搭建一套高可用的Harbor容器镜像仓库

mp.weixin.qq.com/s/hT1HfVUyhYQ4uSe7JiHG6Q

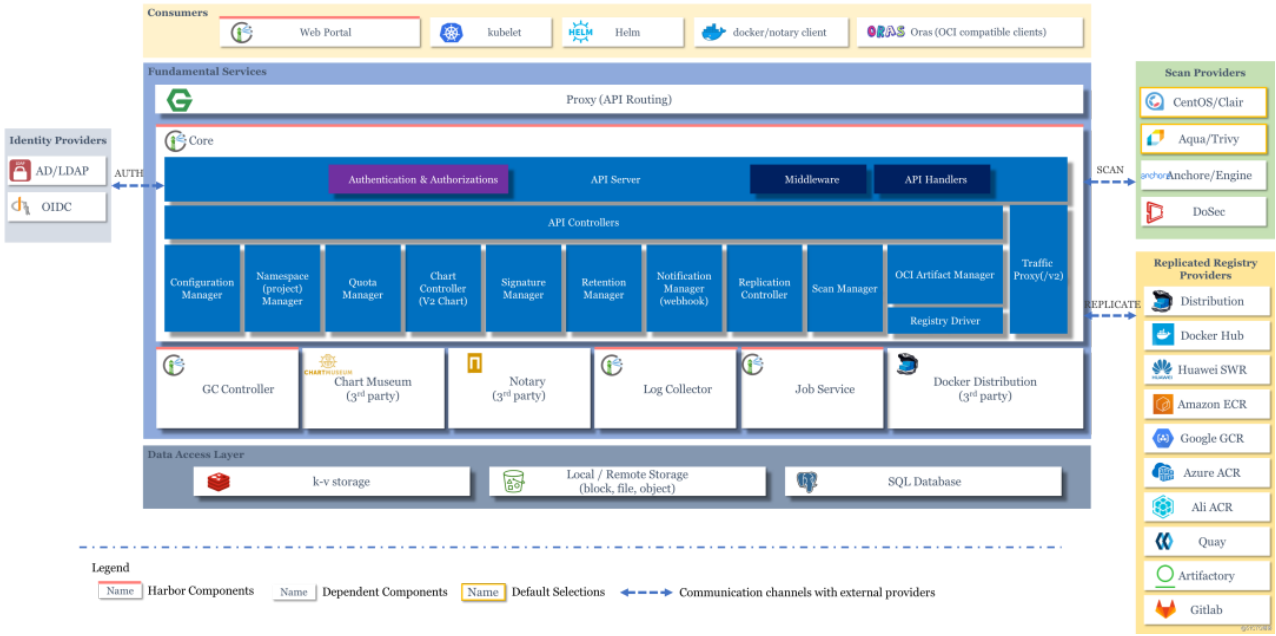
来源：https://blog.51cto.com/zero01/2530940

Harbor简介

Docker容器应用的开发和运行离不开可靠的镜像管理，Docker官方提供了原生的Registry，但其功能比较简单，而且没有可视化界面，自然无法满足企业级的需求。虽然Docker官方也提供了公共的镜像仓库，但是从安全和效率等方面考虑，部署私有环境内的Registry也是非常必要的。

为了解决以上需求，VMware公司推出了Harbor，Harbor 是为企业用户设计的容器镜像仓库开源项目，包括了权限管理(RBAC)、LDAP、审计、安全漏洞扫描、镜像验真、管理界面、自我注册、HA 等企业必需的功能，同时针对中国用户的特点，设计镜像复制和中文支持等功能。

Harbor的架构示意图：



Harbor的GitHub仓库地址如下：

- https://github.com/goharbor/harbor
- Wiki：https://github.com/goharbor/harbor/wiki

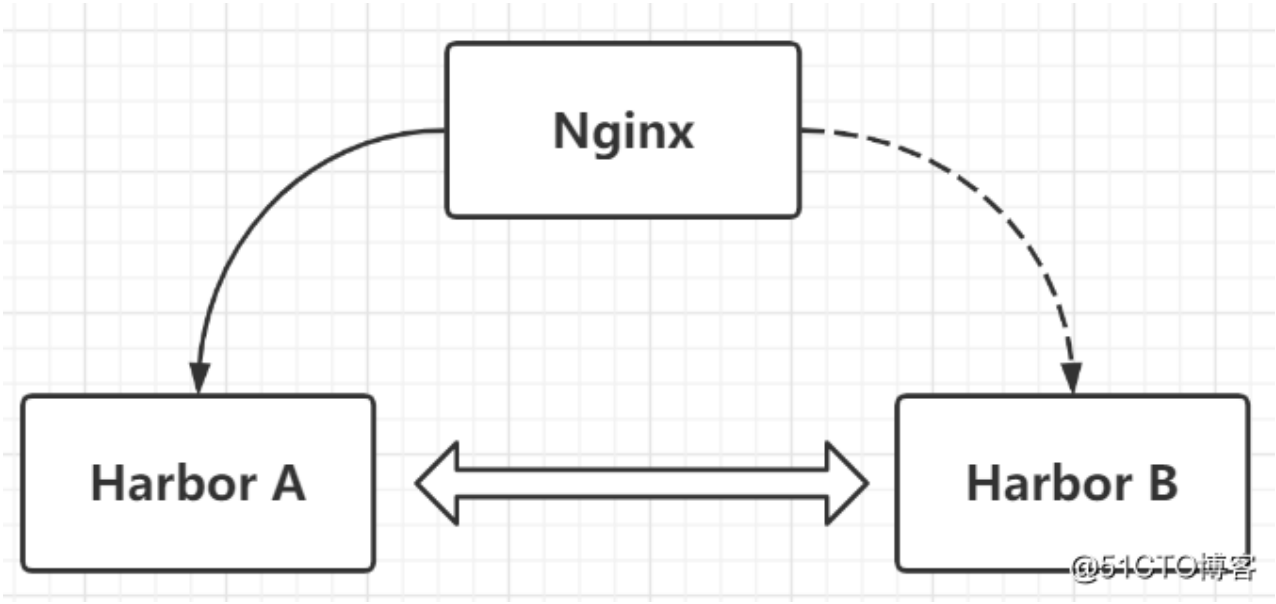
Harbor高可用部署

官方的安装文档：

https://goharbor.io/docs/2.0.0/install-config/

本文采用的高可用方案是Harbor的双主复制，该方案比较简单，需要搭建至少两个Harbor节点，并且节点之间能够互相复制，然后通过nginx代理Harbor节点提供外部访问。这里采用的高可用方案级别没那么高，因为主要是通过Nginx代理其中一个节点，该节点挂掉后需要手动修改Nginx配置文件去代理另一个可用节点。

示意图如下：



所以此方案比较适合中小型公司，而且Harbor主要是给公司内部的开发人员使用的，通常只需要保证分钟级的高可用性就可以了。另外还有一点就是，云环境基本无法使用keepalived，因为云服务商一般不支持自定义外网可访问的虚拟IP，要么就是使用起来非常麻烦。这也是为什么没有采用keepalived的原因之一，当然，如果是部署在内网服务器上也是可以采用keepalived的。

准备工作

我这里使用了三台CentOS-7.7的虚拟机，具体信息如下表：

系统版本	IP地址	节点角色	CPU	Memory	Hostname
CentOS-7.7	192.168.243.138	master	\>=2	\>=2G	m1
CentOS-7.7	192.168.243.139	worker	\>=2	\>=2G	s1
CentOS-7.7	192.168.243.140	worker	\>=2	\>=2G	s2

这三台机器均需事先安装好Docker，由于安装过程比较简单这里不进行介绍，可以参考官方文档：

https://docs.docker.com/engine/install/centos/

安装Harbor（worker节点）

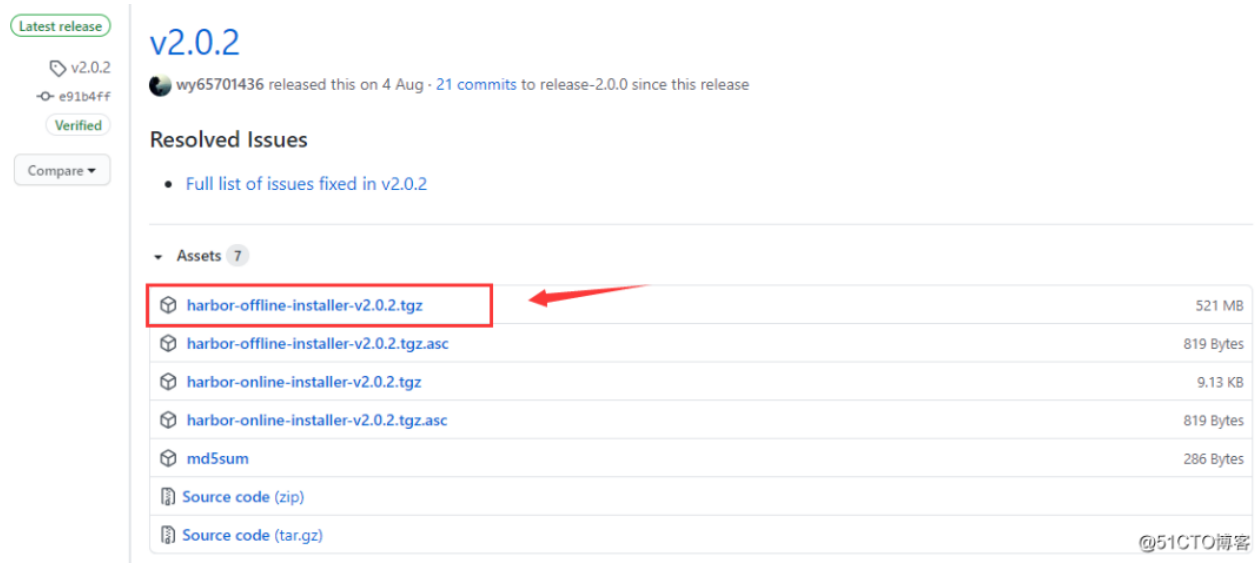
在两台 `worker` 节点上分别安装Harbor，由于官方提供了安装脚本，安装过程还是比较简单的。具体步骤如下：

下载安装包

首先下载官方的离线安装包，当然你能科学上网的话使用在线安装包也可以：

> https://github.com/goharbor/harbor/releases

我这里下载的是`2.0.2`版本的离线安装包：



下载完成后，将压缩包上传到两个`worker`节点：

```
[root@s1 /usr/local/src]# ls
harbor-offline-installer-v2.0.2.tgz

[root@s2 /usr/local/src]# ls
harbor-offline-installer-v2.0.2.tgz
```

然后对其进行解压：

```
$ tar -zxvf harbor-offline-installer-v2.0.2.tgz
```

解压后的目录文件如下：

```
[root@s1 /usr/local/src]# cd harbor
[root@s1 /usr/local/src/harbor]# ls
common.sh  harbor.v2.0.2.tar.gz  harbor.yml.tmpl  install.sh  LICENSE  prepare
[root@s1 /usr/local/src/harbor]#
```

配置harbor

将配置文件模板拷贝一份，并命名为`harbor.yml`，这是默认的配置文件名称：

```
[root@s1 /usr/local/src/harbor]# cp harbor.yml.tmpl harbor.yml
```

编辑`harbor.yml`文件，按照如下说明修改几处配置项：

```
[root@s1 /usr/local/src/harbor]# vim harbor.yml
# 修改为当前所在节点的ip
hostname: 192.168.243.139
# 登录界面的密码
harbor_admin_password: Harbor12345
# harbor的版本号
_version: 2.0.2

# 将https相关的配置给注释掉，这里为了简单只使用http，而且也可以在nginx那一层去做https
# https related config
#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
```

执行安装脚本

准备好配置文件之后，安装`docker-compose`，因为Harbor的安装脚本是基于`docker-compose`去安装的。下载`docker-compose`然后放到`/usr/local/bin/`目录下，再更改一下权限即可：

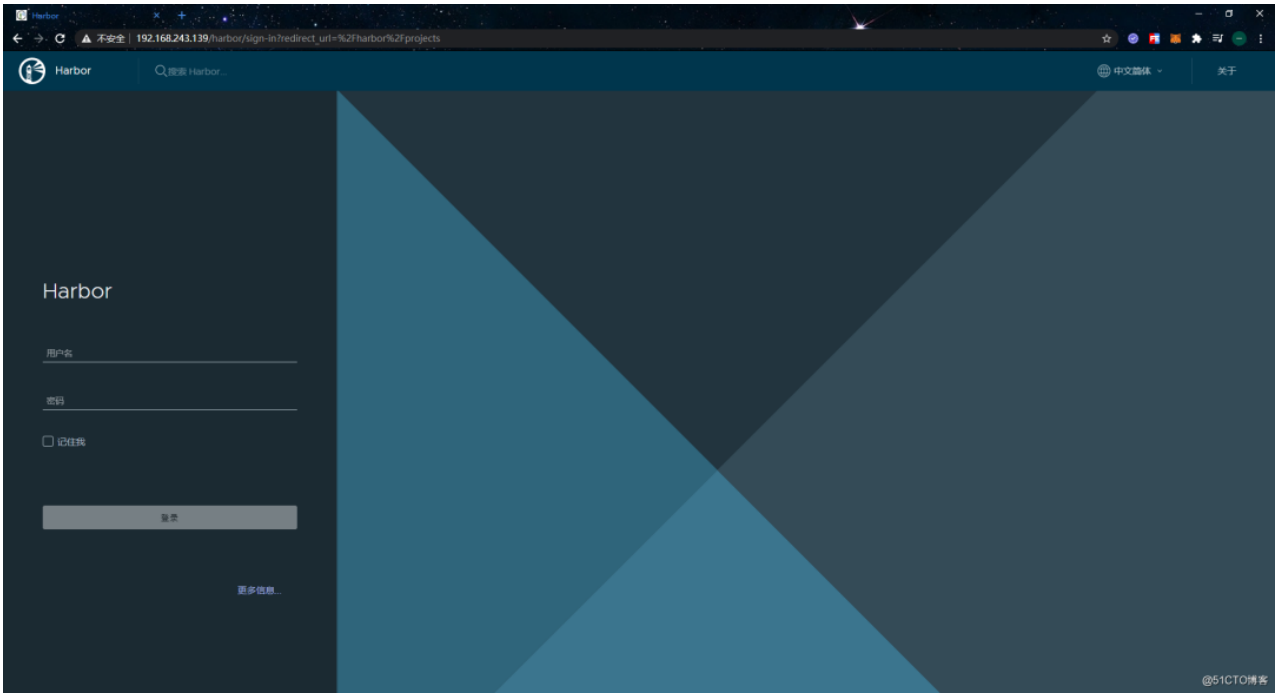
```
[root@s1 ~]# curl -L
"https://github.com/docker/compose/releases/download/1.26.2/docker-compose-$(uname
-s)-$(uname -m)" -o /usr/local/bin/docker-compose
[root@s1 ~]# chmod 755 /usr/local/bin/docker-compose
```

然后就可以运行Harbor的安装脚本了：

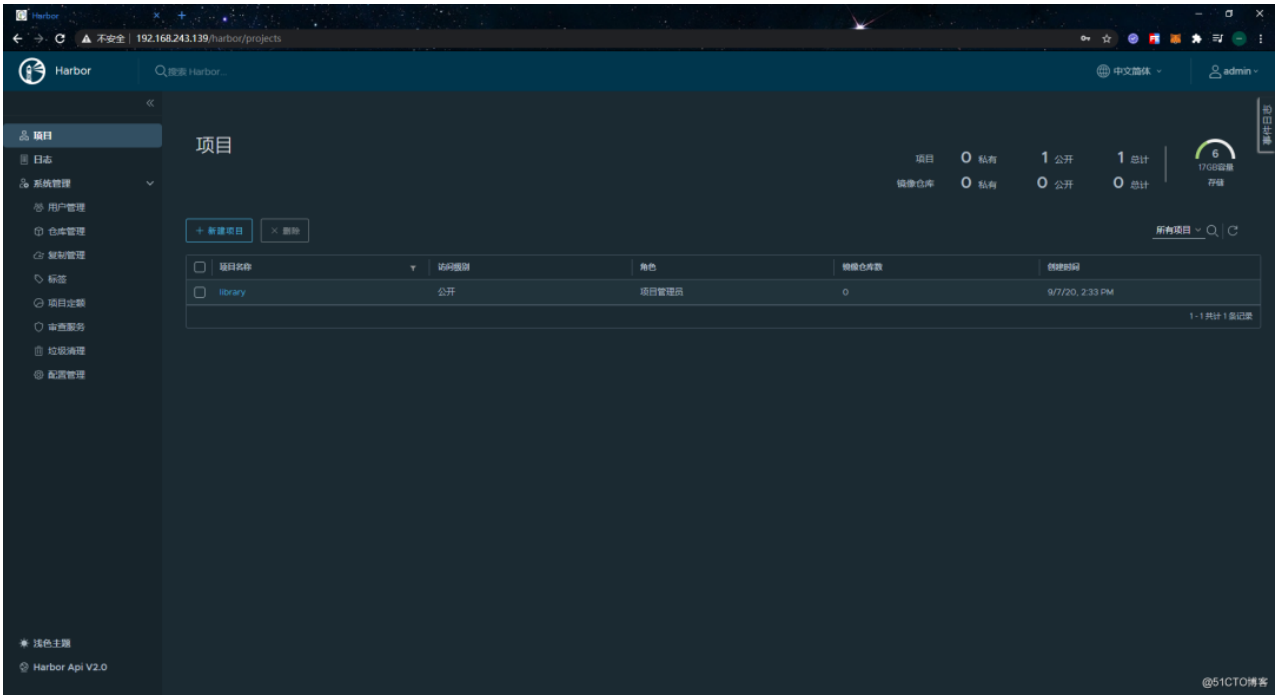
```
[root@s1 /usr/local/src/harbor]# ./install.sh

[Step 0]: checking if docker is installed ...

Note: docker version: 19.03.12

[Step 1]: checking docker-compose is installed ...

Note: docker-compose version: 1.26.2

[Step 2]: loading Harbor images ...
Loaded image: goharbor/prepare:v2.0.2
Loaded image: goharbor/harbor-jobservice:v2.0.2
Loaded image: goharbor/harbor-registryctl:v2.0.2
Loaded image: goharbor/registry-photon:v2.0.2
Loaded image: goharbor/harbor-core:v2.0.2
Loaded image: goharbor/notary-signer-photon:v2.0.2
Loaded image: goharbor/clair-photon:v2.0.2
Loaded image: goharbor/trivy-adapter-photon:v2.0.2
Loaded image: goharbor/harbor-log:v2.0.2
Loaded image: goharbor/nginx-photon:v2.0.2
Loaded image: goharbor/clair-adapter-photon:v2.0.2
Loaded image: goharbor/chartmuseum-photon:v2.0.2
Loaded image: goharbor/harbor-portal:v2.0.2
Loaded image: goharbor/harbor-db:v2.0.2
Loaded image: goharbor/redis-photon:v2.0.2
Loaded image: goharbor/notary-server-photon:v2.0.2

[Step 3]: preparing environment ...

[Step 4]: preparing harbor configs ...
prepare base dir is set to /usr/local/src/harbor
WARNING:root:WARNING: HTTP protocol is insecure. Harbor will deprecate http
protocol in the future. Please make sure to upgrade to https
Generated configuration file: /config/log/logrotate.conf
Generated configuration file: /config/log/rsyslog_docker.conf
Generated configuration file: /config/nginx/nginx.conf
Generated configuration file: /config/core/env
Generated configuration file: /config/core/app.conf
Generated configuration file: /config/registry/config.yml
Generated configuration file: /config/registryctl/env
Generated configuration file: /config/registryctl/config.yml
Generated configuration file: /config/db/env
Generated configuration file: /config/jobservice/env
Generated configuration file: /config/jobservice/config.yml
Generated and saved secret to file: /data/secret/keys/secretkey
Successfully called func: create_root_cert
Generated configuration file: /compose_location/docker-compose.yml
Clean up the input dir

[Step 5]: starting Harbor ...
Creating network "harbor_harbor" with the default driver
Creating harbor-log ... done
Creating harbor-db     ... done
Creating harbor-portal ... done
Creating redis         ... done
Creating registryctl   ... done
Creating registry      ... done
Creating harbor-core   ... done
```

```
Creating harbor-jobservice ... done
Creating nginx             ... done
✔ ----Harbor has been installed and started successfully.----
[root@s1 /usr/local/src/harbor]#
```

安装完成，使用浏览器访问Harbor，正常情况下应能进入登录界面：



默认用户名为admin，密码则为配置文件中定义的密码。登录成功后页面如下：



安装nginx（master）

在两台`worker`节点上安装好Harbor后，接着我们到`master`节点上使用docker搭建一个nginx。拉取nginx的镜像：

```
[root@m1 ~]# docker pull nginx:1.13.12
```

创建一个nginx配置文件，定义一些简单的配置：

```
[root@m1 ~]# mkdir nginx
[root@m1 ~]# cd nginx
[root@m1 nginx]# vim nginx.conf
user nginx;
worker_processes 1;

error_log /var/log/nginx/error.log warn;

pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

stream {
    upstream hub{
        server 192.168.243.139:80;
    }
    server {
        listen 80;
        proxy_pass hub;
        proxy_timeout 300s;
        proxy_connect_timeout 5s;
    }
}
```

> **Tips：**这里只所以只代理其中一个Harbor节点是因为Harbor节点之间的同步存在延迟，而且通常镜像都比较大，所以这个延迟也会比较明显。一般镜像推送完马上就会调度拉取，所以这个延迟时间一般是不可接受的。如果让nginx代理两个节点就会出现一会请求A一会请求B的问题，造成镜像`pull/push`不成功。只代理一个节点也成为了这个方案的缺点，当nginx代理的那个节点宕掉，我们得手动修改nginx的配置代理另一个节点。但由于Harbor是给公司内部的开发人员使用，通常可以允许分钟级别的不可用。

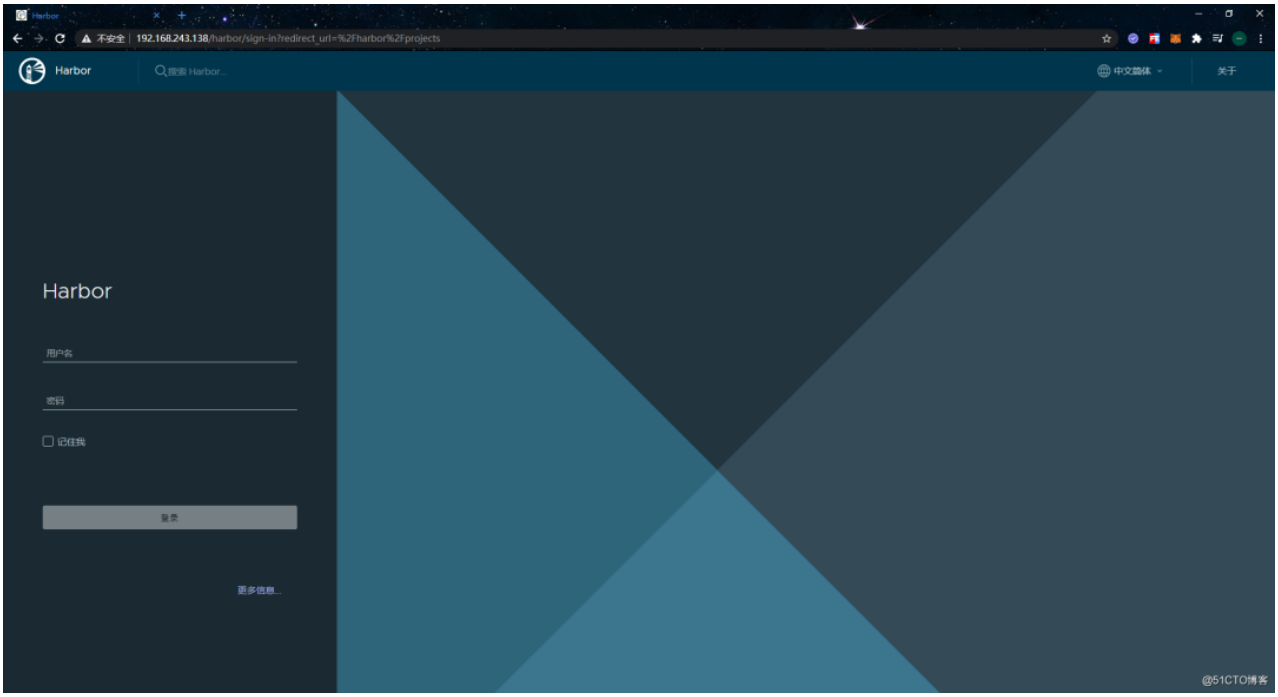
然后为了方便操作，我们写一个简单的启动脚本：

```
[root@m1 nginx]# vim restart.sh
#!/bin/bash

docker stop harbor-nginx
docker rm harbor-nginx
docker run -itd --net=host --name harbor-nginx -v
/root/nginx/nginx.conf:/etc/nginx/nginx.conf nginx:1.13.12
```

执行该脚本：

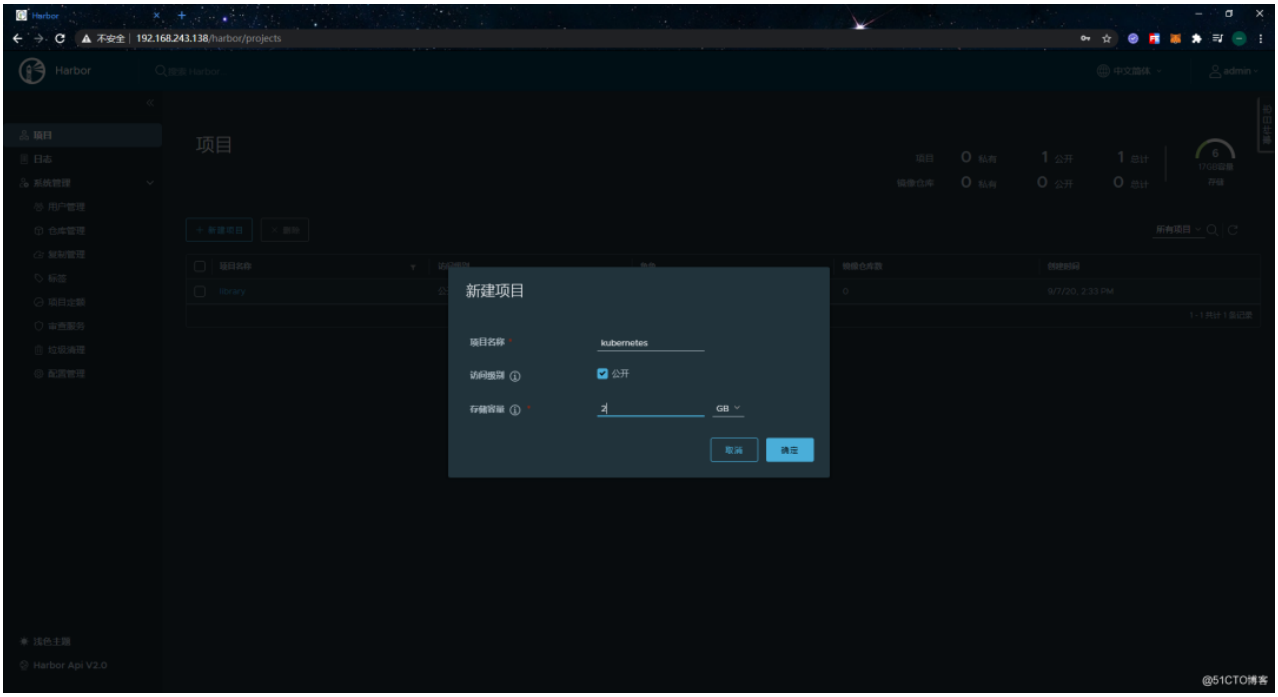
```
[root@m1 ~/nginx]# sh restart.sh
```

使用浏览器访问`master`节点的ip看看能否正常进入到Harbor的登录页：

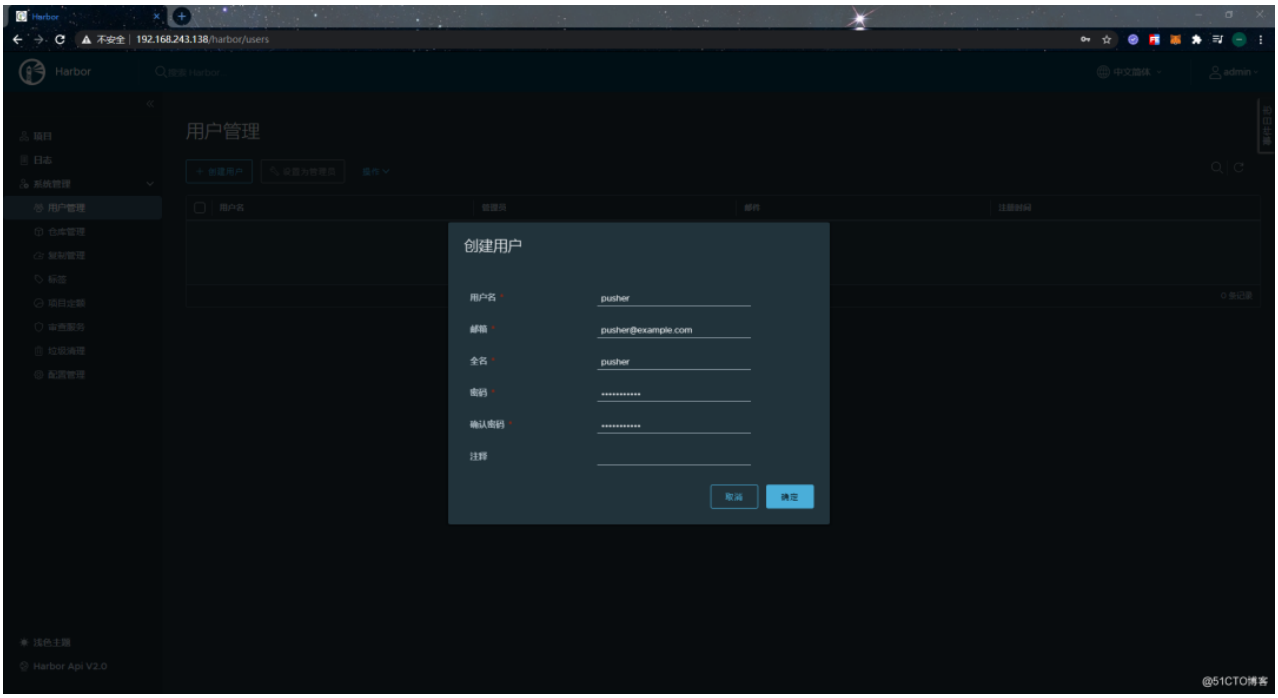


测试Harbor

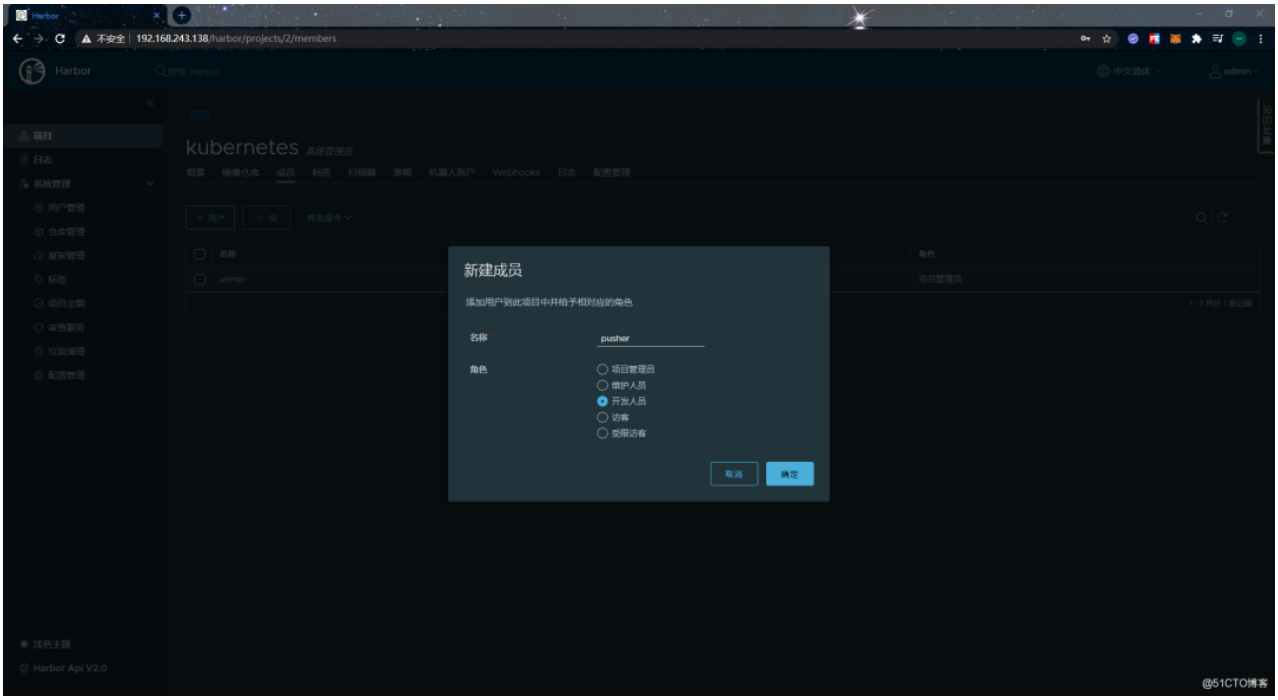
搭建好Harbor之后，我们测试下能否正常使用。将默认的library项目删除掉，然后创建一个新项目：



接着到“用户管理”新建一个用户：



将该用户添加到新建的项目中：



回到命令行上测试一下 `push` 和 `pull` 。由于我们自己搭建的私有仓库默认是不受Docker信任的，所以需要先在配置文件中增加如下配置项让Docker信任该registry：

```
[root@m1 ~]# vim /etc/docker/daemon.json
{
  "insecure-registries": ["192.168.243.138"]
}
[root@m1 ~]# systemctl restart docker
```

登录到我们的Harbor仓库：

```
[root@m1 ~]# docker login 192.168.243.138
Username: pusher
Password:
Login Succeeded
[root@m1 ~]#
```

然后尝试使用命令行`push`一个镜像到Harbor上：

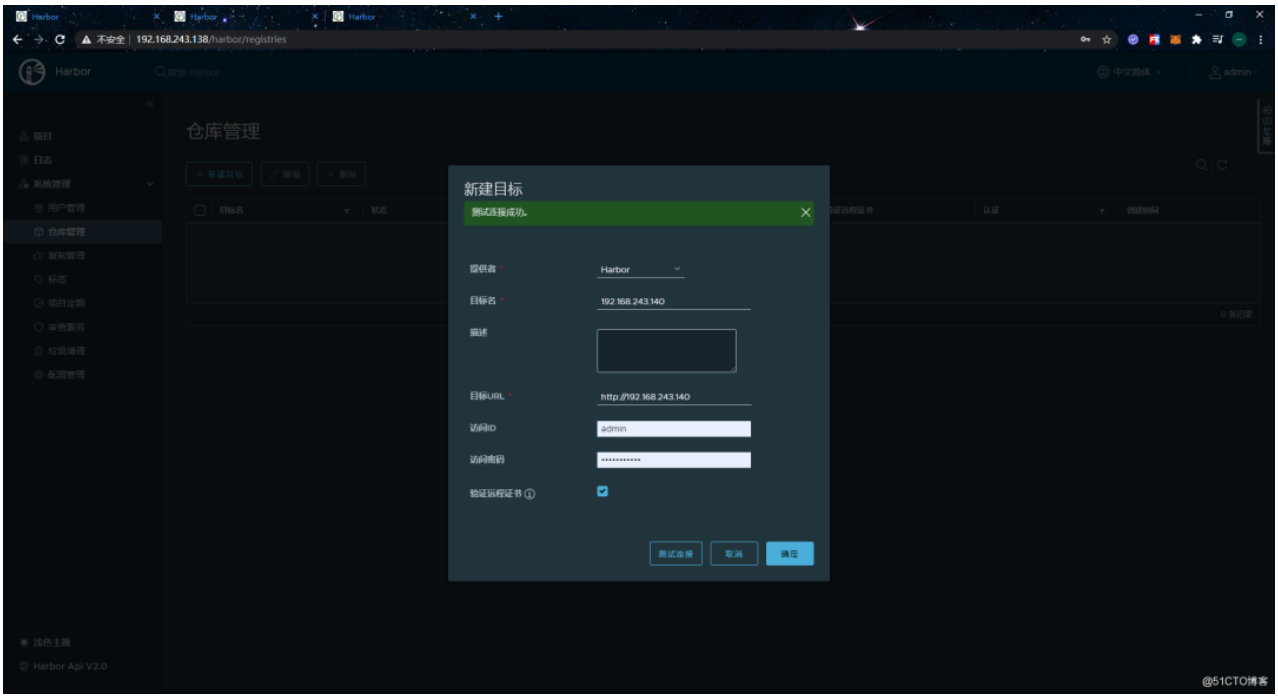
```
[root@m1 ~]# docker tag nginx:1.13.12 192.168.243.138/kubernetes/nginx:1.13.12
[root@m1 ~]# docker push 192.168.243.138/kubernetes/nginx:1.13.12
The push refers to repository [192.168.243.138/kubernetes/nginx]
7ab428981537: Pushed
82b81d779f83: Pushed
d626a8ad97a1: Pushed
1.13.12: digest:
sha256:e4f0474a75c510f40b37b6b7dc2516241ffa8bde5a442bde3d372c9519c84d90 size: 948
[root@m1 ~]#
```

接着测试`pull`，到另一台机器上用同样方式配置`daemon.json`并`docker login`到Harbor，然后使用`docker pull`从Harbor上拉取镜像：

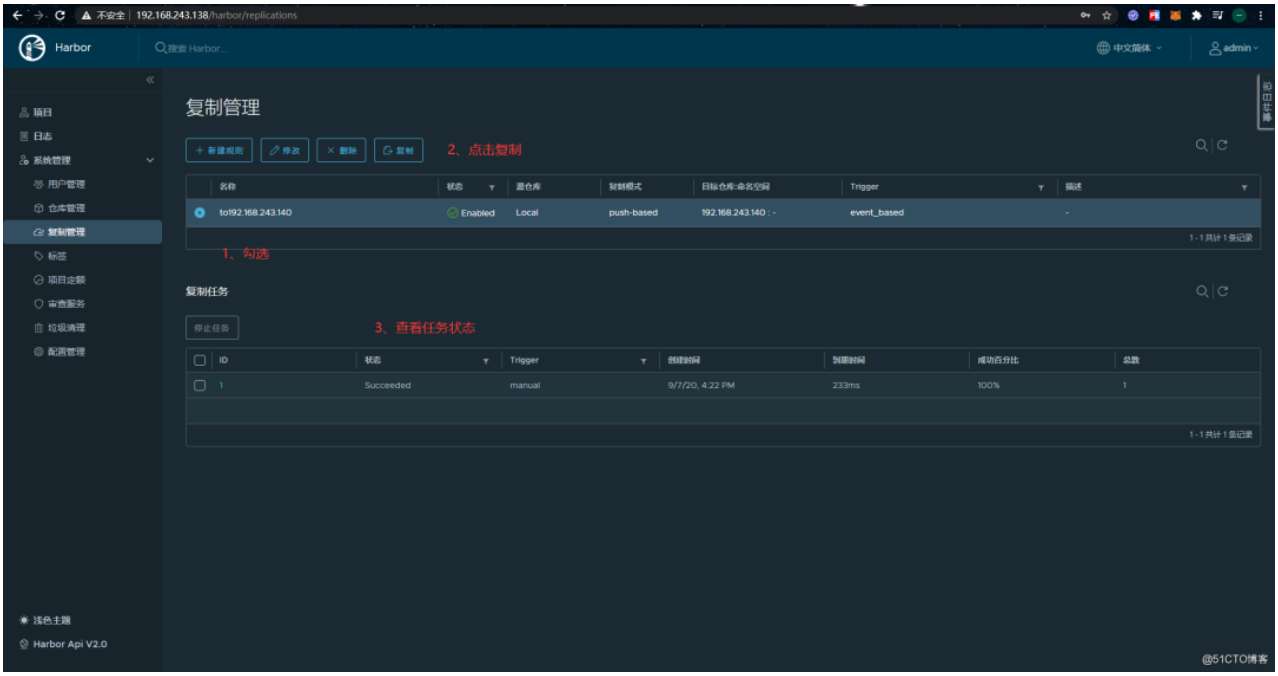
```
[root@s1 ~]# docker pull 192.168.243.138/kubernetes/nginx:1.13.12
1.13.12: Pulling from kubernetes/nginx
f2aa67a397c4: Pull complete
3c091c23e29d: Pull complete
4a99993b8636: Pull complete
Digest: sha256:e4f0474a75c510f40b37b6b7dc2516241ffa8bde5a442bde3d372c9519c84d90
Status: Downloaded newer image for 192.168.243.138/kubernetes/nginx:1.13.12
192.168.243.138/kubernetes/nginx:1.13.12
[root@s1 ~]#
```

配置Harbor节点互相复制

测试完Harbor的基本功能后，我们接下来配置一下Harbor节点之间的互相复制功能，让两个节点能够同步镜像数据。首先到第一个节点上的“仓库管理”界面中新建一个目标，这个目标就是另一台Harbor节点：

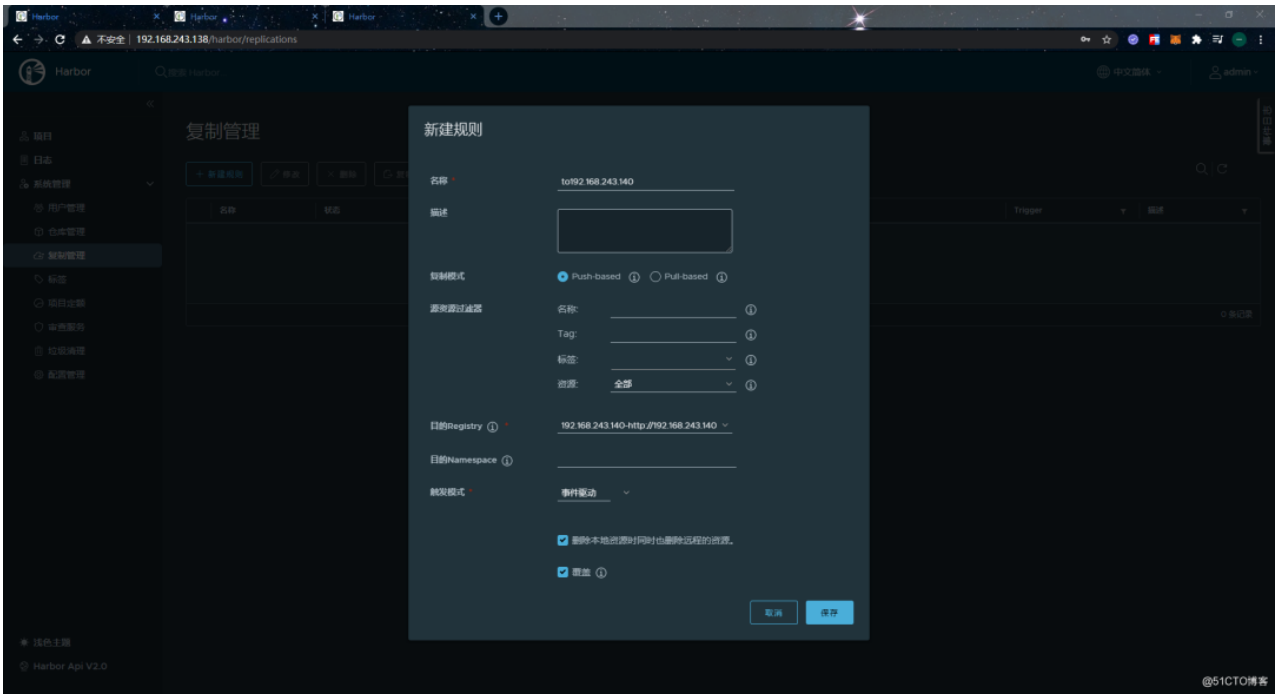


然后到“复制管理”界面中新建复制规则，如下：

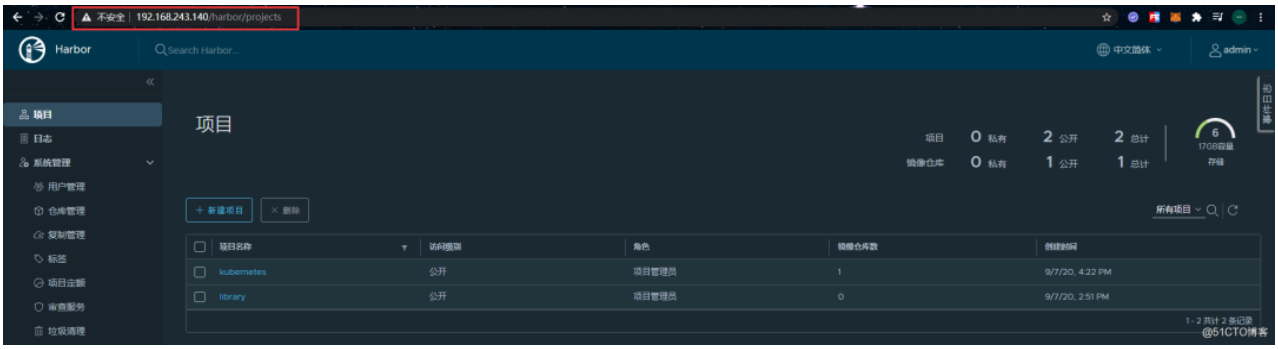


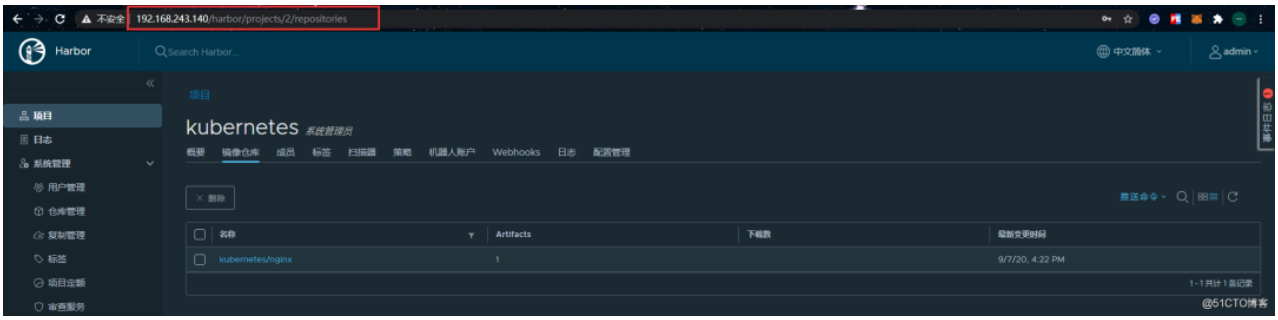
资源过滤器是用于定义只复制哪些镜像的，过滤维度有名称、tag和label。不配置默认复制全部

定义了复制规则后，我们可以在界面上手动触发复制：



复制成功后，到另一个节点上可以看到`kubernetes`这个项目及项目下的镜像都被复制过去了：





同样的，这个节点也需要使用同样的方式配置对另一个节点的复制，由于是一样的步骤，这里就不重复演示了。

- END -

推荐阅读

小团队如何从零搭建一个自动化运维体系？

点亮，服务器三年不宕机