

# 什么是镜像仓库

镜像仓库是一个支持多实例生命周期管理的云原生镜像托管服务, 支持将镜像仓库实例部署至任意云原生基础环境, 同时支持集成外部镜像仓库 (Harbor Registry 、 Docker Registry 和 Jfrog Registry) 。通过镜像仓库服务, 您可以将私有镜像空间分配给一个或多个工作空间 (租户) 使用, 确保私有镜像的安全性, 也可以将镜像空间公开给所有 Kubernetes 命名空间使用, 镜像仓库配合[容器管理](#)服务帮助用户快速部署应用。

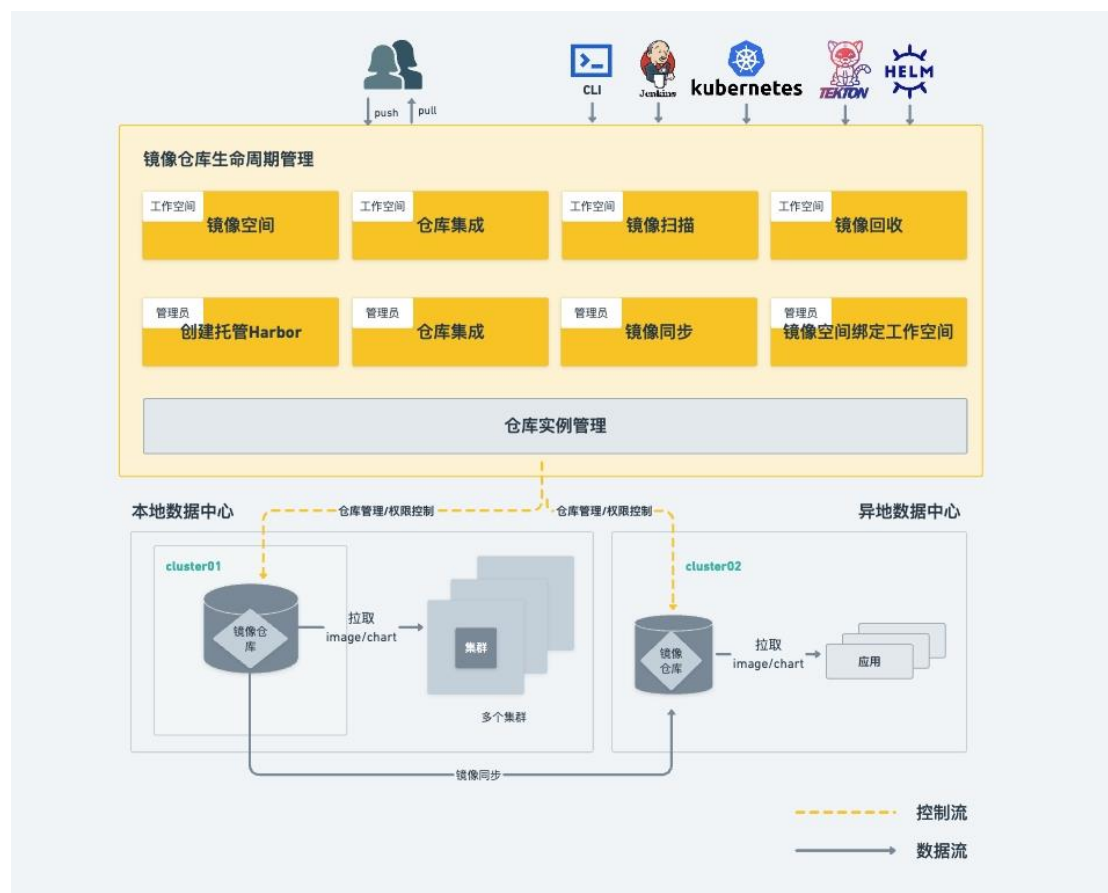
## 功能特性

镜像仓库是 DCE 5.0 商业版所包含的模块, 其功能特性如下:

| 功能特性        | 描述   |
|-------------|--|
| 镜像仓库全生命周期管理 | 通过托管 Harbor 提供镜像仓库的全生命周期管理, 包括镜像仓库的创建、编辑、删除等。              |
| 多维度集成外部仓库   | 支持管理员集成外部仓库后, 将镜像空间分配给一个或多个工作空间 (租户) 使用; 支持工作空间独立集成外部镜像仓库。 |
| 镜像扫描        | 支持镜像扫描功能, 识别   |

| 功能特性    | 描述  |
|---------|---|
|         | 镜像安全风险。   |
| 便捷式应用部署 | 与容器管理模块联动，通过“选择镜像”功能快速选择镜像，完成应用部署。                |
| 镜像同步    | 支持手动/定时两种方式同步镜像，可将镜像同步至目标仓库，或将目标仓库镜像同步至本仓库。       |
| 镜像回收    | 支持设置镜像回收规则来管理镜像仓库中的镜像，通过定时/手动的方式将一些镜像按照一定规则放入回收站。 |

## 产品逻辑架构



逻辑架构图

[下载 DCE 5.0](#) [安装 DCE 5.0](#) [申请社区免费体验](#)

## 镜像仓库 Release Notes

本页列出镜像仓库 Release Notes，便于您了解各版本的演进路径和特性变化。

[Kangaroo]: 镜像仓库的内部开发代号 [Kpanda]: 容器管理的内部开发代号 \*[Harbor]: 一个开源的镜像仓库工具，是 CNCF 毕业项目

## 2024-11-30

### v0.24.0

- **修复** 不同集群使用相同名称的 minio 创建托管 harbor 失败的问题

## 2024-10-31

### v0.23.0

- **新增** 增加 Harbor 仓库开启镜像代理后禁止推送镜像的校验提示
- **新增** Docker Registry 允许自定义镜像空间
- **新增** 删除 Harbor 支持同步删除中间件
- **修复** 创建镜像空间，计数存在延迟问题
- **修复** 托管 Harbor Chartmuseum 没有正确使用 Redis 的问题
- **修复** GetRepository 接口和 ListRepository 接口返回值不一致的问题
- **修复** 删除管理员集成 docker-registry 镜像提示 405 错误的问题
- **修复** 镜像漏洞问题
- **修复** 创建 Harbor Minio 实例会重启问题

## 2024-09-30

### v0.22.0

- **优化** 增加 MinIO 在一个命名空间下只允许创建一个实例的校验
- **优化** 推送镜像增加可推送镜像空间提示
- **优化** 创建托管 Harbor 置灰禁止使用已被使用的中间件

- **修复** 集成或者托管镜像仓库的空间名为纯数字无法删除的问题
- **新增** 支持通过一键创建 Redis/MinIO/PostgreSQL 的方式快速创建托管 Harbor
- **优化** 创建托管 Harbor 时校验中间件实例，不允许二次使用

## 2024-08-30

### v0.21.1

- **新增** 支持通过一键创建 Redis/MinIO/PostgreSQL 的方式快速创建托管 Harbor
- **优化** 创建托管 Harbor 时校验中间件实例，不允许二次使用
- **优化** 适配中间件模块，使通过镜像仓库创建出来的中间件也能被中间件模块纳管
- **优化** 创建托管 Harbor 时，增加集群状态的校验，Harbor 只能创建在运行的集群中
- **修复** 托管 Harbor 编辑删除没有审计日志的问题
- **修复** 创建内置中间件接口拉取镜像报错
- **修复** 开启 OIDC 之后 Admin 不能通过 API 接口操作 Harbor 的问题

## 2024-07-31

### v0.20.0

- **修复** 部分镜像拉取失败导致的在线创建 Harbor 不成功问题
- **修复** Admin 用户权限点与自定义角色权限点数量不一致问题
- **修复** 部分权限溢出问题

## 2024-06-30

### v0.19.0

- **修复** 工作空间无法删除，提示需要解绑镜像空间，实际镜像空间没有该工作空间的绑定关系问题

## 2024-05-31

### v0.18.1

- **优化** 在使用 Harbor 普通管理员集成到镜像仓库后生成的登录指令提示，使提示更准确
- **修复** 应用工作台的镜像选择器接口分页错误

## 2024-04-30

### v0.17.0

- **优化** Docker/Jfrog 类型仓库删除镜像提示优化
- **优化** 镜像下载次数优化
- **优化** 【管理员】视角下 Docker/Jfrog 类型仓库镜像空间列表优化

## 2024-03-29

### v0.16.1

- **新增** 提供通过 LoadBalancer 模式部署 Harbor 的最佳实践文档

- **新增** 支持在 Kpanda Helm 应用更新升级 Kangaroo
- **修复** Docker Registry【镜像删除】删除镜像失败
- **修复** 普通租户（wsadmin 权限）创建镜像回收规则失败
- **修复** 自定义角色托管 Harbor 的创建和删除两个权限点依赖项有误
- **修复** 创建镜像仓库时，如果描述写了中文会报错

## 2023-01-31

### v0.15.0

- **修复** 关联 jfrog 镜像仓库 -> 查询镜像空间报错
- **修复** 生成登录命令失败

## 2023-12-29

### v0.14.0

- **优化** 兼容支持非法的 K8s 用户名
- **修复** 大规模镜像同步时占用较大带宽问题
- **修复** 单个镜像空间中存在大规模的镜像查询延迟较高问题

## 2023-11-30

### v0.13.1

- **修复** 镜像空间绑定工作空间失败问题
- **修复** 生成登录指令时，对所有公开的镜像空间都授权了问题

## 2023-10-31

### v0.12.0

- **新增** 非白金版本创建托管 Harbor 支持快速部署中间件
- **新增** 发布 Harbor Nginx 配置最佳实践
- **新增** 发布镜像仓库资源规划最佳实践
- **优化** 创建托管 Harbor 资源校验优化
- **优化** 安装 Harbor Operator 依赖 cert manager 提示

## 2023-09-06

### v0.11.0

- **新增** 托管 Harbor 状态校验
- **新增** 上线到下载站
- **新增** Nginx 代理最佳实践、资源容量规划最佳实践方案
- **优化** 修复托管 Harbor 描述无法更新问题
- **优化** 点火镜像仓库镜像空间显示不全
- **优化** WS admin 创建回收规则前端报错

## 2023-08-02

### v0.10.0

- **新增** 输出镜像仓库迁移/备份/恢复方案，已通过迁移 release-ci 仓库验证
- **新增** 输出非安全镜像仓库登录最佳实践方案



- **新增** 创建托管 Harbor 支持使用内部中间件 MINIO
- **新增** 适配人大金仓的 PG 模式
- **优化** 创建托管 Harbor 时对 pg、redis 地址进行格式校验
- **优化** 无权限置灰 + 提示优化
- **优化** 集群解绑后的特殊情况处理

## 2023-07-02

### v0.9.0

- **新增** 创建托管 Harbor 支持选择中间件 minio 实例
- **新增** 支持 Ghippo 重点审计功能
- **优化** 删除工作空间（WS）时校验该 WS 下的镜像仓库资源
- **优化** 处理解绑集群后，托管 Harbor 的异常状态
- **修复** Ghippo workspace 资源无法解绑的问题
- **修复** Sidecar 注入问题
- **修复** 创建 Harbor 选择 minio 时，无法推送镜像的问题

## 2023-06-05

### v0.8.0

### 新功能

- **新增** Harbor 类型仓库增加镜像描述信息
- **新增** 镜像空间列表增加实例状态

- **新增** 创建 Harbor 支持随机生成 nodeport 端口号
- **新增** 创建 Harbor 校验 redis、postgresql、s3 是否重复使用
- **新增** project 支持镜像回收功能

## 修复

- **修复** 离线部署托管 Harbor 镜像地址拼接错误
- **修复** Docker Registry 类型 **project** 列表分页问题
- **修复** Docker Registry 类型 repository 搜索问题
- **修复** 机器人账号创建时已存在问题

## 优化

- **优化** 添加 job 清除逻辑
- **优化** 去除获取临时登录指令接口中 project 字段
- **优化** 登录指令
- **优化** 镜像扫描判断逻辑

## 2023-05-08

### v0.7.0

- **新增** 支持页面配置多实例之间的镜像同步
- **新增** 在工作空间中支持关联 jfrog 类型仓库
- **新增** 支持在离线 ARM 环境下创建托管 Harbor
- **修复** 仓库集成 URL 修改后不能被更新的问题

- **修复** Project 从原生 Harbor 页面删除后逻辑不兼容情况
- **优化** 对接 jfrog 接口性能到 2s 内

## 2023-04-07

### v0.6.2

- **新增** 支持全局管理中的自定义角色，对用户授予不同的角色权限
- **新增** 支持生成临时登录指令。
- **新增** 创建托管 Harbor 支持 cpu、memory 校验
- **新增** 创建托管 Harbor 支持编辑镜像扫描器
- **新增** 创建托管 Harbor 支持使用 S3 存储
- **修复** harbor-operator helm 安装字节过大的问题
- **优化** 镜像列表功能页面性能，单独提出 Project 列表页面

## 2023-03-15

### v0.5.0

### 新功能

- **新增** 支持创建托管 Harbor 支持 NodePort 方式暴露，并校验端口是否被占用
- **新增** 支持创建托管 Harbor 支持 https 方式暴露
- **新增** 支持创建托管 Harbor 支持开启 DCE 5.0 ODIC 接入，实现使用 DCE 5.0 用户登录 Harbor
- **新增** 支持创建托管 Harbor 支持校验部署的集群是否安装 harbor-operator
- **新增** 支持创建托管 Harbor 支持使用中间件模块部署的 redis 实例

- **新增** 支持创建托管 Harbor 支持修改 Admin 密码、资源配额、Redis 实例、访问类型
- **新增** 支持创建托管 Harbor 支持自动创建镜像扫描器
- **新增** 支持仓库集成，关联仓库校验用户密码以及用户权限
  - **新增** 支持仓库集成校验用户名密码正确，并且保证用户具备管理员权限
  - **新增** 支持关联仓库校验用户名密码正确
- **新增** 支持仓库集成列表新增模糊查询
- **新增** 支持页面编辑 Project 为公开或者私有
- **新增** 支持 Harbor、Docker Registry 类型的仓库多架构镜像页面显示

## 修复

- **修复** 支持关联仓库接入 https 的 Harbor、Docker Registry 仓库

## 2022-12-30

### v0.4.0

- **新增** 支持基于 Harbor 创建托管镜像仓库
- **新增** 支持多副本部署
- **新增** 支持对镜像仓库的全生命周期管理
- **新增** 支持将托管 harbor 实例部署在平台的任意集群下的任意命名空间
- **新增** 支持平台接入和管理外部 Harbor、Docker Registry 镜像仓库
- **新增** 支持为平台工作空间（租户）单独分配私有镜像空间
- **新增** 支持创建公开/私有镜像空间
- **新增** 支持工作空间（租户）独立接入外部 Harbor、Docker Registry 镜像仓库

- **新增** 支持部署应用时通过镜像选择器选择镜像
- **新增** 支持镜像扫描

## 镜像仓库 FAQ

本页列出使用镜像仓库时常见的一些问题和解决办法。

### DCE 5.0 标准版为什么不能使用中间件部署镜像仓库

部署镜像仓库时会用到一些中间件，但 DCE 5.0 标准版中没有中间件，中间件属于白金版。

### 部署镜像仓库时如何校验配置的中间件网络是否可连接

登录部署 Harbor 的目标集群，在任意节点中执行 ping 命令，测试是否能连接到中间件组件。

### 镜像空间列表看不到私有镜像

镜像仓库 v0.7.0-v0.7.3 和 v0.8.0 存在一个 bug，会导致看不到私有镜像，请升级到更高版本的镜像仓库。

### 在使用中间件 MinIO 部署镜像仓库时报错

在使用中间件 MinIO 部署镜像仓库时，需要先手动通过 MinIO 管理平台创建好 bucket。

### 仓库集成支持的 Harbor 最低版本

在仓库集成时因使用了 Harbor 的功能，对版本有一定要求，目前支持的已知最低版本为

v2.4.0。更早的旧版本将不可用。

## 离线环境镜像扫描器失败

镜像扫描因为依赖漏洞数据，默认是去 [CVE 官网](#) 获取漏洞数据。如果是一个纯离线环境，则不能正常进行漏洞扫描，会执行失败。

trivy

trivy

## 创建托管 Harbor 时第一步集群校验通过后创建 Harbor 仍然出错

目前只校验了集群中是否有 CRD，没有校验 harbor-operator 服务，可能会出现不存在 harbor-operator 服务的情况，导致不能正确地创建 Harbor。

## 本地执行 docker login {ip} 之后报错

执行 docker login {ip} 后出现以下报错：

```
Error response from daemon: Get "https://{ip}/v2/": x509: cannot validate certificate for {ip} because it doesn't contain any IP SANs
```

出现这个错误是因为 registry 是 https 服务，是使用了非签名证书或者不安全证书，就会

提示这个错误。解决办法是在 /etc/docker/daemon.json 配置文件中 "insecure-registries"

加入对应的 IP。

```
"insecure-registries": [  
  "{ip}",  
  "registry-1.docker.io"  
]
```

之后重启 systemctl restart docker。

## 创建托管 Harbor 接入外部 PostgreSQL、Redis，密码含有特殊字符 (!@#\$%^&\*)，导致服务启动失败

目前密码中不能有特殊字符，不然会出现服务启动失败的情况，可以使用大小写字母和数字组合的情况。

## harbor-operator 安装不成功

harbor-operator 安装不成功需要检查这几项，cert-manager 是否安装成功，installCRDs 是否设置为 true。安装 harbor-operator 的 helm 任务是否成功。

## 创建托管 Harbor 可以使用 redis cluster 模式吗

目前 Harbor 仍然不能使用 redis cluster 模式。

## 私有镜像在非镜像仓库模块能看到吗？

镜像仓库是严格按照 DCE 5.0 的权限来执行的，在镜像仓库中某个用户必须要属于某个租户/工作空间，才能看到当前租户下的私有镜像空间，否则即使管理员也看不到。

## 私有镜像绑定到工作空间后查询不到私有镜像

私有镜像绑定工作空间后程序需要异步执行很多逻辑，所以不会马上看到。这个过程会受到系统的影响，如果系统响应较快，则异步执行较快，通常 1 分钟内就能看到。最长应该不会超过 5 分钟。

## 托管 Harbor 创建后能访问了但是状态依然不健康

目前托管 Harbor 页面上的状态和仓库集成的状态是二合一的，当两个状态都为健康的时候才是健康，因此可能出现托管 Harbor 已经可以访问了，但是状态依然不健康，这种情况需要等一个服务探测周期，一个探测周期是 10 分钟，在一个周期后就会恢复如初。

## 创建的托管仓库状态为不健康

### 仓库不健康

#### 仓库不健康

- A1：用户输入的数据库、Redis、S3 存储等信息有误，导致无法连接，可通过查看日志文件进行排查。现象主要是几个核心服务有 Pod 启动失败，可以通过查看日志进一步确认原因。

```
kubecttl -n kangaroo-lrf04 get pods
```

| NAME  |              | READY   | STAT |
|---|--------------|---------|------|
| US  | RESTARTS AGE |         |      |
| trust-node-port-harbor-harbor-chartmuseum-57fdb9cdc-qznwc | 1/1 20h      | Running | 0    |
| trust-node-port-harbor-harbor-core-855f8df46c-cgqb9       | 1/1 20h      | Running | 0    |
| trust-node-port-harbor-harbor-jobservice-6b958dbc57-ks997 | 1/1 20h      | Running | 0    |
| trust-node-port-harbor-harbor-portal-5cf6bf659b-kj6gd     | 1/1 20h      | Running | 0    |
| trust-node-port-harbor-harbor-registry-5ccbf457c5-qrtx5   | 2/2 20h      | Running | 0    |
| trust-node-port-harbor-harbor-trivy-dbd8945-xh6rv         | 1/1 20h      | Running | 0    |
| trust-node-port-nginx-deployment-677c74576-7kmh4          | 1/1 20h      | Running | 0    |

- A2：如果通过 [A1](#) 排查后无误，继续排查 harborcluster 资源是否健康，如下命令查看 harborcluster 资源状态。

```
kubecttl -n kangaroo-lrf04 get harborclusters.goharbor.io
```



| NAME            | PUBLIC URL               | STATUS  |
|-----------------|--------------------------|---------|
| trust-node-port | https://10.6.232.5:30010 | healthy |

- A3：如果 [A2](#) 排查后无误，继续在 `kpanda-global-cluster` 集群上排查 `registrysecrets.kangaroo.io` 资源是否创建，以及 status 情况。

提示: namespace 默认为 `kangaroo-system`。

```
kubectl -n kangaroo-system get registrysecrets.kangaroo.io
```

| NAME             | AGE |
|------------------|-----|
| inte-bz-harbor-1 | 34d |

```
kubectl -n kangaroo-system describe registrysecrets.kangaroo.io inte-bz-harbor-1
```

!!! tip

- 上述 [A1](#a1)、[A2](#a2) 都在托管 Harbor 所在的集群上排查问题，在目标集群上的查看路径为：\_\_仓库实例\_\_ -> \_\_概览\_\_ -> \_\_部署位置\_\_
- 上述 [A3](#a3) 在 ``kpanda-global-cluster`` 集群上验证。

## 创建 Project 或上传镜像后发现页面上的镜像空间和可用存储未增加

这是因为 UI 页面上在 **托管 Harbor** 首页、仓库集成详情中的统计信息是异步获取的数据，会有一定的延迟，最长延迟为 10 分钟。

## 仓库集成后但状态为不健康

### 仓库集成不健康

仓库集成不健康

首先确认实例是否真的健康，如果实例不健康，则需要排查实例的问题；如果实例健康，则通过在 `kpanda-global-cluster` 集群上排查 `registrysecrets.kangaroo.io` 资源是否创建，并排查 status 情况，这样可以初步确认问题所在。

提示: namespace 默认为 `kangaroo-system`。

```
kubectl -n kangaroo-system get registrysecrets.kangaroo.io
```

| NAME           | AGE |
|----------------|-----|
| trust-test-xjw | 34d |

```
kubectl -n kangaroo-system get registrysecrets.kangaroo.io trust-test-xjw -o yaml
apiVersion: kangaroo.io/v1alpha1
kind: RegistrySecret
metadata:
  name: trust-test-xjw
  namespace: kangaroo-system
spec:
  ....
status:
  state:
    lastTransitionTime: "2023-03-29T03:27:31Z"
    message: 'Get "https://harbor.kangaroo.daocloud.io": dial tcp: lookup harbor.kangaroo.daocloud.io
on 10.233.0.3:53: no such host'
    reason: RegistryHealthCheckFail
    status: "False"
    type: HealthCheckFail
```

## 仓库集成后，在镜像列表页面实例中不可查看

请确认仓库集成的资源是否健康，如果不健康是不会在镜像列表页面的实例列表中显示的。

确认方式请参考[仓库集成后不健康的确认方法](#)。

## 在 Kpanda 镜像选择器中选一个私有 Project 镜像但部署时提示镜像拉取失败

- A1: 能在镜像选择器中看到私有 Project 表明 Project 和 Workspace 已经进行了绑定，

此时需要去镜像部署的目标集群 namespace 中确认是否生成名为 registry-secret 的 secret 。

```
kubectl -n default get secret registry-secret
```

| NAME            | TYPE                           | DATA | AGE |
|-----------------|--------------------------------|------|-----|
| registry-secret | kubernetes.io/dockerconfigjson | 1    | 78d |

- A2: 如果确认已经生成名为 registry-secret 的 secret ，则需要确认 secret 中的 **dockerconfigjson** 是否正确。

```
kubectl get secret registry-secret -o jsonpath='{.data.*}' | base64 -d | jq
```

```
{
  "auths": {
    "127.0.0.1:5000": {
      "auth": "YWRtaW46SGFyYm9yMTIzNDU="
    }
  }
}

echo "YWRtaW46SGFyYm9yMTIzNDU=" | base64 -d
admin:Harbor12345
```

## 离线升级镜像仓库模块

本页说明[下载镜像仓库模块](#)后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `__kangaroo__` 字样是镜像仓库模块的内部开发代号。

## 从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 `chart-syncer`

同步镜像到镜像仓库，该方法更加高效便捷。

## chart-syncer 同步镜像到镜像仓库

### 1. 创建 load-image.yaml

!!! note

该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

=== “已安装 chart repo”

若当前环境已安装 `chart repo`，且 `chart-syncer` 也支持将 `chart` 导出为 `tgz` 文件。

```
```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: kangaroo-offline # (1)!
target:
  containerRegistry: 10.16.10.111 # (2)!
  containerRepository: release.daocloud.io/kangaroo # (3)!
repo:
```

```

kind: HARBOR # (4)!
url: http://10.16.10.111/chartrepo/release.daocloud.io # (5)!
auth:
  username: "admin" # (6)!
  password: "Harbor12345" # (7)!
containers:
  auth:
    username: "admin" # (8)!
    password: "Harbor12345" # (9)!
...

```

1. 到执行 `charts-syncer` 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径

2. 需更改为你的镜像仓库 url
3. 需更改为你的镜像仓库
4. 也可以是任何其他支持的 Helm Chart 仓库类别
5. 需更改为 chart repo url
6. 你的镜像仓库用户名
7. 你的镜像仓库密码
8. 你的镜像仓库用户名
9. 你的镜像仓库密码

### === “未安装 chart repo”

若当前环境未安装 chart repo，且 `chart-syncer` 也支持将 chart 导出为 `tgz` 文件，并存放在指定路径。

```

```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: kangaroo-offline # (1)!
target:
  containerRegistry: 10.16.10.111 # (2)!
  containerRepository: release.daocloud.io/kangaroo # (3)!
repo:
  kind: LOCAL
  path: ./local-repo # (4)!
containers:
  auth:
    username: "admin" # (5)!
    password: "Harbor12345" # (6)!
...

```

1. 到执行 `charts-syncer` 命令的相对路径，而不是此 YAML 文件和离线包之间的相对路径

2. 需更改为你的镜像仓库 url

3. 需更改为你的镜像仓库
4. chart 本地路径
5. 你的镜像仓库用户名
6. 你的镜像仓库密码

## 2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

## Docker 或 containerd 直接加载

解压并加载镜像文件。

### 1. 解压 tar 压缩包。

```
tar xvf kangaroo.bundle.tar
```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

### 2. 从本地加载镜像到 Docker 或 containerd。

=== “Docker”

```
```shell
docker load -i images.tar
```
```

=== “containerd”

```
```shell
ctr -n k8s.io image import images.tar
```
```

!!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作，  
加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

## 升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

=== “通过 helm repo 升级”

### 1. 检查镜像仓库 helm 仓库是否存在。

```
```shell
```

```
helm repo list | grep kangaroo
```

```

若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

```
```none
Error: no repositories to show
```
```

1. 添加镜像仓库的 helm 仓库。

```
```shell
helm repo add kangaroo http://{harbor url}/chartrepo/{project}
```
```

1. 更新镜像仓库的 helm 仓库。

```
```shell
helm repo update kangaroo # (1)!
```
```

1. helm 版本过低会导致失败，若失败，请尝试执行 `helm update repo`

1. 选择您想安装的镜像仓库版本（建议安装最新版本）。

```
```shell
helm search repo kangaroo/kangaroo --versions
```
```

```
```none
[root@master ~]# helm search repo kangaroo/kangaroo --versions
NAME                                CHART VERSION  APP VERSION  DESCRIPTION
kangaroo/kangaroo 0.9.0                v0.9.0       A Helm chart for Kangaroo
...
```
```

1. 备份 `--set` 参数。

在升级镜像仓库版本之前，建议您执行如下命令，备份老版本的 `__--set__` 参数。

```
```shell
helm get values kangaroo -n kangaroo-system -o yaml > bak.yaml
```
```

1. 查看版本更新记录，如果 CRD 有更新，更新 kangaroo crds

```
```shell
helm pull kangaroo/kangaroo --version 0.9.0 && tar -zxf kangaroo-0.9.0.tgz
kubectl apply -f kangaroo/crds
```
```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 \_\_global.imageRegistry\_\_ 字段为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade kangaroo kangaroo/kangaroo \
  -n kangaroo-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry
  --version 0.9.0
```
```

### === “通过 chart 包升级”

1. 备份 `--set` 参数。

在升级镜像仓库版本之前，建议您执行如下命令，备份老版本的 \_\_--set\_\_ 参数。

```
```shell
helm get values kangaroo -n kangaroo-system -o yaml > bak.yaml
```
```

1. 查看版本更新记录，如果 CRD 有更新，更新 kangaroo crds。

```
```shell
kubectl apply -f ./crds
```
```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 \_\_global.imageRegistry\_\_ 为当前使用的镜像仓库地址。

```
```shell
```

```
export imageRegistry={你的镜像仓库}
...

```shell
helm upgrade kangaroo . \
  -n kangaroo-system \
  -f ./bak.yaml \
  --set global.imageRegistry=$imageRegistry
...

```

## 管理外部镜像仓库

如果您有一个或多个 Harbor、Docker、Jfrog Artifactory 镜像仓库，完全可以用 DCE 5.0 镜像仓库进行统一管理。视操作员的角色权限，可以使用两种方式：

- 仓库集成（工作空间）：支持 Harbor、Docker、Jfrog Artifactory 三种镜像仓库类型
- 仓库集成（管理员）：支持 Harbor、Docker 两种镜像仓库类型

### 仓库集成（工作空间）

如果是 Workspace Admin，可以通过仓库集成（工作空间）功能，将现有镜像仓库关联到 DCE 5.0 平台，供工作空间的成员使用。简单的操作步骤如下：

1. 以 Workspace Admin 角色登录，从左侧导航栏点击 **仓库集成(工作空间)**，点击右上角的 **仓库集成** 按钮。

仓库集成(工作空间)

仓库集成(工作空间)

2. 填写表单信息后点击 **确定**。

填写表单

填写表单

!!! note



1. Docker Registry 镜像仓库若未设置密码可不填写，Harbor 镜像仓库必须填写用户名/密码。
2. 有关实际操作演示，请参阅[仓库集成视频演示](../../videos/kangaroo.md)。

## 仓库集成(管理员)

如果是 Admin（平台管理员），还可以通过仓库集成功能，将现有镜像仓库集成到 DCE 平台。仓库集成是集中管理平台镜像仓库的入口，对于 Harbor 镜像仓库平台管理员可以通过将镜像空间与工作空间绑定的方式，将某个私有镜像空间分配给一个或者多个工作空间（工作空间下的命名空间）使用。或者将镜像空间设置为公开，供平台所有命名空间使用。

1. 以 Admin 角色登录，在左侧导航栏点击 **仓库集成(管理员)**。

仓库集成

仓库集成

2. 点击右上角的 **仓库集成** 按钮。

仓库集成

仓库集成

3. 选择仓库类型，填写集成名称、仓库地址、用户名和密码后点击 **确定**。

填写表单

填写表单

4. 在集成的仓库列表中，光标悬浮于某个仓库上，点击眼睛图标可查看概览。

查看概览

查看概览

5. 概览页面显示了当前仓库的基本信息、统计信息，还在顶部提供了快速入门，方便管理镜像空间、工作空间、创建应用。

查看概览

## 查看概览

!!! note

Docker Registry 镜像仓库不支持镜像空间与工作空间绑定的能力，若 Docker Registry 未设置密码，

则相当于该 Docker Registry 是公开的，平台所有命名空间创建应用时均能拉取其中的全部镜像。

# 推送镜像

创建托管 Harbor 和镜像空间后，您可以按照如下说明登录并将镜像推送的镜像仓库中；或者登录原生 Harbor，查看原生 Harbor 在每个镜像空间（项目）下提供的引导。

## 推送方式一

前提：已创建托管 Harbor 和镜像空间

您可在本地构建新的容器镜像或从 DockerHub 上获取一个公开镜像用于测试。本文以 DockerHub 官方的 Nginx 最新镜像为例，在命令行工具中依次执行以下指令，即可推送该镜像。请将 library 及 nginx 依次替换为您实际创建的镜像空间名称及镜像仓库名。

### 1. 登录镜像仓库

```
docker login --username=<镜像仓库登录名> <镜像仓库地址>
```

示例：

```
docker login --username=admin http://test.lrf02.kangaroo.com
```

在返回结果中输入镜像仓库密码（创建托管 Harbor 时设置的密码）。

### 2. 给镜像加标签

执行以下命令，给镜像打标签。

```
docker tag <镜像仓库名称>:<镜像版本号> <镜像仓库地址>/<镜像空间名称>/<镜像仓库名称>:<镜像版本号>
```

示例：

```
docker tag nginx:latest test.lrf02.kangaroo.com/library/nginx:latest
```

### 3. 推送镜像

执行以下命令，推送镜像至镜像空间 library 中。

```
docker push <镜像仓库地址>/<镜像空间名称>/<镜像仓库名称>:<镜像版本号>
```

示例：

```
docker push test.lrf02.kangaroo.com/library/nginx:latest
```

### 4. 拉取镜像

执行以下命令，拉取镜像。

```
docker pull <镜像仓库地址>/<镜像空间名称>/<镜像仓库名称>:<镜像版本号>
```

示例：

```
docker pull test.lrf02.kangaroo.com/library/nginx:latest
```

## 推送方式二

前提：已创建托管 Harbor 和镜像空间。

1. 在托管 Harbor 列表页面中，点击目标镜像仓库右侧的 ...，点击 **原生 Harbor**，进入

原生 Harbor 的登录页。

原生 Harbor

原生 Harbor

2. 输入创建托管 Harbor 时设置的用户名和密码进入原生 Harbor。

输入用户名和密码

输入用户名和密码

3. 点击目标镜像空间（项目）的名称，进入镜像空间。

镜像空间

镜像空间

4. 点击右侧的推送命令，查看原生 Harbor 提供的推送命令。

## 查看推送命令

### 查看推送命令

!!! tip

相对于[方式一](#\_2)，原生 Harbor 的推送命令自动填入了镜像仓库地址和镜像空间名称。

# Admin 快速使用镜像仓库为平台服务

在 DCE 5.0 中有托管 Harbor (在 DCE 上自建的 Harbor) 和仓库集成 (集成外部的 Harbor 或 Docker Registry) 两种镜像仓库管理方式：

- [托管 Harbor](#) (建议)
- [仓库集成](#)

DCE 5.0 主推 Harbor 作为镜像仓库提供镜像服务。

## 共享公开镜像

假设您已创建了托管 Harbor 或接入了一个外部 Harbor，按照以下步骤可以将公开镜像共享

给所有命名空间使用（在部署应用时能够通过镜像选择器选择公开的镜像）：

graph TB

create[创建托管 Harbor] --> setpublic[创建镜像空间并设置为公开]

- -> push[推送镜像]

- -> deploy[部署应用]

```
classDef plain fill: #ddd,stroke:#fff,stroke-width:1px,color:#000;
```

```
classDef k8s fill: #326ce5,stroke:#fff,stroke-width:1px,color:#fff;
```

```
classDef cluster fill: #fff,stroke:#bbb,stroke-width:1px,color:#326ce5;
```

```
class create,setpublic,push,deploy cluster;
```

```
click create "https://docs.daocloud.io/kangaroo/hosted/"
```

```
click setpublic "https://docs.daocloud.io/kangaroo/integrate/create-space/"
```

```
click push "https://docs.daocloud.io/kangaroo/quickstart/push/"
```

```
click deploy "https://docs.daocloud.io/kpanda/user-guide/workloads/create-deployment/"
```

预期结果：平台上所有用户在命名空间中部署应用时，均能够通过镜像选择器，选择公开镜像空间中的镜像进行部署。

选择镜像

选择镜像

镜像选择

镜像选择

## 共享私有镜像

假设您已创建了托管 Harbor 或接入了一个外部 Harbor，按照以下步骤可以将私有镜像共享给指定的工作空间（租户）下的命名空间使用（在部署应用时能够通过镜像选择器选择私有的镜像）：

前提条件为：

graph TB

create[已创建工作空间] --> bind[命名空间已绑定工作空间]

```
classDef plain fill: #ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill: #326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill: #fff,stroke:#bbb,stroke-width:1px,color:#326ce5;
```

```
class create,bind cluster;
```

```
click create "https://docs.daocloud.io/hippo/user-guide/workspace/Workspaces/"
click bind "https://docs.daocloud.io/hippo/user-guide/workspace/quota/#_4"
```

操作步骤为：

graph TB

```
create[创建托管 Harbor] --> setpublic[创建镜像空间并设置为公开]
- -> push[推送镜像]
- -> bind[镜像空间绑定工作空间]
- -> deploy[部署应用]
```

```
classDef plain fill: #ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill: #326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill: #fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

class create,setpublic,push,bind,deploy cluster;

click create "https://docs.daocloud.io/kangaroo/hosted/"
click setpublic "https://docs.daocloud.io/kangaroo/integrate/create-space/"
click push "https://docs.daocloud.io/kangaroo/quickstart/push/"
click bind "https://docs.daocloud.io/kangaroo/integrate/integrate-admin/bind-to-ws/"
click deploy "https://docs.daocloud.io/kpanda/user-guide/workloads/create-deployment/"
```

预期结果：仅在该工作空间下的命名空间部署应用时能够通过镜像选择器，选择该镜像空间下的私有镜像进行部署应用。

选择镜像

选择镜像

镜像选择

镜像选择

!!! tip

1. 接入的 Harbor 可按照上述方式达到同样的使用效果。
1. Docker Registry 本身只有公开镜像，因此接入后镜像将公开给所有命名空间使用。

## Workspace Admin 快速使用镜像仓库为工作空间服务

在 DCE 5.0 中，Workspace Admin 能够通过仓库集成(工作空间)方式集成 Harbor 和 Docker Registry 两种外部镜像仓库。集成后工作空间成员能够在镜像列表中看到仓库集成(工作空间)的所有镜像，以及在工作空间下的命名空间中部署应用时能够通过镜像选择器选择该仓库中的镜像进行部署。

假设您已经创建了一个外部 Harbor 或者 Docker Registry 镜像仓库，按照以下步骤可以将

外部 Harbor 或者 Docker Registry 共享给工作空间成员使用。

前提条件为：

1. 您是 Workspace Admin，并且该工作空间下绑定了一些命名空间
2. 您拥有一个或者多个外部镜像仓库（Harbor 或 Docker Registry）

操作步骤为：

graph TB

associate[仓库集成] --> push[推送镜像] --> deploy[部署应用]

```
classDef plain fill: #ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill: #326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill: #fff,stroke:#bbb,stroke-width:1px,color:#326ce5;
```

```
class associate,push,deploy cluster;
```

```
click associate "https://docs.daocloud.io/kangaroo/related-registry/"
click push "https://docs.daocloud.io/kangaroo/quickstart/push/"
click deploy "https://docs.daocloud.io/kpanda/user-guide/workloads/create-deployment/"
```

预期结果：在该工作空间下的命名空间部署应用时能够通过镜像选择器，选择该镜像空间下的镜像进行部署应用。

按钮

按钮

镜像选择

镜像选择

## 离线环境开启镜像安全扫描

Trivy 会使用 admin 账号来拉取要扫描的镜像，在使用前需要确认可以使用 admin 进行 docker login。

## 下载离线镜像包

trivy 有两个版本，目前都需要下载版本 2，在 <https://github.com/aquasecurity/trivy-db> 项目中已经不提供版本 2 直接下载离线的 **trivy-db** 包了，被打包成了 oci [包](#)。

如下使用 oras 工具来下载，先安装 oras，注意区分操作系统版本，如下是下载 linux 平台的：

```
export VERSION="1.0.0"
$ curl -LO "https://github.com/oras-project/oras/releases/download/v${VERSION}/oras_${VERSION}_linux_amd64.tar.gz"
$ mkdir -p oras-install/
$ tar -zxf oras_${VERSION}_*.tar.gz -C oras-install/
$ sudo mv oras-install/oras /usr/local/bin/
$ rm -rf oras_${VERSION}_*.tar.gz oras-install/
```

使用 oras 工具下载 trivy-db：

```
$ oras pull ghcr.io/aquasecurity/trivy-db: 2
db.tar.gz
$ tar -zxf db.tar.gz
# 解压出来有两个文件
db/metadata.json
db/trivy.db
```

## 开启托管 Harbor 离线扫描

### 通过命令修改 trivy

运行以下命令为托管 Harbor 的 Kubernetes 集群修改 YAML 参数：

```
$ kubectl -n {namespace} edit harborclusters.goharbor.io {harbor-name}
# 主要修改 trivy offlineScan 和 skipUpdate 改为 true
trivy:
  offlineScan: true
  skipUpdate: true
```

### 在 UI 界面修改 YAML

edit-harborcluster.png



edit-harborcluster.png

1. 从 **集群列表** 进入某个集群，选择 **自定义资源**
2. 选择 **harborcluster** 的资源
3. 进入托管 Harbor 所在 Namespace
4. 选择 YAML
5. 选择 v1beta1 版本
6. 编辑 YAML：

trivy:

```
offlineScan: true
skipUpdate: true
```

## 上传 trivy.db 和 metadata.json 文件

**先在 trivy pod 中创建对应的目录**  
**/home/scanner/.cache/trivy/db**

upload-trivy-db.png

upload-trivy-db.png

1. 从 **集群列表** 进入某个集群
2. 进入托管 Harbor 所在 Namespace
3. 进入 trivy 的工作负载
4. 点击控制台进入容器（如果有多个副本，每个副本都需要设置）
5. 进入容器后，执行 `cd /home/scanner/.cache/trivy`
6. 执行 `mkdir db`

## 创建好目录之后上传离线包

upload-trivy-offline-db.png

upload-trivy-offline-db.png

- 1.从 **集群列表** 进入某个集群
- 2.进入托管 Hharbor 所在 Namespace
- 3.进入 trivy 的工作负载
- 4.点击上传文件
- 5.在弹窗中写入上传路径 `/home/scanner/.cache/trivy/db`，点击 **确认**
- 6.进入选择文件页面，分别上传 `trivy.db` 和 `metadata.json` 文件

## 镜像空间

DCE 5.0 镜像仓库提供了基于镜像空间的镜像隔离功能。镜像空间分为公开和私有两种类型：

- 公开镜像仓库：所有用户都可以访问，通常存放公开的镜像，默认有一个公开镜像空间。
- 私有镜像仓库：只有授权用户才可以访问，通常存放镜像空间本身的镜像。

您可以选择不同的实例，查看其下的所有镜像空间。

切换实例

切换实例

点击某一个镜像空间的名称，可以查看当前空间内的镜像列表以及镜像回收规则。您可以为当前镜像空间创建镜像回收规则。所有镜像回收规则独立计算并且适用于所有符合条件的镜像，最多支持 15 条回收规则。

镜像空间包含什么

镜像空间包含什么

通过左侧导航栏的 **仓库集成(工作空间)** 进入镜像空间后，点击右侧的 **推送命令** 按钮。

按钮

按钮

可以查看将镜像推送到当前镜像空间的推送命令。

### 推送命令

#### 推送命令

点击 **上传镜像** 按钮，可以上传格式为 tar 或 tar.gz 的镜像文件。注意不要超过 2GB，一个文件对应一个镜像，目前仅支持上传 1.11.2 及以上容器引擎客户端版本制作的镜像压缩包。

### 上传镜像

#### 上传镜像

## 镜像列表

镜像列表以工作空间为维度，展示租户下所有可用的公开镜像和私有镜像。镜像来源主要有两部分：

1. 第一部分是平台集成或托管的镜像仓库中全部的公开镜像，以及通过镜像空间与工作空间绑定而单独分配给该工作空间的私有镜像。
2. 第二部分是工作空间主动关联某镜像仓库而获得的全部公开或私有镜像。

### 主要功能

- 快速部署应用：镜像列表和 **应用工作台** 均以工作空间为维度，因此当您在应用工作台选择同一工作空间部署应用时，可以通过 **选择镜像** 按钮一键获取该工作空间下镜像列表中所有可见的镜像，快速完成部署。
- 细粒度分配：管理员在平台集成或创建托管 Harbor 实例后，可以通过镜像空间与工作空间绑定的方式，将不同的镜像空间分配给不同的工作空间使用，实现细粒度分配。
- 镜像扫描：提供镜像扫描和分析功能，可视化查看漏洞等级及分布情况，提供扫描日志，

便于追踪漏洞。

- 推送镜像：可通过平台提供的命令将镜像推送到有权限的镜像空间中。
- 层级等详情查看：对可见范围内的镜像提供镜像版本、层级信息、创建人、创建时间等详细信息的展示。

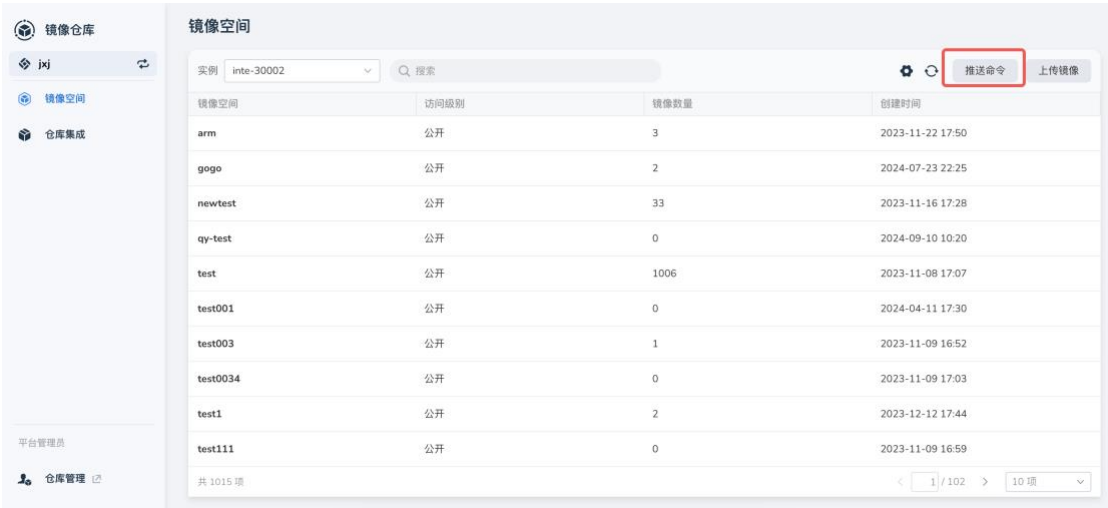
## 功能优势

- 安全可靠：通过镜像空间与工作空间绑定的方式实现细粒度的镜像分配，确保私有镜像只能被指定的工作空间拉取。
- 简单便捷：平台管理员创建托管 Harbor 或集成其他仓库后，若希望共享给所有工作空间及命名空间使用，只需要将镜像仓库公开即可。
- 灵活接入：除了平台级的仓库集成能力，在每个工作空间下还提供了仓库集成功能。集成后，该镜像仓库仅供该工作空间使用。
- 全局联动：应用工作台和容器管理针对镜像仓库提供了 **镜像选择器** 功能，在部署应用时可通过 **选择镜像** 按钮一键获取所有可见的镜像，快速完成部署。

## 推送命令

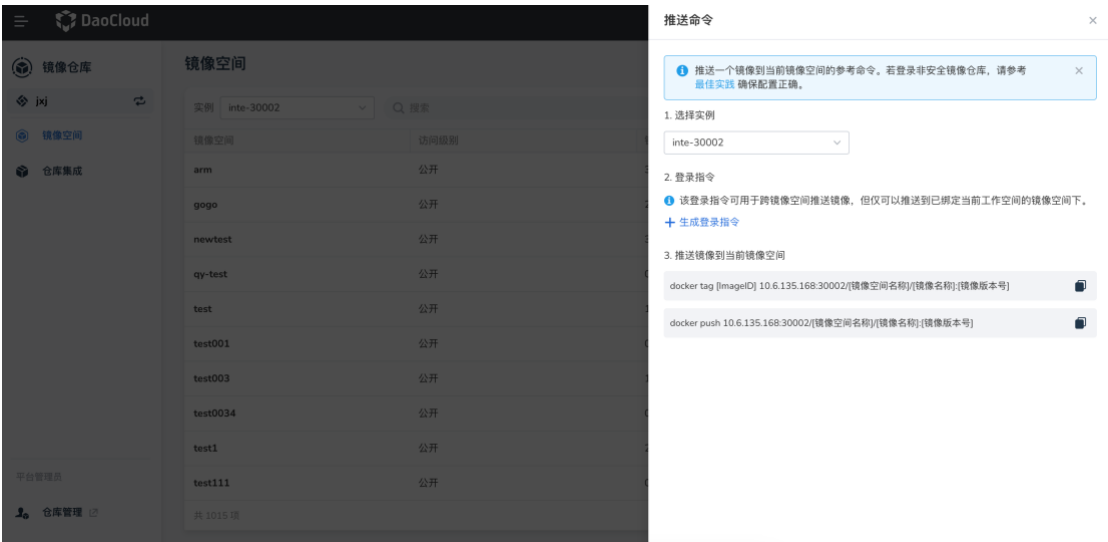
您可以推送一个镜像到当前镜像空间，并设置推送命令。

1. 在镜像空间页面上，点击右侧的 **推送命令** 按钮



点击按钮

2.可以 + 生成登录指令 后，查看推送镜像的命令。




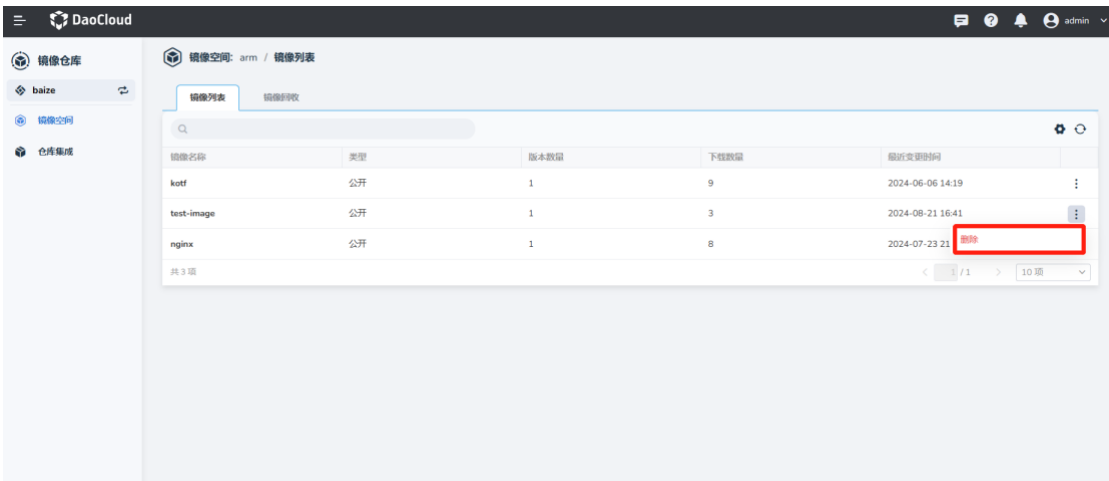
推送命令

推送命令示例：

```
docker tag [ImageID] 10.6.135.168: 30002/[镜像空间名称]/[镜像名称]:[镜像版本号]
docker push 10.6.135.168: 30002/[镜像空间名称]/[镜像名称]:[镜像版本号]
```

删除镜像

1.点击镜像空间右侧的 ，在弹出的菜单中点击 删除



删除操作

- 2. 确认无误后，在二次确认弹窗中输入镜像名称，点击 **删除**



二次确认

下一步: [Artifacts 和描述信息](#)

# Artifacts 和描述信息

在普通用户视图中，您可以查看 Harbor 镜像空间的详情，包括 Artifacts 扫描结果和镜像的详细描述。您可以使用 Markdown 为每个镜像自定义描述信息。

## Artifacts 详情

您可以参阅以下步骤，查看 Artifacts 的详情。

1. 选择一个 Harbor 实例，进入 **镜像空间** 页面，点击某个镜像空间的名称。

进入镜像空间

进入镜像空间

2. 进入镜像列表，点击某个镜像名称。

镜像列表

镜像列表

3. 默认显示 Artifacts 页签，点击某个版本，可以查看更多详情。

Artifacts

Artifacts

版本详情显示了当前 Artifact 的构建历史等信息。

Artifacts

Artifacts

## 自定义描述信息

您可以参阅以下步骤，用 Markdown 自定义每个镜像的描述信息。

1. 接着上一节的步骤，点击 **描述信息** 页签，点击 **编辑** 按钮。

edit

edit

2. 使用 Markdown 语法编写描述信息后，点击 **保存**。

save

save

3. 最终效果如下图。

result

result

# 镜像回收

在镜像仓库中，许多镜像在一段时间或更新到一定版本后，旧版本的镜像可能不再需要，这些多余的镜像会消耗大量的存储容量。如果您是 Workspace Admin，可对工作空间下所有镜像空间中的镜像进行管理。您可以通过设置镜像回收规则来管理镜像仓库中的镜像，通过定时/手动的方式将一些镜像按照一定规则放入回收站。

这里的镜像回收指的是删除镜像并回收创建镜像所占用的资源。当您不再需要一个镜像时，您可以将其删除，这将回收镜像占用的磁盘空间。此过程被称为“镜像回收”。通过镜像回收，您可以释放磁盘空间以便在您的系统上运行其他操作，同时也可以保持系统的整洁和优化。

您可以为当前镜像空间创建镜像回收规则。所有镜像回收规则独立计算并且适用于所有符合条件的镜像，目前 DCE 5.0 镜像仓库最多支持 15 条回收规则。

1. 使用具有 Workspace Admin 角色的用户登录 DCE 5.0，点击左侧导航栏 **镜像空间**，点击列表中的某个名称。

切换实例

切换实例

2. 点击 **镜像回收** 页签，点击 **创建回收规则** 按钮。

点击按钮

点击按钮

!!! Note

只有 Harbor 仓库才支持镜像回收

3. 按提示选择镜像，配置规则。

创建规则



创建规则

4. 返回镜像回收列表，点击右侧的 ，可以禁用、编辑或删除回收规则。

点击按钮

点击按钮

## 镜像扫描

镜像被下载后可以直接使用，为用户提供了很多方便，但是不一定安全，可能被恶意植入后门。用户需要扫描下载的镜像，获知镜像安全信息。

在 (DevOps) CI/CD 过程中，镜像被直接推送到镜像仓库，无法保证镜像的安全性，存在持续安全集成自动扫描的需求。

安全扫描是一种主动的防范措施，能有效避免黑客攻击行为，做到防患于未然，建议定时/手动扫描镜像。

(投产) 容器到生产环境之后，生产环境对容器的安全性要求较高，需要容器的安全性得到保证，因此使用镜像运行容器前，一定要对镜像进行扫描，从而提高安全性。

最后扫描结果应提供更多有关纠正措施的指导。当用户收到容器镜像受到漏洞困扰的坏消息时，需要为自己找出修复方法，漏洞来自哪里？可以做些什么来解决问题？

## 镜像扫描特性

目前 DCE 5.0 的镜像仓库模块支持以下几种扫描镜像：

- 托管的 Harbor 仓库支持 Trivy 扫描。
- 原生的 Harbor 仓库支持 Clair 和 Trivy 扫描，具体取决于用户安装了什么插件。

用户在扫描镜像索引时，会同步扫描被索引的所有镜像，扫描结果为被索引镜像的扫描结果

之和。

## 手动扫描镜像

对于关联和集成的仓库，将出现在镜像列表中。您可以按需手动扫描某些镜像。

1. 进入镜像仓库的 **镜像列表** 中，选择一个实例和镜像空间，点击某一个镜像。

镜像列表

镜像列表

2. 在镜像详情列表中，点击列表右侧的 **！**，在弹出菜单中选择 **扫描**。

扫描

扫描

3. 系统开始扫描镜像，通常状态依次为 **排队中**、**扫描中**、**扫描完成**。

排队中

排队中

扫描中

扫描中

扫描完成

扫描完成

扫描状态包括：

- 未扫描：镜像从未被扫描过。
- 不支持：此镜像不支持扫描。
- 排队中：扫描任务已安排但尚未运行。
- 扫描中：扫描任务正在进行中，并显示进度条。

- 查看日志：扫描任务未能完成。点击 **查看日志** 以查看相关日志。
- 扫描完成：扫描任务成功完成。

4. 扫描完成后，将光标悬浮在扫描完成的比例条上，可以查看扫描详情。

扫描完成

扫描完成

## 扫描原生 Harbor 镜像

集成的原生 Harbor 仓库，支持按 Clair 或 Trivy 进行扫描。

具体操作步骤为：

1. 以平台管理员登录镜像仓库，点击左侧底部的 **仓库集成**。

仓库集成

仓库集成

2. 在集成的仓库列表中，光标悬浮于某个仓库上，点击 **原生 Harbor** 图标。

仓库集成

仓库集成

3. 跳转到原生 Harbor，参阅[扫描 Harbor 镜像](#)。

## 仓库集成(工作空间)

仓库集成(工作空间)是针对工作空间提供的便捷接入仓库功能。 工作空间管理员 (Workspace Admin) 可根据需要为该工作空间灵活接入镜像仓库，供工作空间成员使用。接入后，成员在工作空间下的命名空间部署应用时可通过 **选择镜像** 按钮一键拉取工作空间下所有公开和私有的镜像，实现快速部署应用。

支持集成 3 种仓库：

- Harbor Registry：这是一个开源的企业级 Docker 镜像仓库，提供镜像存储、版本控制、访问控制和安全扫描等功能。它专注于为企业环境提供高度可定制和安全的容器镜像管理解决方案。Harbor Registry 支持跨多个容器编排平台进行集成，并具有丰富的权限管理和审计功能。
- Docker Registry：这是 Docker 官方提供的镜像仓库服务。它作为 Docker 生态系统的一部分，用于存储、分发和管理 Docker 镜像。Docker Registry 提供了基本的镜像存储和版本控制功能，可以通过简单的 API 进行操作。
- JFrog Artifactory：这是一个通用的软件包管理和分发平台，支持各种不同的包类型，包括 Docker 镜像。除了作为 Docker 镜像仓库外，Artifactory 还支持其他软件包格式（如 Maven、npm 等）的存储、分发和管理。Artifactory 提供强大的权限控制、缓存和快速复制等功能，同时具备高度可扩展性和可定制性。与上述两个镜像仓库相比，Artifactory 是一个较为全面的软件包管理平台，适用于跨多种包类型的工作负载。

## 优势

- 灵活便捷：工作空间的管理员均可自主接入一个或多个 Harbor / Docker 类型的镜像仓库，供工作空间成员使用。
- 全局联动：接入后，在应用工作台部署应用时，可通过 **选择镜像** 按钮，一键选择仓库中的镜像，实现快速部署应用。

## 操作步骤

可以参阅[视频教程](#)熟悉以下操作步骤：

1. 使用具有 Workspace Admin 角色的用户登录 DCE 5.0, 从左侧导航栏点击 **镜像仓库** ->

**仓库集成(工作空间) 。**

镜像仓库

镜像仓库

2. 点击右上角的 **仓库集成** 按钮。

仓库集成

仓库集成

3. 填写表单信息后点击 **确定** 。

填写表单

填写表单

!!! note

1. Docker Registry 镜像仓库若未设置密码可不填写，Harbor 仓库必须填写用户名/密码。  
1. 有关实际操作演示，请参阅[仓库集成(工作空间)视频演示](../videos/kangaroo.md)

## 仓库集成(管理员)

仓库集成(管理员)是集中管理平台镜像仓库的入口，既支持集成外部镜像仓库，如 Harbor Registry、Docker Registry；又能够自动集成平台创建的托管 Harbor。仓库集成后，平台管理员可以通过将镜像空间与工作空间绑定的方式，将某个私有镜像空间分配给一个或者多个工作空间（工作空间下的命名空间）。或者将镜像空间设置为公开，供平台所有命名空间使用。

## 主要功能

- 支持集成主流的镜像仓库，如 Harbor Registry、Docker Registry，帮助您集中管理平台级别的镜像仓库。
- 支持通过概览页，快速查看仓库地址、镜像空间数、存储使用量等数据。
- 支持创建并设置镜像空间状态为公开或私有。若镜像空间状态为公开，其下的镜像能够被平台所有命名空间使用。若镜像空间状态为私有，则将镜像空间与一个或多个工作空间绑定后，只有被绑定的工作空间下的命名空间才能够使用该私有镜像，保证私有镜像的安全性。
- 自动集成托管 Harbor，平台创建托管 Harbor 实例后将被自动集成到集成仓库列表，进行统一的管理。

## 功能优势

- 统一管理入口，对集成镜像仓库和托管 Harbor 实例进行统一管理。
- 高安全性，私有镜像只能通过镜像空间与工作空间绑定的方式才能在部署应用时被拉到。
- 方便快捷，镜像空间一旦被设置为公开，则平台范围内的所有命名空间在部署应用时都能够拉取其下的公开镜像。
- 支持主流镜像仓库类型：Harbor Registry、Docker Registry。

## 操作步骤

可以参阅[视频教程](#)熟悉以下操作步骤：

1. 使用具有 Admin 角色的用户登录 DCE 5.0，从左侧导航栏点击 **镜像仓库** -> **仓库集成 (管理员)**。

## 仓库集成

### 仓库集成

2. 点击右上角的 **仓库集成** 按钮。

### 点击按钮

### 点击按钮

3. 选择仓库类型，填写集成名称、仓库地址、用户名和密码，点击 **确定**。

### 填写参数

### 填写参数

!!! note

对于 Harbor 仓库，必须提供 Admin 级别的用户名/密码。

4. 返回仓库集成列表。集成的仓库将带有 **集成**、**健康** 或 **不健康** 等标签。光标悬浮到

某个磁贴上，可以执行 **删除集成**、**编辑** 等操作。

### 更多操作

### 更多操作

下一步：[创建镜像空间](#)

## 创建镜像空间

Harbor 提供了基于镜像空间（project）的镜像隔离功能。镜像空间分为公开和私有两种类型：

- 公开镜像仓库：所有用户都可以访问，通常存放公开的镜像，默认有一个公开镜像空间。
- 私有镜像仓库：只有授权用户才可以访问，通常存放镜像空间本身的镜像。

前提条件：已经创建或集成了一个外部 Harbor 仓库。

1. 使用具有 Admin 角色的用户登录 DCE 5.0，从左侧导航栏点击 **镜像仓库** -> **仓库集成**

(管理员)。

仓库集成

仓库集成

2. 点击某个仓库名称。

点击某个名称

点击某个名称

3. 在左侧导航栏点击 **镜像空间**，在右上角点击 **创建镜像空间**。

创建镜像空间

创建镜像空间

4. 填写镜像空间名称，勾选类型后点击 **确定**。

填写

填写

5. 返回镜像空间列表，提示镜像空间创建成功。

成功

成功

6. 找到刚创建的镜像空间，点击右侧的 **⋮**，可以执行[绑定/解绑工作空间](#)、删除等操作。

其他操作

其他操作

!!! info

- 若镜像空间状态为`公开`，则空间中的镜像能够被平台上的所有 **Kubernetes** 命名空间拉取使用；
- 若镜像空间状态为`私有`，则只有在管理员 **Admin** 将该镜像空间绑定到一个或多个工作空间（租户）后，才能被工作空间（租户）下的 **Kubernetes** 命名空间拉取使用。

下一步：[绑定/解绑工作空间](#)



# 绑定/解绑工作空间

镜像空间有两种类型：公开和私有。公开镜像空间中的镜像为公开镜像，公开镜像可被平台所有用户拉取；私有镜像空间中的镜像为私有镜像，只有绑定了工作空间后，才能被工作空间下的成员拉取，非工作空间成员无法拉取。

- 对于公开镜像，用户在容器管理模块部署应用时能够通过 **选择镜像** 按钮，选择 **镜像仓库** -> **仓库集成** 中所有公开镜像空间中的镜像部署应用，无需任何配置。
- 对于私有镜像，则需要管理员将私有镜像空间分配给工作空间（租户）后，才能够被工作空间下的成员使用，确保私有镜像的安全性。

前提条件：已经创建或集成了一个外部 Harbor 仓库，且已[创建镜像空间](#)。

## 绑定步骤

通常对私有的镜像空间，执行工作空间绑定，便于租户或成员使用其下的镜像。

- 1.使用具有 Admin 角色的用户登录 DCE 5.0，从左侧导航栏点击 **镜像仓库** -> **仓库集成** (管理员) 。

仓库集成

仓库集成

- 2.点击某个仓库名称。

点击某个名称

点击某个名称

- 3.在左侧导航栏点击 **镜像空间** ，在某个镜像空间最右侧，点击 **！** ，选择 **绑定/解绑工作空间** 。

## 绑定/解绑

### 绑定/解绑

若没有绑定任何工作空间，工作空间一栏将显示待分配。

4. 在弹窗中，选择一个或多个工作空间，点击 **确定** 。

### 绑定

#### 绑定

若想取消某个工作空间的绑定，只需在已选中的工作空间列表中，点击 **x** 。

5. 提示绑定/解绑工作空间成功，光标移到工作空间一栏，将显示所绑定的工作空间。

### 绑定

#### 绑定

该工作空间（租户）下的成员将都能拉取这个镜像空间中的镜像。

## FAQ

1. 在 Kubernetes 命名空间部署应用时，通过 **选择镜像** 按钮仍选择不到镜像空间中的镜像。
  - 排查该 Kubernetes 命名空间是否绑定了工作空间（需要绑定）。
  - 排查该镜像空间是否绑定了 Kubernetes 命名空间所在的工作空间（需要绑定）。
  - 排查该镜像空间状态是私有还是公开，切换如下 tab 查找。

### 公开镜像

#### 公开镜像

2. 将镜像空间分配给工作空间使用，与在工作空间下集成仓库有什么区别？

平台管理员 Admin 可以统一管理，批量将一个镜像空间分配给多个工作空间使用，而无需分别绑定。

工作空间管理员 Workspace Admin 可以根据需要自己集成外部镜像仓库给成员使用，而无需全部依赖于平台管理员，使用起来更为灵活。

下一步：[部署应用](#)

## 镜像同步

镜像同步是指将两个或多个镜像仓库之间的镜像进行同步更新的过程。在软件开发或系统管理中，这种同步更新方式常用于确保在多个服务器之间共享相同的软件或操作系统镜像，以便在进行部署时能够确保一致性和减少工作量。通常情况下，通过镜像同步可以实时地将镜像内容同步到其他服务器上，从而保证多个服务器上的镜像都是最新版本。

DCE 5.0 镜像仓库可以让用户创建同步规则，添加目标仓库等。

!!! note

创建同步规则和添加目标仓库需要管理员权限。

## 创建同步规则

1. 从仓库集成或托管 Harbor 页面中，点击某个仓库。

选择一个仓库

选择一个仓库

2. 在左侧导航栏，依次点击 **镜像同步** -> **同步规则**，点击 **创建同步规则** 按钮。

点击按钮

点击按钮

3. 填写各项参数后点击 **确定**。


## 配置参数

### 配置参数

#### 同步模式：

- 推送：将镜像同步到目标仓库
- 拉取：把目标仓库的镜像同步到当前仓库
- 覆盖：指定如果存在相同资源的名称，是否要覆盖 镜像过滤器：
- 名称：按名称过滤当前镜像空间下的资源，不填或 \*\* 将匹配所有资源。
- Tag：按 tag/version 过滤当前镜像空间下的资源，不填或 \*\* 将匹配所有资源。

目标镜像空间：如果不填写，镜像将被放到与源仓库相同的镜像空间中。

4. 返回同步规则列表，新建的规则默认处于启用状态。点击右侧的 ，可以执行同步、

编辑、禁用、删除等操作。

## 更多操作

### 更多操作

## 添加目标仓库

1. 从仓库集成或托管 Harbor 页面中，点击某个仓库。

### 选择一个仓库

#### 选择一个仓库

2. 在左侧导航栏，依次点击 **镜像同步** -> **目标仓库**，点击 **添加目标仓库** 按钮。

### 点击按钮

#### 点击按钮


3. 可以点选已集成的仓库，或自定义仓库。

选择

选择

自定义

自定义

4. 返回目标仓库列表，点击右侧的 ，可以执行编辑、删除等操作。

## 托管 Harbor

Harbor 是一个开源的镜像仓库服务，用于容器镜像、Helm Chart 等符合 OCI 标准的 Artifact 的安全托管及高效分发，能够帮助您跨云原生计算平台（如 Kubernetes 和 Docker）一致且安全地管理 Artifact。DCE 5.0 提供了基于 Harbor 的快捷部署能力，并通过与平台中的应用工作台、容器管理模块打通，与工作空间绑定等一系列便捷通道，实现了一站式高可用、高性能、高效率的部署、管理、使用等全周期的镜像仓库服务。

## 产品功能

- 支持多副本部署实现高可用
- 支持将平台用户导入原生 Harbor 实例
- 提供原生 Harbor 实例入口，用户在平台 UI 的操作与在原生 Harbor 中的操作实时同步生效

步生效

- 支持使用平台自建数据库或接入外部数据库
- 支持使用平台自建 Redis 实例或接入外部 Redis 实例
- 支持指定内部存储或使用外部 S3 兼容对象存储

## 功能优势

- 多镜像仓库实例，满足开发、测试、生产等多种环境，多套镜像仓库的需求。
- 打破模块之间的调用壁垒，支持在应用工作台和容器管理模块部署应用时快速拉取镜像
- 提供统一的管理控制平面，管理员能够在同一界面对多个 Harbor 实例进行全生命周期管理。

## 操作步骤

1. [安装 Harbor Operator](#)
2. [创建托管 Harbor 实例](#)

## 安装 Harbor Operator

托管 Harbor 使用的是 Harbor Operator 技术来进行 Harbor 创建、升级、删除等全生命周期管理。在创建托管 Harbor 之前，需要先在容器管理中安装 Harbor Operator，版本要求最低 1.4.0。

提示：Harbor Operator 依赖 Cert Manager，所以需要先安装好 Cert Manager。

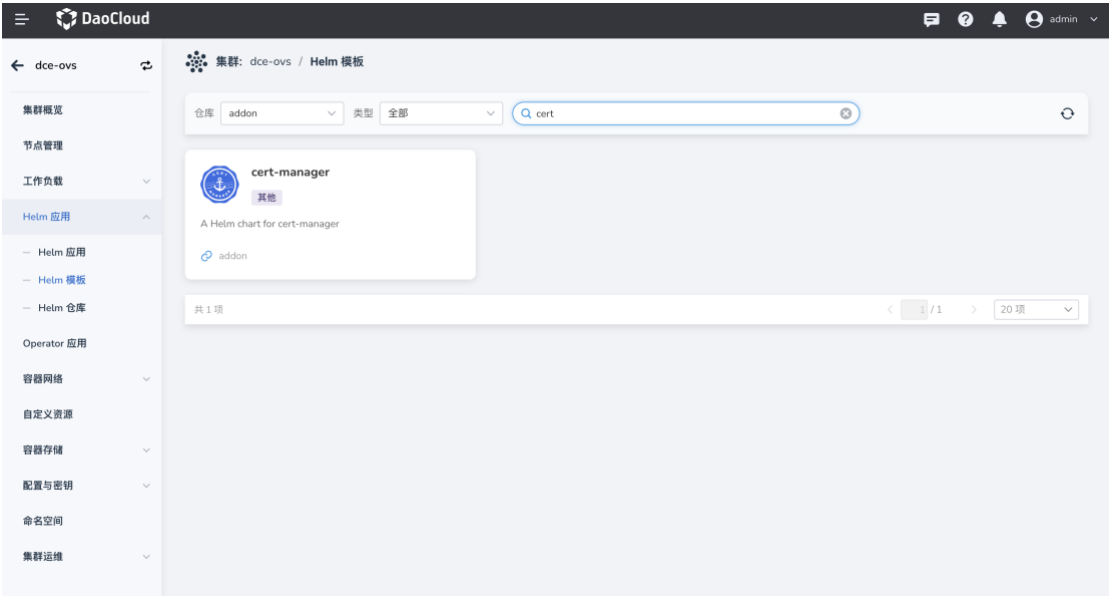
如果在创建 Harbor 实例时，出现以下异常提示，请点击 **前往安装**。（必须先安装 Cert Manager ！）



harbor-operator 异常

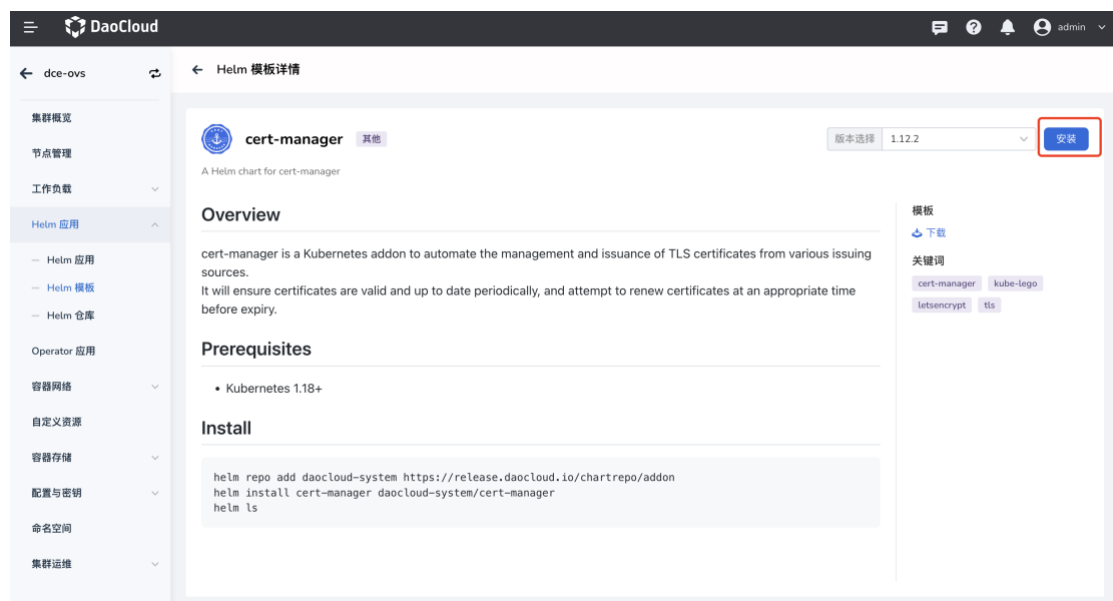
## 第一步

1. 进入 容器管理 的 Helm 应用 -> Helm 模板 ， 找到并点击 cert-manager 卡片。



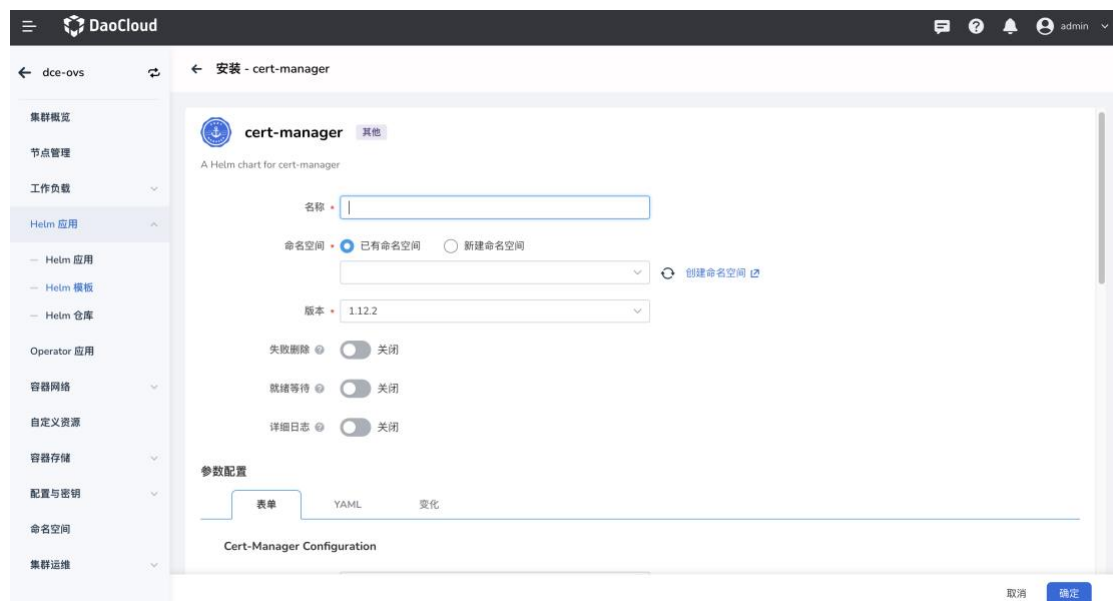
搜索 cert-manager

2. 选择版本， 点击 安装 。



## 安装

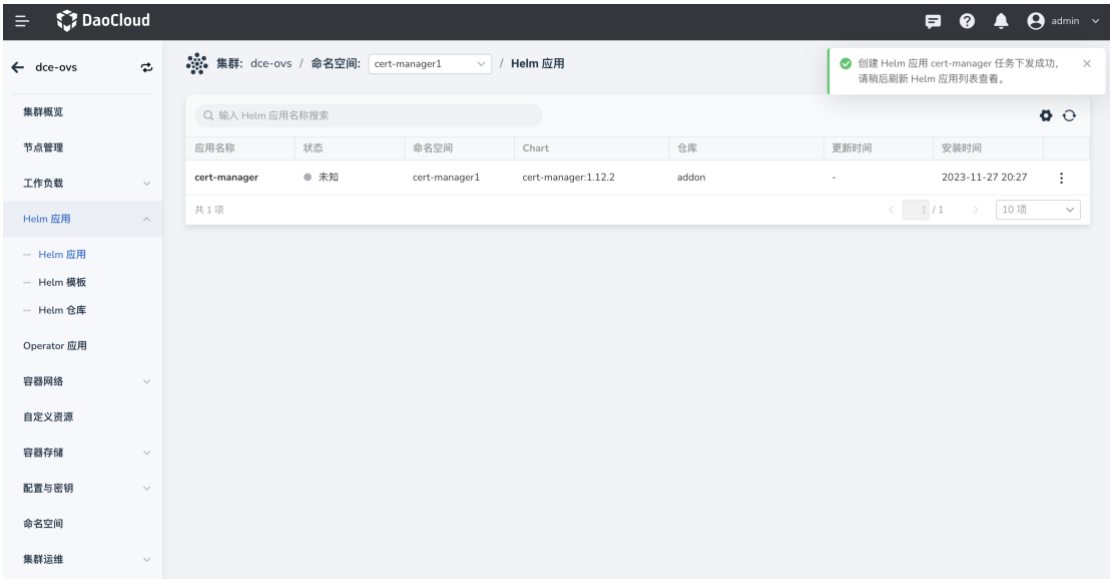
3. 输入名称和命名空间后，点击 **确定**，如果想要添加其他参数，请参考下一节的参数说明。



## 填表

4. 等待安装完成。





安装

第二步

1.进入 容器管理 的 Helm 应用 -> Helm 模板 ，找到并点击 harbor-operator 卡片。

找到 operator

找到 operator

2.选择版本，点击 安装 。

安装

安装

3.输入名称和命名空间后，点击 确定 ，如果想要添加其他参数，请参考下一节的参数说明。

填表

填表

4.等待安装完成。

安装

## 安装

## 参数值

Harbor Operator 在安装过程中有较多参数可以填写和控制，具体参数请参考如下表格内容：

其中 minio-operator.enabled、postgres-operator.enabled 和 redis-operator.enabled 只能为 false。

| Key  | Type   | Default   | Description   |
|--|--------|-----------|---|
| affinity                                   | object | {}        | <p>Expects input structure as per specification</p> <p><a href="https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#affinity-v1-core">https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#affinity-v1-core</a> For example:</p> <pre>{   "nodeAffinity": {     "requiredDuringSchedulingIgnoredDuringExecution": {       "nodeSelectorTerms": [         {           "matchExpressions": [             {               "key": "foo.bar.com/role",               "operator": "In",               "values": [                 "master"               ]             }           ]         }       ]     }   } }</pre> |
| allowPrivilegeEscalation                   | bool   | false     | Allow privilege escalation for the controller Pods  |
| autoscaling.enabled                        | bool   | false     | Whether to enable <a href="#">Horizontal Pod Autoscaling</a>  |
| autoscaling.maxReplicas                    | int    | 100       | Maximum controller replicas   |
| autoscaling.minReplicas                    | int    | 1         | Minimum controller replicas   |
| autoscaling.targetCPUUtilizationPercentage | int    | 80        | CPU usage target for autoscaling  |
| autoscaling.targetMemoryUsagePercentage    | int    | No target | Memory usage target for autoscaling   |

| Key  | Type   | Default | Description  |
|--|--------|---------|--|
| targetMemoryUtilizationPercentage            |        |         |  |
| controllers.chartmuseum.maxReconcile         | int    | 1       | Max parallel reconciliation for ChartMuseum controller                         |
| controllers.common.classname                 | string | ""      | Harbor class handled by the operator. An empty class means watch all resources |
| controllers.common.networkPolicies           | bool   | false   | Whether the operator should manage network policies                            |
| controllers.common.watchChildren             | bool   | true    | Whether the operator should watch children                                     |
| controllers.core.maxReconcile                | int    | 1       | Max parallel reconciliation for Core controller                                |
| controllers.harbor.maxReconcile              | int    | 1       | Max parallel reconciliation for Harbor controller                              |
| controllers.harborConfiguration.maxReconcile | int    | 1       | Max parallel reconciliation for HarborConfiguration controller                 |
| controllers.harborcluster.maxReconcile       | int    | 1       | Max parallel reconciliation for HarborCluster controller                       |
| controllers.jobservice.maxReconcile          | int    | 1       | Max parallel reconciliation for JobService controller                          |
| controllers.notaryserver.maxReconcile        | int    | 1       | Max parallel reconciliation for NotaryServer controller                        |
| controllers.notarysigner.maxReconcile        | int    | 1       | Max parallel reconciliation for NotarySigner controller                        |
| controllers.portal.maxReconcile              | int    | 1       | Max parallel reconciliation for Portal controller                              |

| Key                                  | Type   | Default                    | Description  |
|--------------------------------------|--------|----------------------------|--|
| concile                              |        |                            |  |
| controllers.registry.maxReconcile    | int    | 1                          | Max parallel reconciliation for Registry controller  |
| controllers.registryctl.maxReconcile | int    | 1                          | Max parallel reconciliation for RegistryCtl controller   |
| controllers.trivy.maxReconcile       | int    | 1                          | Max parallel reconciliation for Trivy controller   |
| deploymentAnnotations                | object | {}                         | Additional annotations to add to the controller Deployment   |
| fullnameOverride                     | string | ""                         |  |
| harborClass                          | string | ""                         | Class name of the Harbor operator  |
| image.pullPolicy                     | string | "IfNotPresent"             | The image pull policy for the controller.  |
| image.registry                       | string | "docker.io"                | The image registry whose default is docker.io.   |
| image.repository                     | string | "goharbor/harbor-operator" | The container registry whose default is the chart appVersion.  |
| image.tag                            | string | "dev_master"               | The image tag whose default is the chart appVersion.   |
| imagePullSecrets                     | list   | []                         | Reference to one or more secrets to be used when pulling images<br><a href="https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/">https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/</a> For example:<br>[ {"name": "image-pull-secret"} ] |
| installCRDs                          | bool   | false                      | If true, CRD resources will be installed as part of the Helm chart. If enabled, when uninstalling CRD resources will be deleted causing all installed custom resources to be DELETED   |
| leaderElection.namespace             | string | "kube-system"              | The namespace used to store the ConfigMap for leader election  |
| logLevel                             | int    | 4                          | Set the verbosity of controller.<br>Range of 0 - 6 with 6 being the most verbose. Info level is 4.   |
| minio-operator.enabled               | bool   | false                      | Whether to enabled <a href="#">MinIO Operator</a>  |

| Key   | Type   | Default                                      | Description   |
|---|--------|--|---|
| nameOverride  | string | ""   |   |
| nodeSelector  | object | {}   | <p>Expects input structure as per specification</p> <p><a href="https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#nodeselector-v1-core">https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#nodeselector-v1-core</a> For example:</p> <pre>[ { "matchExpressions":   [ { "key":     "kubernetes.io/e2e-az-name",     "operator": "In", "values":     [ "e2e-az1",       "e2e-az2" ] } ] } ]</pre> |
| podAnnotations  | object | {}   | Additional annotations to add to the controller Pods  |
| podLabels   | object | {}   | Additional labels to add to the controller Pods   |
| podSecurityContext                                      | object | { "runAsNonRoot": true, "runAsUser": 65532 } | <p>Expects input structure as per specification</p> <p><a href="https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#podsecuritycontext-v1-core">https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#podsecuritycontext-v1-core</a> For example: { "fsGroup": 2000, "runAsUser": 1000, "runAsNonRoot": true }</p>  |
| postgres-operator.configKubernetes.secret_name_template | string | "{username}.{cluster}.credentials"           |   |
| postgres-operator.enabled                               | bool   | false  | Whether to enabled <a href="#">Postgres operator</a>  |
| priorityClassName                                       | string | ""   | priority class to be used for the harbor-operator pods  |
| rbac.create   | bool   | true   | Whether to install Role Based Access Control  |
| redis-operator.enabled                                  | bool   | false  | Whether to enabled <a href="#">Redis Operator</a>   |
| redis-operator.image.tag                                | string | "v1.2.0"                                     |   |
| replicaCount  | int    | 1  | Number of replicas for the controller   |

| Key                        | Type   | Default   | Description  |
|----------------------------|--------|---|--|
| resources                  | object | <code>{"limits": {"cpu": "500m", "memory": "300Mi"}, "requests": {"cpu": "300m", "memory": "200Mi"}}</code> | Expects input structure as per specification<br><a href="https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#resourcerequirements-v1-core">https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#resourcerequirements-v1-core</a>  |
| service.port               | int    | 443   | Expose port for WebHook controller   |
| service.type               | string | "ClusterIP"   | Service type to use  |
| serviceAccount.annotations | object | <code>{}</code>   | Annotations to add to the service account  |
| serviceAccount.create      | bool   | true  | Specifies whether a service account should be created  |
| serviceAccount.name        | string | ""  | The name of the service account to use. If not set and create is true, a name is generated using the fullname template   |
| strategy                   | object | <code>{}</code>   | Expects input structure as per specification<br><a href="https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#deploymentstrategy-v1-apps">https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#deploymentstrategy-v1-apps</a> For example: <code>{ "type": "RollingUpdate", "rollingUpdate": { "maxSurge": 0, "maxUnavailable": 1 } }</code> |
| tolerations                | list   | <code>[]</code>   | Expects input structure as per specification<br><a href="https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#toleration-v1-core">https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#toleration-v1-core</a> For example: <code>[ { "key": "foo.bar.com/role", "operator": "Equal", "value": "master", "effect": "NoSchedule" } ]</code>    |
| volumeMounts               | list   | <code>[]</code>   | Expects input structure as per specification<br><a href="https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#volumemount-v1-core">https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#volumemount-v1-core</a> For example: <code>[ { "mountPath": "/test-ebs", "name":</code>  |

| Key     | Type | Default | Description  |
|---------|------|---------|--|
| volumes | list | []      | <div>"test-volume" }]<br/>Expects input structure as per specification<br/><a href="https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#volume-v1-core">https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#volume-v1-core</a> For example:<br/>[ { "name": "test-volume",<br/>"awsElasticBlockStore":<br/>{ "volumeID":<br/>"&lt;volume-id&gt;", "fsType":<br/>"ext4" } } ]</div> |

下一步: [创建托管 Harbor 实例](#)

## 创建托管 Harbor

前提条件: 需要先安装好 [Cert Manager](#) 和 [Harbor Operator](#) 。

!!! note

对于 Harbor 实例, 除了接入管理员账号外, 还可以接入机器人账号达到同样的接入效果。

- 1.使用具有 Admin 角色的用户登录 DCE 5.0, 从左侧导航栏点击 **镜像仓库 -> 托管**

**Harbor 。**



harbor 实例入口

- 2.点击右上角的 **创建实例** 按钮。

## 创建实例

### 创建实例

3. 填写实例名称，选择部署位置后点击 **下一步**（若无部署位置可选，需先前往容器管理创建集群和命名空间）。

## 基本信息

### 基本信息

4. 填写数据库、Redis 实例和镜像 /Charts 存储信息后点击 **下一步**。

若已安装 **中间件** 模块，系统会自动检测 PostgreSQL 和 Redis，如果没有可用的数据库，可以点击创建链接进行创建；若未安装 **中间件** 模块，可选择 **接入外部实例** 的方式使用外部 PostgreSQL 和 Redis。

数据库填写提示：

- 地址：postgres://{host}:{port}，例如 postgres://acid-minimal-cluster.default.svc:5432
- 用户名：填写连接数据库的用户名
- 密码：填写连接数据库的密码

Redis 填写分为单机和哨兵模式：

- 单机模式填写地址：redis://{host}:{port}，需要替换 host、port 两个参数。
- 哨兵模式填写地址：redis+sentinel://{host}:{port}?sentinelMasterId={master\_id}，需要替换 host、port、master\_id 三个参数。
- 密码：按需填写



规格配置

版本: 2.6.3

副本数: ☒ 单副本 ☐ 三副本 ☐ 其它  
请选择选择“单副本”模式。推荐使用多副本模式保证实例高可用。

CPU配额: 请求量 0.2 Core 限制量 0.5 Core

内存配额: 请求量 0.5 GB 限制量 1 GB

数据库: ☒ 内置实例 ☐ 接入外部实例

工作空间: [选择]

PostgreSQL: [创建 PostgreSQL](#)

| PostgreSQL名称 | 状态 | 模式 |
|--------------|----|----|
| 暂无数据         |    |    |

Redis: ☒ 内置实例 ☐ 接入外部实例

工作空间: [选择]

Redis: [创建 Redis](#)

| Redis名称 | 状态 | 模式 |
|---------|----|----|
| 暂无数据    |    |    |

镜像/Charts存储: ☒ Storage Class ☐ MinIO对象存储 ☐ S3兼容对象存储

指定存储: nfs-sc [创建存储池](#)

容量存储: 900G

## 规格配置

5. 填写域名，选择 Ingress 实例，输入管理员密码后点击 **确定**（用户名/密码用于登录原生 Harbor 实例，请妥善保管密码）。

域名填写提示: http://{host}, host 前面的 **http://** 必须要带上。

✓

基本信息

✓

规格配置

3

访问与策略绑定

访问类型

☒ Ingress

☐ NodePort

协议

☐ HTTP

☒ HTTPS

🔒

域名

🌐

https://

证书

▼

[创建证书](#)

选择Ingress实例

▼

🔄

没找到?

[安装 Ingress 实例](#)

管理员信息

用户名

admin

密码

👁

必须包含大小写字母和数字，长度8-20位。

确认密码

👁

访问策略

同步用户

☐ 将用户同步至Harbor实例

1. 将平台中的用户同步至Harbor实例。

2. 同步后用户可以通过平台的用户名/密码登录Harbor实例。

3. 用户在Harbor实例中的权限，需二次授权。

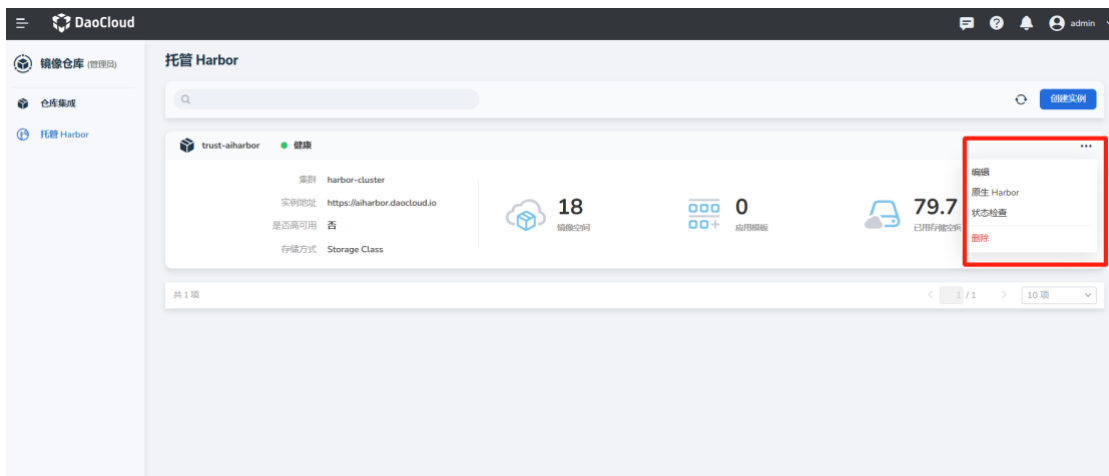
## 访问与策略绑定

6. 返回托管 Harbor 实例列表，新创建的实例默认位于第一个，等待状态从更新中变为健康，即可正常使用。

## 实例列表

## 实例列表

7. 点击某个实例右侧的 ...，可以选择编辑、删除、进入原生 Harbor，或者查看状态。



更多操作

8. **托管 Harbor** 主要是对 Harbor 实例的全生命周期进行管理。创建完成后，系统会自动将该实例集成到 **仓库集成** 列表，您可前往 **仓库集成** 列表进入实例详情进行 创建镜像空间和镜像同步等操作

下一步：[创建镜像空间](#)

## 创建镜像空间

Harbor 提供了基于镜像空间（project）的镜像隔离功能。镜像空间分为公开和私有两种类型：

- 公开镜像仓库：所有用户都可以访问，通常存放公开的镜像，默认有一个公开镜像空间。
- 私有镜像仓库：只有授权用户才可以访问，通常存放镜像空间本身的镜像。

前提条件：已经创建或集成了一个外部 Harbor 仓库。

1. 使用具有 Admin 角色的用户登录 DCE 5.0，从左侧导航栏点击 **镜像仓库** -> **仓库集成 (管理员)**。

仓库集成

仓库集成

2. 点击某个仓库名称。

点击某个名称

点击某个名称

3. 在左侧导航栏点击 **镜像空间**，在右上角点击 **创建镜像空间**。

创建镜像空间

创建镜像空间

4. 填写镜像空间名称，勾选类型后点击 **确定**。


填写

填写

5. 返回镜像空间列表，提示镜像空间创建成功。

成功

成功

6. 找到刚创建的镜像空间，点击右侧的 ，可以执行[绑定/解绑工作空间](#)、删除等操作。

其他操作

其他操作

!!! info

- 若镜像空间状态为`公开`，则空间中的镜像能够被平台上的所有 Kubernetes 命名空间拉取使用；
- 若镜像空间状态为`私有`，则只有在管理员 Admin 将该镜像空间绑定到一个或多个工作空间（租户）后，才能被工作空间（租户）下的 Kubernetes 命名空间拉取使用。

下一步：[绑定/解绑工作空间](#)

## 绑定/解绑工作空间

镜像空间有两种类型：公开和私有。公开镜像空间中的镜像为公开镜像，公开镜像可被平台所有用户拉取；私有镜像空间中的镜像为私有镜像，只有绑定了工作空间后，才能被工作空间下的成员拉取，非工作空间成员无法拉取。

- 对于公开镜像，用户在容器管理模块部署应用时能够通过 **选择镜像** 按钮，选择 **镜像仓库** -> **仓库集成** 中所有公开镜像空间中的镜像部署应用，无需任何配置。
- 对于私有镜像，则需要管理员将私有镜像空间分配给工作空间（租户）后，才能够被工作空间下的成员使用，确保私有镜像的安全性。

前提条件：已经创建或集成了一个外部 Harbor 仓库，且已[创建镜像空间](#)。

## 绑定步骤

通常对私有的镜像空间，执行工作空间绑定，便于租户或成员使用其下的镜像。

- 1.使用具有 Admin 角色的用户登录 DCE 5.0，从左侧导航栏点击 **镜像仓库** -> **仓库集成**  
**(管理员)**。

仓库集成

仓库集成

- 2.点击某个仓库名称。

点击某个名称

点击某个名称

- 3.在左侧导航栏点击 **镜像空间**，在某个镜像空间最右侧，点击 **！**，选择 **绑定/解绑工作空间**。

绑定/解绑

绑定/解绑

若没有绑定任何工作空间，工作空间一栏将显示待分配。

- 4.在弹窗中，选择一个或多个工作空间，点击 **确定**。

绑定

绑定

若想取消某个工作空间的绑定，只需在已选中的工作空间列表中，点击 **x**。

- 5.提示绑定/解绑工作空间成功，光标移到工作空间一栏，将显示所绑定的工作空间。

绑定

绑定

该工作空间（租户）下的成员将都能拉取这个镜像空间中的镜像。

## FAQ

1. 在 Kubernetes 命名空间部署应用时，通过 **选择镜像** 按钮仍选择不到镜像空间中的镜像。

- 排查该 Kubernetes 命名空间是否绑定了工作空间（需要绑定）。
- 排查该镜像空间是否绑定了 Kubernetes 命名空间所在的工作空间（需要绑定）。
- 排查该镜像空间状态是私有还是公开，切换如下 tab 查找。

### 公开镜像

#### 公开镜像

2. 将镜像空间分配给工作空间使用，与在工作空间下集成仓库有什么区别？

平台管理员 Admin 可以统一管理，批量将一个镜像空间分配给多个工作空间使用，而无需分别绑定。

工作空间管理员 Workspace Admin 可以根据需要自己集成外部镜像仓库给成员使用，而无需全部依赖于平台管理员，使用起来更为灵活。

下一步：[部署应用](#)

## 镜像同步

镜像同步是指将两个或多个镜像仓库之间的镜像进行同步更新的过程。在软件开发或系统管理中，这种同步更新方式常用于确保在多个服务器之间共享相同的软件或操作系统镜像，以便在进行部署时能够确保一致性和减少工作量。通常情况下，通过镜像同步可以实时地将镜像内容同步到其他服务器上，从而保证多个服务器上的镜像都是最新版本。

DCE 5.0 镜像仓库可以让用户创建同步规则，添加目标仓库等。

!!! note

创建同步规则和添加目标仓库需要管理员权限。

## 创建同步规则

1. 从仓库集成或托管 Harbor 页面中，点击某个仓库。

选择一个仓库

选择一个仓库

2. 在左侧导航栏，依次点击 **镜像同步** -> **同步规则**，点击 **创建同步规则** 按钮。

点击按钮

点击按钮

3. 填写各项参数后点击 **确定**。

配置参数

配置参数

同步模式：

- 推送：将镜像同步到目标仓库
- 拉取：把目标仓库的镜像同步到当前仓库
- 覆盖：指定如果存在相同资源的名称，是否要覆盖 镜像过滤器：
- 名称：按名称过滤当前镜像空间下的资源，不填或 \*\* 将匹配所有资源。
- Tag：按 tag/version 过滤当前镜像空间下的资源，不填或 \*\* 将匹配所有资源。

目标镜像空间：如果不填写，镜像将被放到与源仓库相同的镜像空间中。

4. 返回同步规则列表，新建的规则默认处于启用状态。点击右侧的 **⋮**，可以执行同步、

编辑、禁用、删除等操作。

更多操作

更多操作

## 添加目标仓库

1. 从仓库集成或托管 Harbor 页面中，点击某个仓库。

选择一个仓库

选择一个仓库

2. 在左侧导航栏，依次点击 **镜像同步** -> **目标仓库**，点击 **添加目标仓库** 按钮。

点击按钮

点击按钮

3. 可以点选已集成的仓库，或自定义仓库。

选择

选择

自定义

自定义

4. 返回目标仓库列表，点击右侧的 **⋮**，可以执行编辑、删除等操作。

## Harbor 迁移指南

本文档将指导您将原始 Harbor 环境迁移至新的集群环境。此迁移案例模拟了一个复杂场景，原始 Harbor 使用 Localpath 存储，新的环境使用 minio 对象存储，其中涉及到了 Harbor、PostgreSQL 和 Minio 的迁移过程。



## 环境准备

- 1.原始 Harbor 环境：使用 Localpath 作为存储方式的 Harbor 容器环境。
- 2.新集群环境：包含新的 PostgreSQL 和 Minio 存储环境。
- 3.合理规划存储等资源
- 4.待迁移 Harbor ：在迁移过程中，**将原始 Harbor 设置为只读模式**，确保数据的一致性和完整性。

## 步骤 1：迁移 PostgreSQL

在这一步骤中，我们将迁移原始 Harbor 所使用的 PostgreSQL 数据库至新的集群环境。

工具安装参见：[PostgreSQL 官网](#)

- 1.使用 pg\_dump 工具备份 Harbor 的 PostgreSQL 数据库：

```
pg_dump --username=<原始数据库用户名> --host=<原始数据库主机> --port=<原始数据库端口> --format=plain --file=harbor_backup.sql <数据库名>
```

请将尖括号中的参数替换为实际的值。例如，如果原始数据库用户名为 postgres，原始

数据库主机为 localhost，端口为 32332，数据库名为 core，则命令应为：

```
pg_dump --username=postgres --host=localhost --port=32332 --format=plain --file=harbor_backup.sql core
```

- 2.将数据库备份文件 harbor\_backup.sql 从原始 Harbor 环境传输到新的 PostgreSQL 环境。

- 3.在新的 PostgreSQL 环境中恢复数据库：

```
psql --host=<新数据库主机> --port=<新数据库端口> --username=<新数据库用户名> --dbname=<数据库名> --file=harbor_backup.sql
```

请将尖括号中的参数替换为实际的值。例如，如果新数据库主机为 localhost，端口为

32209，新数据库用户名为 postgres，数据库名为 core，则命令应为：

```
psql --host=localhost --port=32209 --username=postgres --dbname=core --file=harbor_backup.sql
```

完成以上步骤后，您已经成功将 Harbor 的 PostgreSQL 数据库迁移至新的集群环境。请确保在整个迁移过程中做好备份，并在迁移前做好充分的测试，以确保数据的安全性和业务的连续性。

## 步骤 2：迁移镜像数据至 Minio

在这一步骤中，我们将迁移原始 Harbor 存储的镜像数据至 Minio 存储系统。请按照以下步骤逐步进行：

1. 在新的集群环境中下载并安装 rclone 工具。rclone 是一个强大的命令行工具，可用于在不同对象存储之间进行数据同步和复制。您可以从 [rclone 官方网站](#) 获取安装说明。
2. 配置 rclone，设置连接信息以连接到 Minio 存储。打开终端或命令提示符，并执行命令 `rclone config`，这将引导您完成配置过程。请根据提示，输入配置名称、Minio 存储类型以及访问密钥等信息。确保您已经正确设置了连接信息，并且可以成功连接到 Minio 存储桶。
3. 查看原始 Harbor 的 registry 服务所挂载的具体目录，这个目录存储着镜像文件。
4. 使用 rclone 进行数据迁移。在终端或命令提示符中执行以下命令

```
rclone copy <原始 Harbor 镜像存储目录> <rclone 配置名称>:<Minio 存储桶名称>/harbor/
images/
```

5. 请将尖括号中的参数替换为实际的值。例如，如果 rclone 配置名称为 minio，Minio 存储桶名称为 harbor-images，原始 Harbor 镜像存储目录为 /data/docker/registry/，则命令应为：

```
rclone copy /data/docker/registry/ minio:harbor-images/harbor/images/
```

!!! note

数据迁移过程可能需要较长时间，具体耗时取决于镜像文件的大小和网络速度。请耐心等待数据迁移完成。

## 步骤 3：创建新的 Harbor 实例

在这一步中，我们将基于新的 PostgreSQL 和 Minio 环境创建一个全新的 Harbor 实例，并确保 Harbor 在新集群环境中正常运行。

1. 在新的集群环境中，使用新的 PostgreSQL 和 Minio 连接信息创建一个全新的 Harbor 实例。您可以使用镜像仓库组件进行[实例创建](#)，需要注意的是，创建时填写账号密码需要和原始环境中的保持一致，同时需要选择新环境中的数据库实例以及 minio 存储。
2. 确保新的 Harbor 实例已成功安装并配置。
3. 在新的 Harbor 实例中验证镜像数据是否已经成功迁移至 Minio 存储。
4. 确保新的 Harbor 实例在新集群环境中正常运行，并能够访问并提供镜像服务。
5. 测试新的 Harbor 实例，验证迁移的完整性和正确性。您可以尝试上传、下载镜像，并检查日志和事件等信息，确保 Harbor 正常工作。
6. 如果一切运行正常，恭喜！您已经成功完成了 Harbor 迁移至新的集群环境，并且现在的 Harbor 实例正常运行在 Minio 存储系统上。

请注意，在进行实际的迁移和创建新的 Harbor 实例之前，请确保做好充分的备份并在测试环境中测试迁移过程，以确保数据的安全性和业务的连续性。

## 托管 Harbor 应该选择什么访问类型

在创建托管 Harbor 时，需要选择访问类型。目前支持两种类型：Ingress 和 NodePort。下表说明了这两种访问类型的优缺点；

Ingress

NodePort

## 优点

使用 Ingress 方便管理和使用，域名易记住，服务易迁移。

支持上传 HTTPS 证书，可以实现拉镜像不用配置非安全仓库。

不依赖任何组件，可快速启动服务进行 Demo 和试用。

NodePort 性能较高，没有额外的路由处理。

## 缺点

在私有网络环境需要配合 DNS 服务做域名解析，需要依赖基础网络设施。

Ingress 会对性能产生一定的损失，因为需要进行额外的路由处理。

管理不便，IP 和某一台宿主机强绑定，并且镜像地址变得不可迁移；

只能使用自签证书，在拉取镜像时需要配置非安全仓库。

### !!! tip

- 在 \*\*生产环境\*\* 中，建议使用 Ingress 方式，因为 Ingress 可以使用 HTTPS 协议，可以使用自签证书或者购买证书，这样可以避免在拉取镜像时需要配置非安全仓库的问题。
- 在 \*\*测试环境\*\* 中，建议使用 NodePort 方式，因为 NodePort 方式不依赖任何组件，可快速启动服务进行 Demo 和试用。

## Ingress

### 前提条件：

- 集群必须安装 Ingress 组件，Ingress 组件可以是 Nginx、Traefik、HAProxy 等，具体安装方法请参考 [Ingress](#)。
- 集群必须安装 DNS 服务，DNS 服务可以是 CoreDNS、KubeDNS、Bind 等，具体安装方法请参考 [DNS](#)。
- 集群必须安装 LoadBalancer 组件，LoadBalancer 组件可以是 MetalLB、F5 等，具体安装方法请参考 [LoadBalancer](#)。

### Ingress 流程

Ingress 可以从集群外部暴露 HTTP 和 HTTPS 路由到集群内部的服务，流量路由由 Ingress 资源上定义的规则控制。在托管 Harbor 中 Ingress 使用流程如下图所示。

flowchart LR

```

subgraph cluster[Cluster]
    direction LR
    ingress[Ingress] -->|路由规则| harbor[Harbor Service]
    harbor --> pod1[Pod]
    harbor --> pod2[Pod]
end

```

```

client([Client]) -.->|以 LB 或 NodePort 暴露服务| ingress

```

```

classDef plain fill: #ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill: #326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill: #fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

```

```

class client plain;
class ingress,harbor,pod1,pod2 k8s

```

对于 **私有云**，使用 Ingress 部署完托管 Harbor 之后，需要在 DNS 服务中添加域名解析，将域名解析到 LoadBalancer 的 IP 地址上，这样用户就可以通过域名访问 Harbor 了；添加 DNS 域名解析需要在 Kubernetes 集群内和 Kubernetes 集群外分别进行操作，具体操作步骤如下：

## 在 Kubernetes 集群内添加 DNS 域名解析

在 Kubernetes 集群内，需要在 CoreDNS 组件中添加 DNS 域名解析，具体操作步骤如下：

1. 进入 CoreDNS 组件的 ConfigMap 中，执行如下命令：

```
kubectl -n kube-system edit configmap coredns
```

2. 进入 CoreDNS 组件的 Configmap 中后，执行如下命令，将 harbor.example.com 解析到 LoadBalancer 的 IP 地址上：

```

hosts {
    10.1.1.1 harbor.example.com
}

```

```

        fallthrough
    }
!!! note
`harbor.example.com` 为用户自定义的域名, LoadBalancer 的 IP 地址为 Ingress 的 IP 地址。

```

3. 保存退出, 会重新创建 CoreDNS Pod:

```
kubectl get pod -n kube-system coredns-5c98db65d4-2t2l2
```

## 在 Kubernetes 集群外添加 DNS 域名解析

在集群外也是通过单独部署一个 CoreDNS 组件来实现 DNS 域名解析, 然后配置域名。参照上述第 2 步。

**如果用户的域名是公网域名, 那就不需要做上述步骤, 公网域名能被解析。**

## NodePort

NodePort 是一种访问 Kubernetes 集群中 Service 的方法, 它会在每个 Node 上打开一个端口, 该端口会将流量转发到 Service 的端口上。NodePort 使用流程如下图所示。

flowchart LR

```

subgraph cluster[Cluster]
    direction LR
    harbor[Harbor Service]
    harbor --> pod1[Pod]
    harbor --> pod2[Pod]
end

```

client([Client]) -.->|以 NodePort 暴露服务| harbor

```

classDef plain fill: #ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill: #326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill: #fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

```

```

class client plain;
class ingress,harbor,pod1,pod2 k8s

```

# 登录非安全镜像仓库

当要从非安全的镜像仓库中进行 Pull、Push 时，会遇到 x509: certificate signed by unknown authority 错误提示；这是由于镜像仓库可能是 http 服务，或者 https 的证书是自签名的就会出现这个问题。

我从如下三种运行时来说明如何解决这个问题；我们以要登录 test.registry.com 为例。

## Docker

在 /etc/docker/daemon.json 文件中加入如下配置：

```
{
  "insecure-registries": [
    "test.registry.com",
    "test.registry.com1"
  ]
}
```

修改之后重启 docker 即可，重启命令为：systemctl restart docker

## containerd

containerd 配置非安全仓库有两种方法，一种是从 1.4 版本之后引入的新方法，一种是之前存在直到 2.0 版本的旧方法。

## 新方法

编辑 /etc/containerd/config.toml 文件中的 config\_path 配置项，默认值是 /etc/containerd/certs.d。

```
[plugins."io.containerd.grpc.v1.cri".registry]
  config_path = "/etc/containerd/certs.d"
```

在 /etc/containerd/certs.d 目录下创建一个以 registry 命名的文件夹，并在其中创建一个

hosts.toml 的文件。

```
mkdir /etc/containerd/certs.d/test.registry.com
```

如果 registry 是带端口号的也需要运行：

```
mkdir /etc/containerd/certs.d/127.0.0.1: 5000
```

```
server = "http: //test.registry.com"
```

```
[host."http: //test.registry.com"]
```

```
capabilities = ["pull", "resolve", "push"]
```

```
skip_verify = true
```

配置后不用重启，可以直接使用。

## 旧方法

在 /etc/containerd/config.toml 文件中加入如下的配置：

```
[plugins."io.containerd.grpc.v1.cri".registry.configs."test.registry.com".tls]
```

```
insecure_skip_verify = true
```

配置之后需要重启 containerd，重启命令为：systemctl restart containerd。

## CRI-O

修改 /etc/crio/crio.conf 配置文件：

```
insecure_registries = ["test.registry.com"]
```

重启 crio: systemctl restart crio。

## Harbor Nginx 配置最佳实践

Harbor 内部涉及到 portal 和 core 两个服务对外暴露功能，并且控制流和数据流都经过 Nginx 组件，如果 Nginx 配置不合理就会出现镜像拉取失败、Helm repo 添加或者更新失败。

架构图



架构图

## 场景一：504 Gateway Timeout 报错

假如服务出现超时，有两种可能：第一种，本身很耗时，比如拉取大镜像；第二种，可能服务所在机器负载太高，响应很慢。

### pull/push 大镜像

Nginx proxy 后端服务默认超时时间是 60s，镜像 Pull 或者 Push 都需要经过 Nginx。如果遇到大镜像（比如机器学习类的）push 或者 pull 时间很长，则很容易触发超时时间。此时需要根据服务性能、网络情况来修复这个超时时间，可以在 ConfigMap 中增加下面两条 proxy 后重启 Pod 来解决，如将超时时间改为 15min。

举例：

- 1.在创建托管 harbor 的场景中，使用 Helm 模板部署 ingress-nginx。可以在 **容器管理**

-> **集群详情** -> **配置与密钥** -> **配置项** 中搜索找到 nginx-config。

配置项

配置项

- 2.进入详情，通过 **更新** 按钮，在如下位置增加 proxy 配置，将超时时间修改为 900s

(15min) ，保存后需要重启 Pod。

配置项详情

配置项详情

- 3.proxy 配置 demo。

```
location /v2/ {  
    proxy_send_timeout 900;
```

```
proxy_read_timeout 900;
}
```

## 服务所在机器负载太高

需要排查负载高的原因，再针对问题进行解决。

## 场景二：helm repo 操作失败

原因：Chartmuseum 服务针对每个 repo 有一个 index-cache.yaml 的文件，这个文件记录了这个仓库中所有 Helm Chart 的信息。假如达到了上万个 Helm Chart，会导致 index-cache.yaml 膨胀到几十 MB，在执行 helm repo add/update 时由于文件字节过大导致拉取这个文件失败。

解决办法：

- 增加超时时间和开启 gzip 压缩功能
- 设置 gzip\_types 类型只针对固定类型进行压缩
- 设置 gzip\_min\_length 在小于长度时不进行压缩

举例：

1. 在创建托管 harbor 的场景中，使用 Helm 模板部署 ingress-nginx。可以在 **容器管理**

-> **集群详情** -> **配置与密钥** -> **配置项** 中搜索找到 nginx-config。

配置项

配置项

2. 进入 config 详情，通过 **更新** 按钮，在如下位置增加 proxy 配置，保存后需要重启 Pod。

配置项详情

配置项详情

### 3.通过 Proxy 配置 demo:

```
location /chartrepo/ {  
    proxy_send_timeout 900;  
    proxy_read_timeout 900;  
    gzip on;  
    gzip_min_length 1000;  
    gzip_proxied expired no-cache no-store private auth;  
    gzip_types text/plain text/css application/json application/javascript application/x-javascript text/xml application/xml application/xml+rss text/javascript application/x-yaml text/x-yaml;  
}
```

## 镜像仓库容量资源规划

整个 Harbor 架构可以分为三层：Consumer、Service 和 Data Access Layer。

- Consumer：主要是 Harbor Portal、helm 工具、docker client 工具。
- Service：服务层主要是 Harbor 提供功能的核心服务，如 core、registry、jobserver。
- Data Access：主要是提供数据的持久化存储，比如镜像文件数据、镜像元数据。

### 资源架构

#### 资源架构

在容量规划上体现在服务层和存储层：

- 服务层：主要是服务运行的资源，比如 CPU、Memory
- 存储层：主要是镜像文件数据存储、元数据 DB 存储、缓存 Redis 存储

实际场景下推荐使用估算 + 验证的方式来确定资源规划是否合理。

## 举例

我们先按照存储 500Gi 的镜像数据来进行设置预估。

服务层

不同的资源配置能支撑的服务请求量不同，并且最少设置 2 个服务副本：

| 服务              | CPU | Memory |
|-----------------|-----|--------|
| Harbor 服务（所有服务） | 1 核 | 2 Gi   |

存储层

| 类型    | 存储   |
|-------|--|
| 镜像文件  | 文件系统：使用率不要超过 85%，如实际需要 500Gi 文件，则文件系统至少配置 588Gi 的存储。<br><a href="#">MinIO 对象存储</a> ：MinIO 的实际使用存储率和服务个数、纠删码配置有关。例如文件系统需要配置 588Gi 存储，使用率在 50%，则需要设置 1176Gi 的存储。 |
| DB 存储 | DB 存储：500Gi 的镜像数据，大概申请 50Gi 的 DB 空间即可。   |
| 缓存存储  | Redis 缓存：500Gi 的镜像数据，大概申请 5Gi 的缓存空间即可。   |

验证

我们可以通过压测工具验证资源规划是否满足实际应用场景。

若不确定当前配置是否满足实际应用场景，推荐使用如下压测工具进行验证。 压测工具为 [Harbor pref](#)。

```
git clone https://github.com/goharbor/perf
cd perf
export HARBOR_URL=https://admin:password@harbor.domain # 用户名密码和地址
export HARBOR_VUS=100 # 虚拟用户数
export HARBOR_ITERATIONS=200 # 每个虚拟用户数执行的次数
export HARBOR_REPORT=true # 是否生成报告
go run mage.go
```

## 补充说明

每个公司的镜像情况不一样, 并且层之间会复用, 还要考虑镜像 GC (不启用则不需要考虑), 镜像文件存储针对使用 MinIO 或者文件系统也不一样, 需合理规划资源。

GC (Garbage Collection, 垃圾收集) 是指原生 Harbor 提供的镜像清理能力, 当您从 Harbor 中删除镜像时, 空间不会自动释放。您必须运行垃圾收集, 通过从文件系统中删除清单不再引用的 blob 来释放空间。细节请参阅 [Garbage Collection](#)。

- 不启用 GC: 不断地往镜像仓库存储镜像, 这时不光要考虑现有镜像的存储, 还需要考虑增量以及增长速度。
- 定期 GC: 定期 GC 会缓解镜像仓库的存储压力, 但也要根据实际情况确认是否开启。

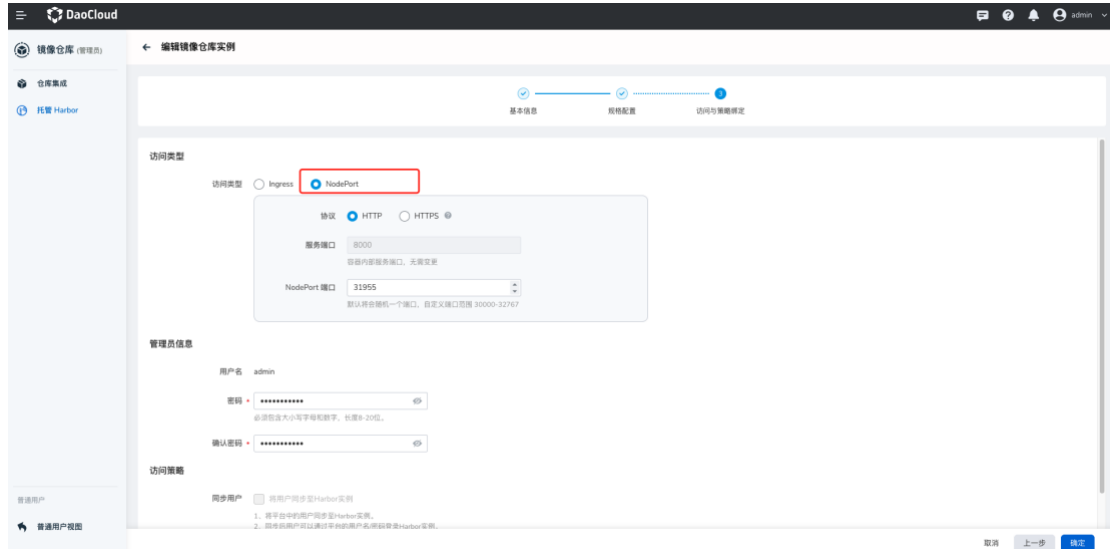
如需开启请前往使用原生 Harbor。

## 通过 LoadBalancer 模式部署 Harbor

目前 DCE 5.0 镜像仓库暂不支持使用 LoadBalancer 方式部署 Harbor, 仅支持使用 Ingress 和 NodePort 方式。本文简要说明如何手工修改访问类型为 LB。

### 1. 创建 NodePort 的 Harbor 服务

首先[创建托管 Harbor](#), 创建一个访问方式为 NodePort 的 Harbor 服务。



nodeport

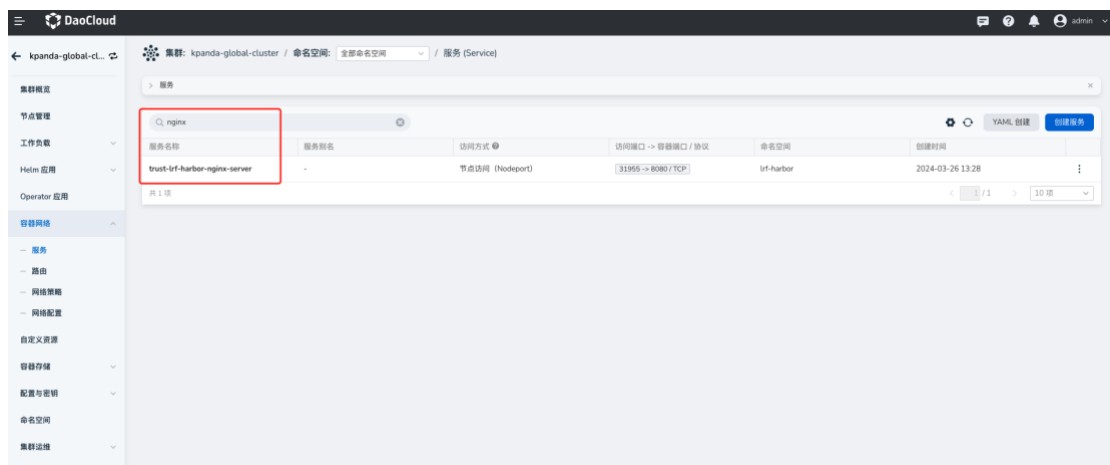
## 2. 创建 LoadBalancer 类型的 svc 资源

在 Harbor 所在集群的 **容器网络** -> **服务** 创建一个 LB 类型的 svc 资源。

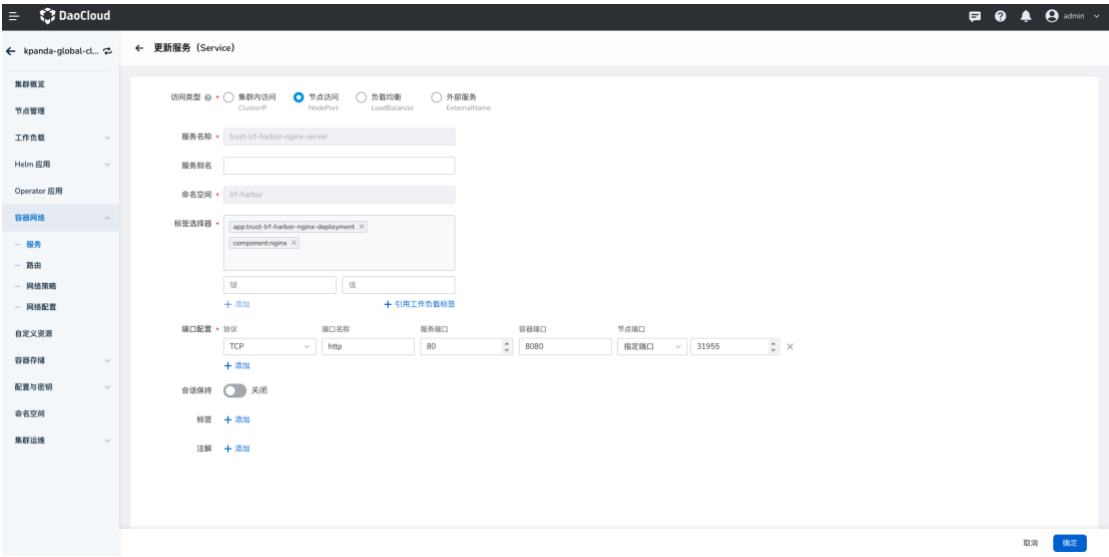
!!! note

创建 LB 类型的 svc 资源时, 标签需要和 nginx svc 的 label 保持一致, 且需要创建在同一个命名空间内。

Harbor 服务创建成功之后, Harbor 所在集群的 **容器网络** -> **服务** 会自动出现 nginx 的 svc 资源。在服务列表中, 点击右侧的 **!** 按钮, 在弹出菜单中选择 **更新**, 可查看 nginx 的 label 信息。



nginx



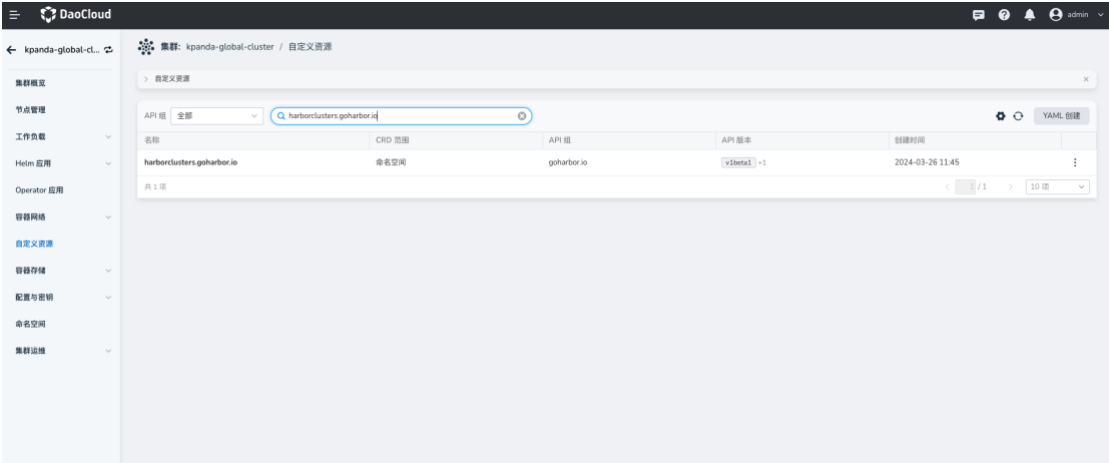
nginxlabel

### 3. 修改 harborclusters.goharbor.io CR

修改 harborclusters.goharbor.io 这个 CR 的 externalUrl 字段，改为 LB 的 IP 即可。

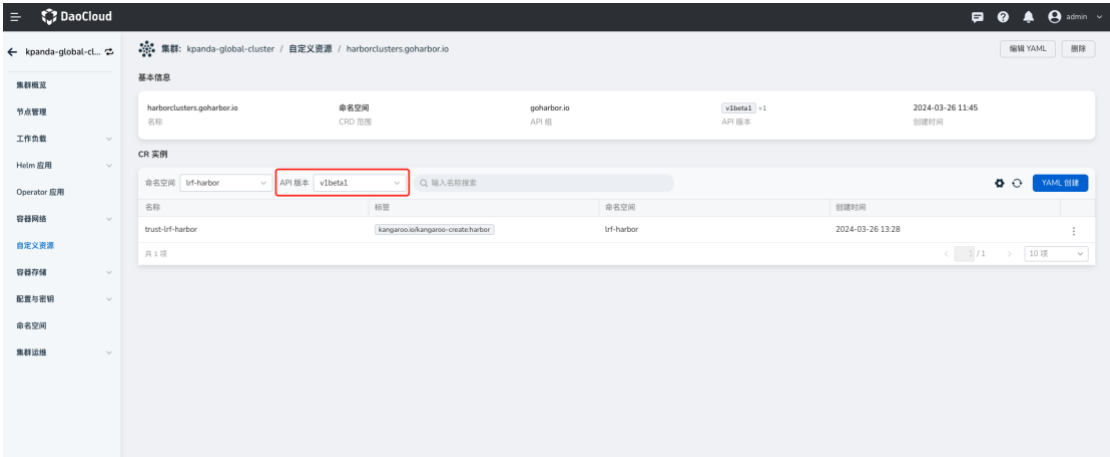
## 通过 UI 修改

- 1.在自定义资源列表，搜索 harborclusters.goharbor.io



crdlist

- 2.点击名称进入详情，选择 API 版本为 **v1beta1** ，修改 API 版本之后保存。



crddetail

# 通过命令行修改

在对应集群中运行以下命令：

```
kubectl edit harborclusters.goharbor.io
```