容器镜像服务(SWR)
25.3.0-HCS
用户指南

文档版本 02

发布日期 2025-04-15

华为云计算技术有限公司

商标声明


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

注意

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

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

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

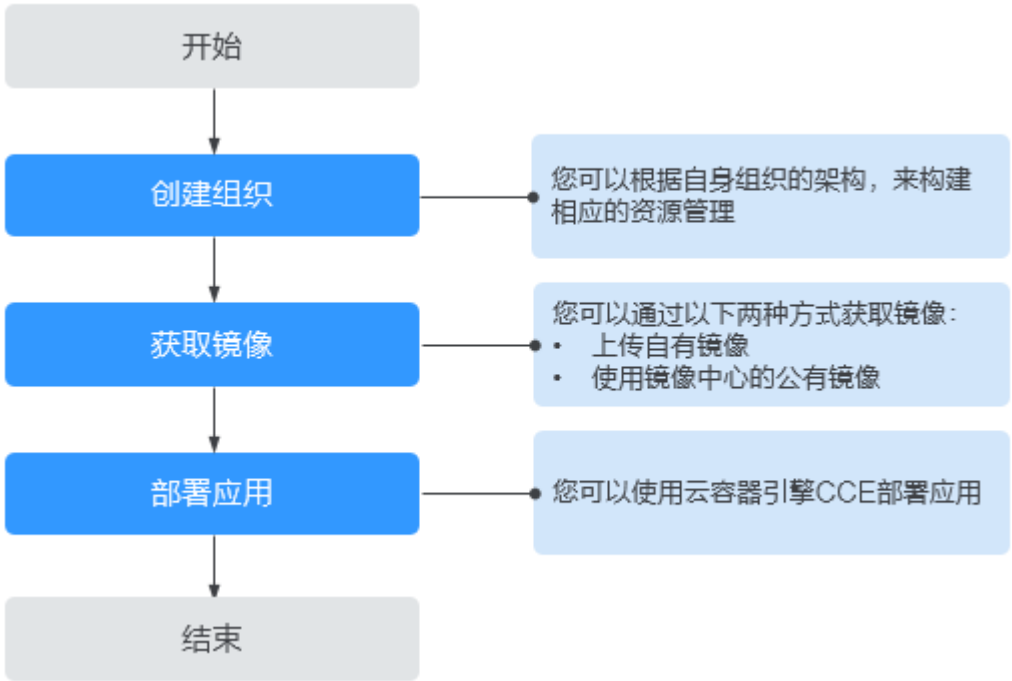
1 用户指南

1.1 欢迎使用容器镜像服务

容器镜像服务（SoftWare Repository for Container，简称SWR）是一种支持镜像全生命周期管理的服务，提供简单易用、安全可靠的镜像管理功能，帮助您快速部署容器化服务。

SWR提供私有镜像库，并支持细粒度的权限管理，可以为不同用户分配相应的访问权限（读取、编辑、管理）。

图 1-1 SWR 使用流程



1.2 容器引擎基础知识

容器引擎（即Docker）是一个开源的引擎，可以轻松的为任何应用创建一个轻量级的、可移植的、自给自足的容器。容器镜像服务兼容原生Docker，支持使用Docker CLI和API管理容器镜像。

安装 Docker

在安装Docker前，请了解Docker的基础知识，具体请参见**Docker Documentation**。

Docker几乎支持在所有操作系统上安装，用户可以根据需要选择要安装的Docker版本，具体请参见**https://docs.docker.com/engine/install/**。

说明

- 容器镜像的存储可以使用容器镜像服务（SWR），SWR支持Docker 1.11.2及以上版本容器引擎上传镜像，建议下载对应版本。
- 安装Docker需要连接互联网，内网服务器需要绑定弹性IP后才能访问。

另外，在Linux操作系统下，可以使用如下命令快速安装Docker。

```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
sudo systemctl daemon-reload
sudo systemctl restart docker
```

制作容器镜像

本节指导您通过Dockerfile定制一个简单的Web应用程序的容器镜像。Dockerfile是一个文本文件，其内包含了一条条的指令（Instruction），每一条指令构建一层，因此每一条指令的内容，就是描述该层应当如何构建。

使用Nginx镜像创建容器应用，在浏览器访问时则会看到默认的Nginx欢迎页面，本节以Nginx镜像为例，修改Nginx镜像的欢迎页面，定制一个新的镜像，将欢迎页面改为“Hello, SWR!”。

步骤1 以root用户登录Docker所在机器。

步骤2 创建一个名为Dockerfile的文件。

mkdir mynginx

cd mynginx

touch Dockerfile

步骤3 编辑Dockerfile。

vim Dockerfile

增加文件内容如下：

```
FROM nginx
RUN echo '<h1>Hello,SWR!</h1>' > /usr/share/nginx/html/index.html
```

Dockerfile指令介绍如下。

- FROM语句：表示使用nginx镜像作为基础镜像，一个Dockerfile中FROM是必备的指令，并且必须是第一条指令。

- RUN语句：格式为RUN <命令>，表示执行echo命令，在显示器中显示一段“Hello, SWR!”的文字。

保存并退出。

步骤4 使用**docker build** [*选项*] <*上下文路径*> 构建镜像。

docker build -t nginx:v1 .

- -t nginx:v1：指定镜像的名称和版本。
- **.**：指定Dockerfile所在目录，镜像构建命令将该路径下所有的内容打包给Docker帮助构建镜像。

步骤5 执行以下命令，可查看到已成功部署的nginx镜像，版本为v1。

docker images

----结束

制作镜像压缩包

本节指导您将容器镜像制作成tar或tar.gz文件压缩包。

步骤1 以root用户登录Docker所在机器。

步骤2 执行如下命令查看镜像。

docker images

查看需要导出的镜像及tag。

步骤3 执行如下命令制作镜像压缩包。

docker save [OPTIONS] IMAGE [IMAGE...]

说明

OPTIONS：--output或-o，表示导出到文件。

压缩包格式为：.tar或.tar.gz。

使用docker save制作镜像压缩包时，请用{image}:{tag}，不要用image id，否则无法在swr页面上传。

示例：

```
$ docker save nginx:latest > nginx.tar
$ ls -sh nginx.tar
108M nginx.tar

$ docker save php:5-apache > php.tar.gz
$ ls -sh php.tar.gz
372M php.tar.gz

$ docker save --output nginx.tar nginx
$ ls -sh nginx.tar
108M nginx.tar

$ docker save -o nginx-all.tar nginx
$ docker save -o nginx-latest.tar nginx:latest
```

----结束

导入镜像文件

本章节将指导你通过docker load命令将镜像压缩包导入为一个镜像。

执行方式有2种：

docker load < 路径/文件名.tar

docker load --input或者-i 路径/文件名.tar

示例：

docker load --input fedora.tar

1.3 访问与使用

步骤1 使用浏览器，以VDC管理员账号登录ManageOne。

- 登录地址：https://*ManageOne运营面的访问地址*。例如，https://console.demo.com。
- 账号密码：VDC管理员对应账号和密码。

步骤2 单击页面左上角的图标，选择区域和资源空间，然后在服务列表中选择“应用服务 > 容器镜像服务SWR”。

步骤3 进入容器镜像服务页面。

----结束

说明

第一次进入前端需要在IAM中被授予SWR Administrator或Tenant Administrator策略才可以进入。

1.4 镜像管理

1.4.1 客户端上传镜像

操作场景

客户端上传镜像，是指在安装了容器引擎客户端的机器上使用docker命令或者ctr命令将镜像上传到容器镜像服务的镜像仓库。如果是**docker**容器引擎客户端则使用docker push命令上传。如果是**containerd**容器引擎客户端则使用ctr push命令上传。

约束与限制

- 使用客户端上传镜像，镜像的每个layer大小不能超过10G。
- 上传镜像的Docker容器引擎客户端必须为1.11.2或以上版本。

前提条件

已创建组织，请参见**创建组织**。

docker 容器引擎客户端

步骤1 制作容器镜像或导入镜像文件。

步骤2 连接容器镜像服务。

1. 登录容器镜像服务控制台。
2. 选择左侧导航栏的“总览”，单击页面右上角的“登录指令”，在弹出的页面中单击复制登录指令。

   说明

   - 此处生成的登录指令有效期为6小时，且每次获取都会刷新。若需要长期有效的登录指令，请参见**获取长期有效登录指令**。获取了长期有效的登录指令后，在有效期内的临时登录指令仍然可以使用。
   - 登录指令末尾的域名为镜像仓库地址，请记录该地址，后面会使用到。
3. 在安装Docker的机器中执行上一步复制的登录指令。

   登录成功会显示“Login Succeeded”。

步骤3 在安装Docker的机器上执行以下命令，为**nginx**镜像打标签。

docker tag [镜像名称1:版本名称1] [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

其中，

- [镜像名称1:版本名称1]：请替换为您所要上传的实际镜像的名称和版本名称。
- [镜像仓库地址]：可在SWR控制台上查询，即**步骤2.2**中登录指令末尾的域名。
- [组织名称]：请替换为您创建的组织。
- [镜像名称2:版本名称2]：请替换为您期待的镜像名称和镜像版本。

示例：

docker tag nginx:v1 {*Image repository address***}/group/nginx:v1**

步骤4 上传镜像至镜像仓库。

docker push [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

示例：

docker push {*Image repository address***}/group/nginx:v1**

终端显示如下信息，表明上传镜像成功。

```
6d6b9812c8ae: Pushed
695da0025de6: Pushed
fe4c16cbf7a4: Pushed
v1: digest: sha256:eb7e3bbd8e3040efa71d9c2cacfa12a8e39c6b2ccd15eac12bdc49e0b66cee63 size: 948
```

返回容器镜像服务控制台，在“我的镜像”页面，执行刷新操作后可查看到对应的镜像信息。

----结束

containerd 容器引擎客户端

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“我的镜像”，单击右侧镜像名称。

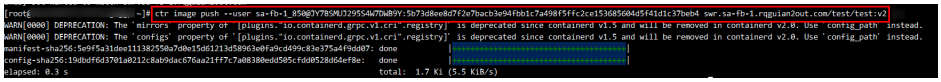
步骤3 在镜像详情页面中，进入“Pull/Push指南”页签，复制containerd容器引擎的镜像上传指令。

说明

该指令将于6个小时后过期。如果想获取长期上传指令请参考**获取containerd容器引擎的长期拉取、推送镜像指令**。

步骤4 以root用户登录containerd引擎所在的虚拟机。

步骤5 在虚拟机中执行**3**复制的镜像上传指令，请修改版本名称为要上传镜像的。



步骤6 检查镜像是否上传成功。

说明

上传时如果遇到上传失败，报错：unexpected status：401 Unauthorized，请参考《**容器镜像服务(SWR) 25.3.0-HCS 维护指南(for 华为云Stack 8.5.1)**》里面的子手册《容器镜像服务(SWR) 25.3.0-HCS 故障管理(for 华为云Stack 8.5.1)》的章节《通过客户端上传失败报错：unexpected status：401 Unauthorized》进行排查解决。

----结束

1.4.2 获取长期有效登录指令

操作场景

本章节介绍如何获取docker容器引擎长期有效的登录指令以及containerd容器引擎长期推送、拉取镜像指令，长期有效访问指令的有效期为永久。

说明

为保证安全，获取登录指令过程建议在开发环境执行。

获取 docker 容器引擎的长期有效登录指令

步骤1 获取区域资源空间名称、镜像仓库地址。

1. 登录管理控制台，单击右上角您的用户名处，单击“个人设置”。
2. 在“资源空间列表”页签中查找当前区域对应的资源空间。
3. 参考**步骤2.2**获取镜像仓库地址，登录指令末尾的域名即为镜像仓库地址。

步骤2 获取AK/SK访问密钥。

说明

访问密钥即AK/SK（Access Key ID/Secret Access Key），表示一组密钥对，用于验证调用API发起请求的访问者身份，与密码的功能相似。如果您已有AK/SK，可以直接使用，无需再次获取。

1. 登录管理控制台，单击右上角您的用户名处，单击“个人设置”。
2. 在“个人设置”页面，单击“管理访问密钥”页签。
3. 单击“新增访问密钥”，新建AK/SK。
4. 单击“确定”，自动下载访问密钥。

   下载成功后，在“credentials”文件中即可获取AK和SK信息。

须知

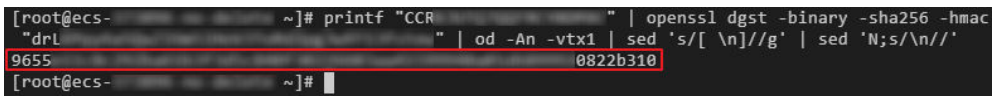
- 每个用户仅允许新增两个访问密钥。
- 为保证访问密钥的安全，访问密钥仅在初次生成时自动下载，后续不可再次通过管理控制台界面获取。请在生成后妥善保管。

步骤3 登录一台linux系统的计算机，执行如下命令获取登录密钥。

printf "*$AK*" | openssl dgst -binary -sha256 -hmac "*$SK*" | od -An -vtx1 | sed 's/[\n]//g' | sed 'N;s/\n//'

其中$AK和$SK为步骤2获取的AK/SK。

图 1-2 示例

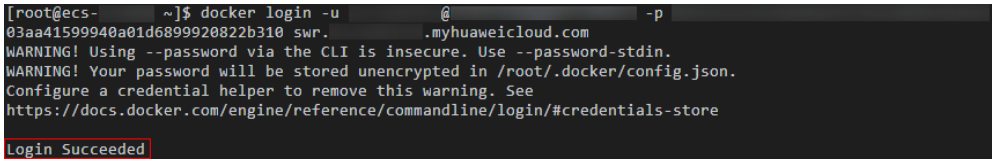


步骤4 使用如下的格式拼接登录指令。

docker login -u [*资源空间名*]@[*AK*] **-p** [*登录密钥*] [*镜像仓库地址*]

其中，资源空间名和镜像仓库地址在步骤1中获取，AK在步骤2中获取，登录密钥为步骤3的执行结果。

图 1-3 长期登录指令



说明

- 登录密钥字符串是经过加密的，无法逆向解密，从-p无法获取到SK。
- 获取的登录指令可在其他机器上使用并登录。

步骤5 使用**history -c**命令清理相关使用痕迹，避免隐私信息泄露。

----结束

获取 containerd 容器引擎的长期拉取、推送镜像指令

步骤1 参考**获取docker容器引擎的长期有效登录指令**中的**步骤1**至**步骤3**获取资源空间名、镜像仓库地址、AK以及登录密钥信息。

步骤2 使用如下的格式拼接长期拉取和推送镜像的指令。

1. 镜像拉取指令拼接

ctr image pull --user [*资源空间名*]@[*AK*]:[*登录密钥*] [*镜像仓库地址*]

其中，资源空间名和镜像仓库地址在**步骤1**中获取，AK在**步骤2**中获取，登录密钥为**步骤3**的执行结果。

2. 镜像推送指令拼接

ctr image push --user [*资源空间名*]@[*AK*]:[*登录密钥*] [*镜像仓库地址*]

其中，资源空间名和镜像仓库地址在**步骤1**中获取，AK在**步骤2**中获取，登录密钥为**步骤3**的执行结果。

说明

- 登录密钥字符串是经过加密的，无法将登录密钥字符串逆向解密成SK。
- 获取的指令可在其他机器上使用并进行镜像上传下载。

----结束

常见问题

登录时如果遇到报错“x509: certificate sigined by unknown authority”，请参考**x509: certificate signed by unknown authority**进行处理。

如果遇到“x509: certificate has expired or is not yet valid”，请参考**x509: certificate has expired or is not yet valid**进行处理。

1.4.3 页面上传镜像

操作场景

本章节介绍如何通过页面上传镜像。从页面上传镜像，是指直接通过控制台页面将镜像上传到容器镜像服务的镜像仓库。

约束与限制

- 每次最多上传10个文件，单个文件大小（含解压后）不得超过2G。
- 仅支持上传1.11.2 及以上版本Docker容器引擎客户端制作的镜像压缩包。

前提条件

- 已创建组织，请参见**创建组织**。
- 镜像已存为tar或tar.gz文件，具体请参见**制作镜像压缩包**。

操作步骤

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“我的镜像”，单击右上角“页面上传”。

步骤3 在弹出的窗口中选择组织，单击“选择镜像文件”，选择要上传的镜像文件。

说明

多个镜像同时上传时，镜像文件会按照顺序逐个上传，不支持并发上传。

图 1-4 上传镜像

步骤4 单击“开始上传”。

待任务进度显示“上传完成”，表示镜像上传成功。

----结束

1.4.4 下载镜像

操作场景

您可以使用docker容器引擎也可以使用containerd容器引擎下载容器镜像服务中的镜像。

docker 容器引擎

步骤1 以root用户登录容器引擎所在的虚拟机。

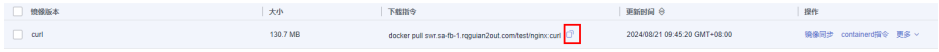
步骤2 参考步骤2获取登录访问权限，连接容器镜像服务。

步骤3 登录容器镜像服务控制台。

步骤4 在左侧导航栏选择“我的镜像”，单击右侧镜像名称。

步骤5 在镜像详情页面中，单击对应镜像版本“下载指令”列的复制图标，复制镜像下载指令。

图 1-5 获取镜像下载指令



步骤6 在虚拟机中执行**步骤5**复制的镜像下载指令。

使用**docker images**命令查看是否下载成功。

```
# docker images
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
xxx/group/nginx               v2.0.0    22f2bf2e2b4f   5 hours ago     22.8MB
```

步骤7 （可选）执行如下命令将镜像保存为归档文件。

docker save [*镜像名称:版本名称*] > [*归档文件名称*]

----结束

containerd 容器引擎

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“我的镜像”，单击右侧镜像名称。

步骤3 在镜像详情页面中，复制操作列的“containerd指令”或者进入“Pull/Push指南”页签，复制containerd容器引擎的镜像下载指令。

说明

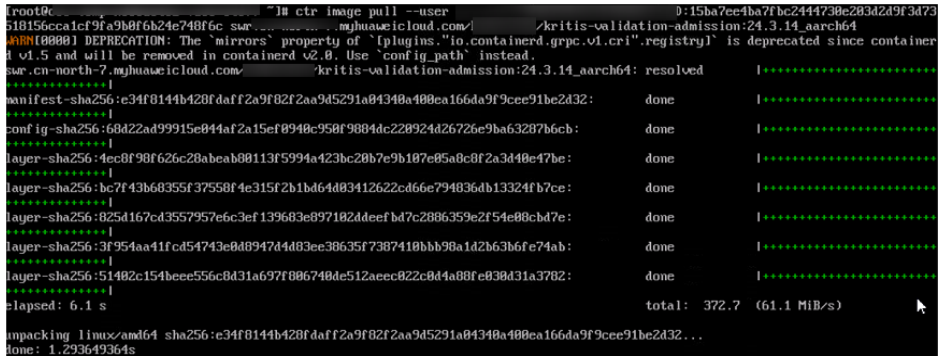
该指令将于6个小时后过期。如果想获取长期下载指令请参考**获取containerd容器引擎的长期拉取、推送镜像指令**。

步骤4 以root用户登录containerd引擎所在的虚拟机。

步骤5 在虚拟机中执行**3**复制的镜像下载指令。

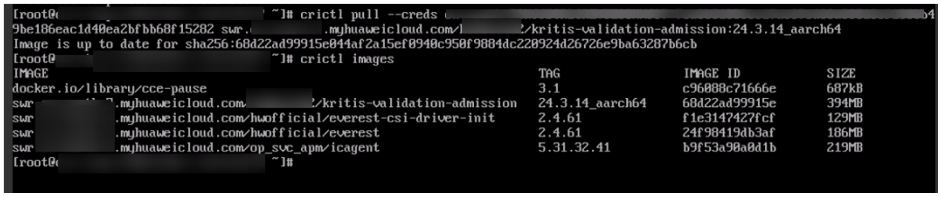
- 复制操作列的“containerd指令”的场景下执行如下：

- 复制“Pull/Push指南”页签containerd容器下载指令的场景下请修改版本名称为要上传镜像的再执行。

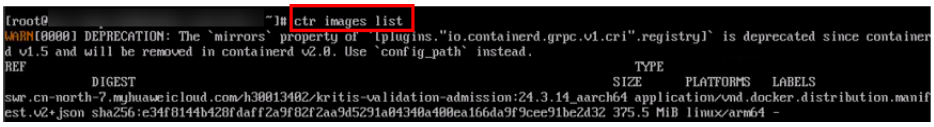


步骤6 查看镜像是否下载成功。

- 复制操作列的“containerd指令”的场景下使用**crictl images** 命令查看是否下载成功。



- 复制“Pull/Push指南”页签containerd容器下载指令的场景下使用**ctr images list**命令查看是否下载成功。



----结束

1.4.5 编辑镜像属性

操作场景

镜像上传后默认为私有镜像，您可以设置镜像的属性，包括镜像的类型（“公开”或“私有”）、类别及描述。

公开镜像所有用户都能下载，私有镜像则受具体权限管理控制。您可以为用户添加授权，授权完成后，用户享有读取、编辑或管理私有镜像的权限，具体请参见**在镜像详情中添加授权**。

操作步骤

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“我的镜像”，单击右侧要编辑镜像的名称。

步骤3 在镜像详情页面，单击右上角“编辑”，在弹出的窗口中根据需要编辑类型（“公开”或“私有”）、类别及描述，然后单击“确定”。

表 1-1 编辑镜像

参数	说明
所属组织	镜像所属组织。
镜像名称	镜像名称。
类型	镜像类型，可选择： • 公开 • 私有 **说明** 公开镜像所有用户都可以下载使用。 • 如果您的机器与镜像仓库在同一区域，访问仓库是通过内网访问。 • 如果您的机器与镜像仓库在不同区域，通过公网才能访问仓库，下载跨区域仓库的镜像需要节点可以访问公网。

参数	说明
类别	镜像分类，可选择： ● 应用服务器 ● Linux ● Windows ● Arm ● 框架与应用 ● 数据库 ● 语言 ● 其他
描述	输入镜像仓库描述，0-30000个字符。

----结束

1.4.6 共享私有镜像

操作场景

镜像上传后，您可以共享**私有镜像**给其他租户，并授予下载该镜像的权限。

租户下的用户需要登录容器镜像服务控制台，在“我的镜像 > 他人共享”页面查看共享的镜像。单击镜像名称，可进入镜像详情页面查看镜像版本、下载指令等。

约束与限制

- 镜像共享功能只支持私有镜像进行共享，不支持公有镜像共享。
- 仅具备该私有镜像管理权限的用户才能共享镜像，被共享者只有只读权限，只能下载镜像。
- 镜像共享功能只能在同一区域内使用，不支持在不同区域间镜像共享。
- 一个私有镜像最多可以共享给300个租户。
- 租户A销户后，如果镜像X曾被共享给租户A，VDC管理员需要在镜像X详情页的共享页签下手动删除已销户的租户记录。否则其仍占用一个配额。

操作步骤

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“我的镜像”，单击右侧镜像的名称。

步骤3 在镜像详情页面选择“共享”页签。

步骤4 单击“共享镜像”，根据表1-2填写相关参数，然后单击“确定”。

表 1-2 共享镜像

参数	说明
共享给	输入租户名称。
截止日期	选择共享截止日期。如勾选“永久有效”，则共享永久有效。
权限	当前仅支持“下载”权限。
描述	输入描述，0-1000个字符。

步骤5 共享完成后，您可以在“我的镜像 > 自有镜像”中，勾选“我共享的镜像”，查看所有共享的镜像。

----结束

1.4.7 添加触发器

操作场景

容器镜像服务可搭配云容器引擎CCE一起使用，实现镜像版本更新时自动更新使用该镜像的应用。您只需要为镜像添加一个触发器。通过触发器，可以在每次生成新的镜像版本时，自动执行更新动作，如：自动更新使用该镜像的应用。

前提条件

更新应用镜像版本之前，请确保已创建容器应用，将镜像部署到云容器引擎CCE。

如未创建，请登录云容器引擎工作负载页面进行创建。

用户在IAM中被授予SWR Administrator或Tenant Administrator策略才有创建触发器的权限。

操作步骤

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“我的镜像”，单击右侧镜像名称，进入镜像详情页。

步骤3 选择“触发器”页签，单击“添加触发器”，根据表1-3填写相关参数，然后单击“确定”。

表 1-3 触发器

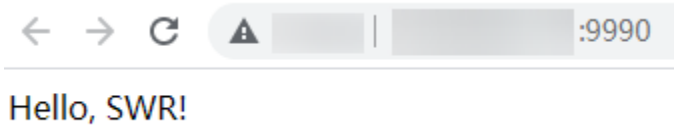
参数	说明
触发器名称	字母开头，由字母、数字、下划线_、中划线-组成，下划线、中划线不能连续且不能作为结尾，1-64个字符。

参数	说明
触发条件	支持如下三种触发条件，当镜像有新版本时，触发部署应用。 ● 全部触发：有新的镜像版本生成或镜像版本发生更新时，触发部署。 ● 指定版本号触发：有指定镜像版本生成或更新时，触发部署。 ● 正则触发：有符合正则表达式的镜像版本生成或更新时，触发部署。正则表达式规则如下： – *****：匹配不包含路径分隔符“/”的任何字段。 – ******：匹配包含路径分隔符“/”的任何字段。 – **?**：匹配任何单个非“/”的字符。 – **{选项1, 选项2, ...}**：同时匹配多个选项。
触发动作	当前仅支持更新容器的镜像，需指定更新的应用，以及该应用下的指定容器镜像。
触发器状态	选择“启用”。
触发器类型	选择“云容器引擎CCE”。
选择应用	选择要更新镜像的容器。

----结束

示例

假设有一个欢迎页面为“Hello, SWR!”的Nginx镜像（版本号为v1），使用该镜像创建了名称为“nginx”的无状态负载，该负载提供对外访问。



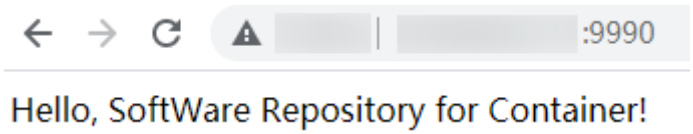
1. 为Nginx镜像添加触发器。

   触发器名称填写“All_tags”，触发条件选择“全部触发”，选择使用了Nginx镜像的无状态负载及容器。

2. Nginx镜像新增一个v2版本，该版本的欢迎页面为“Hello, SoftWare Repository for Container!”。
3. 确认是否触发成功。

   在“触发器”页签，单击 图标，查看触发结果为“成功”。

   工作负载的访问页面已变更为“Hello, SoftWare Repository for Container!”。



1.4.8 添加镜像老化规则

操作场景

镜像上传后，您可以添加镜像老化规则。容器镜像服务提供了如下两种类型的镜像老化处理规则，规则设置完成后，系统会根据已定义的规则自动执行镜像老化操作。

- 存活时间：设置该类型的老化规则后，留存时间超过指定时间的老旧镜像将被删除。
- 版本数目：设置该类型的老化规则后，留存镜像超过指定值时，老旧镜像将被删除。

此外，对于特定版本的镜像可通过添加过滤策略来保留，免受老化规则的影响。

约束与限制

一个镜像仅支持添加一个老化规则。如需添加新的老化规则，需要删除已有老化规则。

操作步骤

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“我的镜像”，单击右侧镜像名称，进入镜像详情页。

步骤3 选择“镜像老化”页签，单击“+”，根据表1-4填写相关参数，然后单击“确定”。

表 1-4 添加镜像老化规则

参数	说明
规则类型	分为存活时间和版本数目。 ● 存活时间：设置该类型的老化规则后，留存时间超过指定时间的老旧镜像将被删除。 ● 版本数目：设置该类型的老化规则后，留存镜像超过指定值时，老旧镜像将被删除。
保留天数	镜像留存的最大天数，可设置为1~365的整数。规则类型设置为“存活时间”时，需要配置此参数。
保留数目	镜像留存的最大数目，可设置为1~1000的整数。规则类型设置为“版本数目”时，需要配置此参数。
过滤标签	输入将被过滤的镜像版本，在应用老化规则前指定版本的镜像将被过滤掉。
过滤正则	输入将被过滤的版本正则式，在应用老化规则前所有版本号满足正则表达式的镜像将被过滤掉。

镜像老化规则添加成功后，系统会立即进行一次查询，清理掉符合老化规则的镜像，且在“老化日志”中显示清理结果。

----结束

1.4.9 镜像自动同步

操作场景

镜像上传后，您可以添加镜像自动同步功能，帮助您把最新推送的镜像自动同步到其他区域镜像仓库内。

说明

镜像自动同步帮助您把最新推送的镜像自动同步到其他区域镜像仓库内，后期镜像有更新时，目标仓库的镜像也会自动更新，但已有的镜像不会自动同步。

约束与限制

在IAM中被授予VDC管理员级别的Tenant Administrator权限才能同步镜像或者配置镜像同步策略。

操作步骤

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“我的镜像”，单击右侧镜像名称。

步骤3 在镜像详情页面单击右上角“镜像自动同步”，添加镜像同步的目标区域和目标组织，添加完成后单击“确定”。

- 目标区域：选择同步的目标区域。
- 目标组织：选择同步的目标组织。
- 覆盖镜像（可选）：

  勾选则表示覆盖，同步相同名称相同版本的镜像时，同步后会替换已有的镜像版本。

  不勾选则表示不覆盖，同步相同名称相同版本的镜像时，会取消同步并提示已存在相同版本镜像。

步骤4 在镜像详情页面的“镜像同步记录”页签下，可查看镜像同步启动时间、镜像版本、状态、同步类型、同步耗时等。

表 1-5 镜像同步失败问题汇总

状态	可能原因	解决方案
失败	区域间管理面节点网络问题，导致镜像同步失败	• 请联系运维工程师确认网络是否存在异常 • 在目标组织清理无用的镜像或者扩大存储空间配额，之后重试 • 请稍后重试
失败：同步超时	区域间管理面节点网络问题，导致镜像同步超时	• 请联系运维工程师确认网络是否存在异常 • 请稍后重试

状态	可能原因	解决方案
失败：镜像已存在	镜像同步时“覆盖镜像”选项未勾选，而目标区域和组织已经存在同名的镜像	• 如果无需覆盖，无需处理，忽略此条记录即可 • 如果需要覆盖，需要删除同步规则，重新创建一个勾选“覆盖镜像”的同步规则

----结束

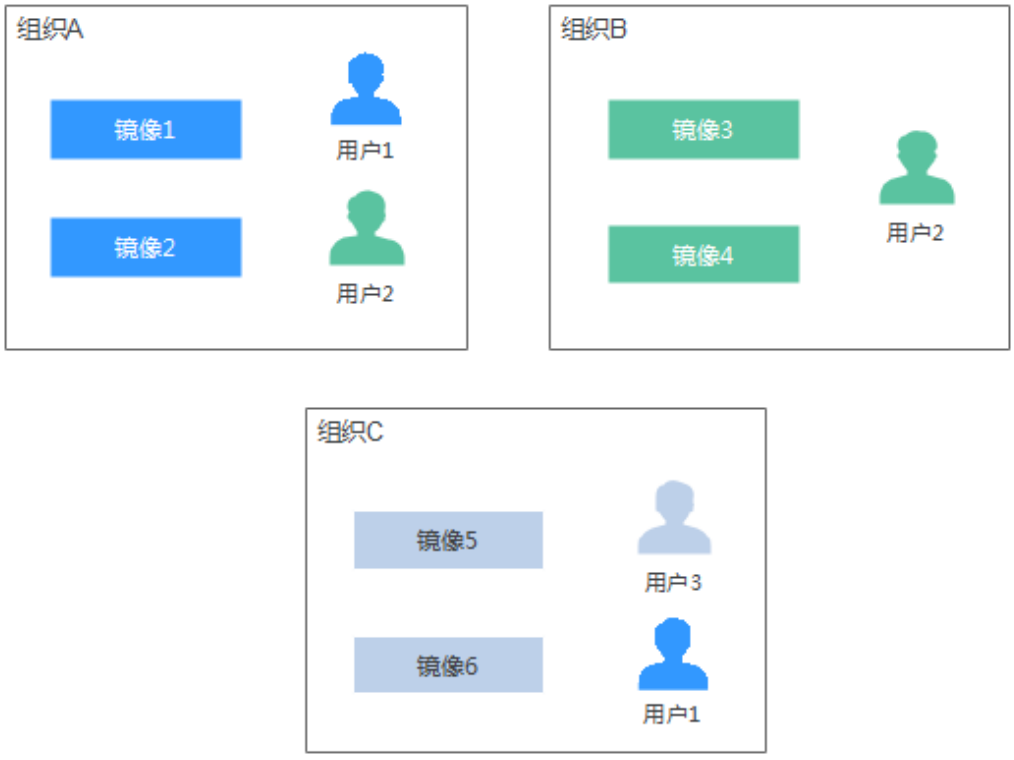
1.5 组织管理

操作场景

组织用于隔离镜像仓库，每个组织可对应一个公司或部门，将其拥有的镜像集中在该组织下。在不同的组织下，可以有同名的镜像。同一用户可属于不同的组织，如图1-6所示。

SWR支持为用户分配相应的访问权限（读取、编辑、管理），具体请参见授权管理。

图 1-6 组织



约束与限制

- 只有角色在IAM中被授予“Tenant Administrator”或“SWR Administrator”策略的用户才能创建组织。

- 不允许创建名称为library的组织，该组织为系统预留，请更换一个其他的名称。

创建组织

容器镜像服务为您提供组织管理功能，方便您根据自身组织架构来构建镜像的资源管理。上传镜像前，请先创建组织。

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“组织管理”，单击右上角“创建组织”，在弹出的页面中填写“组织名称”，然后单击“确定”。

说明

- 组织名称全局唯一，创建组织时如果提示组织已存在，可能该组织名称已被其他用户使用，请重新设置一个组织名称。
- 创建组织时，如系统提示组织已存在，可能是删除租户后， 组织资源存在残留，建议您重新设置一个组织名称。
- 单租户可创建的组织数量为1000个。

----结束

查看组织中的镜像

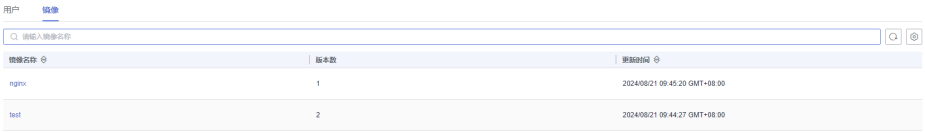
创建组织后，您可以查看当前组织中的镜像。

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“组织管理”，单击右侧组织名称。

步骤3 单击“镜像”页签，查看当前组织中的镜像。

图 1-7 查看组织中的镜像



----结束

删除组织

删除组织前，请先删除组织下的所有镜像。

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“组织管理”，单击待删除组织右上角的图标  然后单击“是”。

----结束

须知

如果需要删除租户，需要先删除组织，直接删除租户会导致其下组织无法清理，否则再次创建同名组织时系统会提示已存在。

设置组织存储空间

为了解决上传镜像没有限制，可能会上传大量无用或者使用率低镜像，浪费SWR空间资源。SWR提供了设置组织(Namespace)下的存储容量配额的功能，您可以根据需要对各个组织进行存储容量的设置。

- 如果您具备swr:namespace:updateNamespaceStorageQuota Action的权限或者有组织管理权限时，可在组织管理页面查看已使用的存储空间和剩余可用存储空间和设置可用存储空间。
- 如果您仅有swr:namespace:getNamespaceStorageQuota Action的权限或者只有组织的读取和编辑权限时只能查看不能设置存储空间。

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“组织管理”，单击待设置组织的组织名称进入组织详情页面，单击设置图标，设置可用存储空间的值，单位为GB。

⚠ 注意

- 该特性首次上线会计算存量组织的已用存储空间，所以会有一段时间无法展示和设置存储空间。该时间依据您的存量数据的数据规模确定，请耐心等待。存量组织处理完后默认没有存储空间配额的限制。
- 当某个组织下存储容量超过配额上限，导致无法上传镜像时，可以通过临时调大组织存储容量配额解决。
- 存在极低概率出现超配额使用情况：当某个组织的存储空间即将达到配额前，如果有多个并发同时上传镜像到该组织时，可能会超过配额，后续上传镜像会被正常拦截。

----结束

1.6 授权管理

操作场景

VDC管理员和VDC业务员都具有Tenant Administrator系统权限，即拥有了SWR的管理员权限，可以管理SWR中的所有组织和所有镜像。

VDC只读管理员如果要查看对应的组织和镜像，需要VDC管理员或VDC业务员为VDC只读管理员授予读取权限。VDC只读管理员当前支持创建组织、在组织中添加授权等功能，但鉴于VDC只读管理员的角色定位，不建议您使用这些功能。

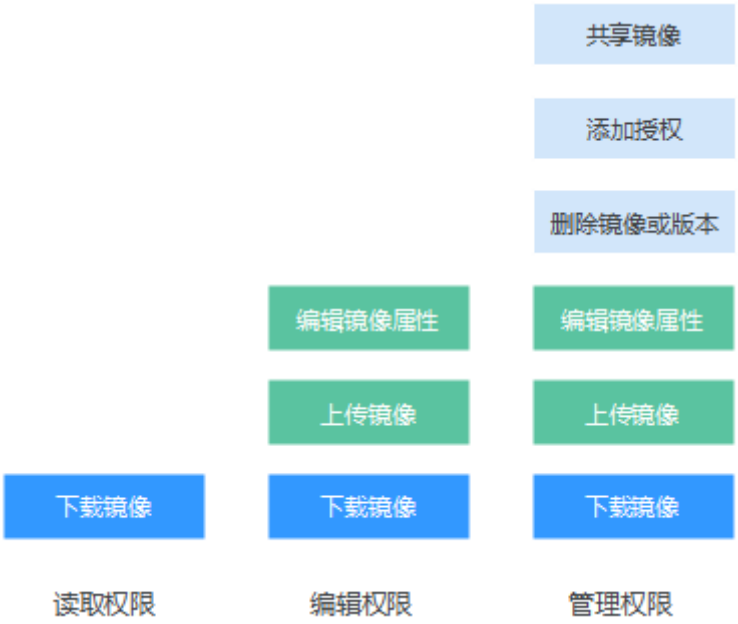
如果VDC管理员或VDC业务员需要给某用户授予对某个镜像的权限或对某个组织中所有镜像的权限，可以添加一个自定义用户组，并且在项目资源空间中添加ServiceStage Developer或SWR Developer权限，然后将该用户添加到自定义用户组。

授权方法

容器镜像服务中给用户添加权限有如下两种方法：

- **在镜像详情中添加授权**，授权完成后，用户享有读取/编辑/管理该镜像的权限。
- **在组织中添加授权**，使用户对组织内所有镜像享有读取/编辑/管理的权限。

图 1-8 用户权限



容器镜像服务中为用户添加的权限有如下三种类型：

- 读取：只能下载镜像，不能上传。
- 编辑：下载镜像、上传镜像、编辑镜像属性。
- 管理：下载镜像、上传镜像、删除镜像或版本、编辑镜像属性、添加授权以及共享镜像。

说明

页面上传镜像功能要求具备组织的编辑或管理权限，在镜像详情中添加的编辑或管理权限不支持页面上传镜像。

在镜像详情中添加授权

在镜像详情中为用户添加授权，授权完成后，该用户享有读取/编辑/管理该镜像的权限。

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“我的镜像”，单击右侧要编辑镜像的名称。

步骤3 在镜像详情页面选择“权限管理”页签。

步骤4 单击“添加授权”，选择用户名称，添加“读取/编辑/管理”的权限，添加后，该用户享有对应权限。

图 1-9 在镜像详情中添加授权

----结束

在镜像详情中修改/删除授权

您还可以在镜像详情中修改用户权限及删除用户权限。

- 修改授权：在“权限管理”页签下用户所在行单击“编辑”，在“权限”所在列选择新的权限，然后单击“保存”。
- 删除授权：在“权限管理”页签下用户所在行单击“删除”，然后单击“确定”。

在组织中添加授权

在组织中为用户添加授权，使用户对组织内所有镜像享有读取/编辑/管理的权限。

只有具备“管理”权限的用户才能添加授权。

步骤1 登录容器镜像服务控制台。

步骤2 在左侧导航栏选择“组织管理”，单击组织名称。

步骤3 在“用户”页签下单击“添加授权”，在弹出的窗口中为用户选择权限，然后单击“确定”。

----结束

在组织中修改/删除授权

您还可以在组织中修改用户权限及删除用户权限。

- 修改授权：在“用户”页签下用户所在行单击“编辑”，在“权限”所在列选择新的权限，然后单击“保存”。

图 1-10 在组织中修改授权

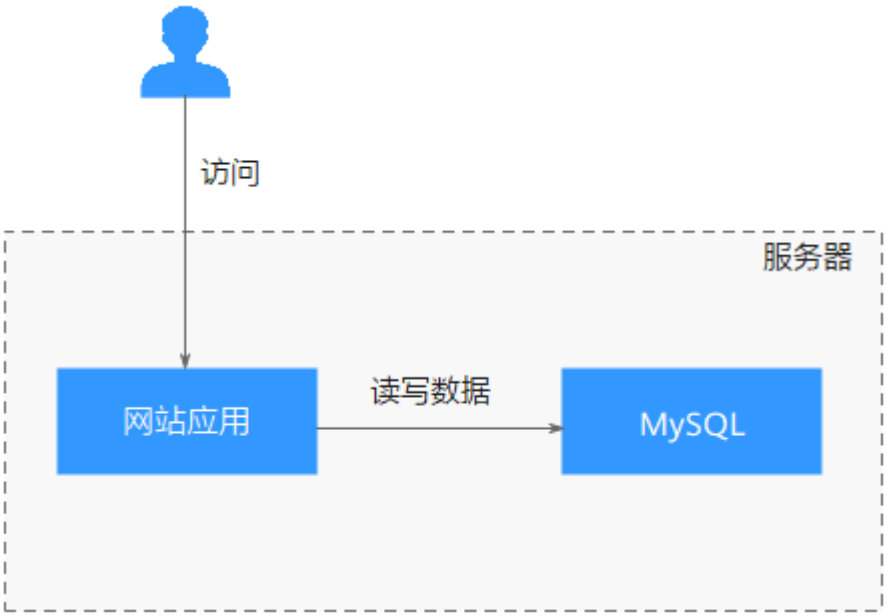
- 在“用户”页签下用户所在行单击“删除”，然后单击“确定”。

2 最佳实践

2.1 编写高效的 Dockerfile

本章基于容器镜像服务实践所编写，将一个单体应用进行容器改造为例，展示如何写出可读性更好的Dockerfile，从而提升镜像构建速度，构建层数更少、体积更小的镜像。

下面是一个常见企业门户网站架构，由一个Web Server和一个数据库组成，Web Server提供Web服务，数据库保存用户数据。通常情况下，这样一个门户网站安装在一台服务器上。



如果把应用运行在一个Docker容器中，那么很可能写出下面这样的Dockerfile来。

```
FROM ubuntu

ADD . /app

RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get install -y nodejs ssh mysql
```

```
RUN cd /app && npm install

# this should start three processes, mysql and ssh
# in the background and node app in foreground
# isn't it beautifully terrible? <3
CMD mysql & sshd & npm start
```

但是这样Dockerfile有很多问题，这里CMD命令是错误的，只是为了说明问题而写。

下面的内容中将展示对这个Dockerfile进行改造，说明如何写出更好的Dockerfile，共有如下几种处理方法。

- **一个容器只运行一个进程**
- **不要在构建中升级版本**
- **将变化频率一样的RUN指令合一**
- **使用特定的标签**
- **删除多余文件**
- **选择合适的基础镜像**
- **设置WORKDIR和CMD**
- **使用ENTRYPOINT（可选）**
- **在entrypoint脚本中使用exec**
- **优先使用COPY**
- **合理调整COPY与RUN的顺序**
- **设置默认的环境变量、映射端口和数据库逻辑卷**
- **使用EXPOSE暴露端口**
- **使用VOLUME管理数据库逻辑卷**
- **使用LABEL设置镜像元数据**
- **添加HEALTHCHECK**
- **编写.dockerignore文件**

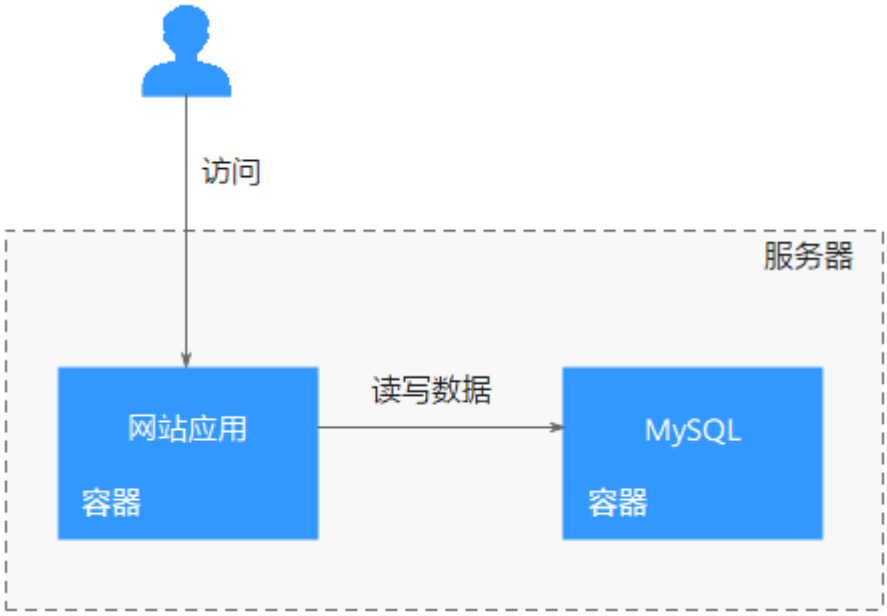
一个容器只运行一个进程

从技术角度讲，Docker容器中可以运行多个进程，您可以将数据库、前端、后端、ssh等都运行在同一个Docker容器中。但是，这样跟未使用容器前没有太大区别，且这样容器的构建时间非常长（一处修改就要构建全部），镜像体积大，横向扩展时非常浪费资源（不同的应用需要运行的容器数并不相同）。

通常所说的容器化改造是对应用整体微服务架构改造，再容器化，这样做可以带来如下好处。

- 单独扩展：拆分为微服务后，可单独增加或缩减每个微服务的实例数量。
- 提升开发速度：各微服务之间解耦，某个微服务的代码开发不影响其他微服务。
- 通过隔离确保安全：整体应用中，若存在安全漏洞，会获得所有功能的权限。微服务架构中，若攻击了某个服务，只可获得该服务的访问权限，无法入侵其他服务。
- 隔离崩溃：如果其中一个微服务崩溃，其他微服务还可以持续正常运行。

如前面的示例可以改造成下面架构，Web应用和MySQL运行在不同容器中。



MySQL运行在单独的镜像中，下面的示例中删掉了MySQL，只安装node.js。

```
FROM ubuntu

ADD . /app

RUN apt-get update
RUN apt-get upgrade -y

RUN apt-get install -y nodejs
RUN cd /app && npm install

CMD npm start
```

不要在构建中升级版本

为了降低复杂性、减少依赖、减小文件大小、节约构建时间，你应该避免安装任何不必要的包。例如，不要在数据库镜像中包含一个文本编辑器。

apt-get upgrade会使得镜像构建非常不确定，在构建时不确定哪些包会被安装，此时可能会产生不一致的镜像。

如果基础镜像中的某个包过时了，你应该联系它的维护者。如果你确定某个特定的包，比如foo需要升级，使用apt-get install -y foo就行，该指令会自动升级foo包。

删掉apt-get upgrade后，Dockerfile如下：

```
FROM ubuntu

ADD . /app

RUN apt-get update

RUN apt-get install -y nodejs
RUN cd /app && npm install

CMD npm start
```

将变化频率一样的 RUN 指令合一

Docker镜像是分层的，类似于洋葱，它们都有很多层，为了修改内层，则需要将外面的层都删掉。Docker镜像有如下特性：

- Dockerfile中的每个指令都会创建一个新的镜像层。
- 镜像层将被缓存和复用。
- Dockerfile修改后，复制的文件变化了或者构建镜像时指定的变量不同了，对应的镜像层缓存就会失效。
- 某一层的镜像缓存失效之后，它之后的镜像层缓存都会失效。
- 镜像层是不可变的，如果在某一层中添加一个文件，然后在下一层中删除它，则镜像中依然会包含该文件，只是这个文件在Docker容器中不可见。

将变化频率一样的指令合并在一起，目的是为了更好的将镜像分层，避免带来不必要的成本。如本例中将node.js安装与npm模块安装放在一起的话，则每次修改源代码，都需要重新安装node.js，这显然不合适。

```
FROM ubuntu

ADD . /app

RUN apt-get update \
    && apt-get install -y nodejs \
    && cd /app \
    && npm install

CMD npm start
```

因此，正确的写法是这样的：

```
FROM ubuntu

RUN apt-get update && apt-get install -y nodejs
ADD . /app
RUN cd /app && npm install

CMD npm start
```

使用特定的标签

当镜像没有指定标签时，将默认使用latest标签。因此，FROM ubuntu指令等同于FROM ubuntu:latest。当镜像更新时，latest标签会指向不同的镜像，这时构建镜像有可能失败。

如下示例中使用16.04作为标签。

```
FROM ubuntu:16.04

RUN apt-get update && apt-get install -y nodejs
ADD . /app
RUN cd /app && npm install

CMD npm start
```

删除多余文件

假设更新了apt-get源，下载解压并安装了一些软件包，它们都保存在“/var/lib/apt/lists/”目录中。

但是，运行应用时Docker镜像中并不需要这些文件。因此需要将它们删除，因为它会使Docker镜像变大。

示例Dockerfile中，删除“/var/lib/apt/lists/”目录中的文件。

```
FROM ubuntu:16.04

RUN apt-get update \
    && apt-get install -y nodejs \
    && rm -rf /var/lib/apt/lists/*

ADD . /app
RUN cd /app && npm install

CMD npm start
```

选择合适的基础镜像

在示例中，选择了ubuntu作为基础镜像。但是只需要运行node程序，没有必要使用一个通用的基础镜像，node镜像应该是更好的选择。

更好的选择是alpine版本的node镜像。alpine是一个极小化的Linux发行版，只有4MB，这让它非常适合作为基础镜像。

```
FROM node:7-alpine

ADD . /app
RUN cd /app && npm install

CMD npm start
```

设置 WORKDIR 和 CMD

WORKDIR指令可以设置默认目录，也就是运行RUN / CMD / ENTRYPOINT指令的地方。

CMD指令可以设置容器创建时执行的默认命令。另外，您应该将命令写在一个数组中，数组中每个元素为命令的每个单词。

```
FROM node:7-alpine

WORKDIR /app
ADD . /app
RUN npm install

CMD ["npm", "start"]
```

使用 ENTRYPOINT（可选）

ENTRYPOINT指令并不是必须的，因为它会增加复杂度。ENTRYPOINT是一个脚本，它会默认执行，并且将指定的命令作为其参数。它通常用于构建可执行的Docker镜像。

```
FROM node:7-alpine

WORKDIR /app
ADD . /app
RUN npm install

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

在 entrypoint 脚本中使用 exec

在前文的entrypoint脚本中，使用了exec命令运行node应用。不使用exec的话，则不能顺利地关闭容器，因为SIGTERM信号会被bash脚本进程吞没。exec命令启动的进程可以取代脚本进程，因此所有的信号都会正常工作。

优先使用 COPY

COPY指令非常简单，仅用于将文件拷贝到镜像中。ADD相对来讲复杂一些，可以用于下载远程文件以及解压压缩包。

```
FROM node:7-alpine

WORKDIR /app

COPY . /app
RUN npm install

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

合理调整 COPY 与 RUN 的顺序

将变化最少的部分放在Dockerfile的前面，这样可以充分利用镜像缓存。

示例中，源代码会经常变化，则每次构建镜像时都需要重新安装NPM模块，这显然不是希望看到的。因此可以先拷贝package.json，然后安装NPM模块，最后才拷贝其余的源代码。这样的话，即使源代码变化，也不需要重新安装NPM模块。

```
FROM node:7-alpine

WORKDIR /app

COPY package.json /app
RUN npm install
COPY . /app

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

设置默认的环境变量、映射端口和数据库逻辑卷

运行Docker容器时很可能需要一些环境变量。在Dockerfile设置默认的环境变量是一种很好的方式。另外，应该在Dockerfile中设置映射端口和数据库逻辑卷。示例如下：

```
FROM node:7-alpine

ENV PROJECT_DIR=/app

WORKDIR $PROJECT_DIR

COPY package.json $PROJECT_DIR
RUN npm install
COPY . $PROJECT_DIR

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

ENV指令指定的环境变量在容器中可以使用。如果你只是需要指定构建镜像时的变量，你可以使用ARG指令。

使用 EXPOSE 暴露端口

EXPOSE指令用于指定容器将要监听的端口。因此，你应该为你的应用程序使用常见的端口。例如，提供Apache web服务的镜像应该使用EXPOSE 80，而提供MongoDB服务的镜像使用EXPOSE 27017。

对于外部访问，用户可以在执行docker run时使用一个标志来指示如何将指定的端口映射到所选择的端口。

```
FROM node:7-alpine

ENV PROJECT_DIR=/app

WORKDIR $PROJECT_DIR

COPY package.json $PROJECT_DIR
RUN npm install
COPY . $PROJECT_DIR

ENV APP_PORT=3000
EXPOSE $APP_PORT

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

使用 VOLUME 管理数据库逻辑卷

VOLUME指令用于暴露任何数据库存储文件、配置文件或容器创建的文件和目录。强烈建议使用VOLUME来管理镜像中的可变部分和用户可以改变的部分。

下面示例中填写一个媒体目录。

```
FROM node:7-alpine

ENV PROJECT_DIR=/app

WORKDIR $PROJECT_DIR

COPY package.json $PROJECT_DIR
RUN npm install
COPY . $PROJECT_DIR

ENV MEDIA_DIR=/media \
    APP_PORT=3000

VOLUME $MEDIA_DIR
EXPOSE $APP_PORT

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

使用 LABEL 设置镜像元数据

你可以给镜像添加标签来帮助组织镜像、记录许可信息、辅助自动化构建等。每个标签一行，由LABEL开头加上一个或多个标签对。

须知

如果你的字符串中包含空格，必须将字符串放入引号中或者对空格使用转义。如果字符串内容本身就包含引号，必须对引号使用转义。

```
FROM node:7-alpine
LABEL com.example.version="0.0.1-beta"
```

添加 HEALTHCHECK

运行容器时，可以指定--restart always选项。这样的话，容器崩溃时，docker daemon会重启容器。对于需要长时间运行的容器，这个选项非常有用。但是，如果容器的确在运行，但是不可用怎么办？使用HEALTHCHECK指令可以让Docker周期性的检查容器的健康状况。只需要指定一个命令，如果一切正常的话返回0，否则返回1。当请求失败时，curl --fail命令返回非0状态。示例如下：

```
FROM node:7-alpine
LABEL com.example.version="0.0.1-beta"

ENV PROJECT_DIR=/app
WORKDIR $PROJECT_DIR

COPY package.json $PROJECT_DIR
RUN npm install
COPY . $PROJECT_DIR

ENV MEDIA_DIR=/media \
    APP_PORT=3000

VOLUME $MEDIA_DIR
EXPOSE $APP_PORT
HEALTHCHECK CMD curl --fail http://localhost:$APP_PORT || exit 1

ENTRYPOINT ["./entrypoint.sh"]
CMD ["start"]
```

编写.dockerignore 文件

.dockerignore的作用和语法类似于.gitignore，可以忽略一些不需要的文件，这样可以有效加快镜像构建时间，同时减少Docker镜像的大小。

构建镜像时，Docker需要先准备context，将所有需要的文件收集到进程中。默认的context包含Dockerfile目录中的所有文件，但是实际上，并不需要.git目录等内容。

示例如下：

```
.git/
```

2.2 跨云 Harbor 同步镜像至 HCS SWR

场景描述

部分客户存在多云场景，并且使用某一家云上的自建Harbor作为镜像仓库。Harbor可以通过公网访问SWR，配置方法参见**公网访问场景**。

背景知识

Harbor是VMware公司开源的企业级Docker Registry管理项目，在开源Docker Distribution能力基础上扩展了例如权限管理（RBAC）、镜像安全扫描、镜像复制等功能。Harbor目前已成为自建容器镜像托管及分发服务的首选。

公网访问场景

步骤1 Harbor上配置镜像仓库。

说明

Harbor在1.10.5以上版本，集成了华为云的SWR对接，只需要在目标（ENDPOINT）上选择就可以。本文以Harbor 2.4.1为例。

1. 新建目标。

2. 填写如下参数。
   - 提供者：必须选“Huawei SWR”。
   - 目标名：自定义。
   - 目标URL：使用SWR的公网域名，格式为https://{*SWR镜像仓库地址*}，例如：https://swr.cn-dg-1.external-c01type1.com。镜像仓库地址获取方法：登录容器镜像服务控制台，进入“我的镜像”，单击“客户端上传”，在弹出的页面即可查看SWR当前Region的镜像仓库地址。
   - 访问ID：遵循SWR的长期有效的认证凭证规则，以“区域项目名称@[AK]”形式填写。
   - 访问密码：遵循SWR的长期有效的认证凭证规则，需要用AK和SK来生成，详细说明请参考**获取长期有效登录指令**。
   - 验证远程证书：建议**取消勾选**。

步骤2 配置同步规则。

1. 新建规则

2. 填写如下参数。
   - 名称：自定义。
   - 复制模式：选择“Push-based”，表示把镜像由本地Harbor推送到远端仓库。
   - 源资源过滤器：根据填写的规则过滤Harbor上的镜像。
   - 目标仓库：选择**步骤1**中创建的目标。
   - 目标

     名称空间：填写SWR上的组织名称。

     仓库扁平化：用以在复制镜像时减少仓库的层级结构，建议选择“替换所有级”。假设Harbor仓库的层级结构为“library/nginx”，目标名称空间为dev-container，“替换所有级”对应的结果为：library/nginx -> dev-container/nginx。
   - 触发模式：选择“手动”。
   - 带宽：设置执行该条同步规则时的最大网络带宽，“-1”表示无限制。

步骤3 创建完成后，选中后单击“复制”即可完成同步。

----结束

2.3 为用户授权 SWR 权限

权限类型

管理权限：可以管理SWR中的所有组织和所有镜像，并可以为SWR开发角色和SWR只读角色进行授权。

开发权限：对自己创建的组织和镜像具有管理权限；也可以通过管理权限用户或其他开发权限用户授权，获得其他组织和镜像的管理/修改/只读权限。

只读权限：对所有组织和镜像只有读取权限。

操作步骤

创建用户组

步骤1 使用浏览器，以运营管理员账号登录ManageOne运营管理平台。

- 登录地址：https://*ManageOne运营面的访问地址*。
- 账号密码：运营管理员对应账号和密码。

步骤2 单击“组织”页签，选择“租户管理”，单击租户名称进入租户详情页。

步骤3 选择“用户组”并单击“创建”，输入用户组名称完成用户组的创建。

用户组权限配置

步骤4 返回用户组列表，单击操作列“添加授权”对用户组进行权限设置。

步骤5 在“资源空间”下选择资源空间，单击“下一步”，选择授权角色。

步骤6 在搜索框中搜索“ServiceStage Developer”角色并选中，如果没有“ServiceStage Developer”就搜索“SWR Developer”角色并选中，单击“确定”添加权限。

为用户授权

步骤7 返回租户详情页面，选择“用户”，在需要添加权限的用户操作列单击“加入到用户组”。

- 给用户添加管理权限

  勾选“VDC管理员”并单击“确定”完成管理员权限的添加。
- 给用户添加开发权限

  勾选**步骤3**创建的用户组并单击“确定”完成开发权限的添加。
- 给用户添加只读权限

  a. 勾选**步骤3**创建的用户组并单击“确定”。

  b. 使用浏览器，以具有SWR管理权限的账号登录ManageOne。

  c. 在页面左上角单击  ，选择“应用服务 > 容器镜像服务SWR”。

  d. 在导航栏左上角下拉框选择区域和资源空间。进入容器镜像服务页面。

  e. 在左侧菜单栏选择“组织管理”，依次单击右侧所有组织名称后的“详情”。

f. 在“用户”页签下单击“添加授权”，在弹出的窗口中为用户选择“读取”权限，然后单击“确定”完成只读权限的添加。

----结束

2.4 使用 image-syncer 迁移镜像

场景描述

当我们处理数量较少的镜像迁移任务时，使用命令行迁移就可以解决这个问题。但是实际生产中涉及到成千上百个镜像，几T的镜像仓库数据时，迁移过程就变得耗时很是漫长，甚至丢失数据。这时，我们可以使用开源镜像迁移工具**image-syncer**来处理这个任务。下面以把HCS环境下的容器镜像仓库中的镜像迁移到敏捷版的镜像仓库中为例进行说明。

操作步骤

步骤1 **下载image-syncer**到执行机上，解压并运行工具。

建议选择非业务节点且已安装docker容器引擎客户端的机器作为执行机，并确保所选执行机与HCS镜像仓库以及敏捷版云容器引擎服务网络互通。

以v1.5.0版本为例，您也可以选择其他版本。

wget https://github.com/AliyunContainerService/image-syncer/releases/download/v1.5.0/image-syncer-v1.5.0-linux-amd64.tar.gz

tar -zvxf image-syncer-v1.5.0-linux-amd64.tar.gz

步骤2 创建镜像仓库的认证信息文件**auth.json**。

image-syncer支持基于Docker Registry V2搭建的docker镜像仓库，按格式填写即可。将源仓库及目标仓库认证信息写入，示例如下。

```
{
    "swr.××××.com": {
        "username": "sa-fb-1_cce******",
        "password": "93e6979253**********67e5383bfb38de21",
        "insecure": true
    },
    "iefagile830.*****.com": {
        "username": "_regis******",
        "password":
"eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJzdUd2aVM4Yz**********kCsOpaBBmA2LvE04xi1JI1g",
        "insecure": true
    }
}
```

其中swr.××××.com和iefagile830.*****.com为源仓库及目标镜像仓库地址，username、password可以在登录命令中获取，获取方法如下：

获取源镜像仓库的登录指令

登录SWR控制台，在总览页面右上角单击“登录指令”，在弹出的窗口中获取登录指令，如下图所示。

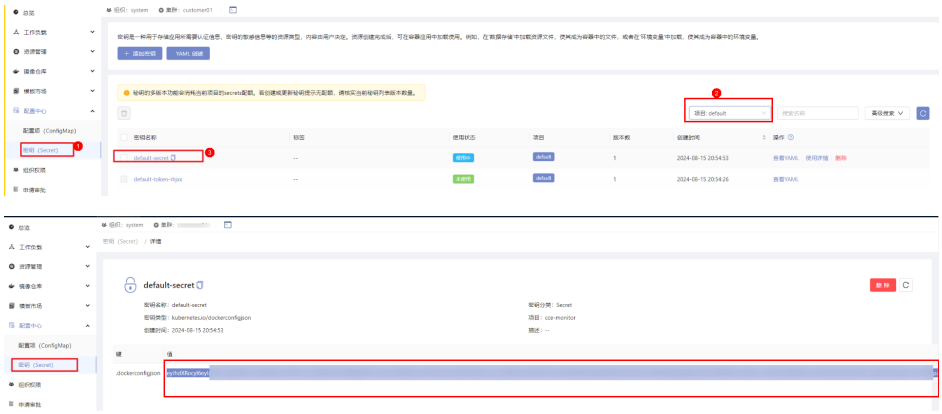
图 2-1 登录指令

获取目标镜像仓库的登录指令

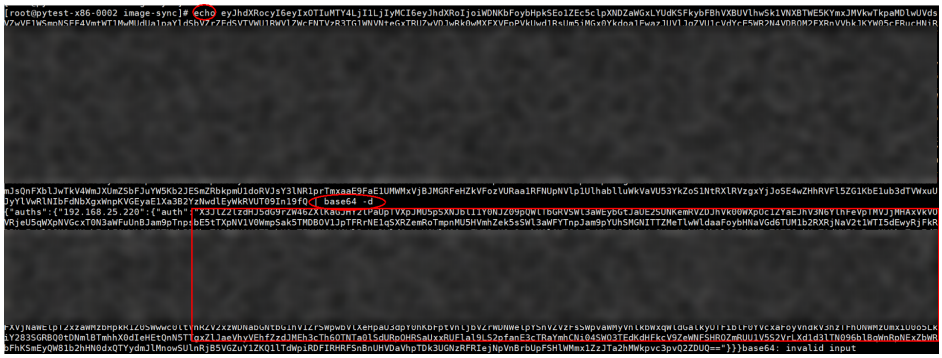
敏捷版的镜像仓库登录指令获取有2种方法。

方法1：

1. 登录敏捷版控制台，在“容器管理平台-配置中心-密钥（Secret）”页面复制default项目下default-secret密钥。



2. 登录敏捷版primary上master节点，通过base64命令进行连续解密。注意：第一次解密内容是上面获取的密钥，第二次解密的内容是第一次解密之后的auth内容，如下图：



注意

因安全性要求，以上示例中所有username和password均有部分内容进行省略，请以控制台获取到的实际用户名和密码为准。

当数据量较大时，镜像迁移过程需要耗费较长的时间，故建议修改敏捷版的令牌有效期时间为24小时。在敏捷版控制台进入“容器管理平台-系统管理-安全配置”修改令牌有效期。

注意

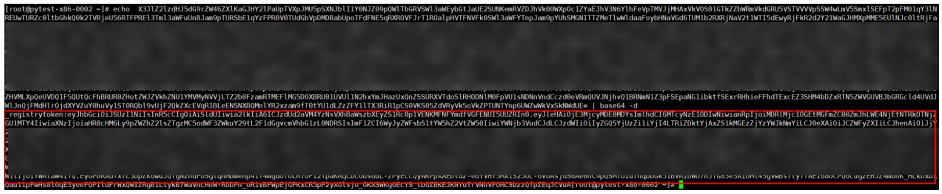
令牌有效期修改后并不能立即生效。修改前已生成的令牌失效后新生成的令牌才能生效 。

方法2：

1. 将primary上master节点中/var/paas/srv/kubernetes/root.key文件拷贝到迁移工具执行机的相同目录下，若迁移工具执行机没有此目录的话需新建相同目录后再拷贝。
2. 参照“镜像仓库-镜像管理”页面的操作步骤在迁移工具执行机下载cce工具并执行cce init和cce login。

3. 之后在迁移工具执行机的文件./docker/config.json里查看密钥。

4. 对上面文件中获取的密钥执行解密。



说明

当迁移的数据量较大时也请参考方法1中的操作修改令牌的有效期时间为24小时。注意：用方法2中获取的密钥随着令牌有效期的修改实时生效。

步骤3 创建同步镜像描述文件**images.json**。

如下示例，左边是源仓库的地址，右边是目的仓库地址。image-syncer还支持其他描述方式，具体请参见**README-zh_CN.md**。

```
{
 "swr.sa-fb******.com/test/nginx:curl":"iefagile830.*****.com/test/test:hcs","iefagile830.*****.com/cce/calico-typha:v3.23.2":"swr.sa-fb******.com/test/nginx:agile"
}
```

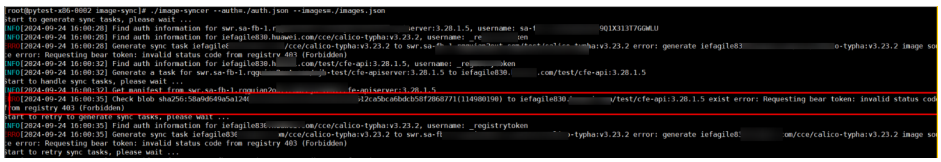
步骤4 执行如下命令将镜像迁移至SWR。

```
./image-syncer --auth=./auth.json --images=./images.json
```

表 2-1 命令行参数说明

参数	说明
--config	设置用户提供的配置文件路径，使用之前需要创建此文件，默认为当前工作目录下的config.json文件。这个参数与 --auth和--images 的作用相同，分解成两个参数可以更好地区分认证信息与镜像仓库同步规则。建议使用 --auth 和 --images。
--images	设置用户提供的镜像同步规则文件所在路径，使用之前需要创建此文件，默认为当前工作目录下的images.json文件。
--auth	设置用户提供的认证文件所在路径，使用之前需要创建此认证文件，默认为当前工作目录下的auth.json文件。
--log	打印出来的log文件路径，默认打印到标准错误输出，如果将日志打印到文件将不会有命令行输出，此时需要通过cat对应的日志文件查看。
--namespace	设置默认的目标namespace，当配置文件内一条images规则的目标仓库为空，并且默认registry也不为空时有效，可以通过环境变量DEFAULT_NAMESPACE设置，同时传入命令行参数会优先使用命令行参数值。
--proc	并发数，进行镜像同步的并发goroutine数量，默认为5。不建议修改该参数。
--retries	失败同步任务的重试次数，默认为2，重试会在所有任务都被执行一遍之后开始，并且也会重新尝试对应次数生成失败任务的生成。一些偶尔出现的网络错误比如io timeout、TLS handshake timeout，都可以通过设置重试次数来减少失败的任务数量。
--registry	设置默认的目标registry，当配置文件内一条images规则的目标仓库为空，并且默认namespace也不为空时有效，可以通过环境变量DEFAULT_REGISTRY设置，同时传入命令行参数会优先使用命令行参数值。

如果执行工具遇到如下403报错，说明密钥失效需重新获取密钥，修改auth.json中密钥信息，重新执行./image-syncer --auth=./auth.json --images=./images.json



步骤5 迁移命令执行后，可登录目标镜像仓库，查看已迁移的镜像。

----结束

2.5 多组织权限隔离

操作场景

当您的业务需要规划多个组织给多个VDC子用户使用，并根据不同VDC子用户的需求，对不同的组织或镜像授予或禁止对应的权限，以实现不同组织镜像权限的有效隔离时，您可以参考本章节的指导进行操作。

下文将以如下场景来演示如何进行配置：

假如存在组织A即nsA，组织B即nsB以及组织C即nsC。并且存在VDC子用户A，VDC子用户B，VDC子用户C，VDC子用户A允许删除组织A以及组织A中的镜像和版本同时禁止删除组织B和C以及组织B和C中的镜像和镜像版本，VDC子用户B允许删除组织B以及组织B中的镜像和版本同时禁止删除组织A和C以及组织A和C中的镜像和镜像版本。

说明

- 多组织权限隔离功能的实现是借助ManageOne提供的角色的权限管理以及给用户授予相关的角色功能来实现的。ManageOne的权限体系如下：

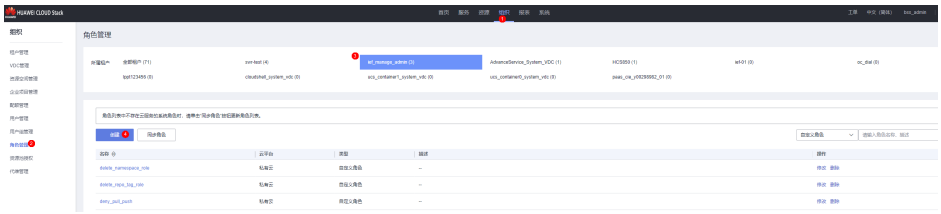
  创建用户 --> 把用户加入用户组 --> 给用户组授予权限角色。
- 当您只需要用户具备SWR的特定权限时，可以直接授予这些权限；但如果需要用户拥有管理员权限（Tenant Administrator或SWR Administrator）,并且仍需要限制他们操作的组织或镜像范围时，您可以在授予管理员权限的同时，进一步细化并限制他们在不同组织和镜像上的具体权限。组织权限隔离采用的即后者的方式。

约束限制

- 修改自定义角色权限后需要退出控制台重新登录使配置生效。
- 请检查确认您的VDC子用户具有Tenant Administrator或者SWR Administrator的权限。

操作步骤

步骤1 使用一级VDC管理员账号（即bss_admin）登录到ManageOne运营面，切换组织后单击”角色管理”选择所属租户，单击”创建”。



步骤2 创建禁止删除组织和镜像角色deny_nsA。

说明

nsA为组织A的名称，该角色权限含义为禁止删除组织A以及组织A下镜像和镜像版本。

输入自定义角色名称: deny_nsA，选择对应的所属租户，作用范围选择资源空间服务，授权配置切换为JSON视图，内容如下，

```
{
    "Version": "1.1",
```

```
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "swr:namespace:deleteNamespace"
      ],
      "Resource": [
        "SWR:*:*:namespace:nsA"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "swr:repo:deleteRepoTag",
        "swr:repo:deleteRepo",
        "swr:repo:createRetention",
        "swr:repo:updateRetention",
        "swr:repo:deleteRetention"
      ],
      "Resource": [
        "SWR:*:*:repo:nsA"
      ]
    }
  ]
}
```

步骤3 创建禁止删除组织和镜像角色deny_nsB。

说明

nsB为组织B的名称，该角色权限含义为禁止删除组织B以及组织B下镜像和镜像版本。

输入自定义角色名称: deny_nsB，并选择对应的所属租户，作用范围选择资源空间服务，授权配置切换为JSON视图，内容如下

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "swr:namespace:deleteNamespace"
      ],
      "Resource": [
        "SWR:*:*:namespace:nsB"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "swr:repo:deleteRepoTag",
        "swr:repo:deleteRepo",
        "swr:repo:createRetention",XX
        "swr:repo:updateRetention",
        "swr:repo:deleteRetention"
      ],
      "Resource": [
        "SWR:*:*:repo:nsB/*"
      ]
    }
  ]
}
```

说明

如果权限配置您使用可视化视图创建，高级设置里的namespace和repo的”了解更多”按钮点击跳转的页面会发生异常，这是由于bss_admin为默认租户，不用于业务访问，不影响您组织权限隔离功能的使用。

步骤4 创建禁止删除组织和镜像角色deny_nsC。

说明

nsC为组织C的名称，该角色权限含义为禁止删除组织C以及组织C下镜像和镜像版本。

输入自定义角色名称: deny_nsC，并选择对应的所属租户，作用范围选择资源空间服务，授权配置切换为JSON视图，内容如下

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Effect": "Deny",
         "Action": [
            "swr:namespace:deleteNamespace"
         ],
         "Resource": [
            "SWR:*:*:namespace:nsC"
         ]
      },
      {
         "Effect": "Deny",
         "Action": [
            "swr:repo:deleteRepoTag",
            "swr:repo:deleteRepo",
            "swr:repo:createRetention",
            "swr:repo:updateRetention",
            "swr:repo:deleteRetention"
         ],
         "Resource": [
            "SWR:*:*:repo:nsC/*"
         ]
      }
   ]
}
```

步骤5 要实现VDC子用户A允许删除组织A以及镜像和版本和禁止删除组织B和C以及镜像和版本，需要使用一级VDC用户给VDC子用户A所在的用户组中添加deny_nsB和deny_nsC角色即可。

步骤6 要实现VDC子用户B允许删除组织B以及镜像和版本和禁止删除组织A和C以及镜像和版本，需要使用一级VDC用户给VDC子用户B所在的用户组中添加deny_nsA和deny_nsC角色授权即可。

步骤7 要实现VDC子用户C允许删除组织C以及镜像和版本和禁止删除组织A和B以及镜像和版本，需要使用一级VDC用户给VDC子用户C所在的用户组中添加deny_nsA和deny_nsB角色授权即可。

表 2-2 VDC 子用户的权限和角色设置

子用户/角色	deny_nsA	deny_nsB	deny_nsC
VDC子用户1（只可删组织A）	-	√	√
VDC子用户2（只可删组织B）	√	-	√
VDC子用户3（只可删组织C）	√	√	-

⚠ 注意

如果后续新增了组织D以及VDC子用户4（只能管控组织D以及其他的镜像和镜像版本），需要一级VDC用户先创建自定义角色deny_nsD，然后重新给已存在的所有其他的子用户（例如上文中的VDC子用户1，VDC子用户2，VDC子用户3）添加deny_nsD的角色。

----结束

3 常见问题

3.1 通用类

3.1.1 产品咨询

SWR 最多能存储多少个镜像?

SWR对存储的镜像数量没有限制，您可以根据需要上传镜像。

SWR 镜像仓库支持 ARM 镜像上传吗?

SWR镜像仓库对镜像的内核架构是没有任何限制的，您上传ARM架构的镜像和上传x86的镜像是没有区别的，直接上传即可。

docker push 使用什么协议将镜像推送到 SWR?

HTTPS协议。

同名同 tag 的镜像上传后会覆盖之前的镜像吗?

会覆盖，保留最新上传的镜像。

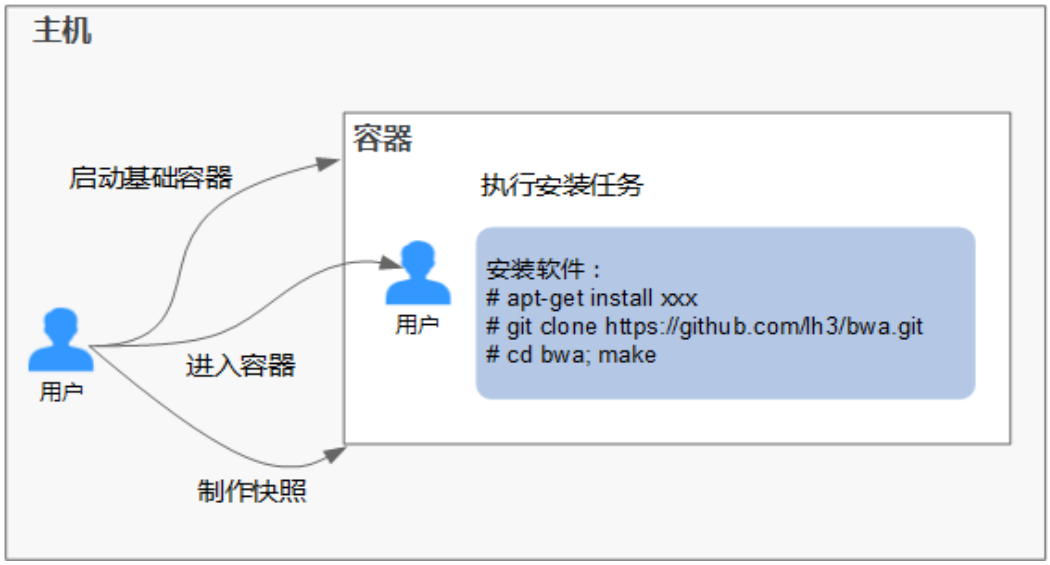
3.1.2 如何制作容器镜像?

自己制作容器镜像，主要有两种方法：

- 制作快照方式获得镜像（偶尔制作的镜像）：在基础镜像上（比如Ubuntu），先登录镜像系统并安装容器引擎软件，然后整体制作快照。
- Dockerfile方式构建镜像（经常更新的镜像）：将软件安装的流程写成Dockerfile，使用docker build构建成容器镜像。

方法一：制作快照方式获得镜像

如果后续镜像没有变化，可采用**方法一**制作镜像。



具体操作如下：

1. 找一台主机，安装容器引擎软件。
2. 启动一个空白的基础容器，并进入容器。
   例如：启动一个CentOS的容器。
   docker run -it centos
3. 执行安装任务。
   yum install XXX
   git clone https://github.com/lh3/bwa.git
   cd bwa;make

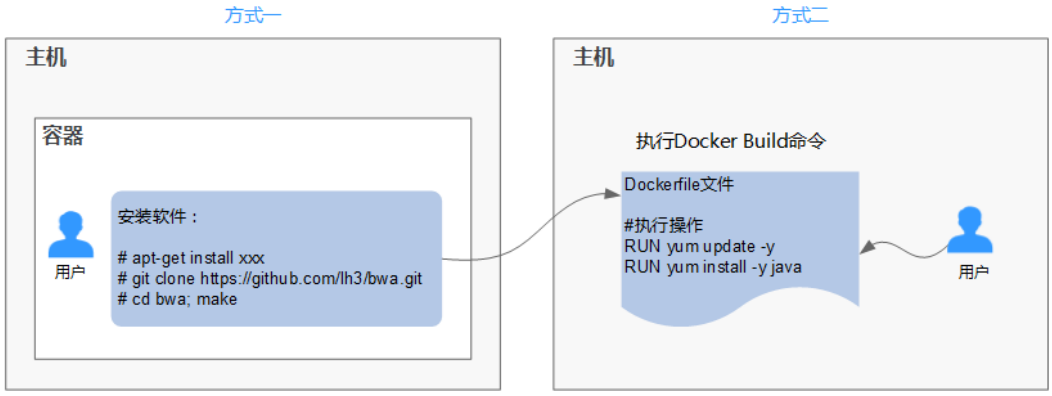
   说明

   请预先安装好Git，并检查本机是否有ssh key设置。
4. 输入**exit**退出容器。
5. 制作快照。
   docker commit -m "xx" -a "test" container-id test/image:tag
   - -a：提交的镜像作者。
   - container-id：**步骤2**中的容器id。可以使用**docker ps -a**查询得到容器id。
   - -m：提交时的说明文字。
   - test/image:tag：仓库名/镜像名:TAG名。
6. 执行**docker images**可以查看到制作完成的容器镜像。

方法二：使用 Dockerfile 方式构建

如果后续镜像经常变更（例如某个软件更新版本），则需要采用**方法二**制作镜像。若仍采用**方法一**制作镜像，则每次变更都需要重新执行**方法一**的命令，操作过程比较繁琐，所以建议使用自动化制作镜像的方法。

其实就是将**方法一**制作镜像的方法，用文件方式写出来（文件名为Dockerfile）。然后执行：**docker build -t test/image:tag.**命令（命令中“.”表示Dockerfile文件的路径），自动完成镜像制作。



简单的Dockerfile示例：

说明

如果依赖外部网络，请搭建网络环境，并保证网络可用。

```
#Version 1.0.1
FROM centos:latest

#设置root用户为后续命令的执行者
USER root

#执行操作
RUN yum update -y
RUN yum install -y java

#使用&&拼接命令
RUN touch test.txt && echo "abc" >>abc.txt

#对外暴露端口
EXPOSE 80 8080 1038

#添加网络文件
ADD https://www.baidu.com/img/bd_logo1.png /opt/

#设置环境变量
ENV WEBAPP_PORT=9090

#设置工作目录
WORKDIR /opt/

#设置启动命令
ENTRYPOINT ["ls"]

#设置启动参数
CMD ["-a", "-l"]

#设置卷
VOLUME ["/data", "/var/www"]

#设置子镜像的触发操作
ONBUILD ADD . /app/src
ONBUILD RUN echo "on build excuted" >> onbuild.txt
```

Dockerfile 基本语法

- FROM：

  指定待扩展的父级镜像（基础镜像）。除注释之外，文件开头必须是一个FROM指令，后面的指令便在这个父级镜像的环境中运行，直到遇到下一个FROM指令。通过添加多个FROM命令，可以在同一个Dockerfile文件中创建多个镜像。

- MAINTAINER：

  声明创建镜像的作者信息：用户名、邮箱，非必须参数。

- RUN：

  修改镜像的命令，常用来安装库、安装程序以及配置程序。一条RUN指令执行完毕后，会在当前镜像上创建一个新的镜像层，接下来对的指令会在新的镜像上继续执行。RUN 语句有两种形式：

  - **RUN yum update**：在**/bin/sh**路径中执行的指令命令。
  - **RUN ["yum", "update"]**：直接使用系统调用**exec**来执行。
  - **RUN yum update && yum install nginx**：使用&&符号将多条命令连接在同一条RUN语句中。

- EXPOSE：

  指明容器内进程对外开放的端口，多个端口之间使用空格隔开。

  运行容器时，通过设置参数**-P**（大写）即可将EXPOSE里所指定的端口映射到主机上其他的随机端口，其他容器或主机可以通过映射后的端口与此容器通信。

  您也可以通过设置参数**-p**（小写）将Dockerfile中EXPOSE中没有列出的端口设置成公开。

- ADD：

  向新镜像中添加文件，这个文件可以是一个主机文件，也可以是一个网络文件，也可以是一个文件夹。

  - 第一个参数：源文件（夹）。
    - 如果是相对路径，必须是相对于Dockerfile所在目录的相对路径。
    - 如果是URL，会将文件先下载下来，然后再添加到镜像里。
  - 第二个参数：目标路径。
    - 如果源文件是主机上的zip或者tar形式的压缩文件，容器引擎会先解压缩，然后将文件添加到镜像的指定位置。
    - 如果源文件是一个通过URL指定的网络压缩文件，则不会解压。

- VOLUME：

  在镜像里创建一个指定路径(文件或文件夹)的挂载点，这个容器可以来自主机或者其它容器。多个容器可以通过同一个挂载点共享数据，即便其中一个容器已经停止，挂载点也仍可以访问。

- WORKDIR：

  为接下来执行的指令指定一个新的工作目录，这个目录可以是绝对目录，也可以是相对目录。根据需要，WORKDIR可以被多次指定。当启动一个容器时，最后一条WORKDIR指令所指的目录将作为容器运行的当前工作目录。

- ENV：

  设置容器运行的环境变量。在运行容器的时候，通过设置**-e**参数可以修改这个环境变量值，也可以添加新的环境变量。

  例如：

  docker run -e WEBAPP_PORT=8000 -e WEBAPP_HOST=www.example.com ...

- CMD：

  用来设置启动容器时默认运行的命令。

- ENTRYPOINT：

  用来指定容器启动时的默认运行的命令。区别在于：运行容器时添加在镜像之后的参数，对ENTRYPOINT是拼接，CMD是覆盖。

  - 若在Dockerfile中指定了容器启动时的默认运行命令为**ls -l**，则运行容器时默认启动命令为**ls -l**，例如：
    - **ENTRYPOINT ["ls", "-l"]**：指定容器启动时的程序及参数为**ls -l**。
    - **docker run centos**：当运行centos容器时，默认执行的命令是**docker run centos ls -l**
    - **docker run centos -a**：当运行centos容器时拼接了-a参数，则默认运行的命令是**docker run centos ls -l -a**
  - 若在Dockerfile中指定了容器启动时的默认运行命令为**--entrypoint**，则在运行容器时若需要替换默认运行命令，可以通过添加**--entrypoint**参数来替换Dockerfile中的指定。例如：

    docker run gutianlangyu/test --entrypoint echo "hello world"

- USER：

  为容器的运行及RUN、CMD、ENTRYPOINT等指令的运行指定用户或UID。

- ONBUILD：

  触发器指令。构建镜像时，容器引擎的镜像构建器会将所有的ONBUILD指令指定的命令保存到镜像的元数据中，这些命令在当前镜像的构建过程中并不会执行。只有新的镜像使用FROM指令指定父镜像为当前镜像时，才会触发执行。

  使用FROM以这个Dockerfile构建出的镜像为父镜像，构建子镜像时：

  ONBUILD ADD . /app/src：自动执行**ADD . /app/src**

3.1.3 如何制作镜像压缩包?

使用docker save命令可将容器镜像制作成tar或tar.gz文件压缩包，具体命令格式如下：

docker save [OPTIONS] IMAGE [IMAGE...]

OPTIONS说明：--output、-o，表示导出到文件。

示例：

```
$ docker save nginx:latest > nginx.tar
$ ls -sh nginx.tar
108M nginx.tar

$ docker save php:5-apache > php.tar.gz
$ ls -sh php.tar.gz
372M php.tar.gz

$ docker save --output nginx.tar nginx
$ ls -sh nginx.tar
108M nginx.tar

$ docker save -o nginx-all.tar nginx
$ docker save -o nginx-latest.tar nginx:latest
```

3.2 镜像管理类

3.2.1 长期有效的登录指令与临时登录指令的区别是什么?

- 临时的登录指令代指24个小时后会过期失效，不能再被使用的登录指令。
- 长期有效的登录指令有效期为永久。

获取了长期有效的登录指令后，在有效期内的临时登录指令仍然可以使用。

长期有效的登录指令与临时登录指令均不受限制，可以多台机器多人同时登录。

3.2.2 为什么通过客户端和页面上传的镜像大小不一样?

问题描述

假设在本地Docker客户端上制作了一个nginx镜像，版本号为v2.0.0，使用**docker images**命令查询SIZE为**22.8MB**：

```
$ docker images
REPOSITORY                    TAG          IMAGE ID       CREATED        SIZE
nginx                         v2.0.0       22f2bf2e2b4f   9 days ago     22.8MB
```

1. 通过客户端（即执行docker push命令）上传该镜像至SWR镜像仓库，查询大小为**9.5MB**。
2. 在本地Docker客户端将镜像打包为tar格式，将nginx.tar下载至本地后，通过页面方式上传至SWR镜像仓库，查询大小为**23.2MB**。

可以发现，通过客户端和页面上传的镜像大小不一样。

原因分析

使用客户端上传的镜像每一层layer都进行了tgz压缩，而页面上传的镜像包是经过Docker打包的，每一层layer只进行了打包，没有压缩。所以两种方式上传的镜像大小显示会不一致。

3.2.3 控制台页面的容器镜像可以下载到本地吗?

控制台页面的容器镜像不能直接下载至本地，您可以参考以下方法操作：

1. 在镜像详情页获取镜像的下载指令。
2. 在安装了Docker客户端的机器上执行上一步的指令下载镜像。
   示例：
   docker pull {*Image repository address***}/group/nginx:v1**
3. 将镜像保存为.tar或.tar.gz格式的文件。
   示例：
   docker save nginx:v1 > nginx.tar
4. 下载文件至本地。

3.3 故障类

3.3.1 为什么登录指令执行失败?

登录指令执行失败有以下几种情况：

1. 容器引擎未安装正确，报如下图所示错误：

   “docker: command not found”

   解决方法：重新安装容器引擎。

   - 由于容器镜像服务支持Docker容器引擎1.11.2及以上版本上传镜像，建议下载对应版本。
   - 安装容器引擎需要连接互联网，内网服务器需要绑定弹性IP后才能访问。

2. 临时登录指令已过期或登录指令中区域资源空间名称、AK、登录密钥错误，报如下图所示错误：

   “unauthorized: authentication required”

   解决方法：登录容器镜像服务控制台，在左侧菜单栏选择“我的镜像”，单击右侧“客户端上传”获取登录指令。

   a. 获取临时的登录指令：单击“生成临时登录指令”，在弹出的页面中单击 复制登录指令。

   b. 获取长期有效的登录指令：单击“如何获取长期有效登录指令”，参考其中的指导获取。

3. 登录指令中镜像仓库地址错误，报如下所示错误：

   “Error llgging in to v2 endpoint, trying next endpoint: Get https://{{endpoint}}/v2/: dial tcp: lookup {{endpoint}} on xxx.xxx.xxx.xxx:53 : no such host”

   解决方法：

   a. 修改登录指令中的镜像仓库地址。

   b. 获取临时的登录指令：方法请参见2。

4. **x509: certificate has expired or is not yet valid**

   长期有效登录指令中AK/SK被删除导致，请使用有效的AK/SK生成登录指令。

5. **x509: certificate signed by unknown authority**

   问题原因：

   容器引擎客户端和SWR之间使用HTTPS的方式进行通信，客户端会对服务端的证书进行校验。如果服务端证书不是权威机构颁发的，则会报如下错误：x509: certificate signed by unknown authority

   解决方法：

   如果用户信赖服务端，跳过证书认证，那么可以手动配置Docker的启动参数，配置方法如下：

   - CentOS：

     修改“/etc/docker/daemon.json”文件（如果没有，可以手动创建），在该文件内添加如下内容：

     ```
     {
       "insecure-registries": ["{镜像仓库地址}"]
     }
     ```

   - Ubuntu：

     修改“/etc/default/docker”文件，在DOCKER_OPTS配置项中增加如下内容：

     ```
     DOCKER_OPTS="--insecure-registry {镜像仓库地址}"
     ```


- EulerOS：

  修改“/etc/sysconfig/docker”文件，在INSECURE_REGISTRY配置项中增加如下内容：

  ```
  INSECURE_REGISTRY='--insecure-registry {镜像仓库地址}'
  ```

  说明

  镜像仓库地址支持域名和IP形式。

  - 域名形式的镜像仓库地址获取方式：参考2获取临时登录指令，末尾的域名即为镜像仓库地址。
  - IP：可通过ping镜像仓库地址（域名形式）获取。

  配置完成后，执行**systemctl restart docker**重启容器引擎。

6. **denied: Authenticate Error**

   用户无编程访问权限，需要使用管理员账号登录IAM服务，单击用户名称进入用户详情页面，单击访问方式参数后面的 按钮，修改访问方式同时勾选编程访问和管理控制台访问。

7. **denied: Not allow to login、upload or download image**

   用户大批量并发上传镜像或者攻击服务，系统把用户拉黑，用户无法登录和上传下载镜像。请在30分钟之后重新尝试。

3.3.2 为什么使用客户端上传镜像失败?

denied: you do not have the permission

问题现象：使用客户端上传镜像，报如下所示错误：

“denied: you do not have the permission”

问题原因：

- 该组织名已被其他用户注册或当前SWR组织数量已超过配额。
- docker login命令使用VDC业务员的AK、SK生成，没有对应组织的权限。

解决方法：

- 该组织名已被其他用户注册时：建议您先创建组织然后再上传镜像。
- SWR组织数量超过配额时：单个用户的组织数量限制为1000个，您可以将镜像上传到已存在的组织下。
- 没有对应组织的权限：使用账号授权后，可以正常推送。

tag does not exist: xxxxxx 或 An image does not exist locally with the tag: xxxxxx

问题现象：使用客户端上传镜像，报如下图所示错误：

“tag does not exist: xxxxxx”

或

“An image does not exist locally with the tag: xxxxxx”

问题原因：上传的镜像或镜像版本不存在。

解决方法：通过docker images查看本地镜像，确认要上传的镜像名称及版本后，重新上传镜像。

name invalid: 'repository' is invalid

问题现象：使用客户端上传镜像，报如下图所示错误：

“name invalid: 'repository' is invalid”

问题原因：组织命名或镜像命名不规范。

解决方法：以下分别是组织名（namespace）和仓库名（repository）的命名正则表达式：

namespace：^([a-z]+(?:(?:(?:_|__|[-]*)[a-z0-9]+)+)?)$，长度范围为：1-64；

repository：^([a-z0-9]+(?:(?:(?:_|__|[-]*)[a-z0-9]+)+)?)$，长度范围为：1-128。

您可以按照上述命名规范，重新指定上传的组织和镜像名称。

偶现推送镜像超时

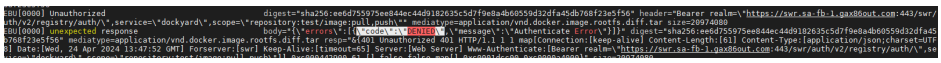
问题现象：偶现推送镜像超时

问题原因：您在国内机器往国外推镜像，网速慢，偶现连接失败。

使用 containerd 容器引擎客户端在 VPC 内上传镜像报 401 鉴权失败

问题现象：在VPC内通过ctr上传镜像至SWR，会报401鉴权失败

```
ctr --debug -n k8s.io images push -k --user {project_id}@{AK}:{SK} swr.sa-fb-1.gax86out.com/test/image:v5
```



说明

HCS 8.1.0或更早版本的swr，通过升级方式升级到新版本后，在使用ctr命令上传镜像时可能会存在该问题。

问题原因：SWR侧返回了一个与Containerd客户端请求Header的Host不一致的Location，导致Containerd不携带认证信息向SWR发起请求，最终鉴权失败。

解决方法：

请根据下面的指导完成ReverseLB和PodLB的规避。

- ReverseLB规避指导

步骤1 以opsadmin用户依次登录ReverseLB_Nginx_Server-01 、ReverseLB_Nginx_Server-02节点执行如下操作。

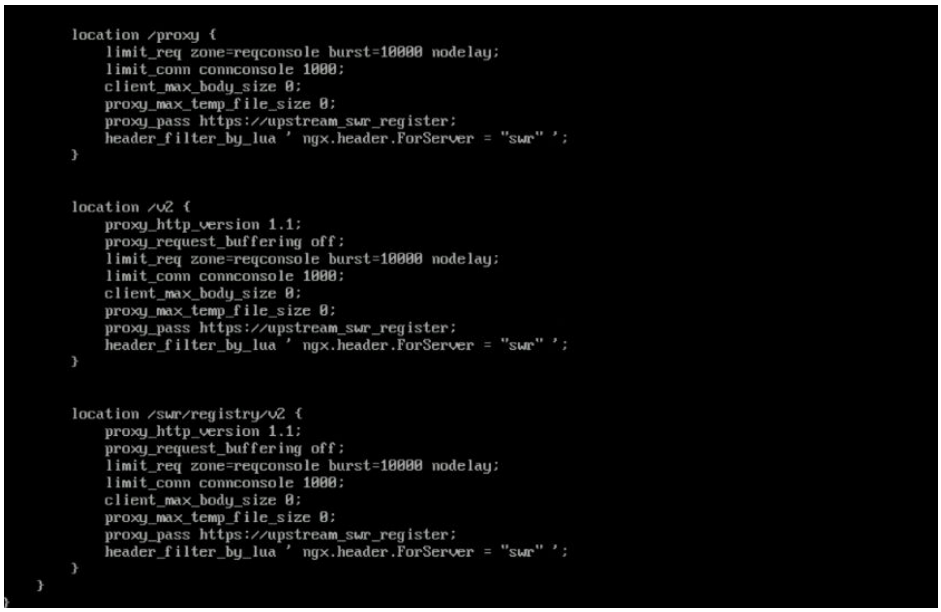
步骤2 切换到root用户。

```
su - root
```

步骤3 修改配置文件 /opt/cloud/nginx/conf/nginx.conf,为指定的location设置“X-Forwarded-Host”头。

```
vim /opt/cloud/nginx/conf/nginx.conf
```

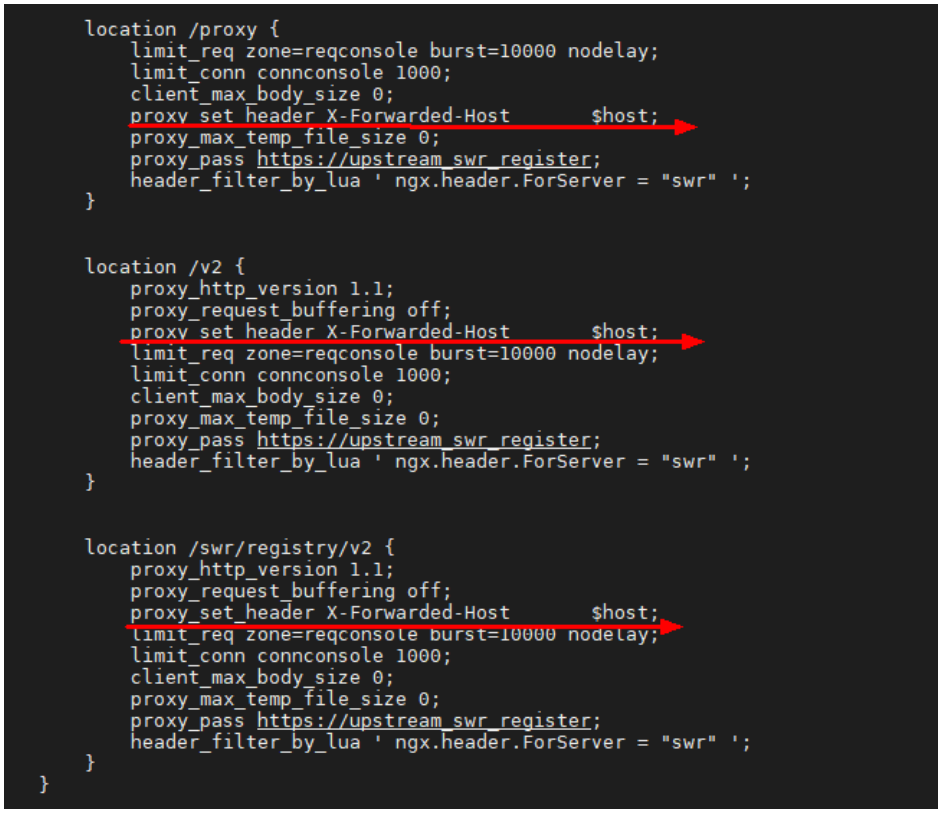
步骤4 找到SWR的location，并进行修改，如下所示：



每一个location在client_max_body_size后增加一行：

```
proxy_set_header X-Forwarded-Host       $host;
```

修改后的效果如下图：



步骤5 修改完成后，先按“Esc”按钮，然后输入:wq! 保存并退出编辑模式。

步骤6 执行如下命令，重启nginx服务完成加固。

```
su - onframework -c "/opt/cloud/nginx/bin/nginx_monitor.sh rstart"
```

----结束

- PodLB规避指导

步骤1 以opsadmin用户依次登录PodLB_Nginx_Server-01 、PodLB_Nginx_Server-02节点执行如下操作。

步骤2 切换到root用户。

```
su - root
```

步骤3 修改配置文件 /opt/cloud/nginx/conf/nginx.conf,为指定的location设置“X-Forwarded-Host”头。

```
vim /opt/cloud/nginx/conf/nginx.conf
```

步骤4 找到SWR的location，并进行修改，如下所示：

```
    location /proxy {
        limit_req zone=reqconsole burst=10000 nodelay;
        limit_conn connconsole 1000;
        client_max_body_size 0;
        proxy_max_temp_file_size 0;
        proxy_pass https://upstream_swr_register;
        header_filter_by_lua ' ngx.header.ForServer = "swr" ';
    }


    location /v2 {
        proxy_http_version 1.1;
        proxy_request_buffering off;
        limit_req zone=reqconsole burst=10000 nodelay;
        limit_conn connconsole 1000;
        client_max_body_size 0;
        proxy_max_temp_file_size 0;
        proxy_pass https://upstream_swr_register;
        header_filter_by_lua ' ngx.header.ForServer = "swr" ';
    }


    location /swr/registry/v2 {
        proxy_http_version 1.1;
        proxy_request_buffering off;
        limit_req zone=reqconsole burst=10000 nodelay;
        limit_conn connconsole 1000;
        client_max_body_size 0;
        proxy_max_temp_file_size 0;
        proxy_pass https://upstream_swr_register;
        header_filter_by_lua ' ngx.header.ForServer = "swr" ';
    }
  }
}
```

每一个location在client_max_body_size后增加一行：

```
proxy_set_header X-Forwarded-Host       $host;
```

修改后的效果如下图：


```
    location /proxy {
        limit_req zone=reqconsole burst=10000 nodelay;
        limit_conn connconsole 1000;
        client_max_body_size 0;
        proxy_set_header X-Forwarded-Host       $host;
        proxy_max_temp_file_size 0;
        proxy_pass https://upstream_swr_register;
        header_filter_by_lua ' ngx.header.ForServer = "swr" ';
    }


    location /v2 {
        proxy_http_version 1.1;
        proxy_request_buffering off;
        proxy_set_header X-Forwarded-Host       $host;
        limit_req zone=reqconsole burst=10000 nodelay;
        limit_conn connconsole 1000;
        client_max_body_size 0;
        proxy_max_temp_file_size 0;
        proxy_pass https://upstream_swr_register;
        header_filter_by_lua ' ngx.header.ForServer = "swr" ';
    }


    location /swr/registry/v2 {
        proxy_http_version 1.1;
        proxy_request_buffering off;
        proxy_set_header X-Forwarded-Host       $host;
        limit_req zone=reqconsole burst=10000 nodelay;
        limit_conn connconsole 1000;
        client_max_body_size 0;
        proxy_max_temp_file_size 0;
        proxy_pass https://upstream_swr_register;
        header_filter_by_lua ' ngx.header.ForServer = "swr" ';
    }
}
```

步骤5 修改完成后，先按“Esc”按钮，然后输入:wq! 保存并退出编辑模式。

步骤6 执行如下命令，重启nginx服务完成加固。

```
su - onframework -c "/opt/cloud/nginx/bin/nginx_monitor.sh rstart"
```

----结束

不允许用户上传镜像

问题现象：使用客户端上传镜像，报如下所示错误：

“Not allow to login、upload or download image”

问题原因：用户大批量并发上传镜像或者攻击服务，系统把用户拉黑，用户无法登录和上传下载镜像。

解决方法：

3.3.3 为什么通过页面上传镜像失败?

SWR对镜像的命名和地址有严格的规范。如果镜像的命名不规范或镜像地址不规范都会导致镜像上传失败。

镜像格式不合法

问题现象：通过页面上传镜像，出现“镜像格式不合法”的报错。

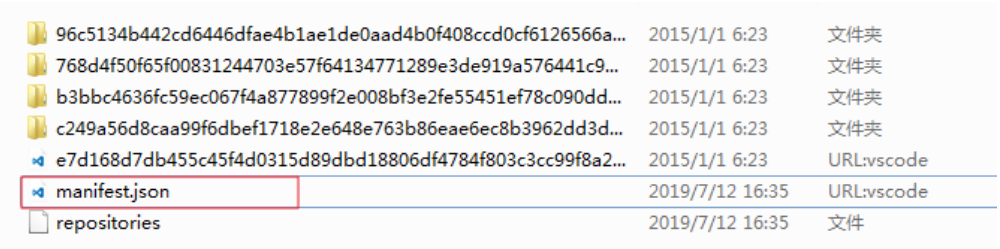
问题原因：镜像地址不规范，导致上传失败。

镜像地址各个部分的含义如下，最后的tag（版本号）可省略，如果省略则表示latest版本，其余部分均不可省略，且不可多余。

样例：registry.example.com/repo_namespace/repo_name:tag

- registry.example.com为容器镜像服务的镜像仓库地址，此处为示例，以实际地址为准。
- repo_namespace为组织名称，命名正则表达式为^([a-z]+(?:(?:(?:_|__|[-]*)[a-z0-9]+)+)?)$，长度范围为：1-64。
- repo_name:tag为镜像名称和版本号，镜像命名正则表达式为^([a-z0-9]+(?:(?:(?:_|__|[-]*)[a-z0-9]+)+)?)$，长度范围为：1-128。

您可以将镜像解压，打开文件manifest.json文件查看RepoTags字段的值是否符合上述规范。



解决方法：按照命名规范，重新给镜像打tag，然后使用docker save命令保存镜像，然后再使用页面上传。

不允许用户上传镜像

问题现象：使用页面上传镜像，报如下所示错误：

“Not allow to login、upload or download image”

问题原因：用户大批量并发上传镜像或者攻击服务，系统把用户拉黑，用户无法登录和上传下载镜像。

解决方法：

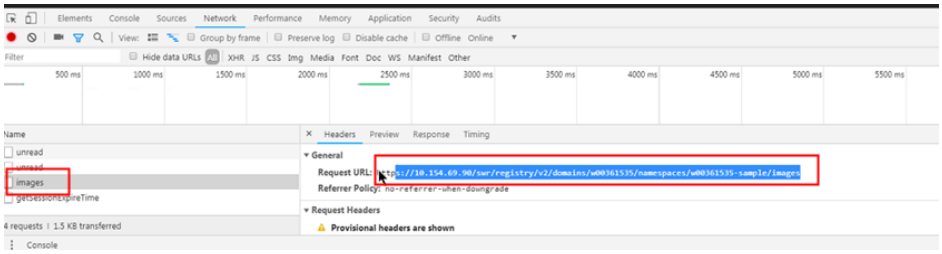
客户端上传镜像失败，报错：网络异常，请检查网络状况

问题现象：客户端上传镜像失败，报错：网络异常，请检查网络状况。

问题原因：浏览器没有信任待访问的SWR服务地址。

解决方法：

1. 排查确认并解决问题。
   - 打开F12, 重新上传镜像，找到红色的images接口。



   - 复制请求失败的地址，即images接口的Headers里面的Request URL的值。打开新的页面粘贴该地址并进行访问。

- 如果提示需要信任该站点，同意即可，之后返回结果404，至此问题已解决。
- 如果images接口访问超时，则说明本机网络和SWR服务不通，请联系管理员进行处理。

2. 重新上传镜像即可。

一直卡在上传界面直到超时

问题现象：通过页面上传镜像，一直卡在上传界面直到超时。

问题原因：镜像命名不规范，导致上传失败。

解决方法：您可以按照镜像命名规范修改镜像名称后，重新上传镜像。

须知

SWR判定镜像名是否合法不是以用户在界面上传镜像时的文件名为依据，而是依据镜像包中的repositories和manifest.json文件。

3.3.4 为什么镜像拉取失败?

x509: certificate signed by unknown authority

问题现象：使用docker pull拉取镜像，报错“x509: certificate signed by unknown authority”。

问题原因：

- 容器引擎客户端和SWR之间使用HTTPS的方式进行通信，client会对服务端的证书进行校验，如果客户端安装的根证书不完整，会报如下错误：“x509: certificate signed by unknown authority”。
- 容器引擎客户端配置了Proxy导致。

解决方法：

- 如果您信赖服务端，跳过证书认证，那么可以手动配置容器引擎的启动参数，配置如下（其中地址配置成需要的即可，选择一个配置即可）：
  - /etc/docker/daemon.json（如果没有可以手动创建），在该文件内添加如下配置（注意缩进，2个空格）：
    ```
    {
      "insecure-registries":["镜像仓库地址"]
    }
    ```
  - /etc/sysconfig/docker：
    ```
    INSECURE_REGISTRY='--insecure-registry=镜像仓库地址'
    ```

  添加配置后执行如下命令重启：**systemctl restart docker**或**service restart docker**。
- 执行**docker info**命令，检查proxy配置是否正确，修改为正确的proxy配置。

x509: certificate has expired or is not yet valid

问题现象：使用docker pull拉取镜像，报错“x509: certificate has expired or is not yet valid”。

问题原因：

非通过CCE创建的节点因未配置 docker/conatinerd 白名单且未使用商用证书，导致工具校验证书不通过。如果CCE集群节点有同样问题，请联系运维工程师处理。

解决方法：

如果您有有效的商用证书请替换有效的商用证书即可。

规避方法：

如果您暂时没有商用证书也可通过以下方式进行规避：

- **docker 容器引擎客户端规避方法**

```
修改docker.service配置
# 修改service配置
vim /usr/lib/systemd/system/docker.service
# 按i增加配置，${}要修改成环境对应的值，注意只是增加配置，不要修改原有的
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock \
    --insecure-registry swr.${region_id}.${external_global_domain_name}
# esc保存退出
:wq
systemctl daemon-reload
# 重新启动docker
systemctl restart docker
```

注意：swr.${region_id}.${external_global_domain_name} 即swr页面生成docker login指令最后面那一段的地址

说明

windows 或 mac 版本文件所在的位置：C:\Program Files\Docker\Docker\resources \windows-daemon-options.json 或 C:\ProgramData\docker\config\daemon.json 。也可以直接在页面单击设置：Docker Engine，把域名加入到 insecure-registries 中，写法与上面相同，也需要应用和重启。

- **containerd 容器引擎客户端规避方法**

```
# 输入命令编辑
vim /etc/containerd/config.toml
# 按i输入，找到[plugins."io.containerd.grpc.v1.cri".registry]，在下级目录添加以下内容，${}要修改成环境对应的值
[plugins."io.containerd.grpc.v1.cri".registry.configs]
    [plugins."io.containerd.grpc.v1.cri".registry.configs."swr.${region_id}.${external_global_domain_name}".tls]
        insecure_skip_verify = true
# esc 保存退出
:wq
# 重新启动containerd
systemctl restart containerd
```

注意：swr.${region_id}.${external_global_domain_name} 即swr页面生成docker login指令最后面那一段的地址。

> **⚠ 注意**
>
> containerd 容器引擎客户端场景下，如果按上述方案修改了后再次使用ctr image pull/push拉取仍然报错，可以直接在push或pull命令末尾追加空格 -k来规避；如果是使用nerdctl工具，请在命令末尾追加--insecure-registry来规避。

Error: remote trust data does not exist

问题现象：使用docker pull拉取镜像，报错“Error: remote trust data does not exist”。

问题原因：客户端开启镜像签名验证，而镜像没有镜像签名层。

解决方法：查看环境变量是否设置了**DOCKER_CONTENT_TRUST=1**，如果设置了，请修改“/etc/profile”文件将DOCKER_CONTENT_TRUST=1去掉，然后执行**source /etc/profile**生效。

不允许用户下载镜像

问题现象：使用客户端下载镜像，报如下所示错误：

“Not allow to login、upload or download image”

问题原因：用户大批量并发上传镜像或者攻击服务，系统把用户拉黑，用户无法登录和上传下载镜像。

解决方法：

3.3.5 为什么已有镜像自动同步不成功?

镜像自动同步是帮助您把最新推送的镜像自动同步到其他区域镜像仓库内，后期镜像有更新时，目标仓库的镜像也会自动更新，但已有的镜像不会自动同步。

已有镜像的同步，需要手动操作：

1. 在镜像详情页面的“镜像版本”页签选择具体的镜像版本后，单击“镜像同步”。
2. 在弹出的对话框中，选择镜像同步的目标区域、目标组织及是否覆盖。

3.3.6 他人共享的镜像，是否可以再进行镜像同步?

问题描述：是否可以把账号A共享给账号B的镜像，再同步给账号B下的不同区域?

原因分析：不可以。账号A共享给账号B的镜像，被共享者只有该镜像的只读权限，即只能下该镜像。而同步该镜像需要账号及具有管理员权限的用户才能操作。

3.3.7 为什么镜像自动同步后在同步区域看不到镜像?

问题描述

镜像自动同步后在同步区域看不到镜像。例如：创建自动同步任务，将regionA的nginx_01镜像同步到regionB区域，同步成功后，在regionB区域看不到nginx_01镜像。

原因分析

当手动同步单个版本镜像或批量手动同步多个版本镜像时，同步才是及时生效的。

如果您创建的是自动同步任务，同步是在下一次版本变动或者有镜像变动时才会同步到另外一个区域，所以当您上传一个新版本nginx_01镜像后，您在regionB区域才能看到新上传的镜像。

3.3.8 为什么创建组织失败?

问题现象：创建组织失败，页面提示该组织已经存在，但在“组织管理”页面没有查询到该组织。

问题原因：组织名称全局唯一，即当前区域下，组织名称唯一。创建组织时如果提示组织已存在，可能该组织名称已被其他用户使用。

解决办法：请重新设置一个组织名称。