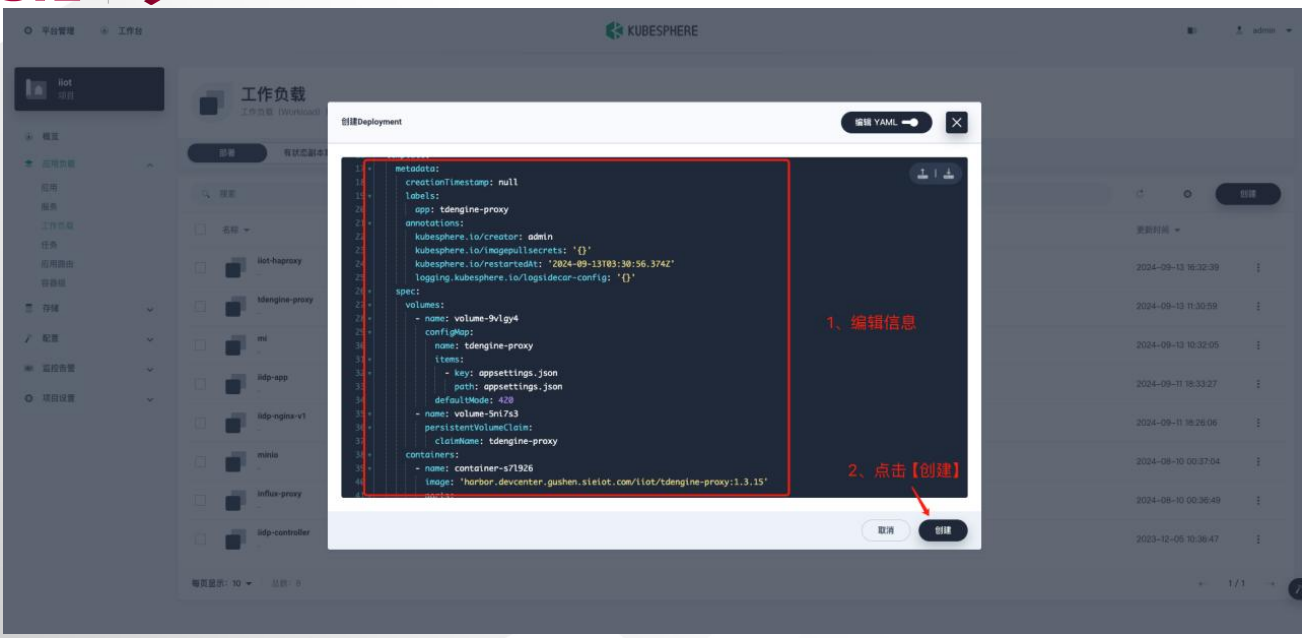


- 项目页面左侧选择【应用负载】->【工作负载】菜单，右上角点击【创建】按钮创建 tdengine-proxy 负载。




KUBESPHERE
工作负载
创建Deployment
编辑 YAML
基本信息
容器组设置
存储设置
高级设置
名称 *
tdengine-proxy
别名
1、输入tdengine-proxy
描述
2、下一步
取消
下一步


创建Deployment
编辑 YAML
基本信息
容器组设置
存储设置
高级设置
1、开启直接编辑 YAML
容器组副本数量
容器
添加容器
更新策略
滚动更新（推荐）
滚动更新设置
容器组安全上下文
容器组调度规则
添加元数据
取消
上一步
下一步



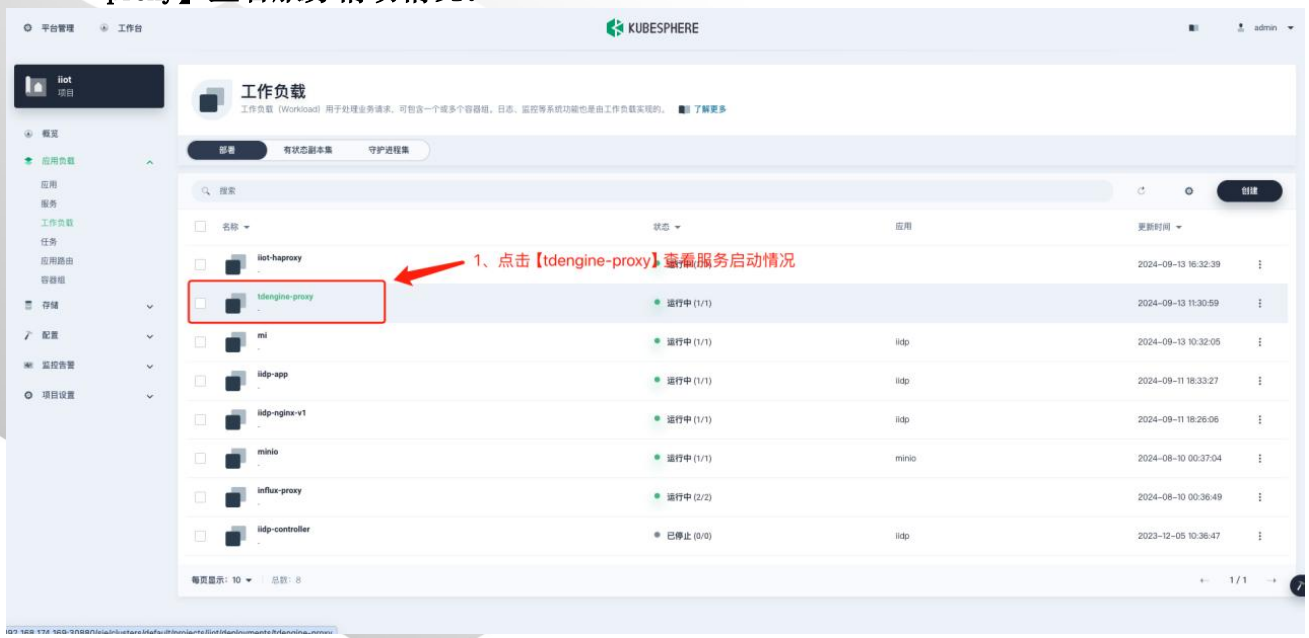
配置文件内容（TDengineProxy 镜像版本号根据实际修改，此处使用 1.3.15 版本）：

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: tdengine-proxy
  namespace: iiot
  labels:
    app: tdengine-proxy
  annotations:
    deployment.kubernetes.io/revision: '42'
    kubesphere.io/creator: admin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tdengine-proxy
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: tdengine-proxy
      annotations:
        kubesphere.io/creator: admin
        kubesphere.io/imagepullsecrets: '{}'
        kubesphere.io/restartedAt: '2024-09-13T03:30:56.374Z'
        logging.kubesphere.io/logsidecar-config: '{}'
    spec:
      volumes:
```

```
      - name: volume-9vlgy4
        configMap:
          name: tdengine-proxy
          items:
            - key: appsettings.json
              path: appsettings.json
          defaultMode: 420
      - name: volume-5ni7s3
        persistentVolumeClaim:
          claimName: tdengine-proxy
  containers:
    - name: container-s71926
      image: 'harbor.devcenter.gushen.sieiot.com/iiot/tdengine-proxy:1.3.15'
      ports:
        - name: http-80
          containerPort: 80
          protocol: TCP
      resources: {}
      volumeMounts:
        - name: volume-9vlgy4
          readOnly: true
          mountPath: /app/appsettings.json
          subPath: appsettings.json
        - name: volume-5ni7s3
          mountPath: /app/tmp
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      imagePullPolicy: IfNotPresent
  restartPolicy: Always
  terminationGracePeriodSeconds: 30
  dnsPolicy: ClusterFirst
  serviceAccountName: default
  serviceAccount: default
  securityContext: {}
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchLabels:
                app: tdengine-proxy
            topologyKey: kubernetes.io/hostname
  schedulerName: default-scheduler
```

```
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 25%
  revisionHistoryLimit: 10
  progressDeadlineSeconds: 600
```

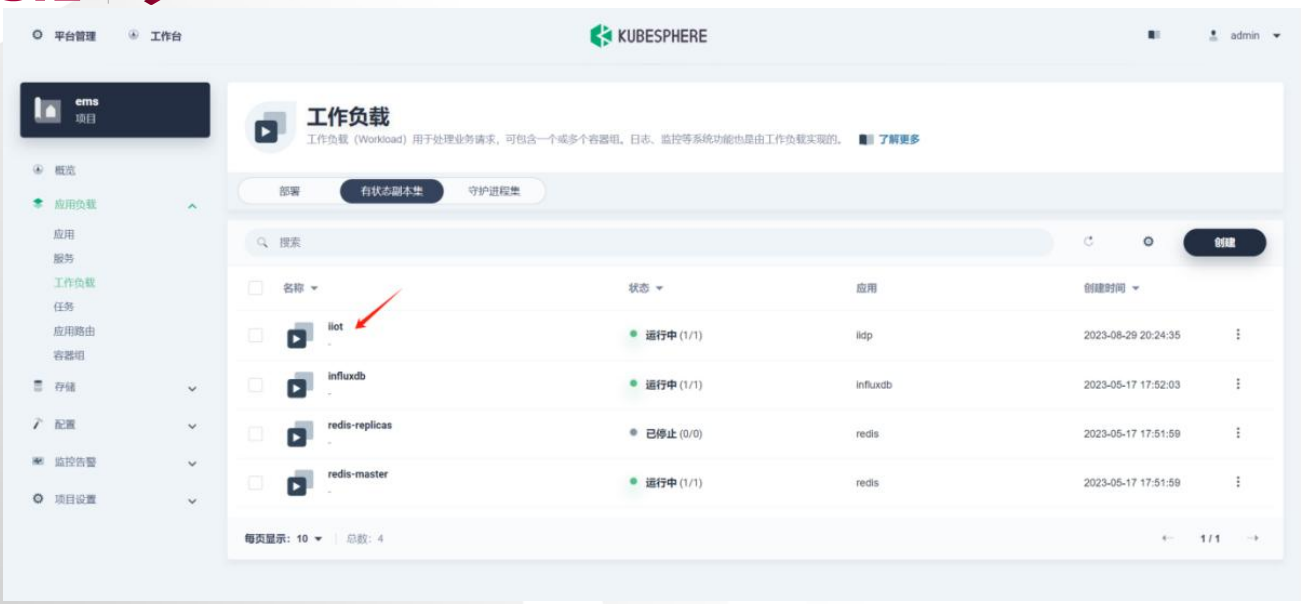
- 项目管理页面左侧选择【应用负载】->【工作负载】菜单，中间区域选择【tdengine-proxy】查看服务情动情况。

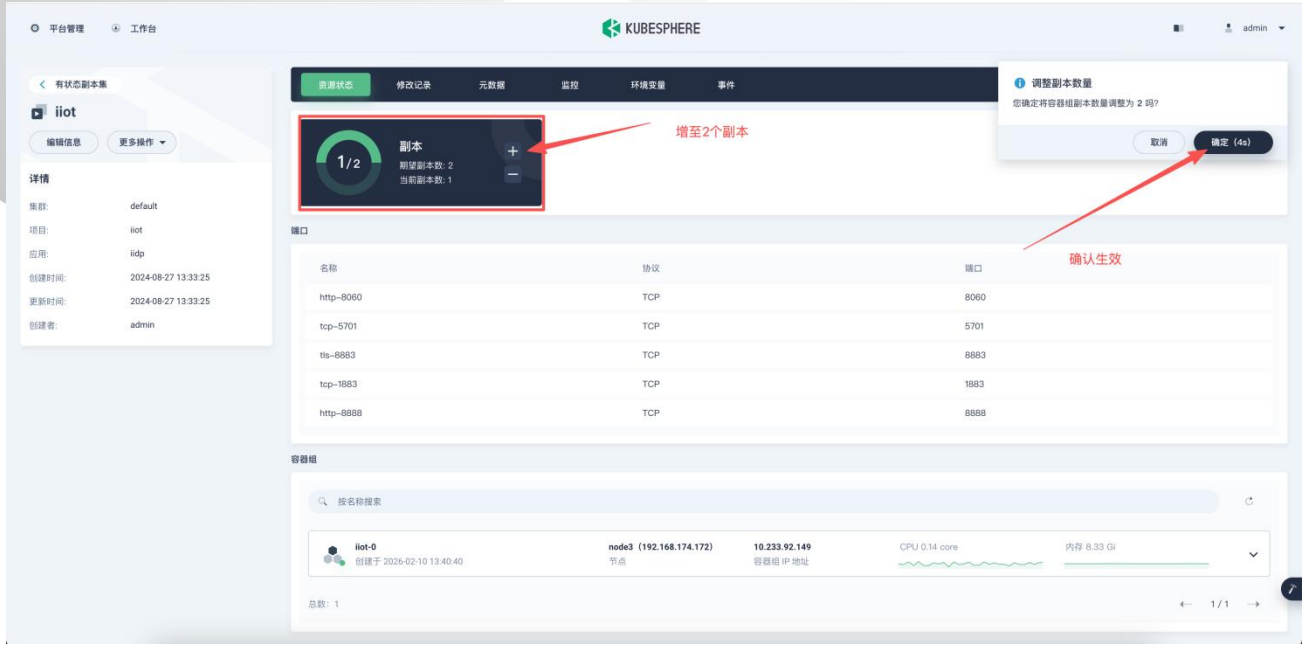


3.3.6. 环境配置

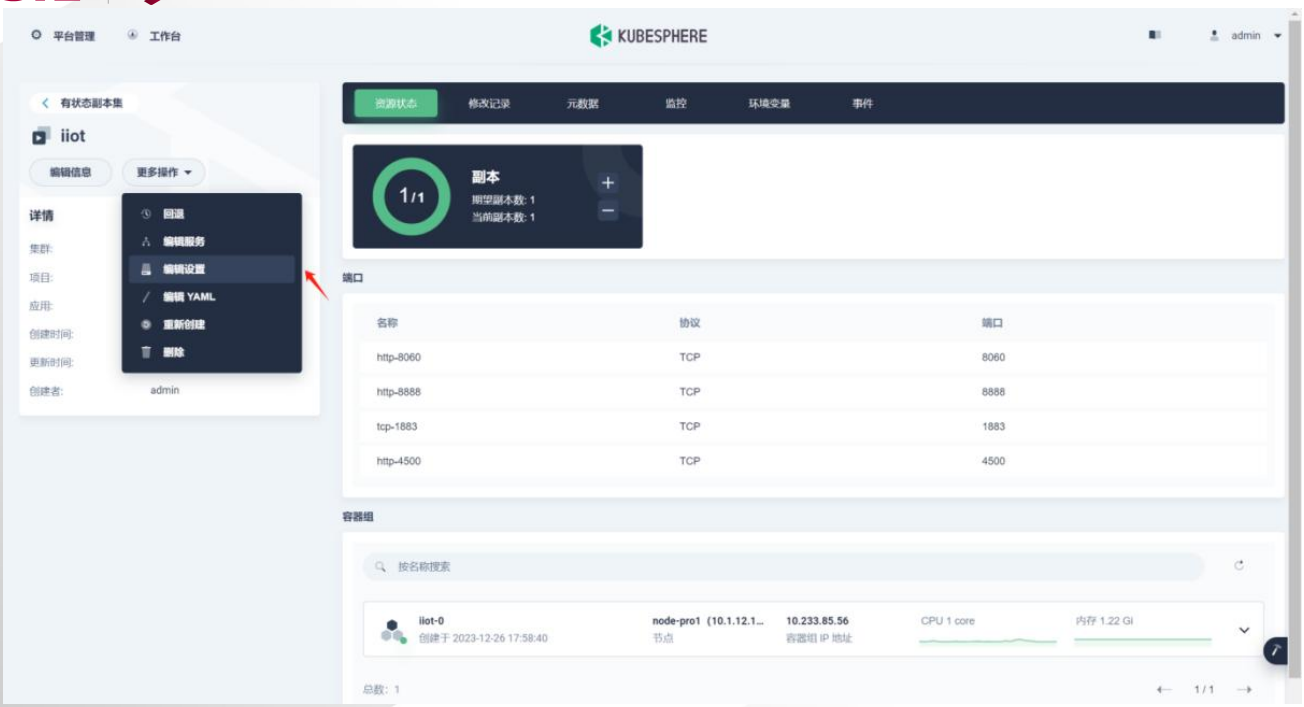
3.3.6.1. 工作负载配置

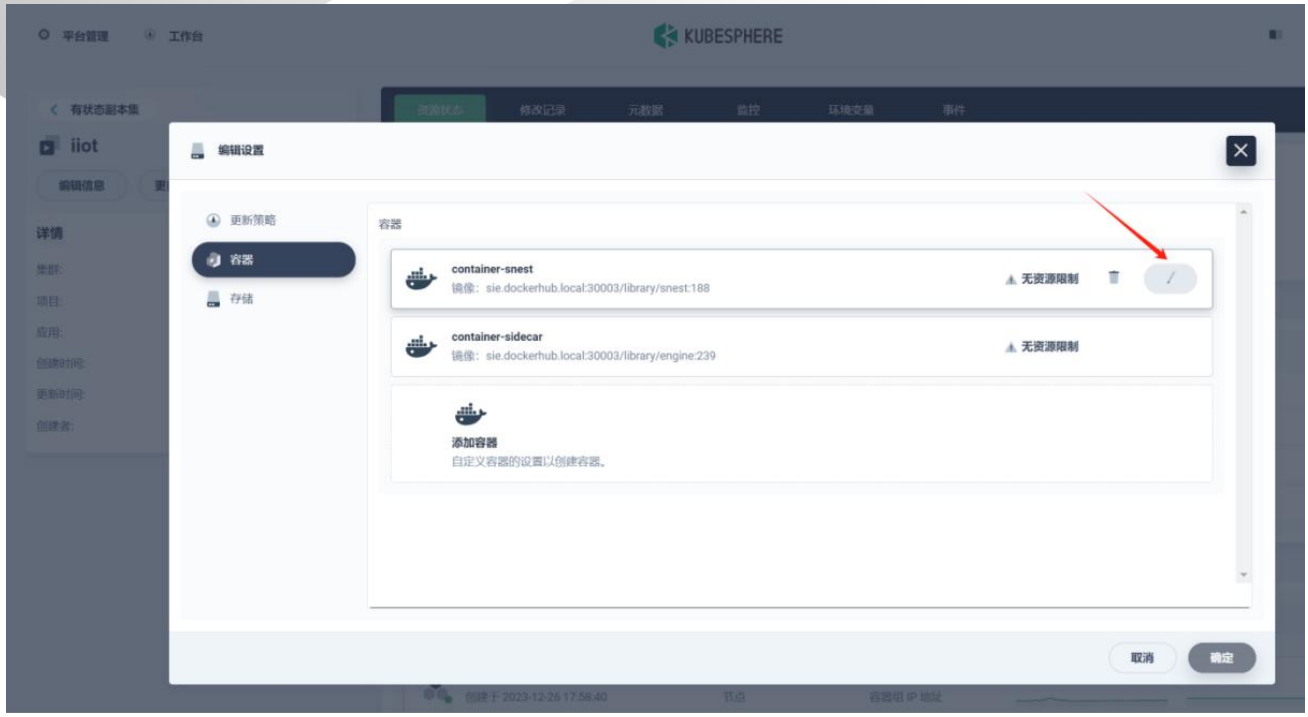
- 登录 kubesphere 管理平台，进入项目所属集群页面，选择【应用负载】→【工作负载】→【有状态副本集】，依次选择 iiot 副本集，点击副本右侧“+”号，将副本数增至 2 个，再点击右上角弹框【确认】按钮保存。

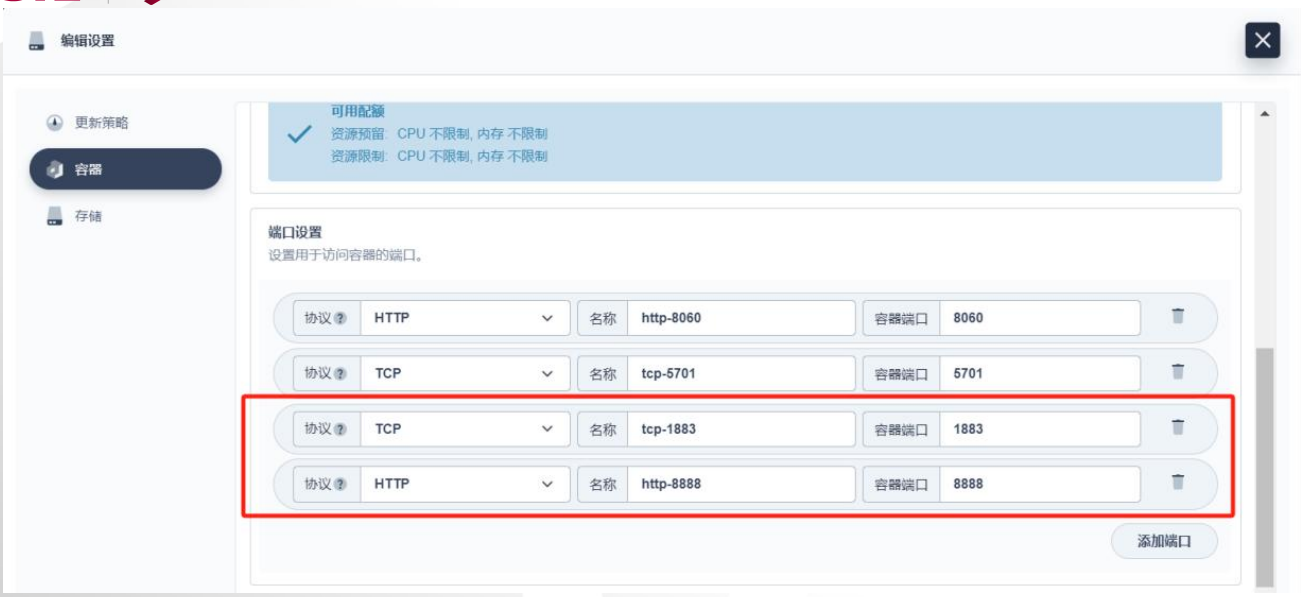




- 页面左上角选择【更多操作】→【编辑设置】，左侧选择【容器】，点击 container-snest 容器右侧【编辑】按钮，在【端口设置】部分参考截图新增 TCP-1883 和 HTTP-8888 配置，点击“√”，再点击【确定】按钮保存。


平台管理
工作台
KUBESPHERE
admin
有状态副本集
iiot
编辑信息
更多操作
详情
回退
编辑服务
编辑设置
编辑 YAML
重新创建
删除
资源状态
修改记录
元数据
监控
环境变量
事件
副本
期望副本数: 1
当前副本数: 1
端口
名称
协议
端口
http-8060
TCP
8060
http-8888
TCP
8888
tcp-1883
TCP
1883
http-4500
TCP
4500
容器组
按名称搜索
iiot-0
创建于 2023-12-26 17:58:40
node-pro1 (10.1.12.1...
节点
10.233.85.56
容器组 IP 地址
CPU 1 core
内存 1.22 Gi
总数: 1
1 / 1


平台管理
工作台
KUBESPHERE
有状态副本集
iiot
详情
编辑设置
更新策略
容器
存储
container-snest
镜像：sie.dockerhub.local:30003/library/snest:188
无资源限制
container-sidecar
镜像：sie.dockerhub.local:30003/library/engine:239
无资源限制
添加容器
自定义容器的设置以创建容器。
取消
确定



- 紧接上一步操作，在【环境变量】部分参考截图新增/修改 SW_OPTS 参数，参数值：-XX:NewRatio=1 -XX:InitialHeapSize=4g -XX:MaxHeapSize=16g -XX:ActiveProcessorCount=8 -XX:ParallelGCThreads=8 -XX:+UseStringDeduplication -XX:+UseCompressedOops -XX:MaxMetaspaceSize=512m -XX:NativeMemoryTracking=summary -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/apps/iiot/

- 页面左上角选择【更多操作】→【编辑 YAML】，在【securityContext】同级新增反亲和配置，点击“√”，再点击【确定】按钮保存（以下代码中的“iiot1”为当前工作负载名称）。

```
affinity:
  podAntiAffinity:
```

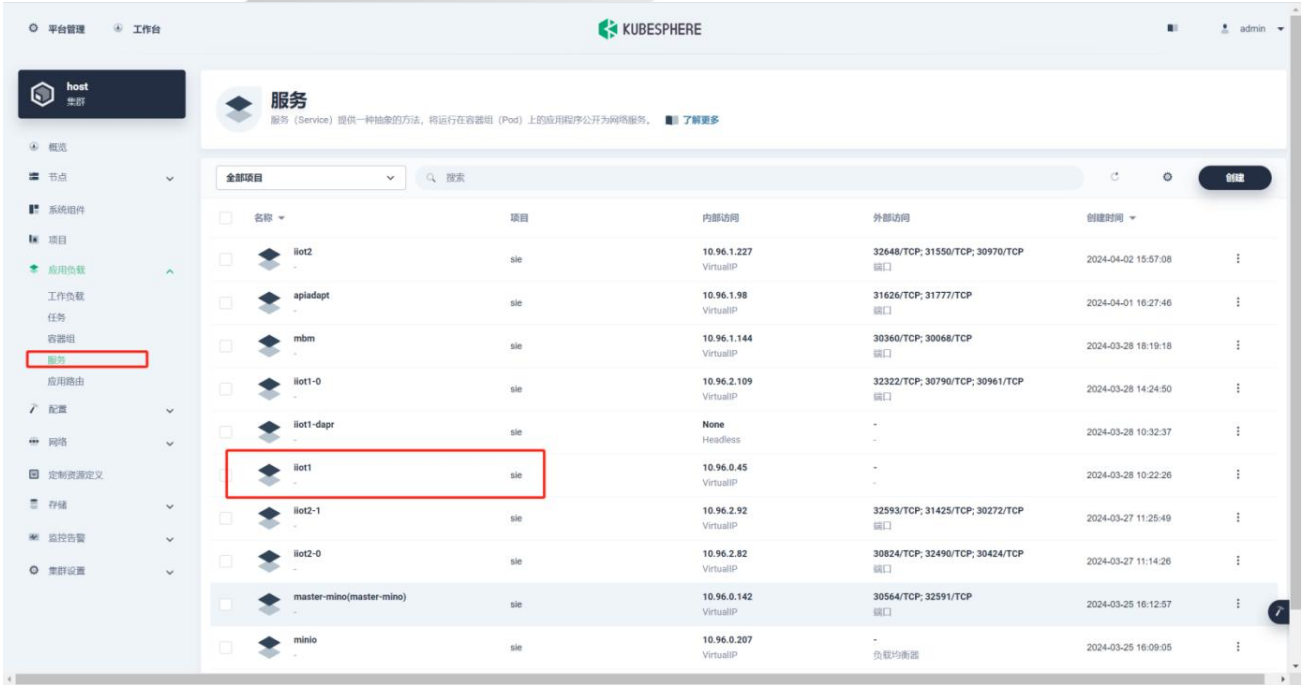
```
requiredDuringSchedulingIgnoredDuringExecution:
  - labelSelector:
      matchExpressions:
        - key: app
          operator: In
          values:
            - iiot1
    topologyKey: kubernetes.io/hostname
```

注意：如果安装了多个 iiot 服务，则按以上步骤重复查找相应工作负载并执行。

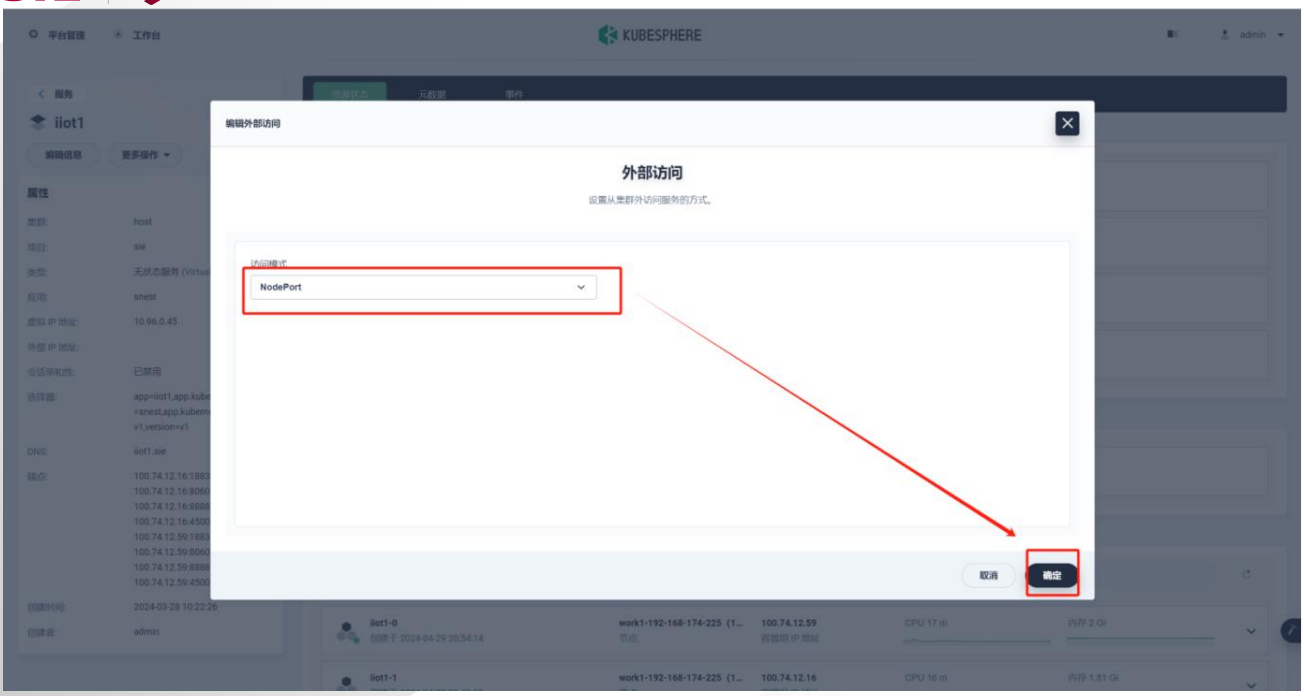
3.3.6.2. 服务配置

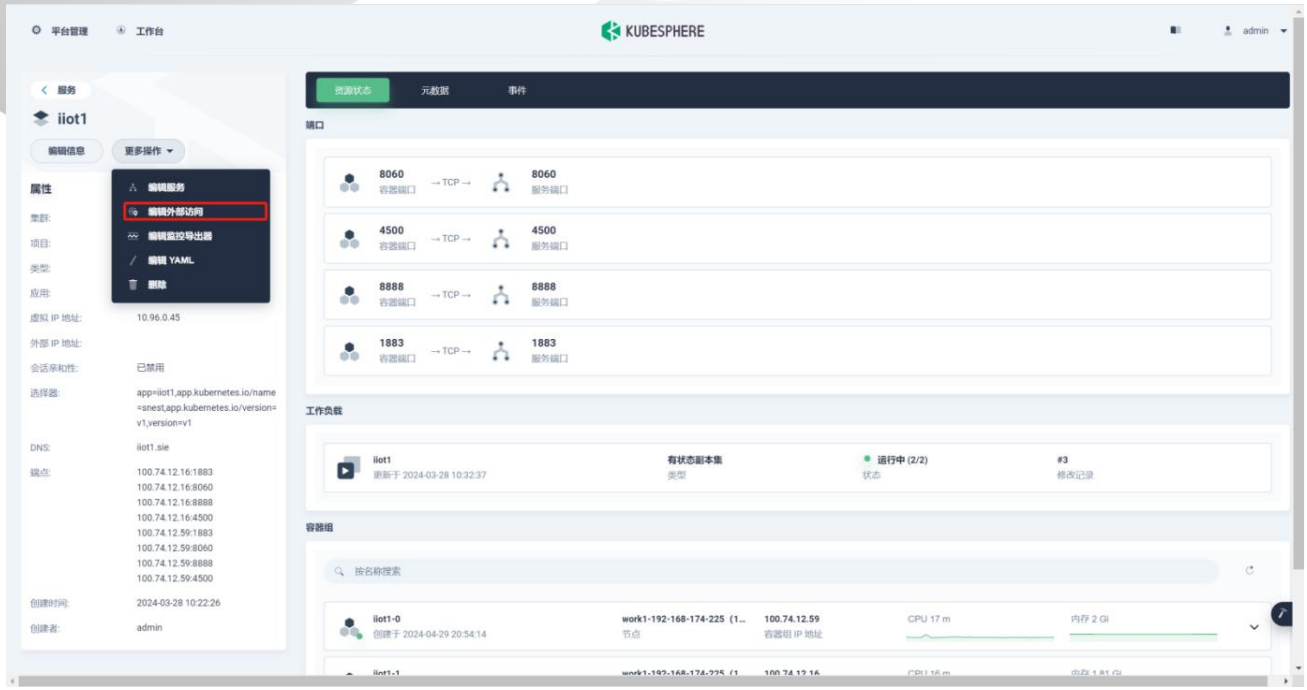
3.3.6.2.1. 配置负载服务

- 登录 kubesphere 管理平台，进入项目所属集群页面，选择【应用负载】→【服务】，查找 iiot1 服务（IIDP 部署 iiot1 服务自动生成），点击进入服务详情页面。

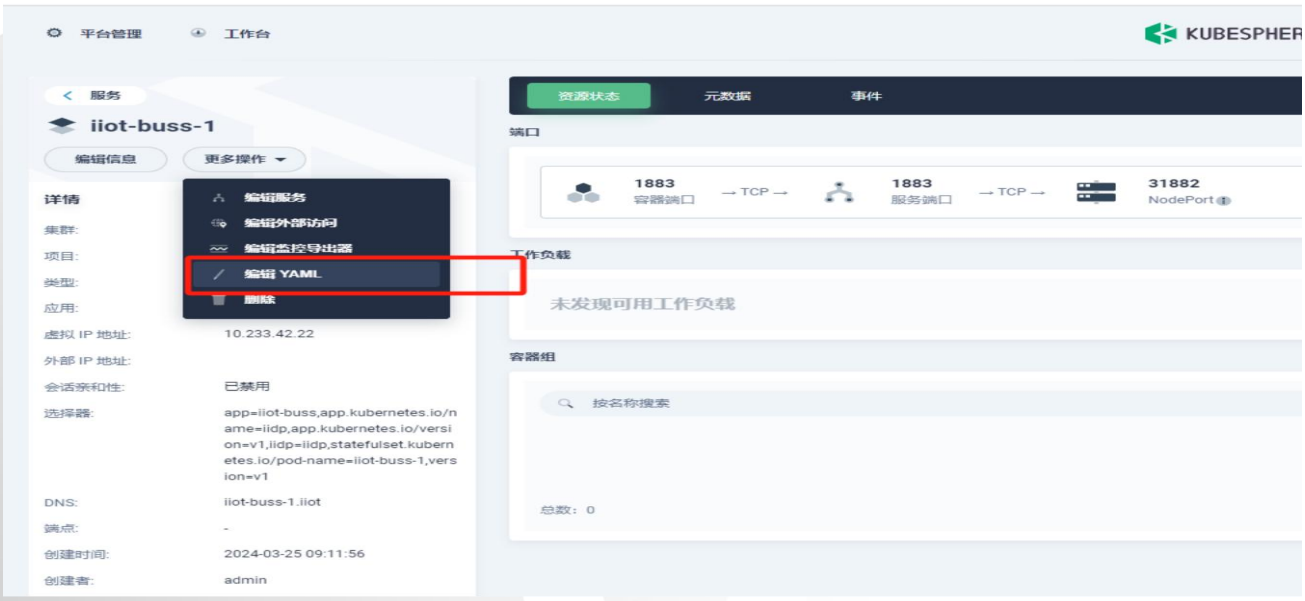


- 页面左上角选择【更多操作】→【编辑外部访问】，选择“NodePort”并点击【确定】按钮保存。





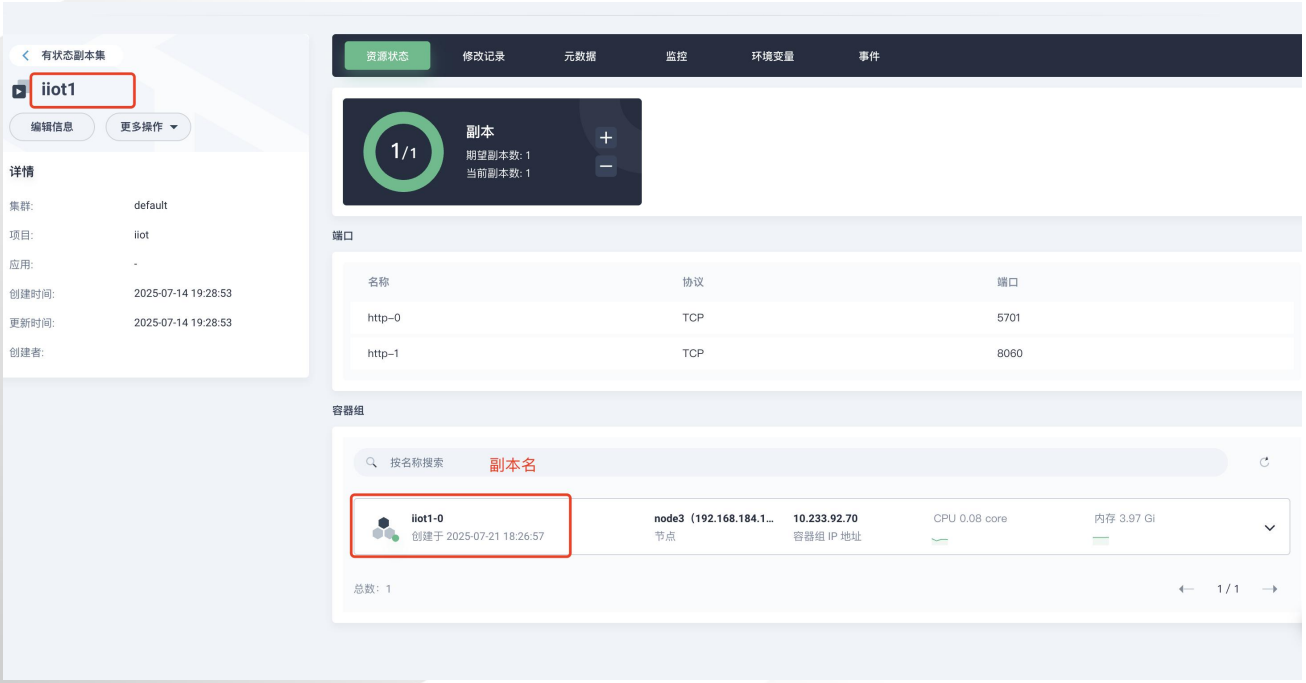
- 页面左上角选择【更多操作】→【编辑 YAML】，按端口开放清单，修改其中的对外开放端口号，并点击【确认】按钮保存。



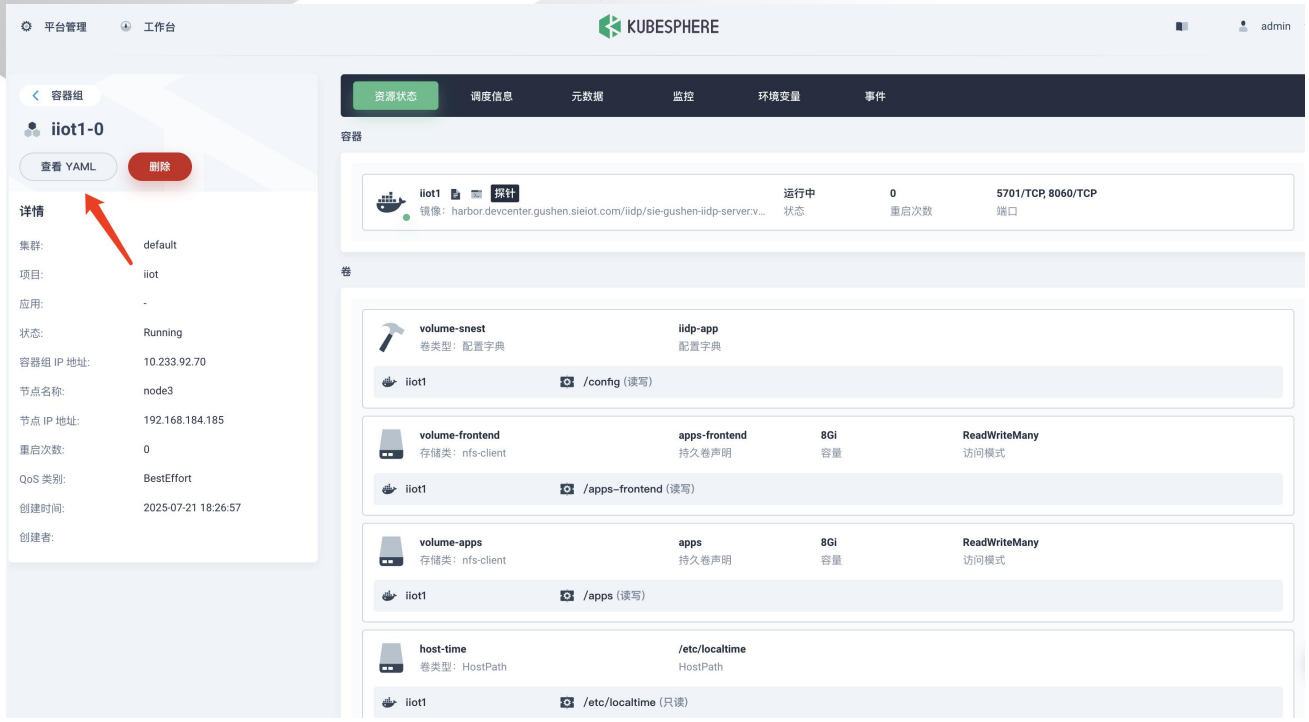
注意：如果在 IIDP 应用市场安装了多个 iiot 服务，则按以上步骤重复查找相应服务并执行。

3.3.6.2.2. 配置副本服务

- 登录 kubesphere 管理平台，进入项目所属集群页面，选择【应用负载】→【工作负载】→【有状态副本集】，查看有多少个 iiot 副本集，然后在【应用负载】→【服务】给每个 iiot 副本（假设 IIDP 安装了 iiot1 服务，则副本名称类似：iiot1-0、iiot1-1、...）创建一个服务。



➢ 点击副本名跳转到副本容器组页面，点击查看 YAML。



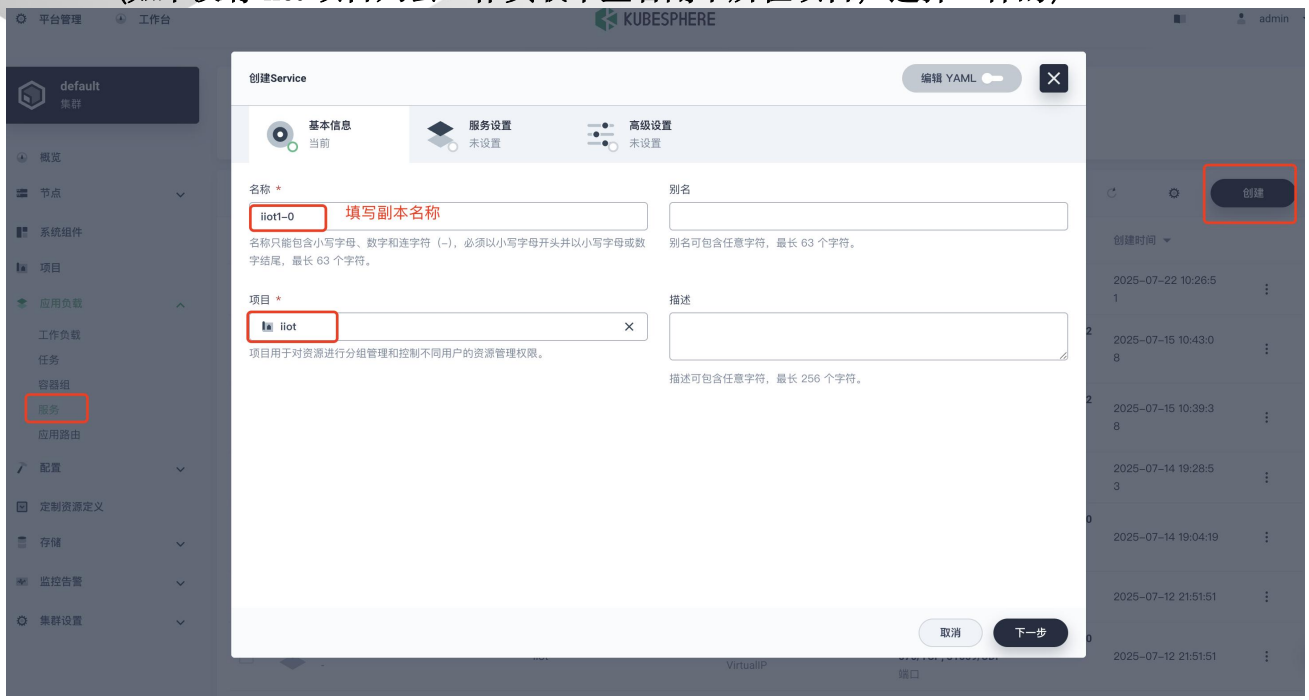
➢ 复制 yaml 文件中这三行之后要用

查看 YAML

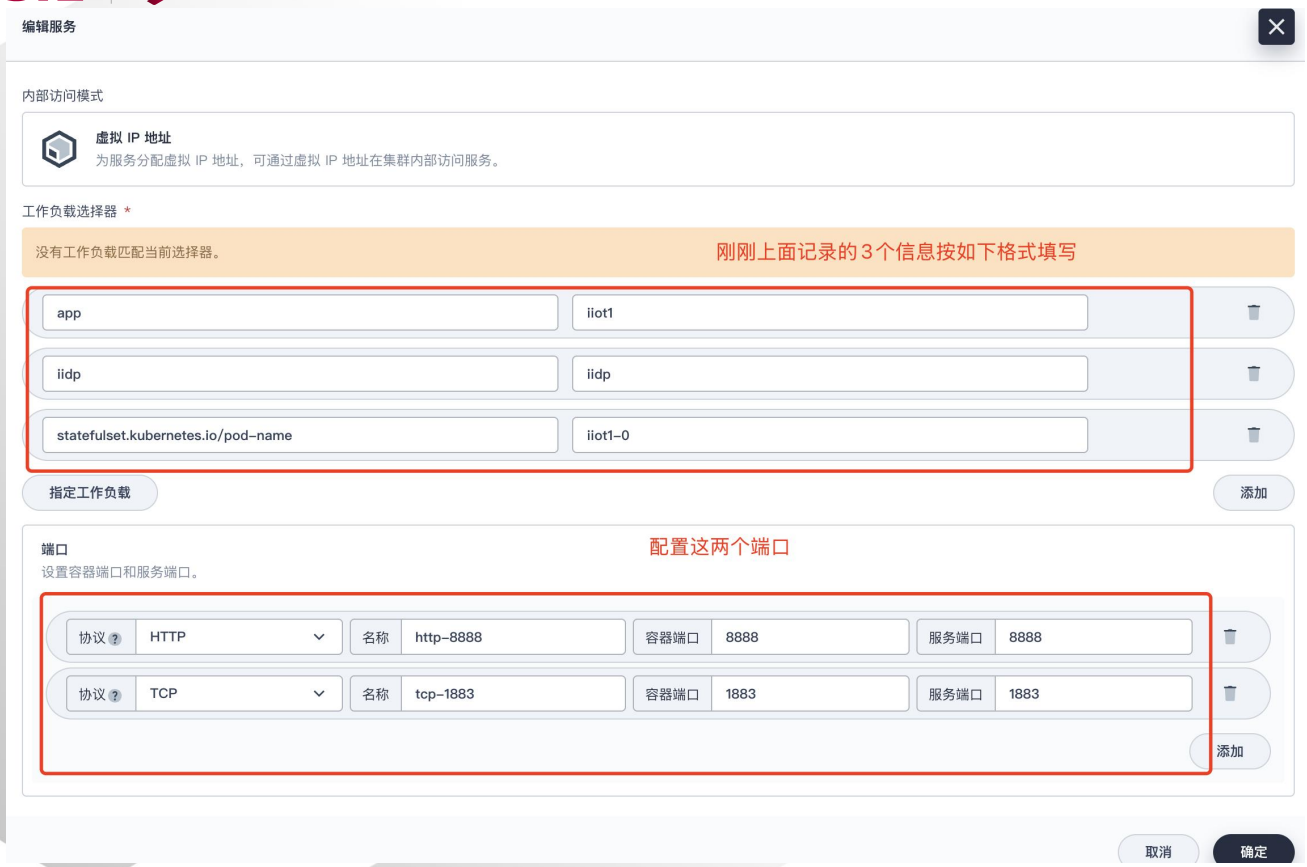
```yaml
kind: Pod
apiVersion: v1
metadata:
  name: iiot1-0
  generateName: iiot1-
  namespace: iiot
  labels:
    app: iiot1
    controller-revision-hash: iiot1-795d8d5555
    iidp: iidp
    statefulset.kubernetes.io/pod-name: iiot1-0
  annotations:
    cni.projectcalico.org/containerID: 4bc17144986846cbe95e2ab4d48735504b4e1f93406ad0282643f1a3e257316f
    cni.projectcalico.org/podIP: 10.233.92.70/32
    cni.projectcalico.org/podIPs: 10.233.92.70/32
    iidp.com/restartedAt: '2025-07-21T10:21:28.024Z'
    kubesphere.io/imagepullsecrets: '{}'
spec:
  volumes:
    - name: volume-snest
      configMap:
        name: iidp-app
        defaultMode: 420
    - name: volume-frontend
      persistentVolumeClaim:
        claimName: apps-frontend
    - name: volume-apps
      persistentVolumeClaim:
        claimName: apps
    - name: host-time
      hostPath:
        path: /etc/localtime
        type: ''
    - name: kube-api-access-xb57n
```

复制这三行，之后要用

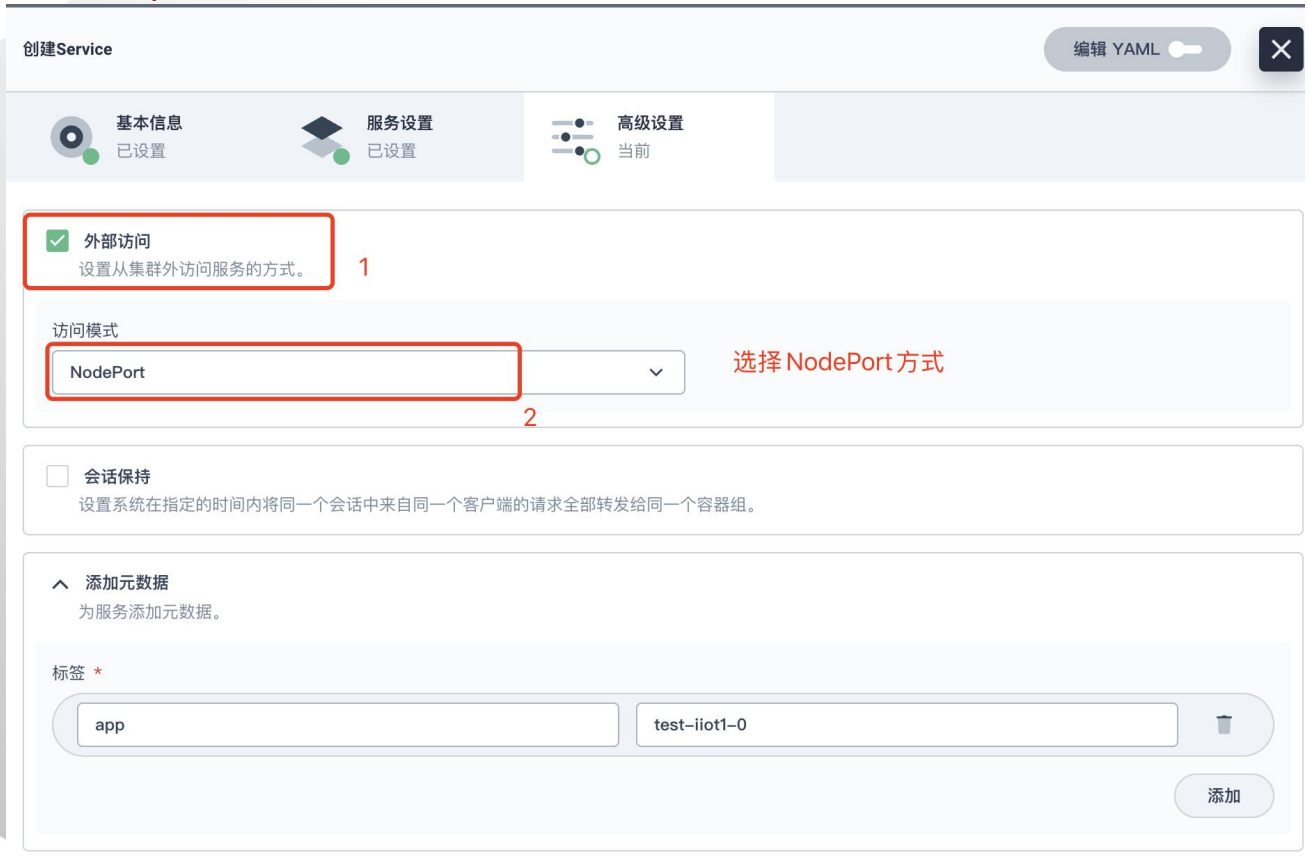
- 进入当前项目的服务列表，点击“创建”，填写和副本名一样的名称，选择 iiot 项目(如果没有 iiot 项目则去工作负载中查看副本所在项目，选择一样的)

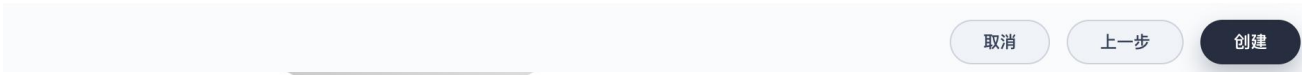


- 下一步，按照图片方式填写

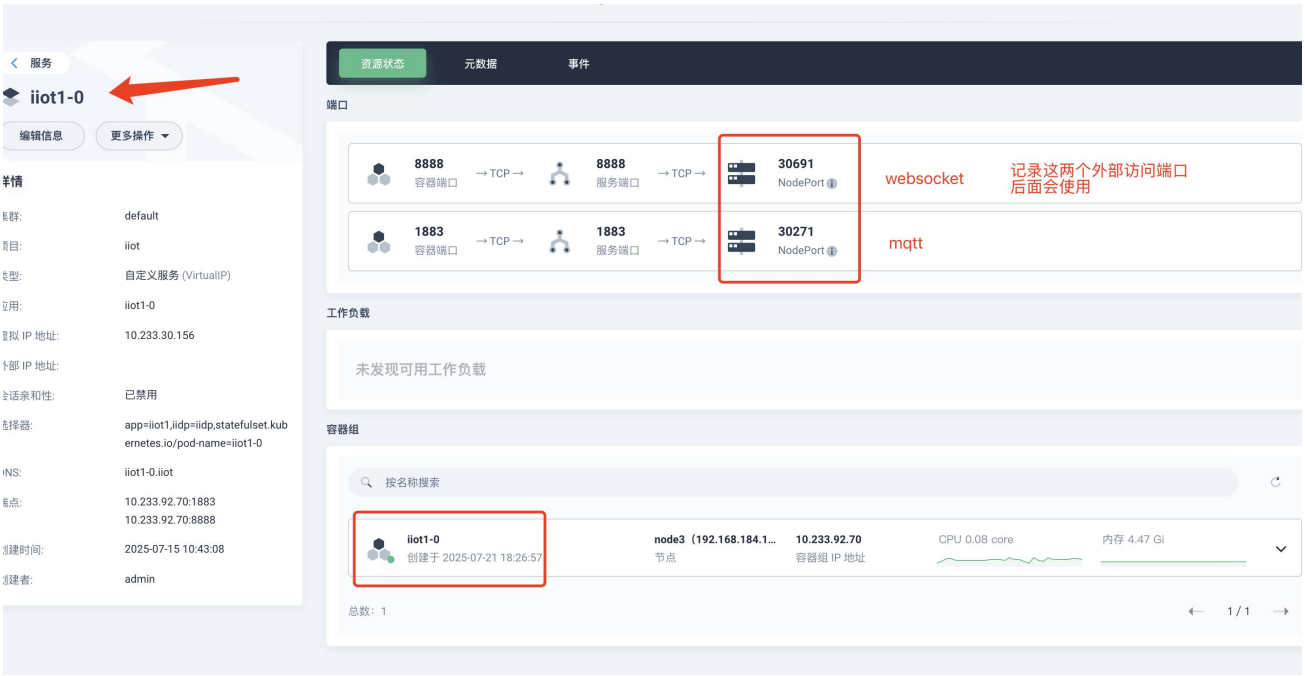


- 下一步，选择 NodePort 访问模式，点击创建





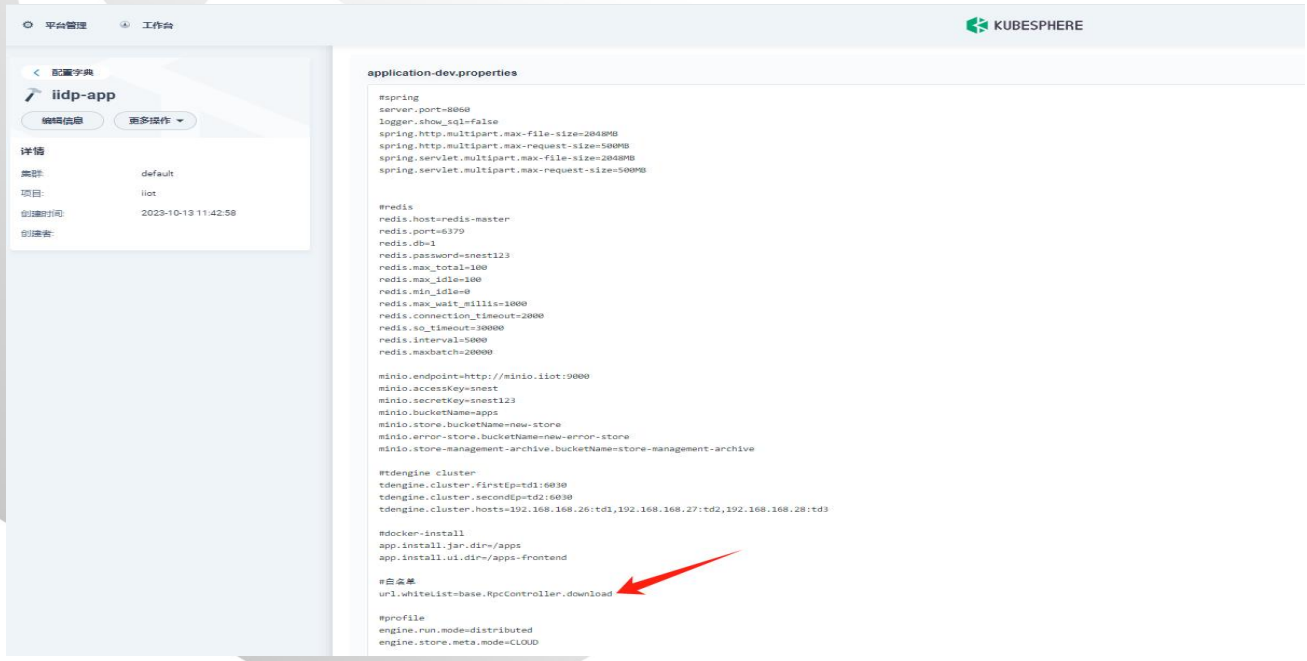
➢ 再次点击进入 iiot1-0 服务，记录服务分配的 2 个类型的外部访问端口，之后在系统配置时要使用



- 到此一个副本已配置完成，若有多个 iiot 服务和副本需要对不同服务、副本重复以上配置。

3.3.6.3. 白名单配置

- 检查平台配置文件 application-dev.properties 中是否存在如下配置，如果不存在，需添加并重启容器（配置路径：【配置】→【配置字典】→【iidp-app】）



配置内容：

```
# 白名单
url.whiteList=base.RpcController.download
```

3.3.7. 系统配置

使用租户管理员账号登录 IIOT 系统，在【系统运维】-->【系统配置】页面，修改以下配置项。

表 3-1 系统配置表

配置键	说明	示例
server.ip	可以访问平台的 ip 地址或域名地址	192.168.175.192
server.api	可以访问平台 api 接口的 url 地址	http://192.168.175.192:30666/api/root/master
iiot1.websocket	服务 iiot1 对外开放的 websocket 地址	ws://192.168.175.192:30788/ws
iiot1.mqtt	服务 iiot1 对外开放的 mqtt 地址	192.168.175.192:31881

iiot2.websocket	服务 iiot1 对外开放的 websocket 地址	ws://192.168.175.192:30888/ws
iiot2.mqtt	服务 iiot1 对外开放的 mqtt 地址	192.168.175.192:31882
iiot1.http	服务 iiot1 向外暴露的 http 地址（按需）	http://192.168.174.228:30134
iiot2.http	服务 iiot2 向外暴露的 http 地址（按需）	http://192.168.174.228:30135
iiot1.coap	服务 iiot1 向外暴露的 coap 地址（按需）	coap://192.168.174.228:30809
iiot2.coap	服务 iiot2 向外暴露的 coap 地址（按需）	coap://192.168.174.228:30810
iiot1.iiot1-0.mqtt	服务 iiot1 的 iiot1-0 副本向外暴露的 mqtt 地址 对应 4.2.3.3 章节配置副本服务记录的 mqtt 端口，不同副本的服务端口不一样	192.168.174.169:30460
iiot1.iiot1-1.mqtt	服务 iiot1 的 iiot1-1 副本向外暴露的 mqtt 地址 对应 4.2.3.3 章节配置副本服务记录的 mqtt 端口，不同副本的服务端口不一样	192.168.174.169:30461
iiot2.iiot2-0.mqtt	服务 iiot2 的 iiot2-0 副本向外暴露的 mqtt 地址 对应 4.2.3.3 章节配置副本服务记录的 mqtt 端口，不同副本的服务端口不一样	192.168.174.169:30560
iiot2.iiot2-1.mqtt	服务 iiot2 的 iiot2-1 副本向外暴露的 mqtt 地址 对应 4.2.3.3 章节配置副本服务记录的 mqtt 端口，不同副本的服务端口不一样	192.168.174.169:30561
iiot1.iiot1-0.websocket	服务 iiot1 的 iiot1-0 副本向外暴露的 websocket 地址 对应 4.2.3.3 章节配置副本服务记录的 websocket 端口，不同副本的服务端口不一样	ws://192.168.174.169:32248/ws
iiot1.iiot1-1.websocket	服务 iiot1 的 iiot1-1 副本向外暴露的 websocket 地址 对应 4.2.3.3 章节配置副本服务记录的 websocket 端口，不同副本的服务端口不一样	ws://192.168.174.169:32249/ws
iiot2.iiot2-	服务 iiot2 的 iiot2-0 副本向外	ws://192.168.174.169:32348/ws

0.websocket	暴露的 websocket 地址 对应 4.2.3.3 章节配置副本服务记录的 websocket 端口，不同副本的服务端口不一样	
iiot2.iiot2-1.websocket	服务 iiot2 的 iiot2-1 副本向外暴露的 websocket 地址 对应 4.2.3.3 章节配置副本服务记录的 websocket 端口，不同副本的服务端口不一样	ws://192.168.174.169:32349/ws

以上配置中涉及的 ip 修改为部署服务器的 ip，端口取章节【端口开放清单】准备的信息。

- 参数调优

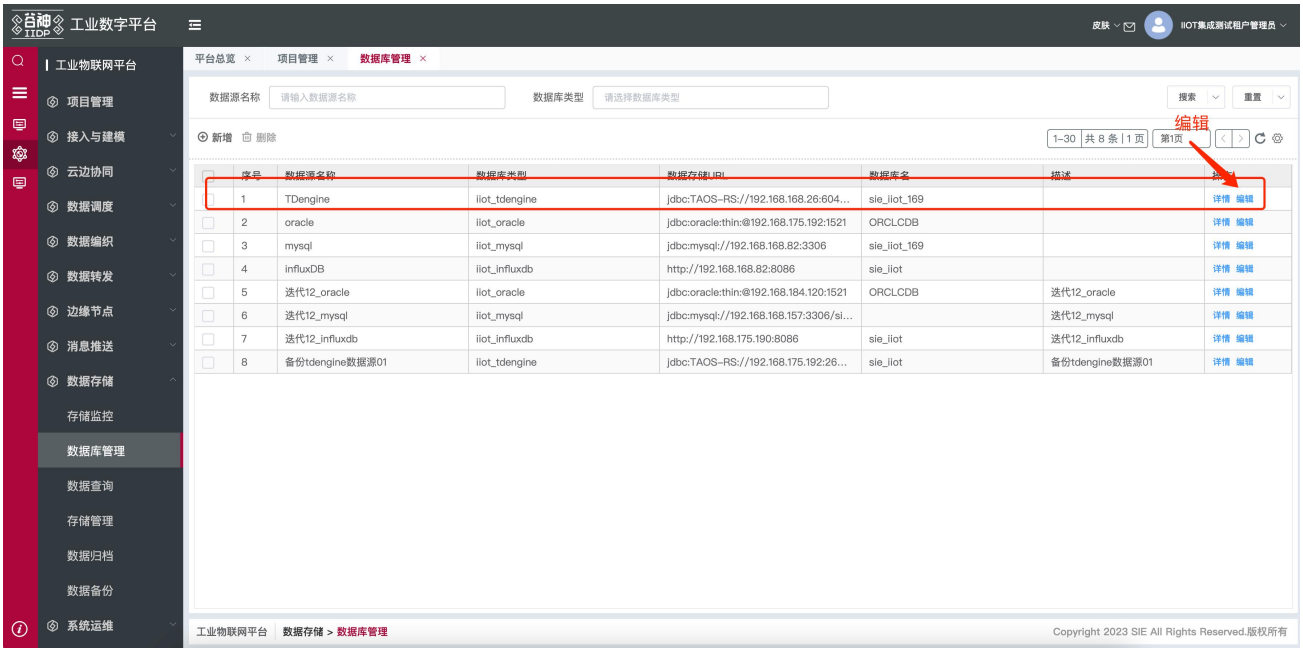
分析数据上报的测点数据类型，针对性进行调优。例如上报测点的数据类型 double 较多，可以适量增加 double 类型的存储配置，对应配置为 store.double.thread 和 store.double.interval，分别为 double 类型数据存储任务的线程数量和存储间隔（毫秒），线程数可以调整为 2-4，存储间隔可以适当缩短，最小不要低于 1000（毫秒）

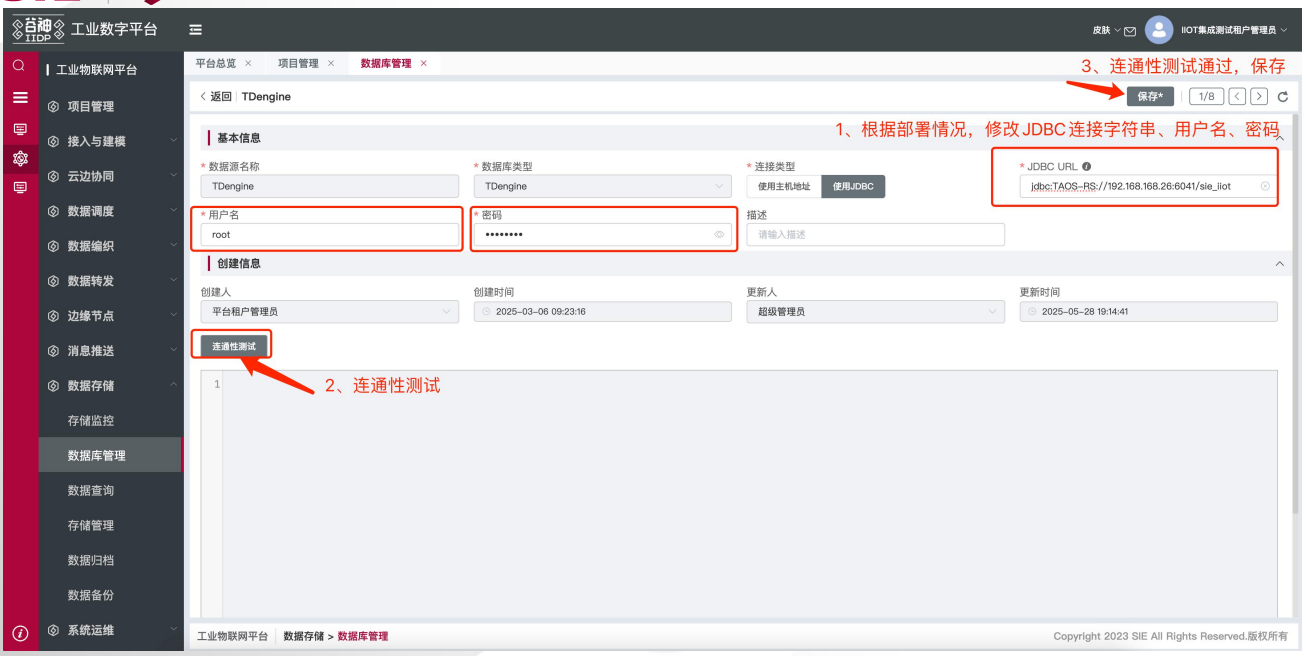
注意：参数调优配置修改后需要手动重启服务。

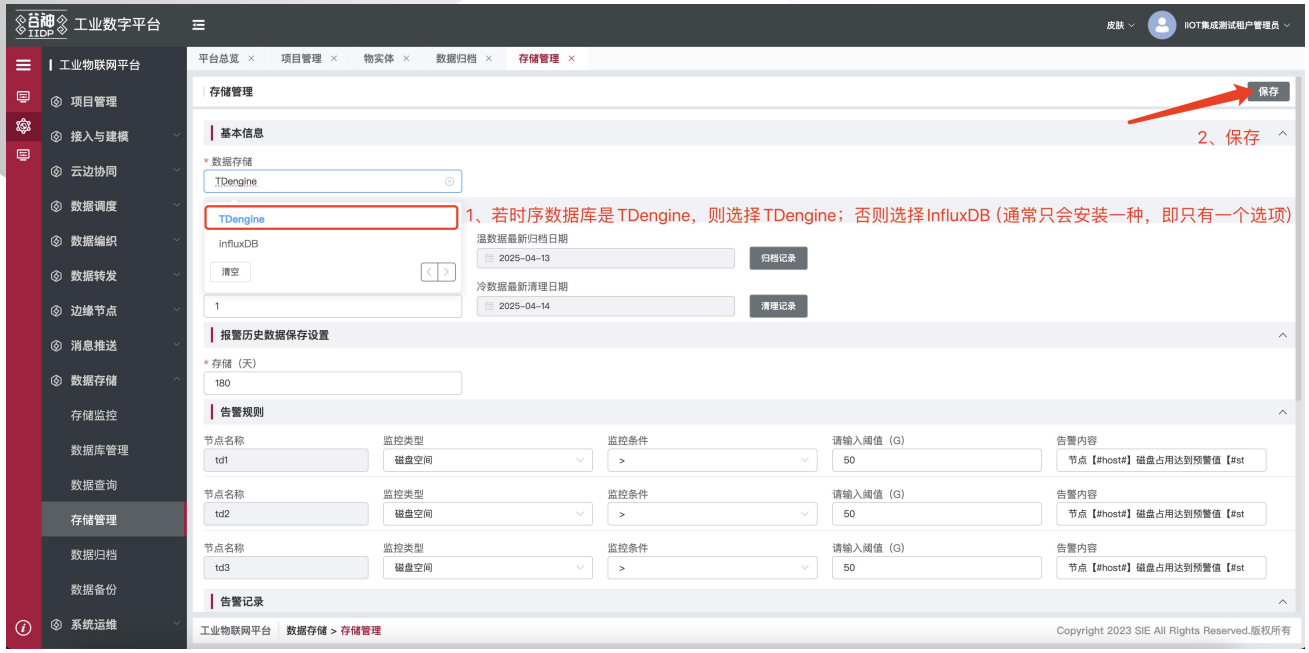
配置键	说明	示例
store.double.thread	定时执行时序数据存储-double 类型任务的线程数量，默认为 1，修改后需要重启服务	4
store.double.interval	定时执行时序数据存储-double 类型任务的存储间隔（毫秒），默认值为 2000，修改后需要重启服务	1000

3.3.8. 存储配置

- 使用租户管理员账号登录 IIOT 系统，在【数据存储】-->【数据库管理】页面，修改时序数据库的信息。







4. 部署时钟同步

NTP 是用来使计算机时间同步化的一种协议，全称是 Network Time Protocol。它可以在大规模的设备范围内同步矫正时间到几 ms 级别的精度，在网络稳定的局域网内，精度甚至可以达到微秒级别。 ntp 端口使用的是 123。

NTP 时间同步配置需要配置服务端与客户端，服务端是源时间地址，客户端的时间会同步服务端时间。

举例说明：

服务端 IP：192.168.1.201
客户端 IP：192.168.1.201 ~ 192.168.1.203
以下所有的配置都基于此信息配置，用户根据实际机器 IP 情况进行配置。

注意：若部署 TDengine 集群，务必保障集群各节点的始终同步。

4.1. 服务端配置

服务端可以作为时间同步的源服务器，客户端服务器可以使用 ntp 同步到服务端的时间。

- **安装 ntp、ntpdate 软件。**

```
yum install -y ntp ntpdate
```

- **修改 ntp 配置文件：etc/ntp.conf。**

```
# 1. 注释默认时钟源配置，添加中国国家授时中心时钟源
#server 0.centos.pool.ntp.org iburst
#server 1.centos.pool.ntp.org iburst
#server 2.centos.pool.ntp.org iburst
#server 3.centos.pool.ntp.org iburst

server 0.cn.pool.ntp.org iburst
server 1.cn.pool.ntp.org iburst
server 2.cn.pool.ntp.org iburst
server 3.cn.pool.ntp.org iburst

# 2. 设置允许上层时钟服务器主动修改本机时间
restrict 0.cn.pool.ntp.org nomodify notrap noquery
restrict 1.cn.pool.ntp.org nomodify notrap noquery
restrict 2.cn.pool.ntp.org nomodify notrap noquery
restrict 3.cn.pool.ntp.org nomodify notrap noquery

# 3. 配置外部时钟源不可用时，以本地时间作为时间服务
server 127.127.1.0
fudge 127.127.1.0 stratum 10

# 4. 配置客户端访问策略（假设网段是：192.168.1.201~192.168.1.203）
restrict 192.168.1.0 mask 255.255.255.0 nomodify notrap
```

```
# 5. 把系统时间写入主板(即使服务器关机或者断电)
SYNC_HWCLOCK=yes
```

- **启动服务端 ntp 服务。**

```
systemctl stop firewalld
systemctl enable ntpd
systemctl restart ntpd
```

4.2. 客户端配置

- **安装 ntp、ntpdate 软件。**

```
yum install -y ntp ntpdate
```

- **修改 ntp 配置文件：etc/ntp.conf。**

```
# 1. 注释默认时钟源配置，添加服务端时钟源
#server 0.centos.pool.ntp.org iburst
#server 1.centos.pool.ntp.org iburst
#server 2.centos.pool.ntp.org iburst
#server 3.centos.pool.ntp.org iburst

server 192.168.1.201

# 2. 设置允许上层时钟服务器主动修改本机时间
restrict 192.168.1.201 nomodify notrap noquery

# 3. 配置外部时钟源不可用时，以本地时间作为时间服务
server 127.127.1.0
fudge 127.127.1.0 stratum 10

# 4. 把系统时间写入主板(即使服务器关机或者断电)
SYNC_HWCLOCK=yes
```

- **启动服务端 ntp 服务。**

```
systemctl stop firewalld
systemctl enable ntpd
systemctl restart ntpd
```

- **同步一次时间。**

```
/usr/sbin/ntpdate -u 192.168.1.201
```

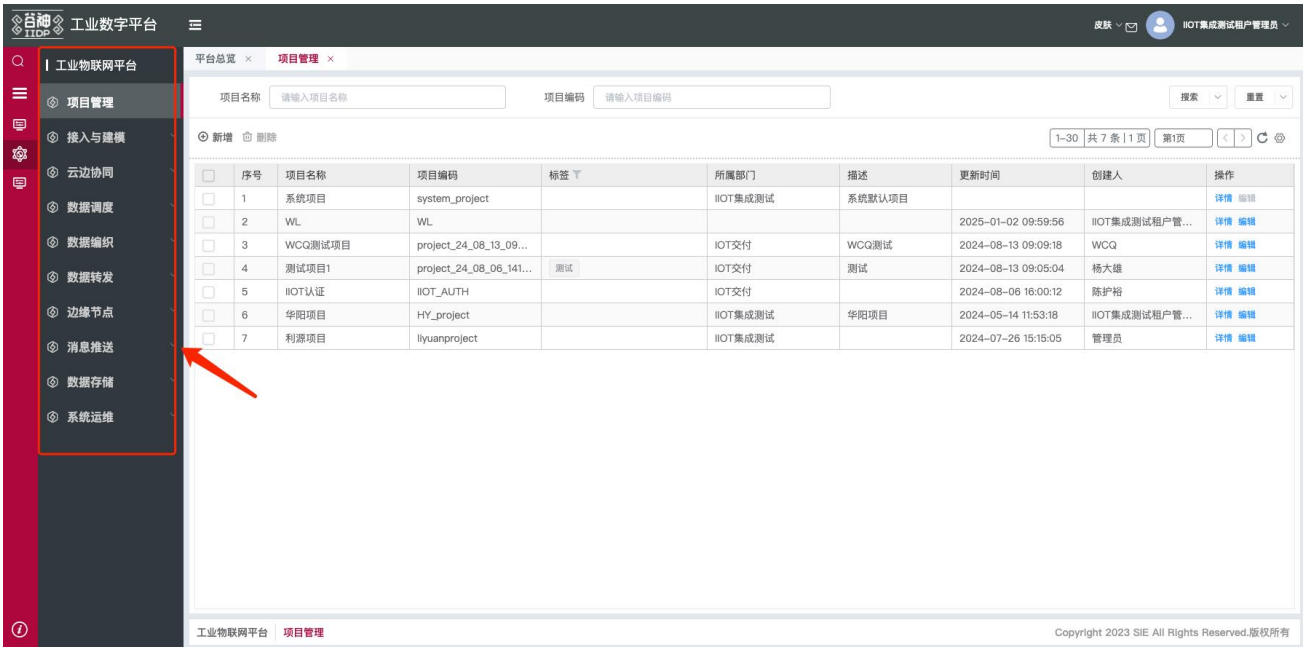
- **配置定时任务同步时间（crontab –e）。**

```
*/10 * * * * /usr/sbin/ntpdate 192.168.0.1
```

5. 部署验证

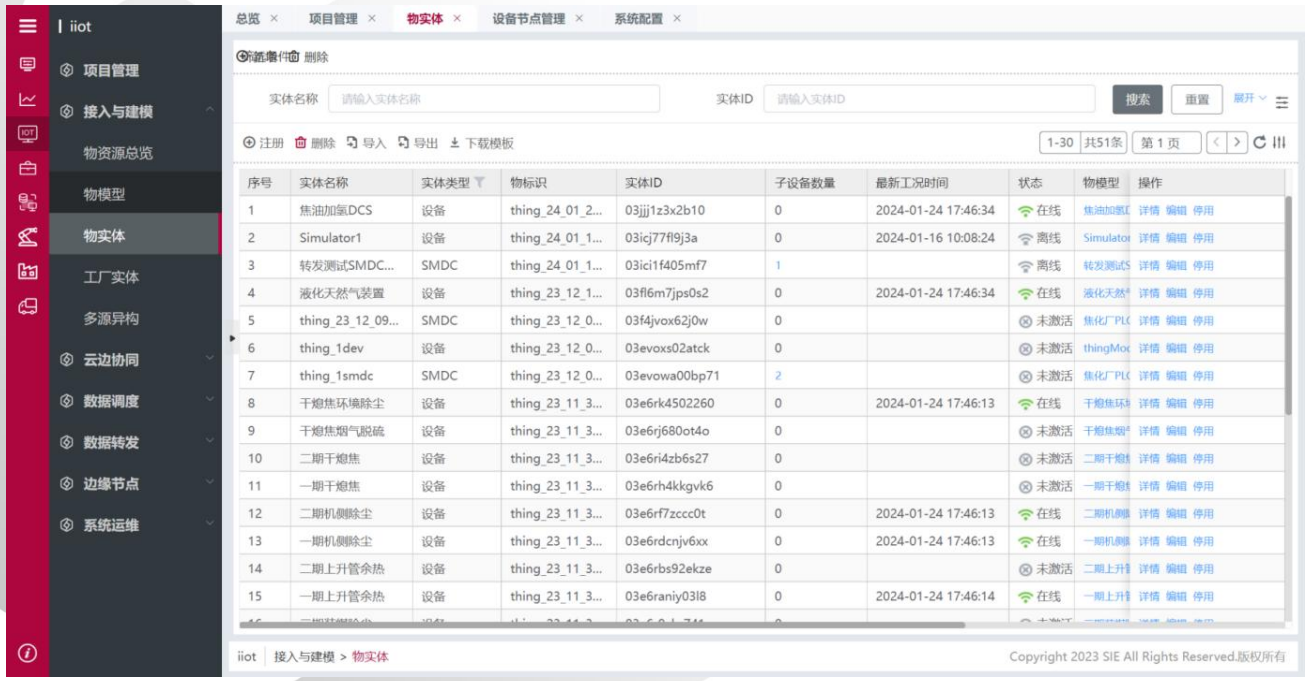
5.1. 菜单验证

使用租户管理员账号登录 IIDP 系统，选择工业物联网（即 IIOT）系统菜单。分别点击各模块菜单，若没有报错，则标识安装正常。



5.2. 核心功能验证

选择【接入与建模】菜单，创建测试用物模型、物实体，使用 SMDC 开发时创建工程（参考 SMDC 使用文档），上报数据到 IIOT 平台。



5.3. 数据存储验证

进入**【数据存储】**->**【存储管理】**菜单，右上角点击**【连通性测试】**按钮，验证数据存储是否正常。