1. 如何防止容器的数据因为容器销毁造成的数据丢失

使用卷或者绑定挂载的方式，将写入到容器的数据永久保存

2. docker导出镜像有两种方式，分别是什么，有什么区别，为什么（提示：两种方式导出的镜像大小有明细差异）

在 Docker 中，export 和 save 都可以用于导出容器或镜像，但它们的作用和原理不同，因此会导致导出文件的大小差异很大

Docker export （导出容器）
作用：
将运行中的容器文件系统以扁平化的形式导出为一个 .tar 文件。
只包含最终容器内的文件系统，不包含历史层（layers）、元数据、Dockerfile、环境变量、镜像历史等

使用场景：
将某个容器的当前状态迁移到另一台机器
导出的内容相当于 Linux 文件系统，恢复时需要手动导入。

导出文件大小
较小，因为只包含当前容器的最终状态
导入时不重建镜像层

Docker save （导出镜像）
作用：
将镜像的所有层（layers）和元数据导出为一个 .tar 文件

包含：
层（layers）：多层结构，便于构建时缓存使用。
元数据：如 Dockerfile、构建命令历史、环境变量等。

使用场景
完整备份或迁移镜像，包括构建历史和所有依赖。
在没有网络的环境下，通过文件导入镜像

导出文件大小
较大，因为包含镜像的所有层和元数据
尤其是基于多个层构建的镜像，大小可能成倍增加。

文件大小差异的原因总结
docker export 是将容器内的最终状态导出，不包含中间构建层和历史记录，只导出最终文件系统，因此体积较小
docker save 会将镜像的所有构建层和元数据信息一起导出，每个层都会独立导出，因此体积较大。

3. 我在镜像管理服务推送了第一版镜像，然后你先去仓库管理服务器上下载，然后我后面又升级了一个版本，是同一个服务，当然，image的版本号肯定会增加，然后发现第一版下载很慢，第二次就很快，为什么?

因为 Docker 镜像的分层存储机制和缓存 使得第二次拉取相同基础的镜像会更快，主要原因如下

Docker 镜像的分层机制
ocker 镜像是由多个只读的 层（layers） 组成，每个层只包含它自己更改的内容，所有层 按顺序堆叠 起来形成最终的镜像。

第一版镜像下载时，所有层都需要从远程仓库拉取，可能因为网络带宽、镜像大小等因素，速度较慢。
第二次下载（新版本）时，Docker 只会拉取新版本中新增或修改的层，而未变更的层会 复用本地缓存，导致下载速度更快。

本地镜像缓存
你第一次拉取镜像时，Docker 会将它存储在本地的 Docker image cache 里（通常在 /var/lib/docker 目录下）。

如果 新版本的镜像和旧版本共用相同的基础层，那么 Docker 只会下载变化的部分，而不会重新拉取已有的层。
这样，第二次拉取镜像时，只需要获取 新的或修改的部分，下载时间大幅减少

Docker Registry 的增量传输
如果你使用的是 支持层缓存的镜像仓库（如 Harbor、Docker Hub、Alibaba Container Registry (ACR)），那么：
仓库端也会缓存 镜像层，并且利用增量传输技术，只传输新增的层
这样即使是从远程仓库拉取，也会更快

4. docker有几种网络模型，分别是什么

Docker 的网络支持5种网络模式:

- **none**
- **host**
- **bridge**
- **container**
- **network-name**

5. dockerfile中，add和copy的区别是什么?

ADD可以看做是增强版的COPY，不仅支持COPY，还**支持自动解压缩**。可以将复制指定的到容器中的

6. dockerfile中，比较run, entrypoint, cmd 三者的区别

RUN和COPY的区别

RUN 命令是在构建镜像时执行的命令
RUN 可以写多个，每一个RUN指令都会建立一个镜像层
每个 Dockerfile 只能有一条 CMD 命令。如指定了多条，只有最后一条被执行

ENTRYPOINT和CMD的区别

ENTRYPOINT后面的指令会追加，作为前面指令的参数
CMD后面如果有后续输入，则会替换之前设置的指令
如果ENTRYPOINT和CMD共存，则CMD的值看作是ENTRYPOINT的参数

ENTRYPOINT和CMD之间配合使用

ENTRYPOINT负责执行环境初始化
CMD作为容器启动的默认进程

7. 分析bridge模式下，docker上部署个容器，它上面的流量流入路径，和流量流出路径分别是什么?

外部流量进入容器:

假设你在宿主机运行以下命令：
```
docker run -d -p 8080:80 nginx
```

外部请求（如 curl http://host_ip:8080） → 宿主机物理网卡 (eth0) → iptables/NAT（端口映射） → docker0 虚拟网桥（或对应的网络模式） → veth 对 → 容器内的 eth0 → 容器内应用接收请求

Docker 容器向外部发送流量
容器内的应用 产生流量 → 容器内 eth0 接口 → veth 对 → docker0 虚拟网桥 → 宿主机的 iptables/NAT → 宿主机物理网卡 eth0 → 外部网络

8. 总结dockerfile的最佳实践（最少写3种）

1. 不要安装无效软件包
2. 应简化镜像中同时运行的进程数，理想状况下，每个镜像应该只有一个进程
3. 当无法避免同一镜像运行多进程时，应选择合理的初始化进程(init process)
4. 最小化层级数
   - 最新的docker只有RUN,COPY,ADD，创建新层，其他指令创建临时层，不会增加镜像大小
     - 比如`EXPOSE指令`不会生成新层
   - 多条RUN命令可通过连接符连接成一条指令集以及减少层数
   - 通过多段构建减少镜像层数
5. 把多行参数按字母排序，可以减少可能出现的重复参数，并且提高可读性
6. 编写dockerfile时，应该把变更频率低的编译指令优先构建以便放在镜像底层以有效利用build cache
7. 复制文件时，每个文件应独立复制，这确保某个文件变更时，只影响该文件对应的缓存

总结：
目标：易管理，少漏洞，镜像小，层级少，利用缓存

9. 实验：编写dockerfile多阶段构建Nginx

```
ARG VERSION=3.20.0
FROM alpine:$VERSION
LABEL maintainer="mystical<mysticalrecluse@gmail.com>"

ENV NGINX_VERSION=1.26.1
ENV NGINX_DIR=/apps/nginx

ADD nginx-$NGINX_VERSION.tar.gz /usr/local/src

RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.ustc.edu.cn/' /etc/apk/repositories
&& \
    apk update && apk --no-cache add gcc make libgcc libc-dev libcurl lib-utils
pcre-dev zlib-dev libnfs pcre pcre2 net-tools curl pstree wget libevent libevent-
dev iproute2 openssl-dev && \
    cd /usr/local/src/nginx-$NGINX_VERSION && \
    ./configure --prefix=${NGINX_VERSION} --user=nginx --group=nginx --with-
http_ssl_module --with-http_v2_module --with-http_realip_module --with-
http_stub_status_module --with-http_gzip_static_module --with-pcre --with-stream
--with-stream_ssl_module --with-stream_realip_module && \
    make && make install && \
    rm -rf /usr/local/src/nginx-$NGINX_VERSION

COPY nginx.conf ${NGINX_DIR}/conf/nginx.conf

FROM alpine:$VERSION
ENV NGINX_DIR=/apps/nginx
COPY --from=0 ${NGINX_DIR}/ ${NGINX_DIR}/
RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.ustc/edu.cn/' /etc/apk/repositories
\
    && apk update && apk --no-cache add tzdate pcre pcre2 \
    && ln -s /usr/share/zoneinfo/Asia/Shanghai /etc/localtime \
    && addgroup -g 888 -S nginx \
    && adduser -u 888 -G nginx -D -S -s /sbin/nologin nginx \
    && chown -R nginx.nginx ${NGINX_DIR}/ \
    # 在容器化环境中，推荐的做法是将应用程序的日志输出到标准输出和标准错误。容器运行时（如
Docker）会捕获这些日志并将它们存储在宿主机上。这样，日志就可以被宿主机上的日志收集和处理系统统一
管理。
    && ln -sf /dev/stdout ${NGINX_DIR}/logs/access.log \
    && ln -sr /dev/stderr ${NGINX_DIR}/logs/error.log
EXPOSE 80 443
CMD ["nginx","-g","daemon off;"]
```

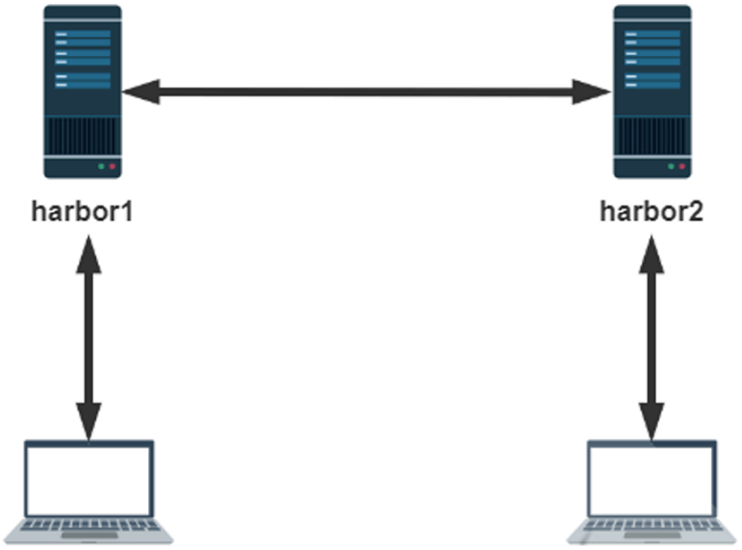
10. 搭建Harbor，并实现Harbor的高可用，并将第9题中构建的Nginx镜像推送到Harbor上，同时观察另一台Harbor上是否进行了同步

安装 Harbor

下载地址： https://github.com/vmware/harbor/releases

环境准备：共四台主机

- 两台主机harbor服务器，地址：10.0.0.101|102
- 两台主机harbor客户端上传和下载镜像



安装 docker

使用脚本安装，脚本内容如下

```
#!/bin/bash
#
#**************************************************************************
#Author:            wangxiaochun
#QQ:                29308620
#Date:              2022-10-14
#FileName:          install_docker_offline.sh
#URL:               http://www.wangxiaochun.com
#Description:       The test script
#Copyright (C):     2022 All rights reserved
#**************************************************************************

#支持在线和离线安装

DOCKER_VERSION=26.1.4
#DOCKER_VERSION=26.0.0
#DOCKER_VERSION=24.0.7
#DOCKER_VERSION=24.0.5
#DOCKER_VERSION=23.0.3
#DOCKER_VERSION=20.10.19
```

```
URL=https://mirrors.tuna.tsinghua.edu.cn
#URL=https://mirrors.aliyun.com
#URL=https://download.docker.com

color () {
    RES_COL=60
    MOVE_TO_COL="echo -en \\033[${RES_COL}G"
    SETCOLOR_SUCCESS="echo -en \\033[1;32m"
    SETCOLOR_FAILURE="echo -en \\033[1;31m"
    SETCOLOR_WARNING="echo -en \\033[1;33m"
    SETCOLOR_NORMAL="echo -en \E[0m"
    echo -n "$1" && $MOVE_TO_COL
    echo -n "["
    if [ $2 = "success" -o $2 = "0" ] ;then
        ${SETCOLOR_SUCCESS}
        echo -n $"  OK  "
    elif [ $2 = "failure" -o $2 = "1"  ] ;then
        ${SETCOLOR_FAILURE}
        echo -n $"FAILED"
    else
        ${SETCOLOR_WARNING}
        echo -n $"WARNING"
    fi
    ${SETCOLOR_NORMAL}
    echo -n "]"
    echo
}

prepare () {
    if [ ! -e docker-${DOCKER_VERSION}.tgz ];then
        #wget ${URL}/docker-
ce/linux/static/stable/x86_64/docker-${DOCKER_VERSION}.tgz
        wget ${URL}/docker-
ce/linux/static/stable/x86_64/docker-${DOCKER_VERSION}.tgz
    fi
    [ $? -ne 0  ] && { echo "文件下载失败"; exit; }
}

install_docker () {
    tar xf docker-${DOCKER_VERSION}.tgz -C /usr/local/
    cp /usr/local/docker/* /usr/local/bin/
    cat > /lib/systemd/system/docker.service <<-EOF
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network-online.target firewalld.service
Wants=network-online.target

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/local/bin/dockerd -H unix://var/run/docker.sock
ExecReload=/bin/kill -s HUP \$MAINPID
```

```
# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity
# Uncomment TasksMax if your systemd version supports it.
# Only systemd 226 and above support this version.
#TasksMax=infinity
TimeoutStartSec=0
# set delegate yes so that systemd does not reset the cgroups of docker
containers
Delegate=yes
# kill only the docker process, not all processes in the cgroup
KillMode=process
# restart the docker process if it exits prematurely
Restart=on-failure
StartLimitBurst=3
StartLimitInterval=60s

[Install]
WantedBy=multi-user.target
EOF
    systemctl daemon-reload
}

config_docker () {
    mkdir -p /etc/docker
    tee /etc/docker/daemon.json <<-'EOF'
    {
      "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
     }
EOF
    #systemctl restart docker

}

start_docker (){
    systemctl enable --now docker
    docker version && color "Docker 安装成功" 0 ||  color "Docker 安装失败" 1
}


config_docker_completion () {
    wget -P /etc/bash_completion.d
http://www.wangxiaochun.com:8888/testdir/docker/docker_completion
    #source /etc/bash_completion.d/docker_completion
}


prepare

install_docker

config_docker

start_docker
```

```
config_docker_completion
```

先安装docker compose

```
# Add Docker's official GPG key:
apt-get update
apt-get install ca-certificates curl
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
chmod a+r /etc/apt/keyrings/docker.asc

echo "GPG OVER"

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" |
\
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# 官方仓库配置好后，执行下面的指令
[root@ubuntu2204 ~]#apt install -y docker-compose-plugin
```

下载Harbor安装包并解压缩

```
[root@ubuntu2204 ~]#wget
https://github.com/goharbor/harbor/releases/download/v2.12.2/harbor-offline-
installer-v2.12.2.tgz

[root@ubuntu2204 ~]#mkdir /apps
[root@ubuntu2204 ~]#tar xvf harbor-offline-installer-v2.12.2.tgz -C /apps/
harbor/harbor.v2.12.2.tar.gz
harbor/prepare
harbor/LICENSE
harbor/install.sh
harbor/common.sh
harbor/harbor.yml.tmpl
```

编辑 harbor 配置文件

```
[root@ubuntu2204 harbor]#cd /apps/harbor
[root@ubuntu2204 harbor]#cp harbor.yml.tmpl harbor.yml
[root@ubuntu2204 harbor]#vim harbor.yml
# 更改配置文件的主机名，这里为了实验简单，故而使用ip
# hostname: harbor.mystical.org
```

```
hostname: 10.0.0.100

# 将https注视掉
#https:
  # https port for harbor, default is 443
#  port: 443
  # The path of cert and key files for nginx
#  certificate: /your/certificate/path
#  private_key: /your/private/key/path

# 更改harbor的登录密码
harbor_admin_password: 123456

# 更改数据目录
# The default data volume
data_volume: /data/harbor

[root@ubuntu2204 harbor]#ls
common.sh  harbor.v2.12.2.tar.gz  harbor.yml  harbor.yml.tmpl  install.sh
LICENSE  prepare

# 使用prepare拉取镜像并生成配置文件
[root@ubuntu2204 harbor]#./prepare

# 使用install.sh
[root@ubuntu2204 harbor]#./install.sh

# 查看
[root@ubuntu2204 harbor]#docker compose ps
NAME                IMAGE                                   COMMAND
  SERVICE       CREATED          STATUS                             PORTS
harbor-core         goharbor/harbor-core:v2.12.2          "/harbor/entrypoint.…"
  core          26 seconds ago   Up 19 seconds (health: starting)
harbor-db           goharbor/harbor-db:v2.12.2            "/docker-entrypoint.…"
  postgresql    26 seconds ago   Up 20 seconds (health: starting)
harbor-jobservice   goharbor/harbor-jobservice:v2.12.2    "/harbor/entrypoint.…"
  jobservice    26 seconds ago   Up 9 seconds (health: starting)
harbor-log          goharbor/harbor-log:v2.12.2           "/bin/sh -c /usr/loc…"
  log           26 seconds ago   Up 24 seconds (health: starting)
127.0.0.1:1514->10514/tcp
harbor-portal       goharbor/harbor-portal:v2.12.2        "nginx -g 'daemon of…"
  portal        26 seconds ago   Up 20 seconds (health: starting)
nginx               goharbor/nginx-photon:v2.12.2         "nginx -g 'daemon of…"
  proxy         26 seconds ago   Up 18 seconds (health: starting)   0.0.0.0:80-
>8080/tcp, [::]:80->8080/tcp
redis               goharbor/redis-photon:v2.12.2         "redis-server /etc/r…"
  redis         26 seconds ago   Up 21 seconds (health: starting)
registry            goharbor/registry-photon:v2.12.2      "/home/harbor/entryp…"
  registry      26 seconds ago   Up 20 seconds (health: starting)
registryctl         goharbor/harbor-registryctl:v2.12.2   "/home/harbor/start.…"
  registryctl   26 seconds ago   Up 20 seconds (health: starting)
```

实现开机自动启动 harbor

旧版 Harbor 开机不会自动启动，可以创建 service 文件实现

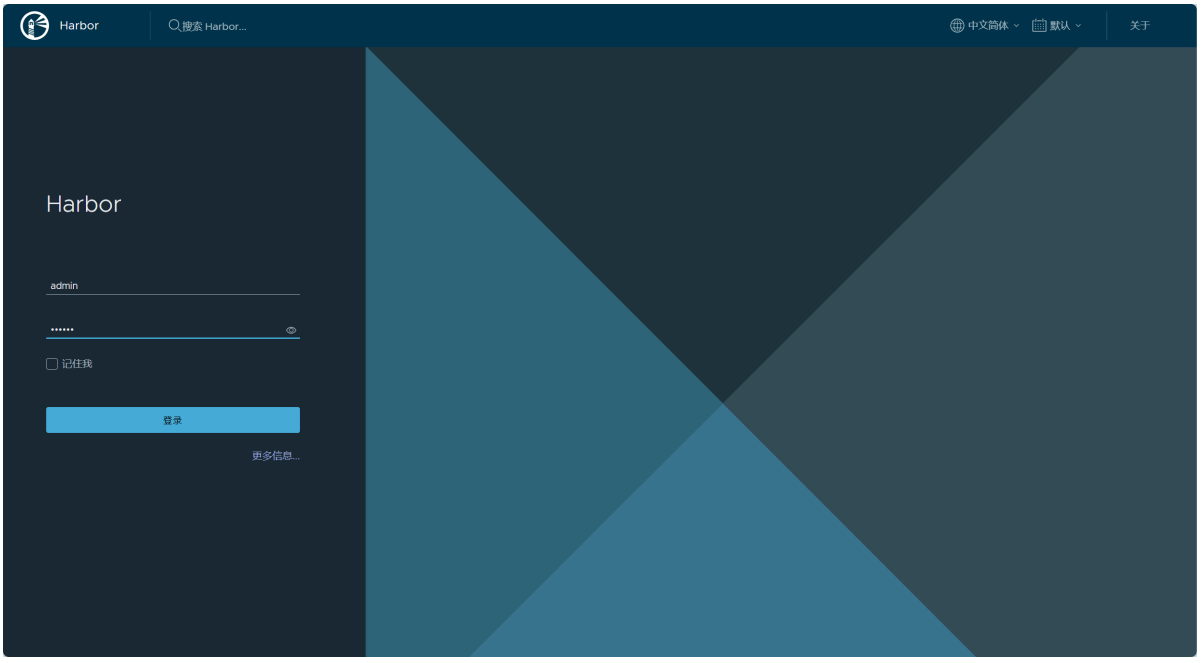
新版 Harbor 默认开机自动，比如：harbor-v2.9.1

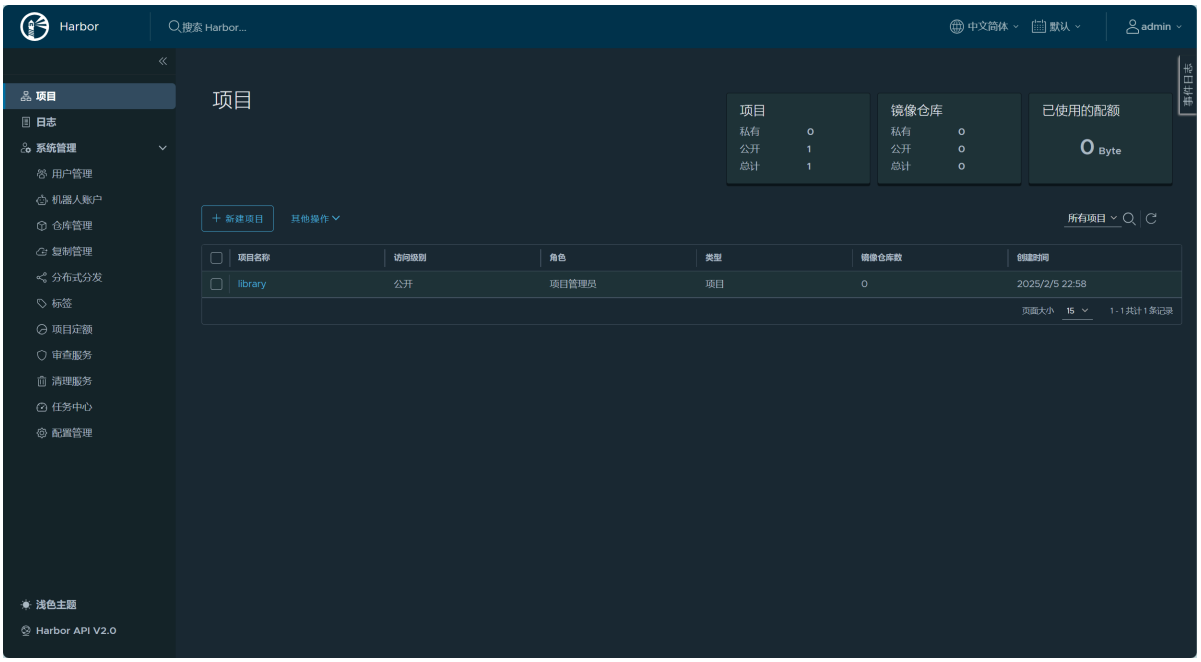
```
[root@harbor ~]#vim /lib/systemd/system/harbor.service
[Unit]
Description=Harbor
After=docker.service systemd-networkd.service systemd-resolved.service
Requires=docker.service
Documentation=http://github.com/vmware/harbor

[Service]
Type=simple
Restart=on-failure
RestartSec=5
ExecStart=/usr/bin/docker-compose -f  /apps/harbor/docker-compose.yml up
ExecStop=/usr/bin/docker-compose -f /apps/harbor/docker-compose.yml down

[Install]
WantedBy=multi-user.target

[root@harbor ~]#systemctl daemon-reload
[root@harbor ~]#systemctl enable harbor
```

登录 harbor 主机网站

用浏览器访问：http://10.0.0.100/

- 用户名：admin
- 密码：即前面harbor.yml中指定的密码

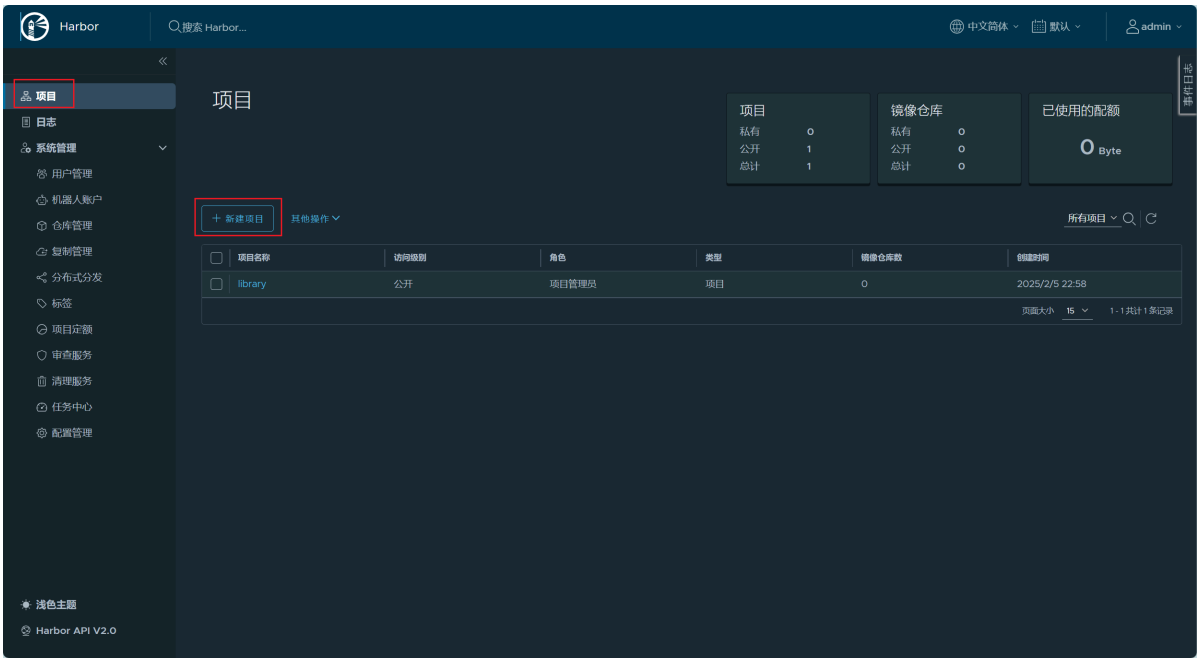


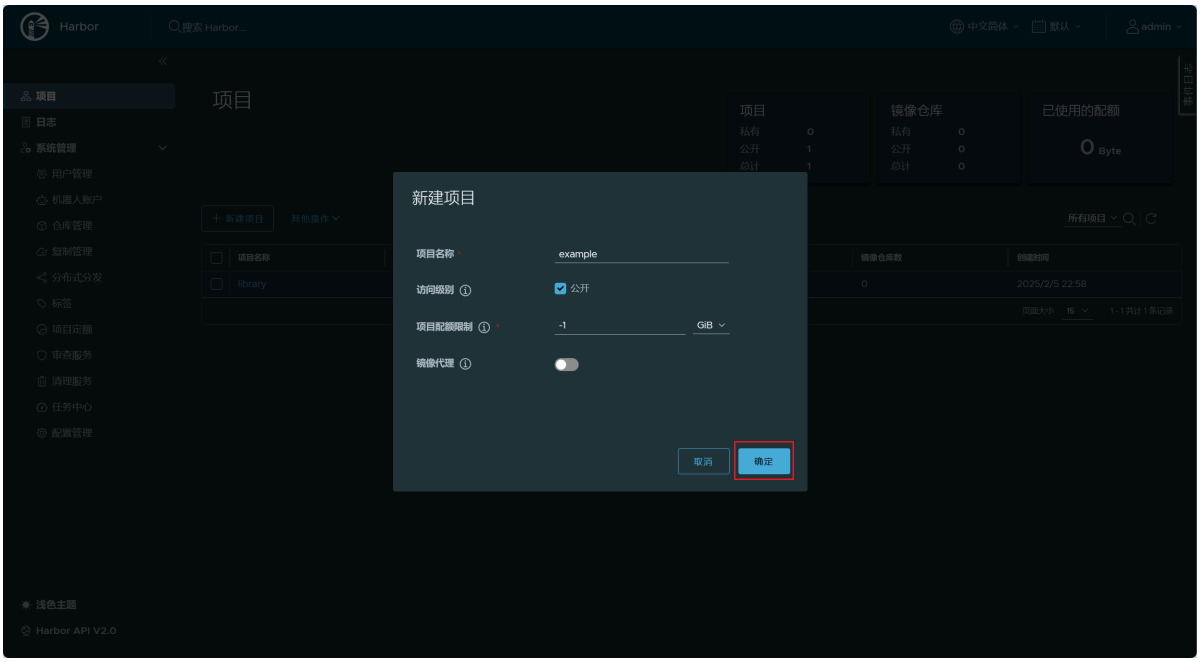


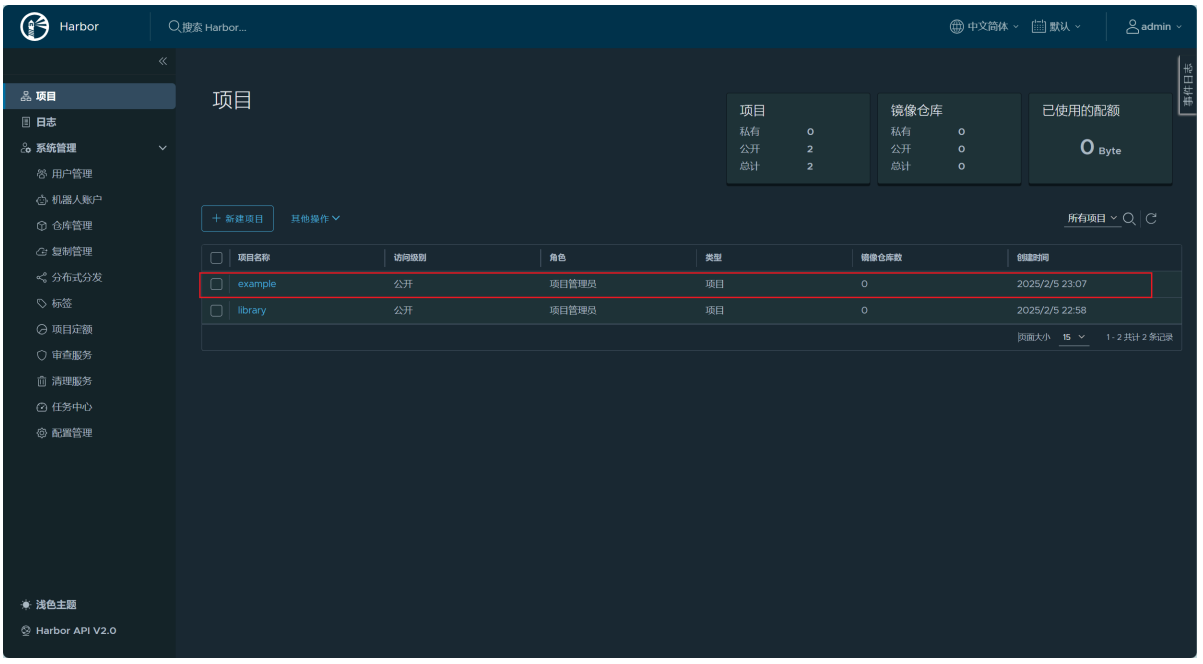
使用单主机 Harbor

建立项目

harbor上必须先建立项目，才能上传镜像

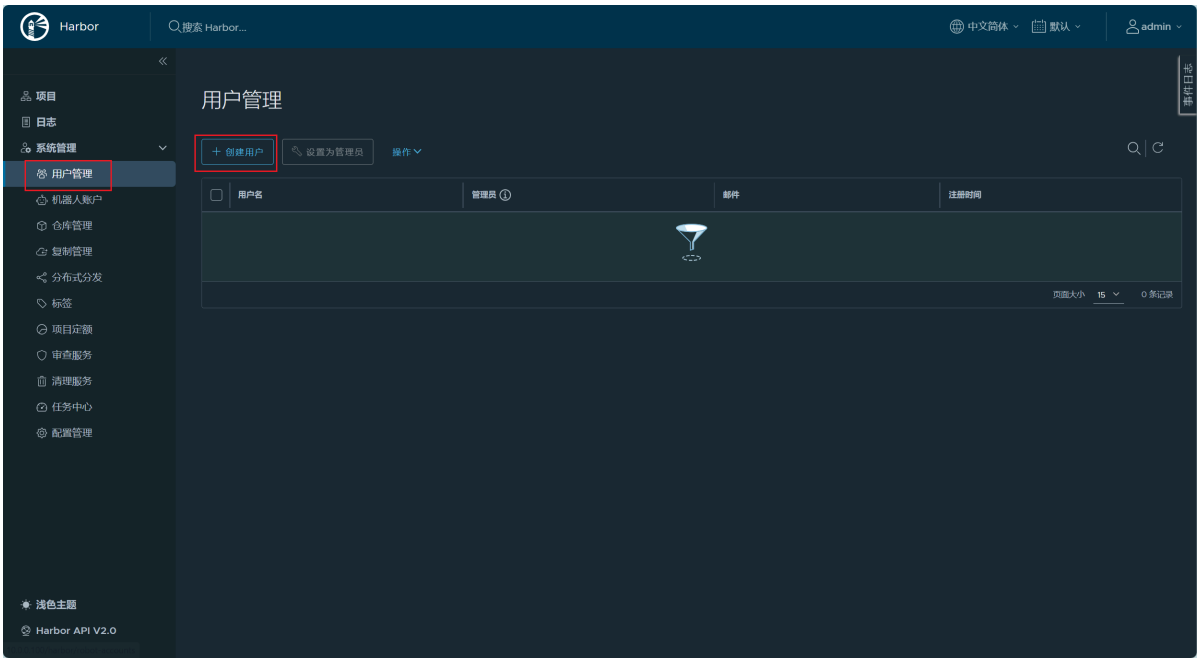


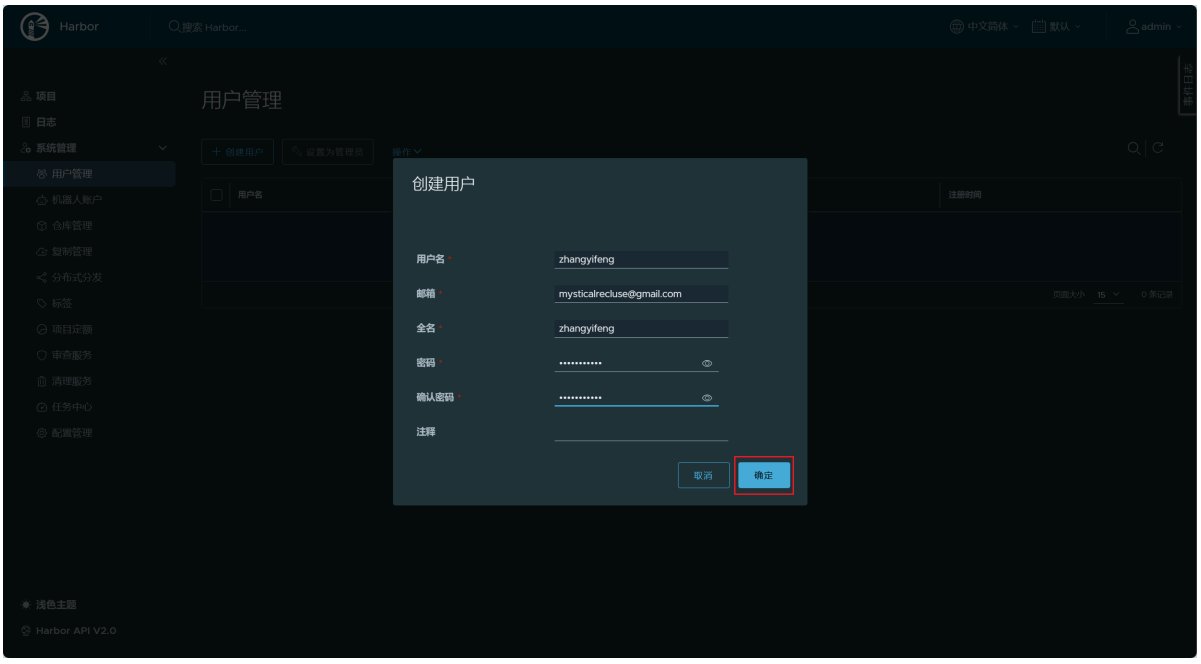


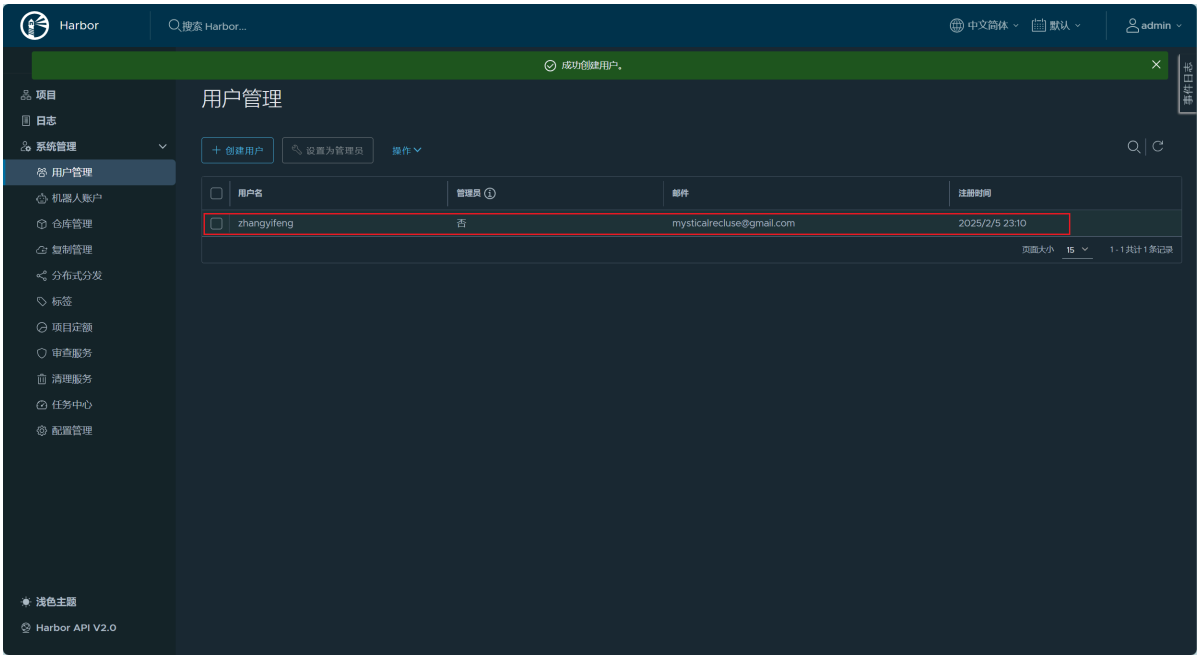


创建用户和项目授权

创建用户

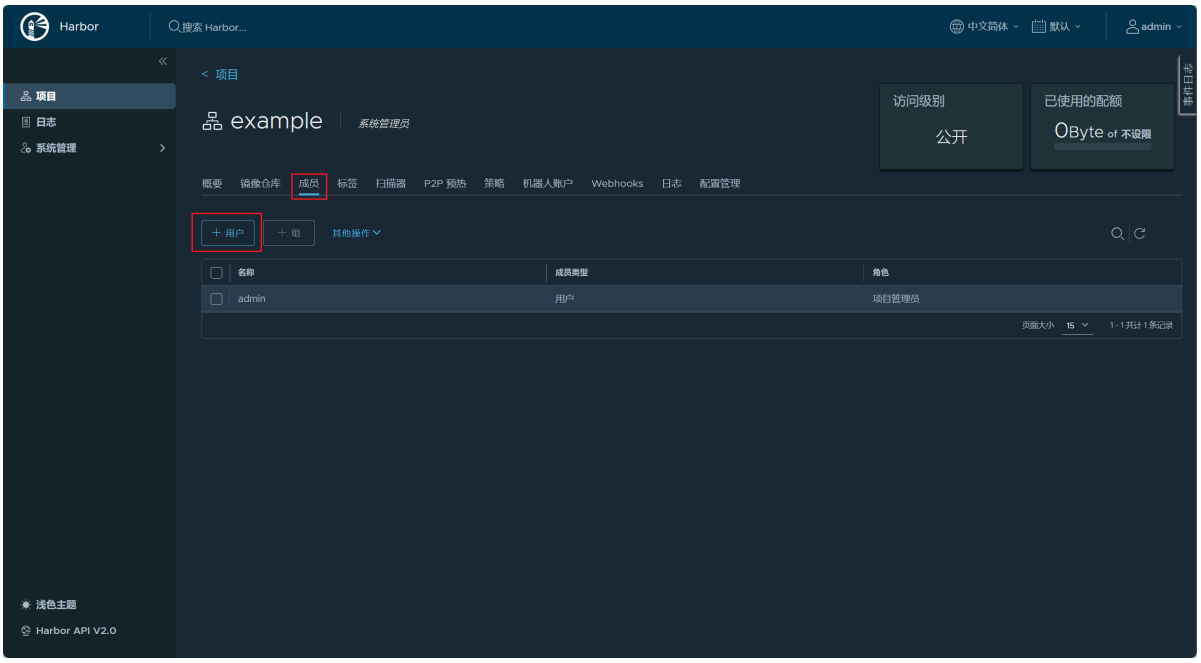

Harbor
搜索 Harbor...
中文简体
默认
admin
项目
日志
系统管理
用户管理
机器人账户
仓库管理
复制管理
分布式分发
标签
项目定额
审查服务
清理服务
任务中心
配置管理
浅色主题
Harbor API V2.0
用户管理
+ 创建用户
设置为管理员
操作
用户名
管理员
邮件
注册时间
页面大小 15
0 条记录

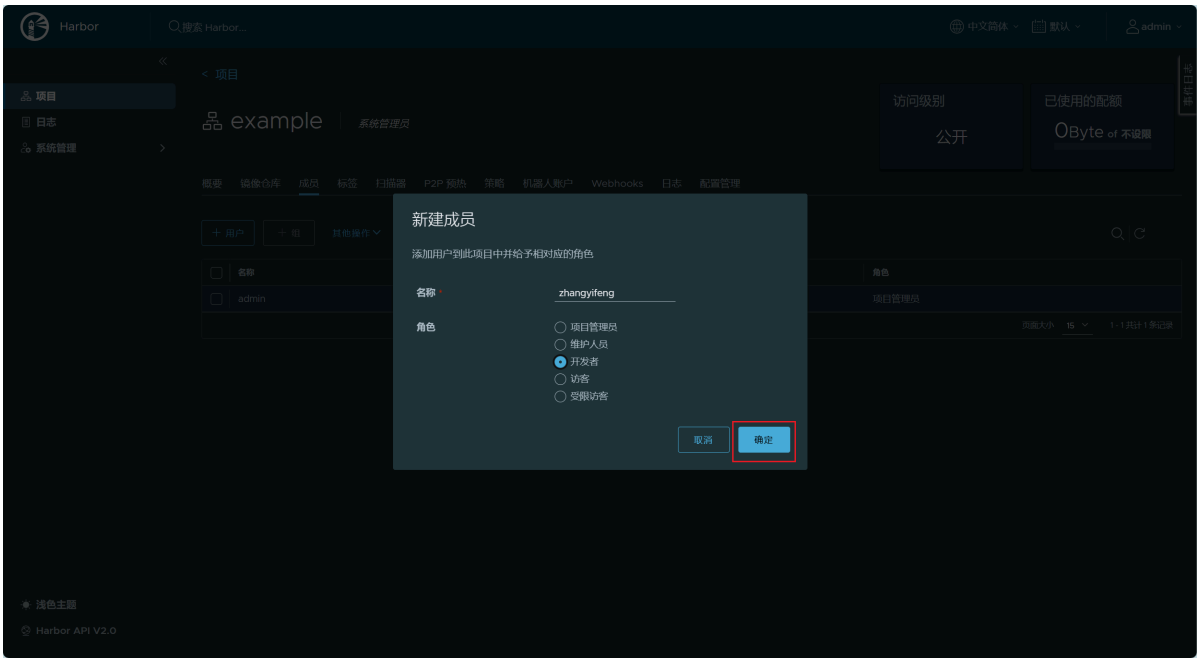

Harbor
用户管理
创建用户
用户名 *
zhangyifeng
邮箱 *
mysticalrecluse@gmail.com
全名 *
zhangyifeng
密码 *
确认密码 *
注释
取消
确定

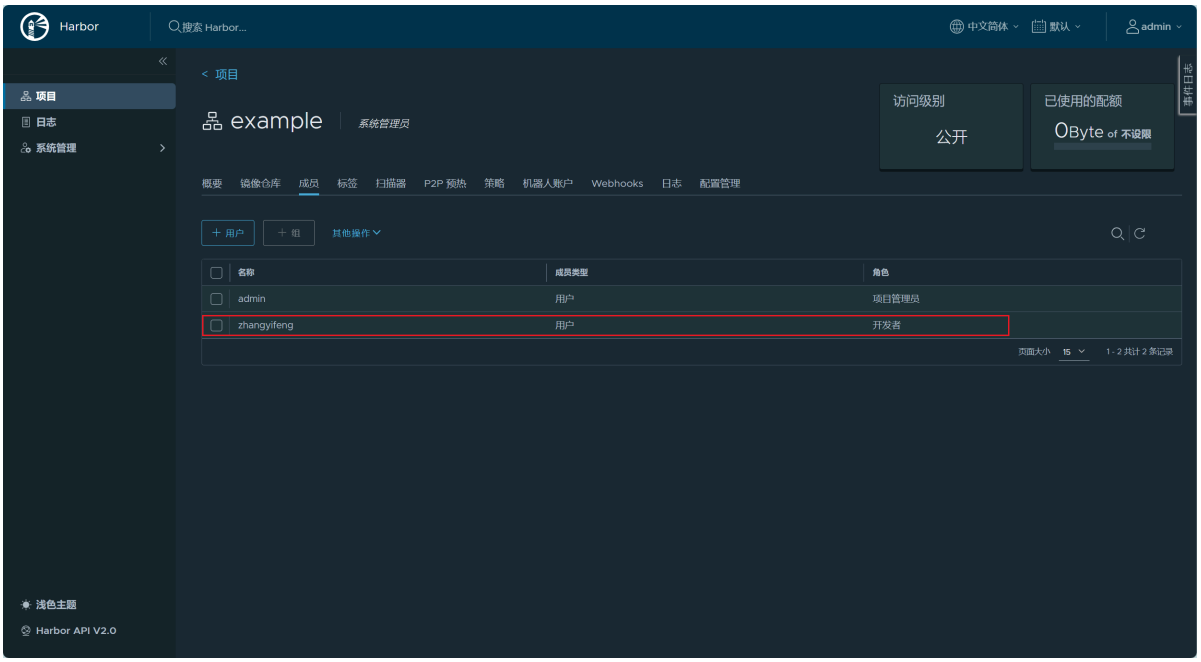

Harbor
成功创建用户。
用户管理
+ 创建用户
设置为管理员
操作
用户名
管理员
邮件
注册时间
zhangyifeng
否
mysticalrecluse@gmail.com
2025/2/5 23:10
1 - 1 共计 1 条记录

在项目内对用户授权

进入对应的项目，，注意：至少是开发者以上才能上传镜像

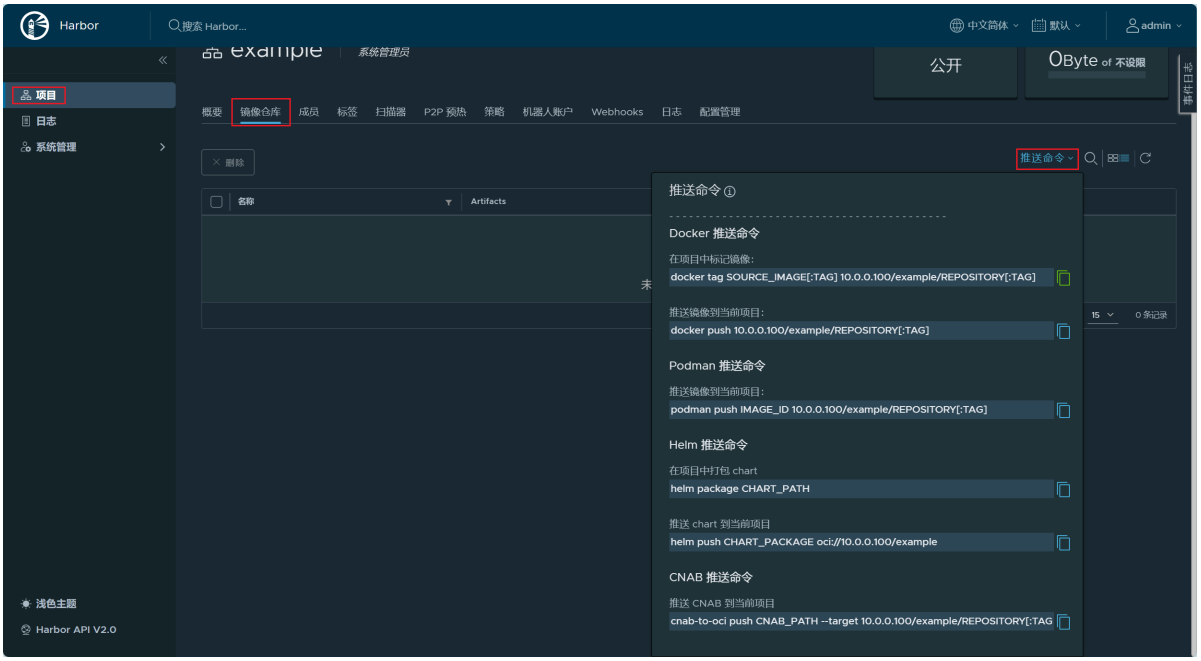






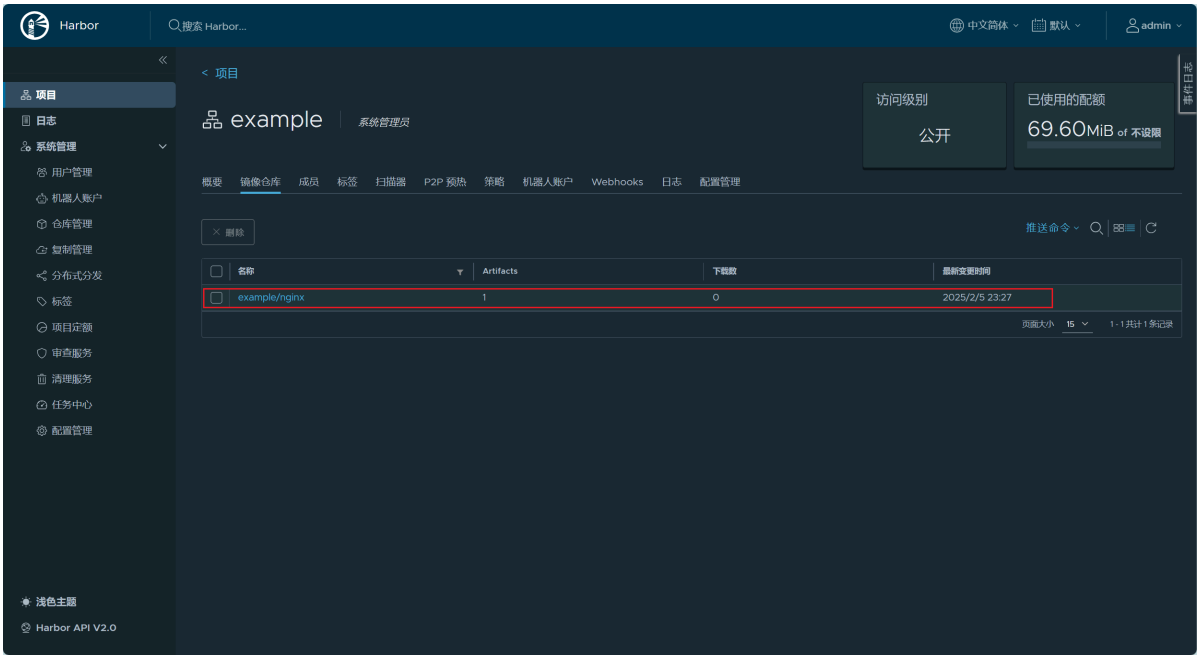
命令行登录 Harbor

查看推送地址与相关命令



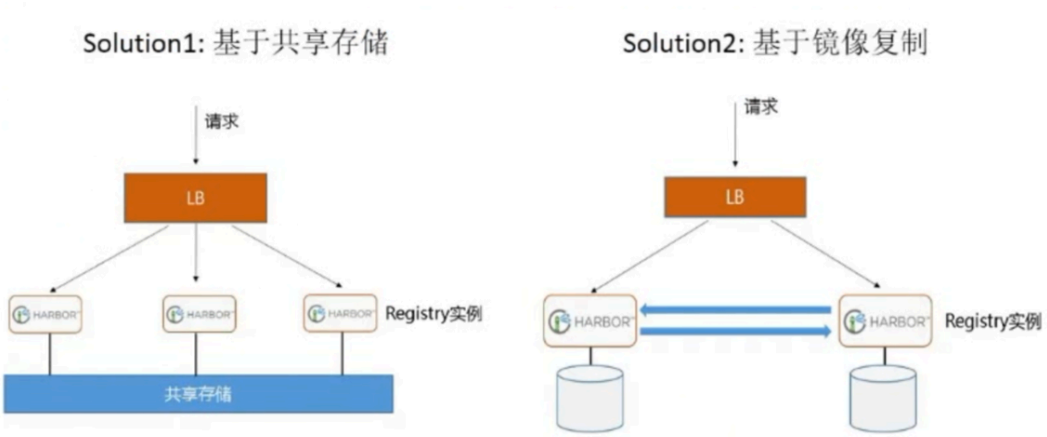

```
# 推送镜像前，要将镜像改名
# docker tag SOURCE_IMAGE[:TAG] 10.0.0.100/example/REPOSITORY[:TAG]
[root@ubuntu2204 ~]#docker tag nginx:latest 10.0.0.100/example/nginx:latest

# 目前推送会报错
# 首先默认会使用https推送，但是我们这边使用http，所以需要将host加入Insecure Registries中
# 这里不安全的仓库，指的就是使用http，而不是https
[root@ubuntu2204 ~]#cat /etc/docker/daemon.json
{
   "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"],
   "insecure-registries": ["harbor.mystical.org", "10.0.0.100"]
}

# 重启docker服务
```

```
[root@ubuntu2204 ~]#systemctl restart docker

# 查看
[root@ubuntu2204 ~]#docker info
......
 Insecure Registries:
  10.0.0.100
  harbor.mystical.org
  127.0.0.0/8
......

# 推送镜像前，要登录harbor
[root@ubuntu2204 ~]#docker login 10.0.0.100
Username: zhangyifeng
Password:
# 只需登录一次，登录信息会记录在config.json文件中，无需反复登录
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded

# 推送镜像
[root@ubuntu2204 ~]#docker push 10.0.0.100/example/nginx:v0.1
The push refers to repository [10.0.0.100/example/nginx]
e2eb04df0bda: Pushed
1b78ffef68d1: Pushed
16649054d94a: Pushed
a280e15d559d: Pushed
0b2dafd61482: Pushed
2cdcaebcf23c: Pushed
7914c8f600f5: Pushed
latest: digest:
sha256:0b2c307c84395005578e9e35ee4bc6c00387cd4573e8098d656245895dc1f7d5 size:
1778

# 浏览器查看
```



```
# 在10.0.0.100查看harbor数据目录
[root@ubuntu2204 harbor]#du -sh /data/harbor
119M    /data/harbor

# 在其他服务器上拉去harbor上的镜像
# 将其加入insecure-registries
[root@ubuntu2204 ~]#cat /etc/docker/daemon.json
{
   "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"],
   "insecure-registries": ["harbor.mystical.org", "10.0.0.100"]
}

# 重启docker服务
[root@ubuntu2204 ~]#systemctl restart docker

# 拉取镜像
[root@ubuntu2204 ~]#docker pull 10.0.0.100/example/nginx:v0.1
latest: Pulling from example/nginx
d2eb42b4a5eb: Pull complete
ee083de5ceda: Pull complete
5afd6583b29c: Pull complete
8c2914db26a3: Pull complete
1e8aefce6919: Pull complete
a982d09283a6: Pull complete
ab571a6216e3: Pull complete
Digest: sha256:0b2c307c84395005578e9e35ee4bc6c00387cd4573e8098d656245895dc1f7d5
Status: Downloaded newer image for 10.0.0.100/example/nginx:latest
10.0.0.100/example/nginx:latest
```

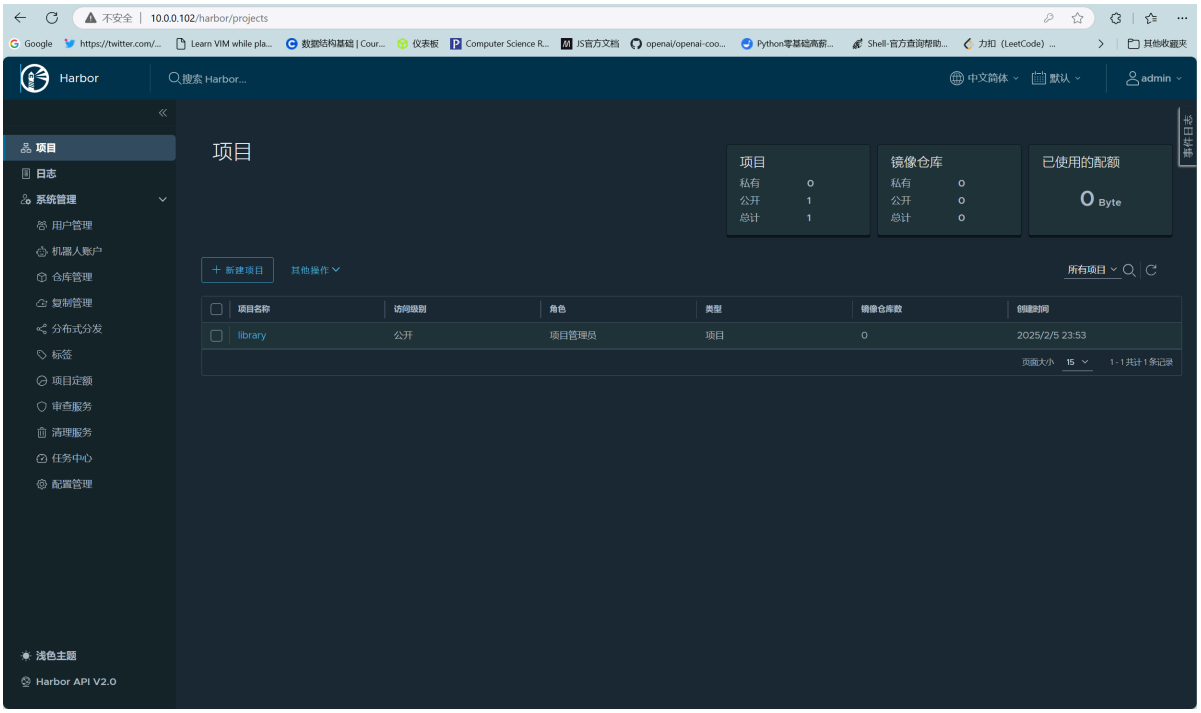
实现 Harbor 高可用



Harbor支持基于策略的Docker镜像复制功能，这类似于MySQL的主从同步，其可以实现不同的数据中心、不同的运行环境之间同步镜像，并提供友好的管理界面，大大简化了实际运维中的镜像管理工作，已经有用很多互联网公司使用harbor搭建内网docker仓库的案例，并且还有实现了双向复制功能

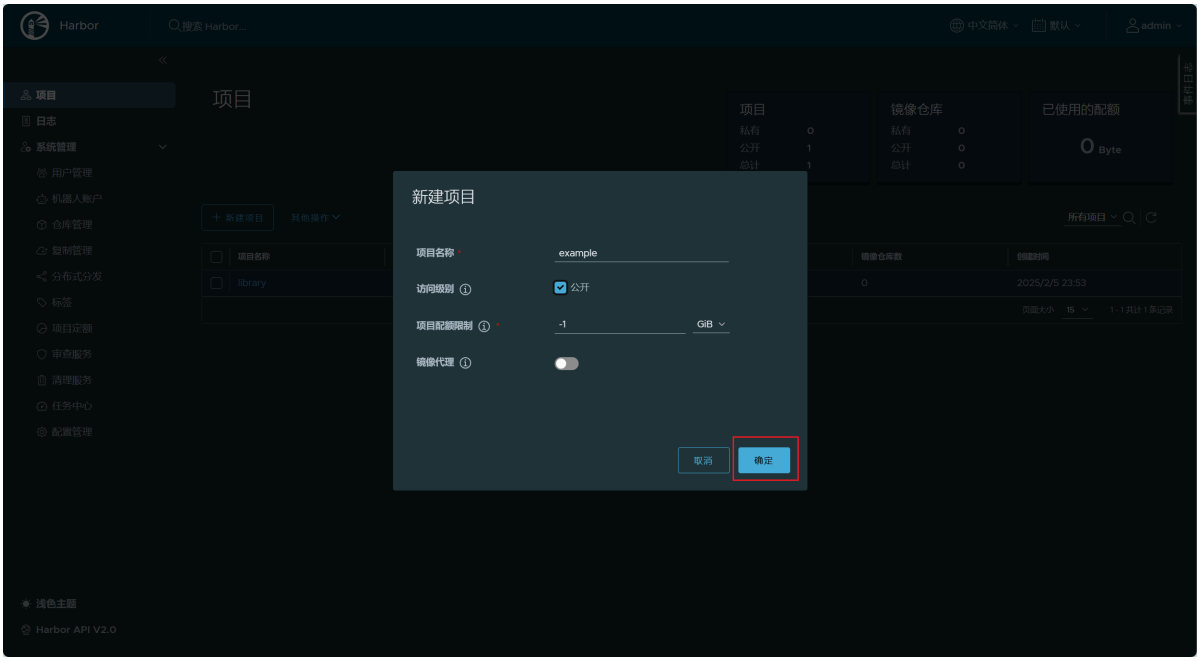
安装第二台 harbor主机

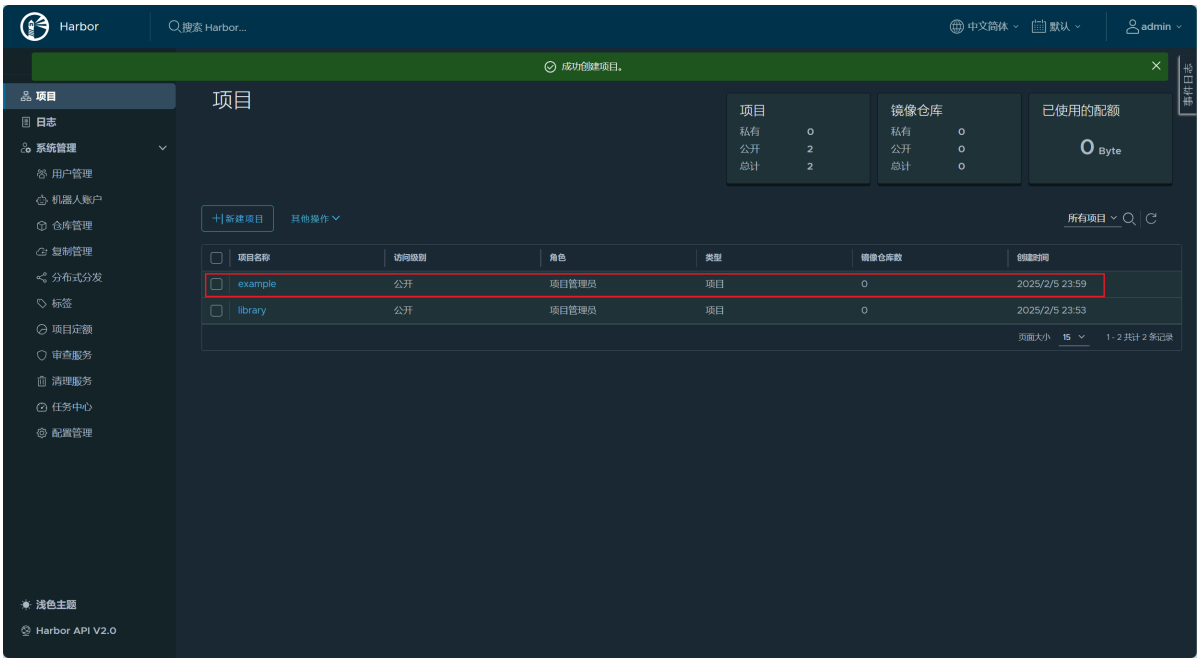
在第二台主机上安装部署好harbor，并登录系统



第二台harbor上新建项目

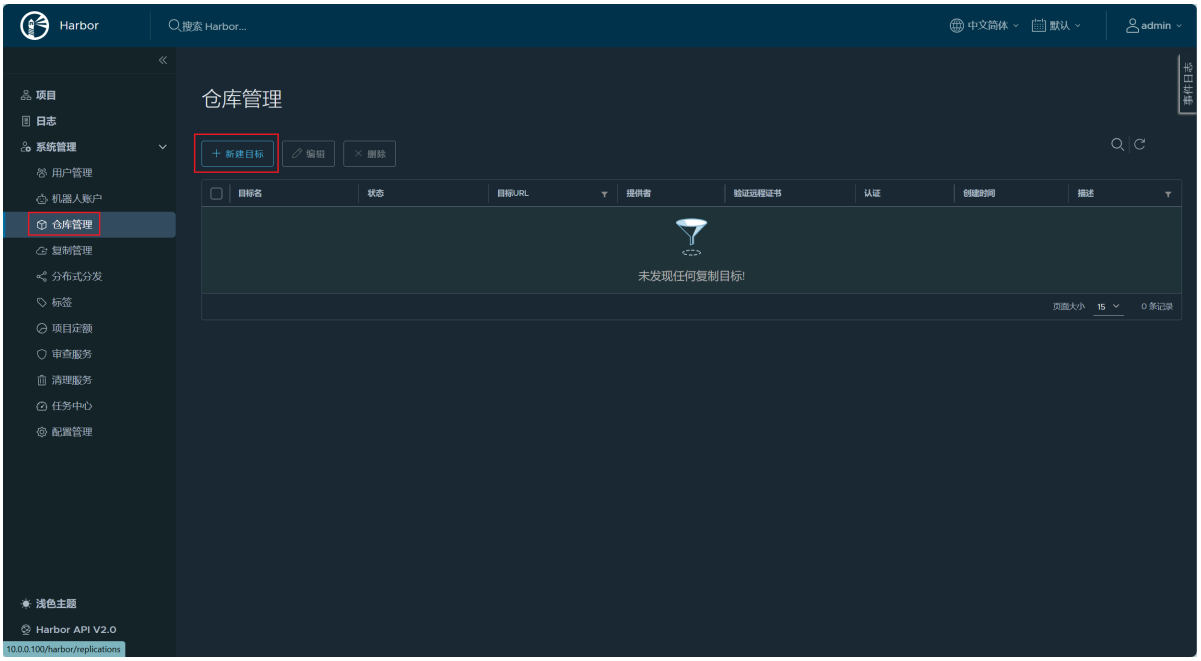
参考第一台harbor服务器的项目名称，在第二台harbor服务器上新建与之同名的项目

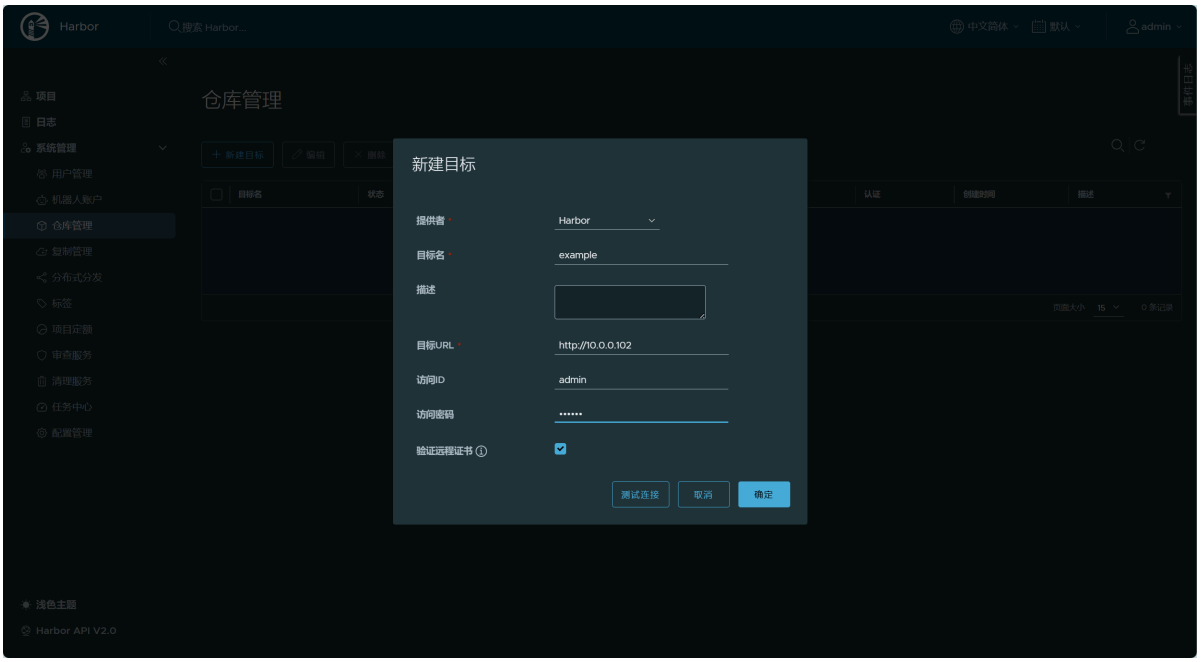


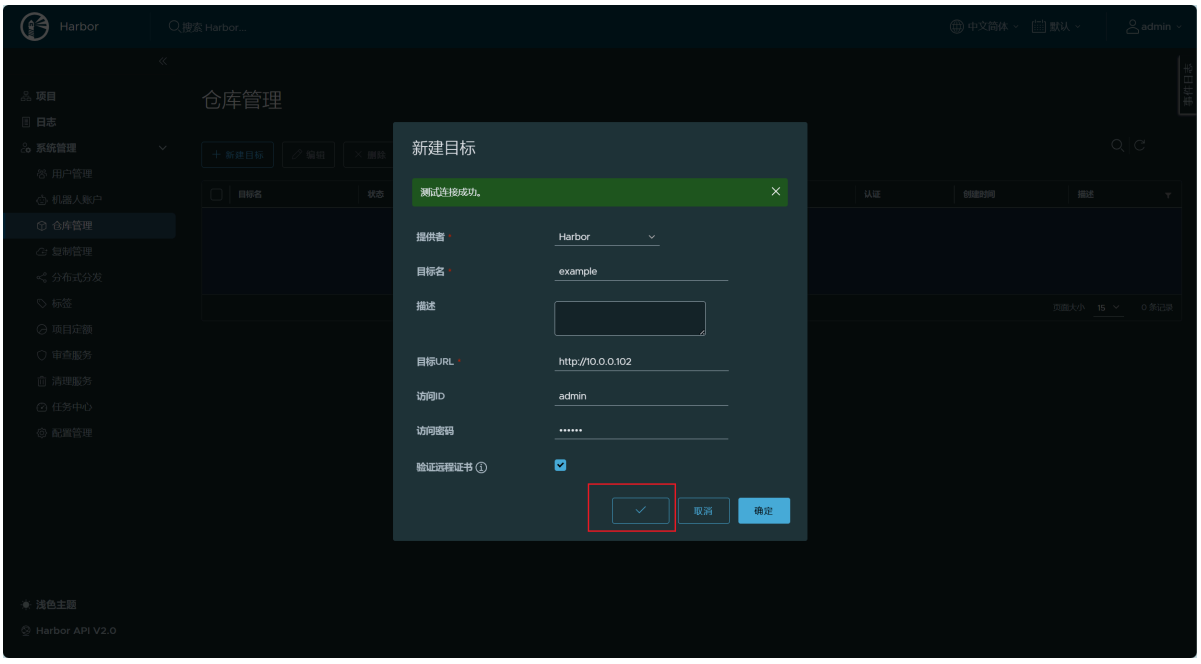


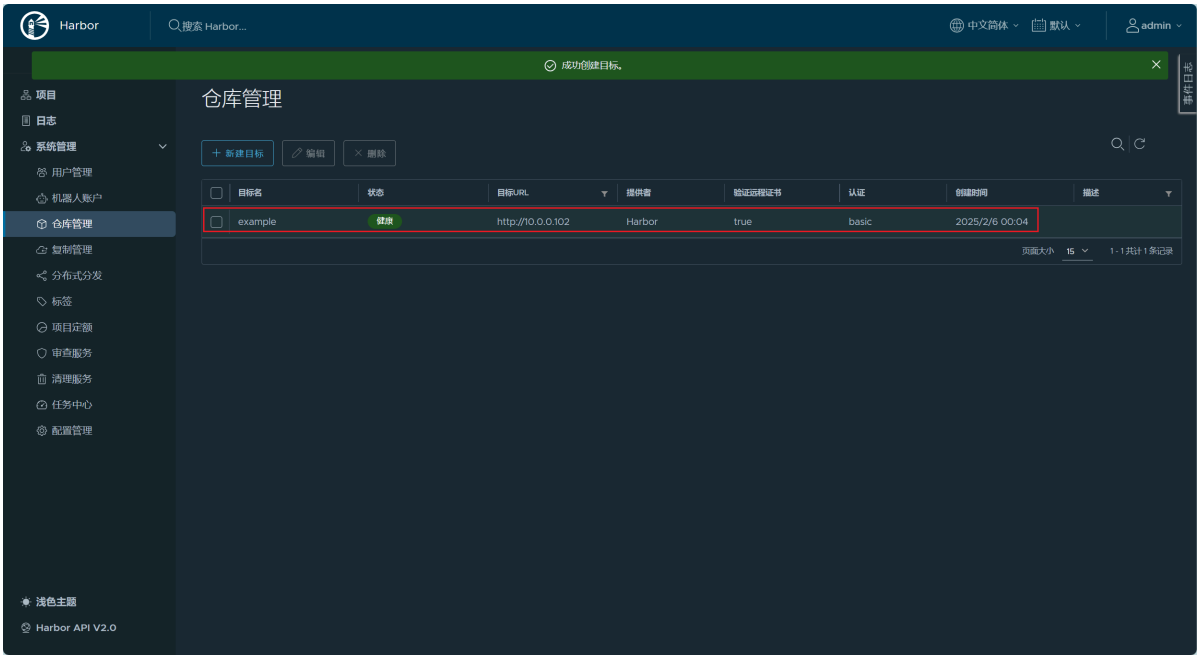
在源主机上新建目标

注意：有镜像的，将要传递镜像给目的主机的服务器为源主机，该实验中为10.0.0.100

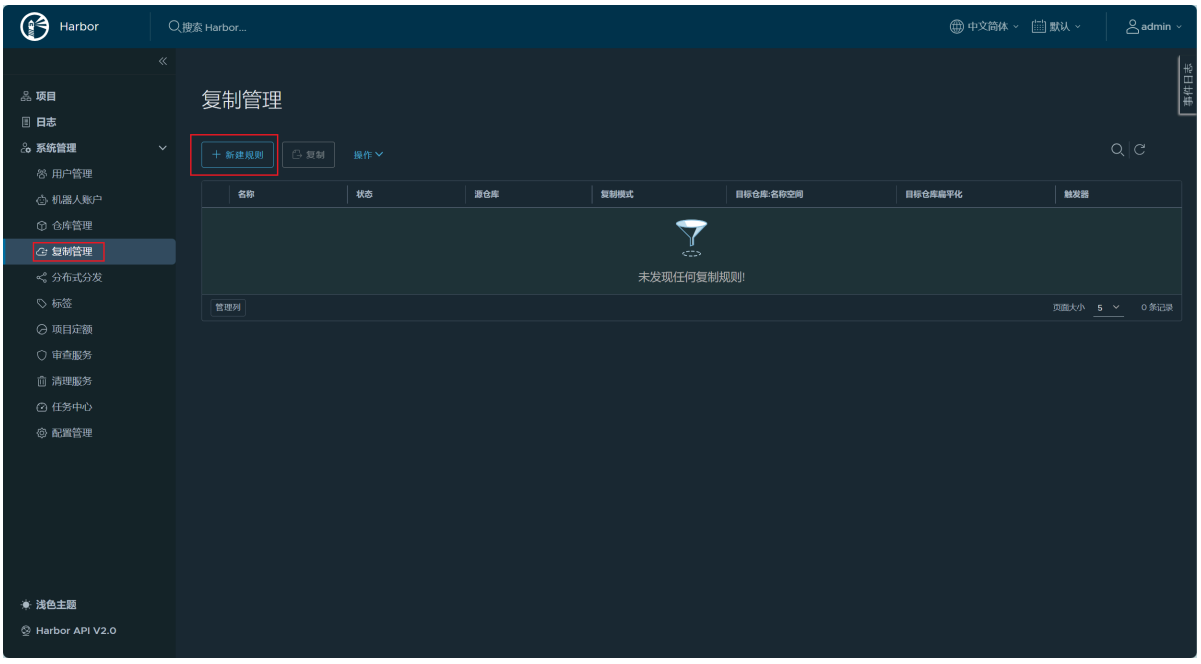



Harbor
搜索 Harbor...
中文简体
默认
admin
项目
日志
系统管理
用户管理
机器人账户
仓库管理
复制管理
分布式分发
标签
项目定额
审查服务
清理服务
任务中心
配置管理
浅色主题
Harbor API V2.0
仓库管理
新建目标
编辑
删除
目标名
状态
认证
创建时间
描述
新建目标
提供者 *
Harbor
目标名 *
example
描述
目标URL *
http://10.0.0.102
访问ID
admin
访问密码
验证远程证书
测试连接
取消
确定

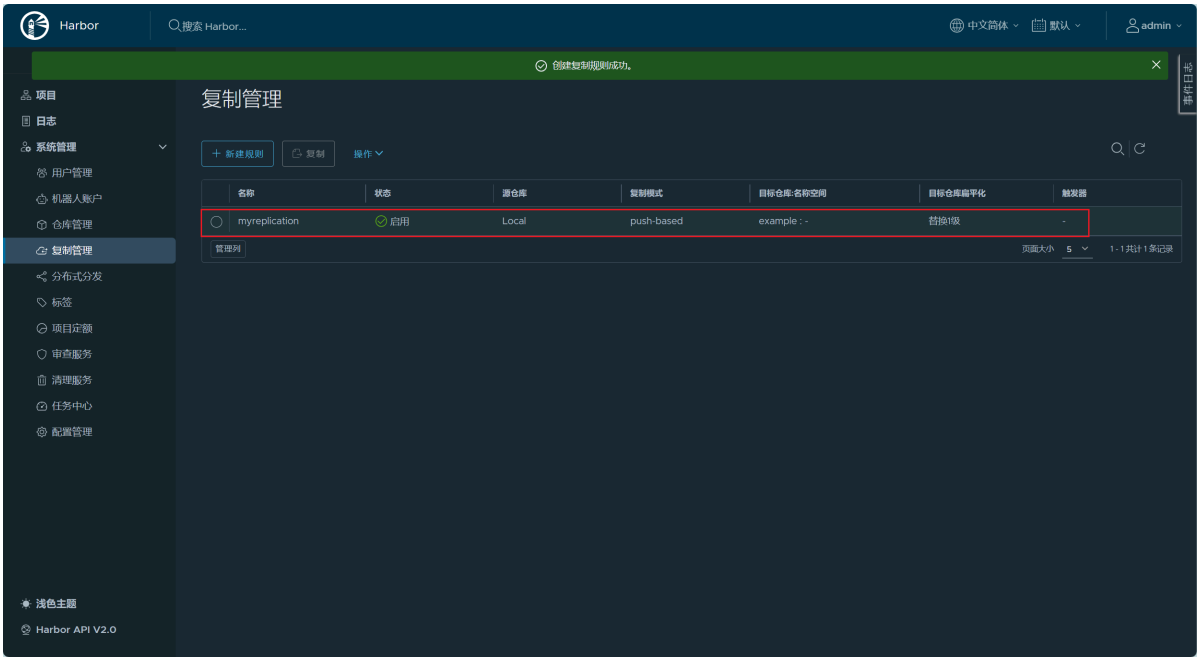

Harbor
搜索 Harbor...
中文简体
默认
admin
项目
日志
系统管理
用户管理
机器人账户
仓库管理
复制管理
分布式分发
标签
项目定额
审查服务
清理服务
任务中心
配置管理
浅色主题
Harbor API V2.0
仓库管理
新建目标
编辑
删除
目标名
状态
认证
创建时间
描述
新建目标
测试连接成功。
提供者 *
Harbor
目标名 *
example
描述
目标URL *
http://10.0.0.102
访问ID
admin
访问密码
验证远程证书
取消
确定


Harbor
搜索 Harbor...
中文简体
默认
admin
成功创建目标。
项目
日志
系统管理
用户管理
机器人账户
仓库管理
复制管理
分布式分发
标签
项目定额
审查服务
清理服务
任务中心
配置管理
浅色主题
Harbor API V2.0
仓库管理
新建目标
编辑
删除
目标名
状态
目标URL
提供者
验证远程证书
认证
创建时间
描述
example
健康
http://10.0.0.102
Harbor
true
basic
2025/2/6 00:04
页面大小 15
1 - 1 共计 1 条记录

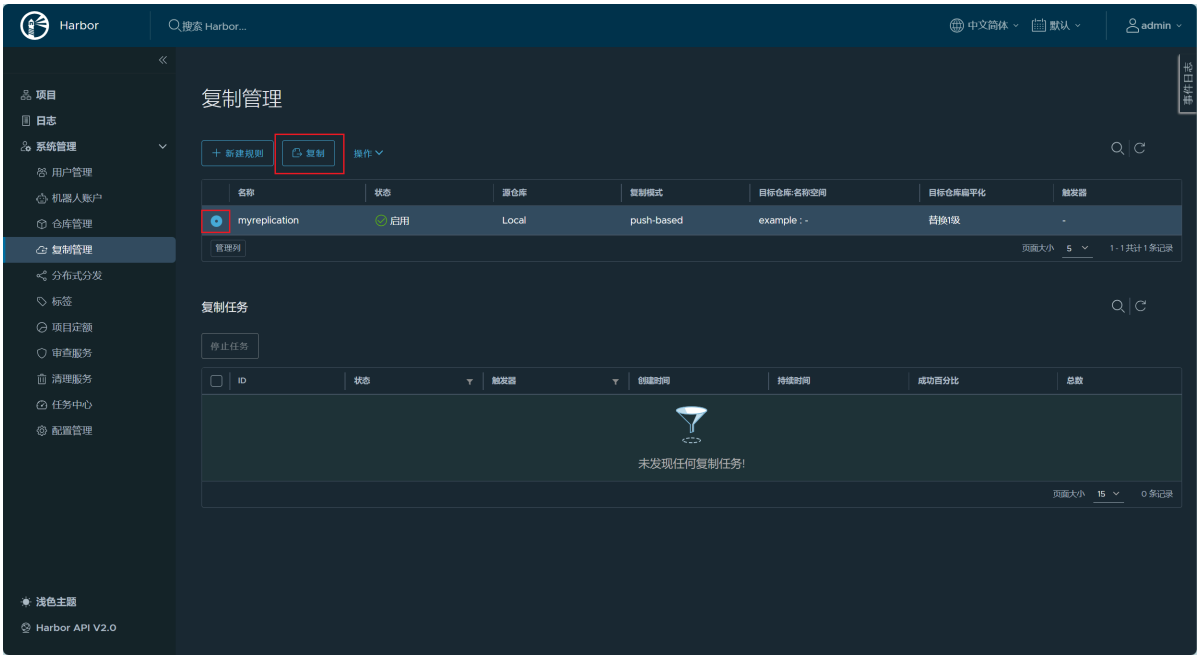
复制管理

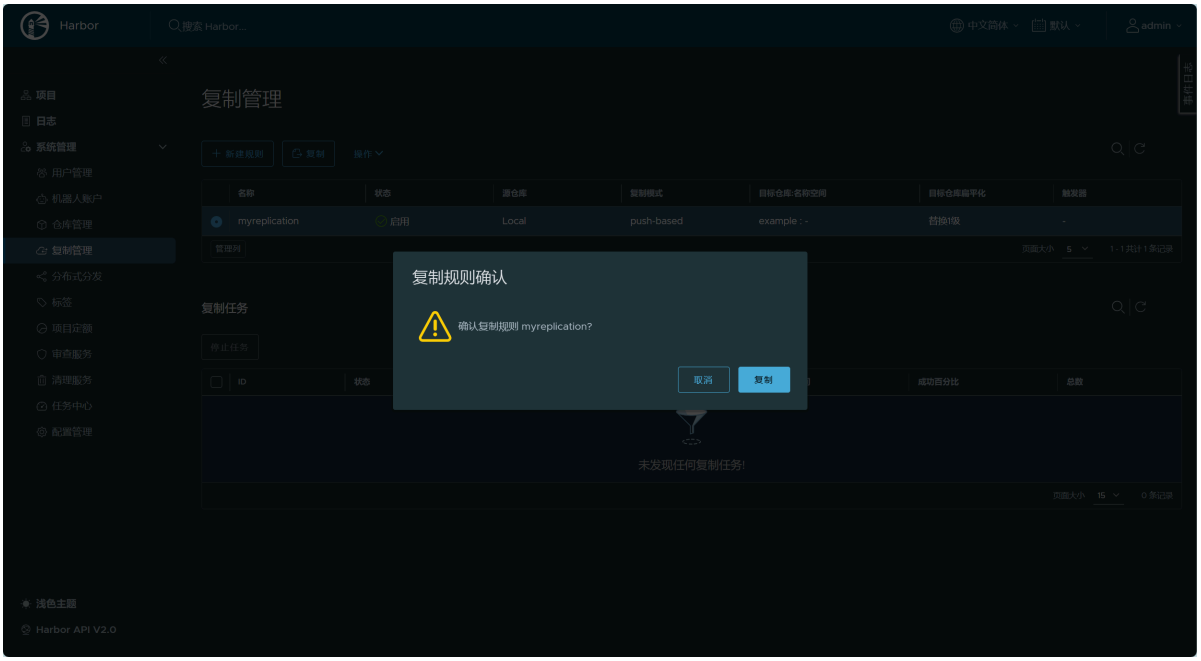

Harbor
搜索 Harbor...
中文简体
默认
admin
项目
日志
系统管理
用户管理
机器人账户
仓库管理
复制管理
分布式分发
标签
项目定额
审查服务
清理服务
任务中心
配置管理
浅色主题
Harbor API V2.0
复制管理
新建规则
复制
操作
名称
状态
源仓库
复制模式
目标仓库:名称空间
目标仓库扁平化
触发器
未发现任何复制规则!
管理列
页面大小
5
0 条记录

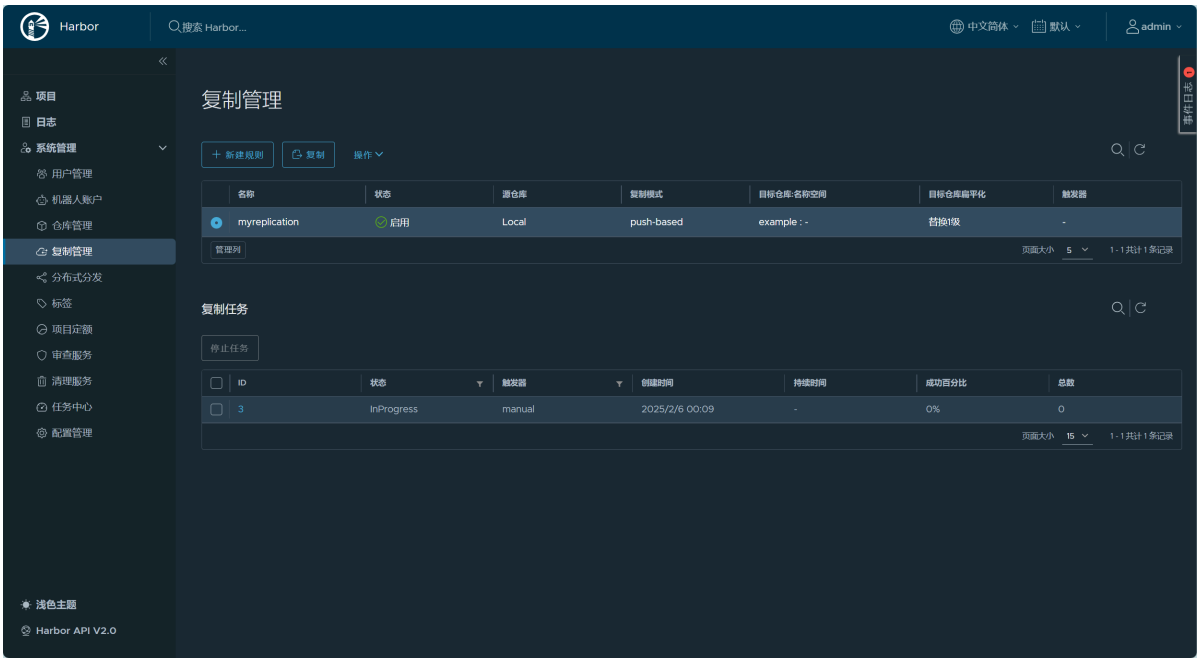

新建规则
名称
myreplication
描述
复制模式
Push-based
Pull-based
源资源过滤器
名称:
Tag:
匹配
标签:
资源:
全部
目标仓库
example-http://10.0.0.102
目标
名称空间:
仓库扁平化:
替换1级
触发模式
事件驱动
删除本地资源时同时也删除远程的资源。
带宽
-1
Kbps
选项
覆盖
分块复制
取消
保存

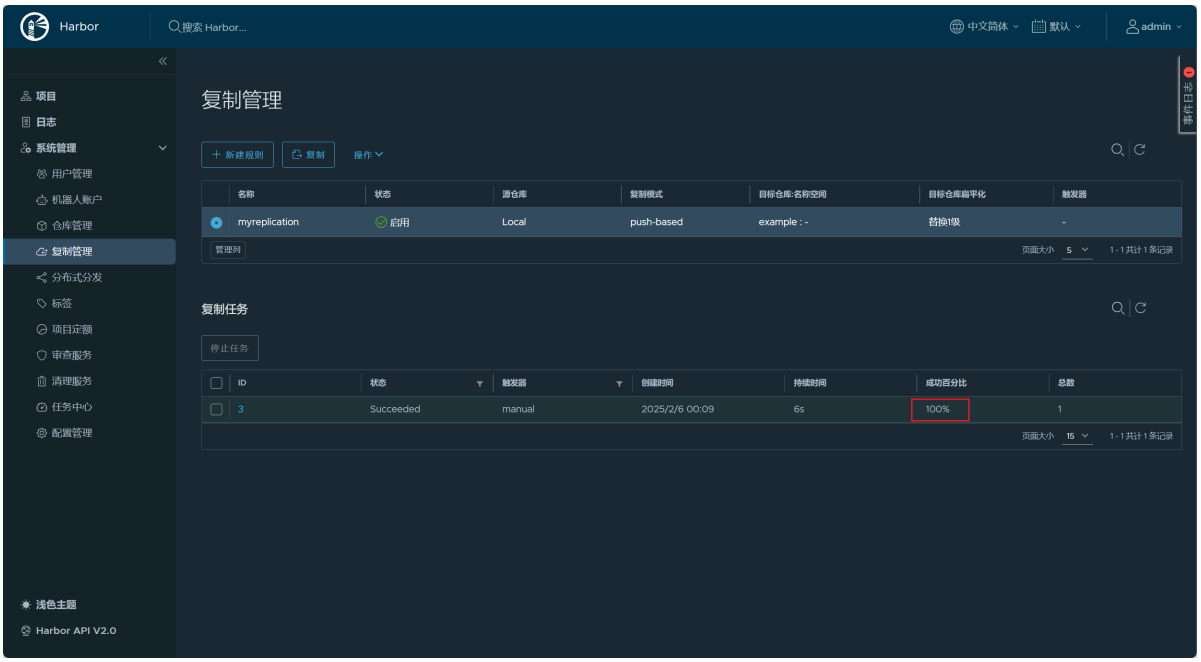

创建复制规则成功。
复制管理
新建规则
复制
操作
名称
状态
源仓库
复制模式
目标仓库:名称空间
目标仓库扁平化
触发器
myreplication
启用
Local
push-based
example : -
替换1级
管理列
页面大小
5
1 - 1 共计 1 条记录

初始手动触发复制

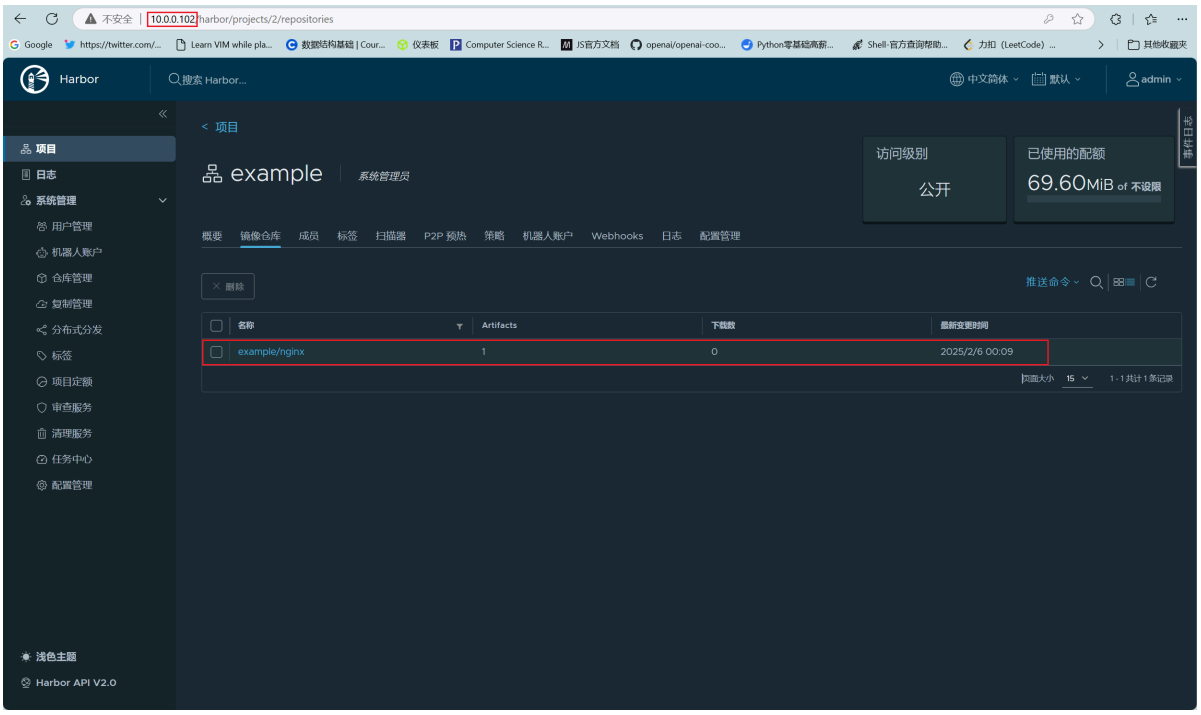

Harbor
搜索 Harbor...
中文简体
默认
admin
项目
日志
系统管理
用户管理
机器人账户
仓库管理
复制管理
分布式分发
标签
项目定额
审查服务
清理服务
任务中心
配置管理
浅色主题
Harbor API V2.0
复制管理
新建规则
复制
操作
名称
状态
源仓库
复制模式
目标仓库:名称空间
目标仓库扁平化
触发器
myreplication
启用
Local
push-based
example : -
替换1级
-
管理列
页面大小 5
1 - 1 共计 1 条记录
复制任务
停止任务
ID
状态
触发器
创建时间
持续时间
成功百分比
总数
未发现任何复制任务!
页面大小 15
0 条记录
事件日志


Harbor
复制规则确认
确认复制规则 myreplication?
取消
复制
复制管理
复制任务
未发现任何复制任务!


Harbor
搜索 Harbor...
中文简体
默认
admin
项目
日志
系统管理
用户管理
机器人账户
仓库管理
复制管理
分布式分发
标签
项目定额
审查服务
清理服务
任务中心
配置管理
浅色主题
Harbor API V2.0
复制管理
新建规则
复制
操作
名称
状态
源仓库
复制模式
目标仓库:名称空间
目标仓库扁平化
触发器
myreplication
启用
Local
push-based
example : -
替换1级
-
管理列
页面大小 5
1 - 1 共计 1 条记录
复制任务
停止任务
ID
状态
触发器
创建时间
持续时间
成功百分比
总数
3
InProgress
manual
2025/2/6 00:09
-
0%
0
页面大小 15
1 - 1 共计 1 条记录


Harbor
搜索 Harbor...
中文简体
默认
admin
项目
日志
系统管理
用户管理
机器人账户
仓库管理
复制管理
分布式分发
标签
项目定额
审查服务
清理服务
任务中心
配置管理
浅色主题
Harbor API V2.0
复制管理
新建规则
复制
操作
名称
状态
源仓库
复制模式
目标仓库:名称空间
目标仓库扁平化
触发器
myreplication
启用
Local
push-based
example : -
替换1级
-
管理列
页面大小 5
1 - 1 共计 1 条记录
复制任务
停止任务
ID
状态
触发器
创建时间
持续时间
成功百分比
总数
3
Succeeded
manual
2025/2/6 00:09
6s
100%
1
页面大小 15
1 - 1 共计 1 条记录

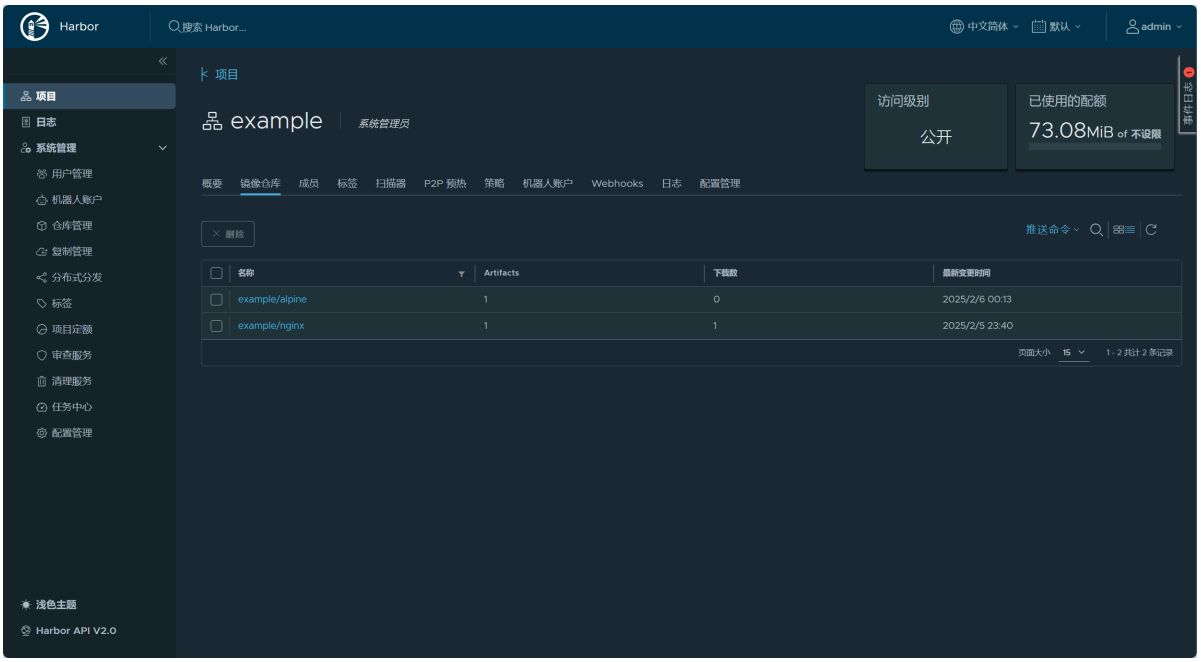
在目标服务器上查看



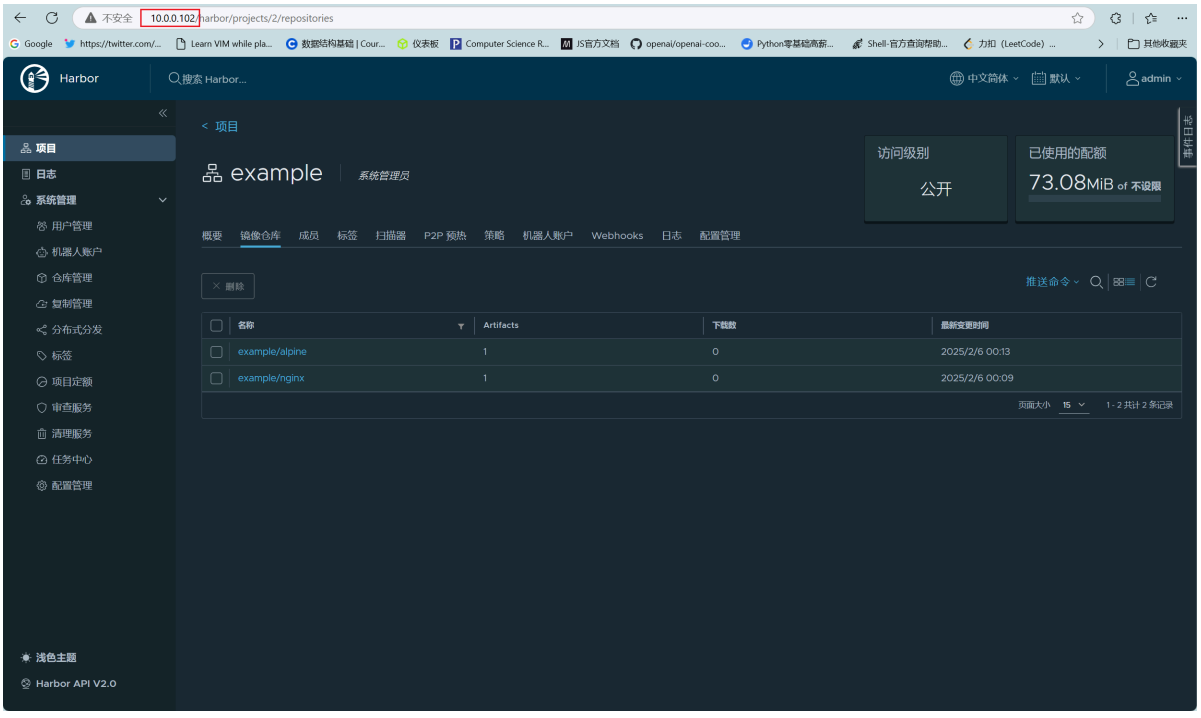
在镜像上传至源服务器，观察目标服务器

```
[root@ubuntu2204 ~]#docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
1f3e46996e29: Pull complete
Digest: sha256:56fa17d2a7e7f168a043a2712e63aed1f8543aeafdcee47c58dcffe38ed51099
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest

[root@ubuntu2204 ~]#docker tag alpine:latest 10.0.0.100/example/alpine:latest
[root@ubuntu2204 ~]#docker push 10.0.0.100/example/alpine:latest
The push refers to repository [10.0.0.100/example/alpine]
a0904247e36a: Pushed
latest: digest:
sha256:c10f729849a3b03cbf222e2220245dd44c39a06d444aa32cc30a35c4c1aba59d size: 527
```

查看源harbor



查看目标服务器



反向再做一遍，实现双向复制