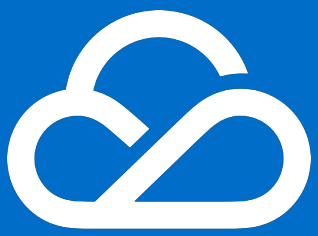
腾讯云容器服务

镜像仓库

产品文档



腾讯云

【版权声明】

【商标声明】

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

镜像仓库

镜像仓库概述

镜像仓库概述

镜像仓库用于存放Docker镜像，Docker镜像用于部署容器服务，每个镜像有特定的唯一标识（镜像的Registry地址+镜像名称+镜像Tag）。

镜像类型

目前镜像支持Docker Hub官方镜像和用户私有镜像以及TencnetHub镜像。

使用帮助

- 镜像仓库基本操作
- 如何构建docker镜像
- 如何搭建私有镜像仓库
- 使用DockerHub加速器

镜像仓库基本教程

开通镜像仓库

1. 登录 腾讯云容器服务控制台 。
2. 单击左侧导航栏中的【镜像仓库】，单击下拉列表页的【我的镜像】。初次使用“我的镜像库”，需要先进行初始化。
   - 命名空间：命名空间是您创建的私人镜像地址的前缀。命名空间将用于对您的容器镜像进行分类。
   - 用户名：默认是当前用户的账号 ID。
   - 密码：密码用于通过 docker login 来登陆腾讯云容器镜像仓库。

创建镜像

1. 在 我的镜像库 下 我的创建 页面中单击【+新建】。

2. 设置镜像仓库的基本信息。
   - 名称：新建镜像仓库的名称。
   - 类型：选择私有或公有类型。
   - 命名空间：命名空间是您创建的镜像地址的前缀。
   - 描述：创建镜像仓库相关信息。该信息将显示在 镜像信息 页面。

推送镜像到镜像仓库

1. 在 我的镜像 页面，单击名称（如图中 ewe）。

2. 单击【上传镜像】。

3. 进入 使用指引 页面，根据提示步骤将镜像推送到镜像仓库。

登录腾讯云 docker registry

[username]是您的腾讯云 QQ 号，输入密码后即登陆完成,密码是您开通 namespace 时设置的密码。

$ sudo docker login --username=[username] ccr.ccs.tencentyun.com

上传镜像

$ sudo docker tag [ImageId] ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[tag]
$ sudo docker push ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[tag]

- [ImageId] 请根据您的实际镜像 ID 信息进行填写。

- [tag] 请根据您的镜像版本信息进行填写。
- [namespace] 是开通镜像仓库时填写的命名空间。
- [ImageName] 是在控制台创建的镜像名称。

下载镜像

[username] 是您的腾讯云 QQ 号，输入密码后即登陆完成,密码是您开通 namespace 时设置的密码。

```
$ sudo docker login --username=[username] ccr.ccs.tencentyun.com
```

下载镜像。[tag] 请根据您的镜像版本信息进行填写。

```
$ sudo docker pull ccr.ccs.tencentyun.com/[namespace]/[ImageName]:[tag]
```

创建服务

镜像创建完成后，可直接在此镜像下创建服务。

1. 在 我的镜像 页面，单击右侧【创建服务】。
2. 进入创建服务页面，有关创建服务的基本操作请参阅 服务的基本操作。

删除镜像

1. 在 我的镜像 页面，单击右侧【删除】。
2. 单击【确定】。

> 注意：
> 删除镜像仓库时，该镜像仓库下的所有镜像版本将一并销毁，请提前备份好数据。

构建配置

1. 在 我的镜像 页面，单击右侧【构建配置】。
2. 进入创建服务页面，有关构建配置的操作请参阅 构建规则设置。

使用DockerHub加速器

Docker软件源地址：

https://mirror.ccs.tencentyun.com

。

CCS 集群 CVM 主机

无需手动配置，在创建节点时会自动安装 Docker 服务，配置 Mirror 镜像。配置项如下：

[root@VM_1_2_centos ~]# cat /etc/docker/dockerd
IPTABLES="--iptables=false"
STORAGE_DRIVER="--storage-driver=overlay2"
IP_MASQ="--ip-masq=false"
LOG_LEVEL="--log-level=warn"
REGISTRY_MIRROR="--registry-mirror=https://mirror.ccs.tencentyun.com"

CVM 主机配置

Linux：

- 适用于 Ubuntu14.04、Debian、CentOS6 、Fedora 和 OpenSUSE
  版本，其他版本可能有细微不同。
  修改 Docker 配置文件

  /etc/default/docker

  ，如下：DOCKER_OPTS="--registry-mirror=https://mirror.ccs.tencentyun.com"

- 适用于 Centos7 版本。

修改 Docker 配置文件

vi /etc/sysconfig/docker

，如下：OPTIONS='--registry-mirror=https://mirror.ccs.tencentyun.com'

注意：
Docker 1.3.2 版本以上才支持 Docker Hub Mirror 机制，如果您还没有安装 Docker 或者版本过低，请安装或升级版本。

Windows：

如果你使用的是 Boot2Docker，进入 Boot2Docker Start Shell 并执行：

sudo su echo "EXTRA_ARGS=\"–registry-mirror=http://https://mirror.ccs.tencentyun.com""> > /var/lib/boot2docker/profile exit # 重启Boot2Docker

启动 Docker

执行如下命令

sudo service docker start

如何搭建私有镜像仓库

本文档旨在介绍如何通过 Docker Compose 搭建一个简单的 registry 环境，使用 DockerHub 官方镜像，registry 镜像版本 为

registry:2.5.0

， nginx 镜像版本为

nginx:1.11.5

。这里主要介绍 registry 环境的搭建及使用，更详细的企业级 registry 服务器的搭建可参阅开源的 Harbor。

registry 概述

registry 是 Docker 的镜像存储服务，DockerHub 上的 registry 镜像见 Registry 官方镜像 ，更多详细信息请转至 GitHub 查看最新源码 。

搭建 registry

- 在服务器上执行如下命令安装 Docker，这里选择腾讯云（Ubuntu Server 14.04.1 LTS 64位）镜像来创建服务器。

  curl -fsSL https://get.docker.com/ | sh

- 安装 Docker Compose
  Docker Compose 是一个定义及运行多个 Docker 容器的工具。使用 Docker Compose 只需要在一个配置文件中定义多个 Docker 容器，再使用一条命令将多个容器启动，Docker Compose 会通过解析容器间的依赖关系，按先后顺序启动所定义的容器。有关 Docker Compose 详情请转至 GitHub 了解。

  curl -L
  https://github.com/docker/compose/releases/download/1.8.0/docker-compose-$(uname

-s)-$(uname -m) > /usr/local/bin/docker-compose
chmod a+x /usr/local/bin/docker-compose

- 启动 registry 服务，此例中包含 nginx 和 registry 两个容器，涉及的配置文件请参见附录。

  docker-compose up -d

- 停止服务。

  docker-compose stop

- 重启服务。

  docker-compose restart

- 下线服务。

  docker-compose down

镜像基本操作

上传镜像

- 因为搭建的 registry 服务使用 HTTP 协议，所以 Docker 启动参数需要配置

  --insecure-registry localhost

  选项，修改

/etc/default/docker

文件如下：

DOCKER_OPTS="--insecure-registry localhost"

- 重启 Docker。

  service docker restart

- 拉取上传镜像 docker pull；docker tag；docker push（tag 默认为 latest）。

  docker pull hello-world
  docker tag hello-world localhost/library/hello-world
  docker push localhost/library/hello-world

下载镜像

```
docker pull localhost/library/hello-world
```

删除镜像

```
docker rmi localhost/library/hello-world
```

获取镜像

获取镜像仓库列表

```
# curl http://localhost/v2/_catalog
{"repositories":["library/hello-world"]}
```

未上传镜像前的输出如下：

```
# curl http://localhost/v2/_catalog
{"repositories":[]}
```

获取镜像 tag 列表

```
# curl -X GET http://localhost/v2/library/hello-world/tags/list
{"name":"library/hello-world","tags":["latest"]}
```

获取镜像 manifests 信息

```
# curl -H "Accept: application/vnd.docker.distribution.manifest.v2+json" -X GET
http://localhost/v2/library/hello-world/manifests/latest
{
"schemaVersion": 2,
"mediaType": "application/vnd.docker.distribution.manifest.v2+json",
"config": {
"mediaType": "application/vnd.docker.container.image.v1+json",
"size": 1473,
"digest": "sha256:c54a2cc56cbb2f04003c1cd4507e118af7c0d340fe7e2720f70976c4b75237dc"
},
"layers": [
{
"mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
"size": 974,
"digest": "sha256:c04b14da8d1441880ed3fe6106fb2cc6fa1c9661846ac0266b8a5ec8edf37b7c"
}
```

```
]
}
```

其中

c54a2cc56cbb2f04003c1cd4507e118af7c0d340fe7e2720f70976c4b75237dc

即为执行 docker images 时显示的 IMAGE ID。
layers 表示了镜像的层次关系，可以通过 layers 中的 digest 来拉取 blob，详情见下面获取镜像 blob。

获取镜像 blob

在上面获取 hello-world:latest 镜像的 manifests 信息中只有一个 layer，以此为例来说明如何获取镜像 blob。拉取的结果显示获取的 blob 与文件 sha256 是一致的。执行 docker pull 实际上就是先获取到镜像的 manifests 信息，再拉取 blob。

```
# curl -s -X GET http://localhost/v2/library/hello-world/blobs/sha256:c04b14da8d1441880ed3fe6106fb2cc6fa1c9661846ac0266b8a5ec8edf37b7c -o hello-world.blob
# ls -l hello-world.blob
-rw-r--r-- 1 root root 974 Nov 23 09:56 hello-world.blob
# sha256sum hello-world.blob
c04b14da8d1441880ed3fe6106fb2cc6fa1c9661846ac0266b8a5ec8edf37b7c hello-world.blob
```

删除镜像

删除镜像（ soft delete ）

首先通过 curl -i 参数获取到镜像的

Docker-Content-Digest

，registry 2.3 版本及以后的版本必须在 header 中指定

Accept: application/vnd.docker.distribution.manifest.v2+json

，否则默认返回的是 schema1 的 digest，与 schema2 的 digest 不同，使用不指定上述头信息返回的 digest 删除时会返回 404。

```
 # curl -i -H "Accept: application/vnd.docker.distribution.manifest.v2+json" -X GET http://localhost/v2/library/hello-world/manifests/latest

 HTTP/1.1 200 OK
 Server: nginx/1.11.5
 Date: Wed, 23 Nov 2016 02:17:51 GMT
 Content-Type: application/vnd.docker.distribution.manifest.v2+json
 Content-Length: 524
 Connection: keep-alive
 Docker-Content-Digest: sha256:a18ed77532f6d6781500db650194e0f9396ba5f05f8b50d4046b294ae5f83aa4
 Docker-Distribution-Api-Version: registry/2.0
 Etag: "sha256:a18ed77532f6d6781500db650194e0f9396ba5f05f8b50d4046b294ae5f83aa4"

 {
 "schemaVersion": 2,
 "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
 "config": {
 "mediaType": "application/vnd.docker.container.image.v1+json",
 "size": 1473,
 "digest": "sha256:c54a2cc56cbb2f04003c1cd4507e118af7c0d340fe7e2720f70976c4b75237dc"
 },
 "layers": [
 {
 "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
 "size": 974,
 "digest": "sha256:c04b14da8d1441880ed3fe6106fb2cc6fa1c9661846ac0266b8a5ec8edf37b7c"
```

```
}
]
}
```

根据上一步返回的

Docker-Content-Digest

删除，返回 202 表示删除成功。

```
# curl -k -v -s -X DELETE http://localhost/v2/library/hello-world/manifests/sha256:a18ed77532f6d6781500db650194e0f9396ba5f05f8b50d4046b294ae5f83aa4

* Hostname was NOT found in DNS cache
* Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> DELETE /v2/library/hello-world/manifests/sha256:a18ed77532f6d6781500db650194e0f9396ba5f05f8b50d4046b294ae5f83aa4 HTTP/1.1
> User-Agent: curl/7.35.0
> Host: localhost
> Accept: */*
>
< **HTTP/1.1 202 Accepted**
* Server nginx/1.11.5 is not blacklisted
< Server: nginx/1.11.5
< Date: Wed, 23 Nov 2016 02:29:59 GMT
< Content-Type: text/plain; charset=utf-8
< Content-Length: 0
< Connection: keep-alive
< Docker-Distribution-Api-Version: registry/2.0
<
* Connection #0 to host localhost left intact
```

确认结果：

```
# curl -X GET http://localhost/v2/library/hello-world/tags/list
{"name":"library/hello-world","tags":null}
```

删除镜像（ hard delete ）

在上一步中，只是删除了镜像的 manifests 信息，引用的 blob
还在占用磁盘空间，执行如下命令可以查看可以删除的 blob。

```
docker exec -it myregistry_registry_1 /bin/registry garbage-collect --dry-run /etc/registry/config.yml
```

要删除 blob，释放磁盘空间，需要执行下面的命令。

> 注意：
> 在执行下面的命令时 registry 必须是只读模式（只读模式可在 registry
> 配置文件中设置），否则可能会导致数据不一致。
>
> ```
> docker exec -it myregistry_registry_1 /bin/registry garbage-collect /etc/registry/config.yml
> ```

附录

目录结构

```
.
|-- config
| |-- nginx
| | `-- nginx.conf
| `-- registry
| `-- config.yml
```

```
`-- docker-compose.yml
```

nginx.conf

```
worker_processes auto;

events {
  worker_connections 1024;
  use epoll;
  multi_accept on;
}

http {
  tcp_nodelay on;

  # this is necessary for us to be able to disable request buffering in all cases
  proxy_http_version 1.1;


  upstream registry {
  server registry:5000;
  }


  server {
  listen 80;

  # disable any limits to avoid HTTP 413 for large image uploads
  client_max_body_size 0;

  location /v1/ {
```

```
  return 404;
  }

  location /v2/ {
  proxy_pass http://registry/v2/;
  proxy_set_header Host $http_host;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

  # When setting up Harbor behind other proxy, such as an Nginx instance, remove the below line if the proxy already has similar settings.
  proxy_set_header X-Forwarded-Proto $scheme;

  proxy_buffering off;
  proxy_request_buffering off;

  }

  }
}
```

config.yml

```
version: 0.1
log:
  level: debug
  fields:
  service: registry
storage:
  cache:
  layerinfo: inmemory
```

```
  filesystem:
  rootdirectory: /var/lib/registry
  maintenance:
  uploadpurging:
  enabled: false
  readonly:
  enabled: false
  delete:
  enabled: true
http:
  addr: :5000
  secret: yoursecret
```

docker-comose.yaml

```
version: '2'
services:
  registry:
  image: library/registry:2.5.0
  restart: always
  volumes:
  - /data/registry:/var/lib/registry
  - ./config/registry/:/etc/registry/
  environment:
  - GODEBUG=netdns=cgo
  command:
  ["serve", "/etc/registry/config.yml"]
  proxy:
  image: library/nginx:1.11.5
  restart: always
  volumes:
```

```
- ./config/nginx:/etc/nginx
ports:
- 80:80
- 443:443
depends_on:
- registry
```

如何构建docker镜像

说明

DockerHub 提供了大量的镜像可用，详情可查看 DockerHub 官网。

Docker 容器的设计宗旨是让用户在相对独立的环境中运行独立的程序。

Docker 容器程序在镜像内程序运行结束后会自动退出。如果要令构建的镜像在服务中持续运行，需要在创建服务页面指定自身持续执行的程序，如：业务主程序，main 函数入口等。

由于企业环境的多样性，并非所有应用都能在 DockerHub 找到对应的镜像来使用。
你可以通过以下教程了解到如何将应用打包成Docker镜像。

Docker 生成镜像目前有两种方式：

- 通过 Dockerfile 自动构建镜像；
- 通过容器操作，并执行 Commit 打包生成镜像。

Dockerfile 自动编译生成（推荐使用）

以 Dockerhub 官方提供的 WordPress 为例，转到 github 查看详情 >>

其 Dockfile 源码如下：

```
FROM php:5.6-apache

# install the PHP extensions we need
RUN apt-get update && apt-get install -y libpng12-dev libjpeg-dev && rm -rf /var/lib/apt/lists/* \
  && docker-php-ext-configure gd --with-png-dir=/usr --with-jpeg-dir=/usr \
  && docker-php-ext-install gd mysqli opcache

# set recommended PHP.ini settings
# see https://secure.php.net/manual/en/opcache.installation.php
```

```
RUN { \
  echo 'opcache.memory_consumption=128'; \
  echo 'opcache.interned_strings_buffer=8'; \
  echo 'opcache.max_accelerated_files=4000'; \
  echo 'opcache.revalidate_freq=2'; \
  echo 'opcache.fast_shutdown=1'; \
  echo 'opcache.enable_cli=1'; \
  } > /usr/local/etc/php/conf.d/opcache-recommended.ini

RUN a2enmod rewrite expires

VOLUME /var/www/html

ENV WORDPRESS_VERSION 4.6.1
ENV WORDPRESS_SHA1 027e065d30a64720624a7404a1820e6c6fff1202

RUN set -x \
  && curl -o wordpress.tar.gz -fSL
"https://wordpress.org/wordpress-${WORDPRESS_VERSION}.tar.gz" \
  && echo "$WORDPRESS_SHA1 *wordpress.tar.gz" | sha1sum -c - \
# upstream tarballs include ./wordpress/ so this gives us /usr/src/wordpress
  && tar -xzf wordpress.tar.gz -C /usr/src/ \
  && rm wordpress.tar.gz \
  && chown -R www-data:www-data /usr/src/wordpress

COPY docker-entrypoint.sh /usr/local/bin/
RUN ln -s usr/local/bin/docker-entrypoint.sh /entrypoint.sh # backwards compat

# ENTRYPOINT resets CMD
ENTRYPOINT ["docker-entrypoint.sh"]
CMD ["apache2-foreground"]
```

通过上述 Dockerfile 文件可以了解到，内置执行了许多的 Linux 命令来安装和部署软件。

在终端创建一个文件夹来保存该 Dockerfile 文件，并通过 docker build 命令来构建镜像。

```
[root@VM_88_88_centos worldpress]# docker build ./
Sending build context to Docker daemon 3.072 kB
Step 1 : FROM php:5.6-apache
Trying to pull repository docker.io/library/php ...
5.6-apache: Pulling from docker.io/library/php
386a066cd84a: Pull complete
269e95c6053a: Pull complete
.....
```

通过 docker images 命令即可查看到构建完成的镜像。

```
[root@VM_88_88_centos worldpress]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
worldpress latest 9f0b470b5ddb 12 minutes ago 420 MB
docker.io/php 5.6-apache eb8333e24502 5 days ago 389.7 MB
```

使用 Dockerfile 来构建镜像有以下建议：

1. 尽量精简，不安装多余的软件包。
2. 尽量选择 Docker 官方提供镜像作为基础版本，减少镜像体积。
3. Dockerfile 开头几行的指令应当固定下来，不建议频繁更改，有效利用缓存。
4. 多条 RUN 命令使用'\'连接，有利于理解且方便维护。
5. 通过 -t 标记构建镜像，有利于管理新创建的镜像。
6. 不在 Dockerfile 中映射公有端口。
7. Push 前先在本地运行，确保构建的镜像无误。

执行 Commit 实现打包生成镜像

通过 Dockerfile 可以快速构建镜像，而通过 commit
生成镜像可以解决应用在部署过程中有大量交互内容，通过 Dockerfile 难以构建的问题。

通过 commit 构建镜像操作如下：

1. 运行基础镜像容器，并进入console。[root@VM_88_88_centos ~]# docker run -i -t centos
   [root@f5f1beda4075 /]#

2. 安装需要的软件，并添加配置。

   [root@f5f1beda4075 /]# yum update && yum install openssh-server
   Loaded plugins: fastestmirror, ovl
   base | 3.6 kB 00:00:00
   extras | 3.4 kB 00:00:00
   updates | 3.4 kB 00:00:00
   (1/4): base/7/x86_64/group_gz | 155 kB 00:00:00
   (2/4): extras/7/x86_64/primary_db | 166 kB 00:00:00
   (3/4): base/7/x86_64/primary_db | 5.3 MB 00:00:00
   (4/4): updates/7/x86_64/primary_db
   ......
   ......
   ......
   Dependency Installed:
   fipscheck.x86_64 0:1.4.1-5.el7 fipscheck-lib.x86_64 0:1.4.1-5.el7 openssh.x86_64 0:6.6.1p1-25.el7_2 tcp_wrappers-libs.x86_64 0:7.6-77.el7
   Complete!

3. 配置完成后打开新终端保存该镜像。

   shell
   [root@VM_88_88_centos ~]# docker ps
   CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
   f5f1beda4075 centos "/bin/bash" 8 minutes ago Up 8 minutes hungry_kare
   [root@VM_88_88_centos ~]# docker commit f5f1beda4075 test:v1.0
   sha256:65325ffd2af9d574afca917a8ce81cf8a710e6d1067ee611a87087e1aa88e4a4

```
[root@VM_88_88_centos ~]#
[root@VM_88_88_centos ~]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
test v1.0 65325ffd2af9 11 seconds ago 307.8 MB
```

镜像构建

镜像构建概述

镜像构建概述

容器持续集成提供在腾讯云容器平台上，自动、手动构建容器镜像的功能。

自动构建

容器镜像自动构建基于github或者gitlab代码仓库，
要求代码仓库里面必须包含Dockerfile文件
。用户需要先注册github
或者gitlab服务器的token，如果代码仓库使用的
是gitlab，gitlab服务器要求必须能够公网访问
。用户可以针对代码仓库设置自动构建规则，当用户往代码仓库发起push操作时，如果符合自动构建规则，那么就会在腾讯云容器平台上进行容器镜像的自动构建，并将构建出来的容器镜像自动推送到腾讯云容器镜像仓库中。
自动构建需要执行以下步骤：

- 第一步：源代码仓库授权
- 第二步：构建规则设置
- 第三步：提交代码自动执行构建

手动构建

手动构建分两类：

- 基于github和gitlab代码仓库
  跟自动构建类似，同样代码仓库需要包含Dockerfile文件，以及如果代码仓库位于gitlab的话，gitlab需要支持公网访问。跟自动构建不同的是，自动构建的构建规则设置了，用户对仓库发起push操作时，会自动构建容器镜像，而手动构建需要用户在控制台上手动点击构建才会构建。

- 基于上传的Dockerfile
  用户可以在镜像仓库的控制台上传一个Dockerfile文件，腾讯云容器平台根据这个Dockerfile来构建容

器镜像。

构建说明

- 对于git仓库中的Dockerfile或者用户手动上传的Dockerfile，如果Dockerfile里依赖了外部资源，外部资源也必须能够公网访问。
- 手动构建和自动构建都是在腾讯云容器平台上进行的，用户无需提供构建环境和服务器资源。

源代码仓库授权

如果用户需要从 Git 代码仓库来构建容器镜像，首先必须进行源代码授权。目前我们支持 Github 仓库和 Gitlab 仓库授权。

操作步骤

1. 登录 腾讯云容器服务控制台 。
2. 单击左侧导航栏中的 【镜像仓库】，在下拉列表中单击【我的镜像】，进入 我的镜像库 页面，单击【源代码授权】。

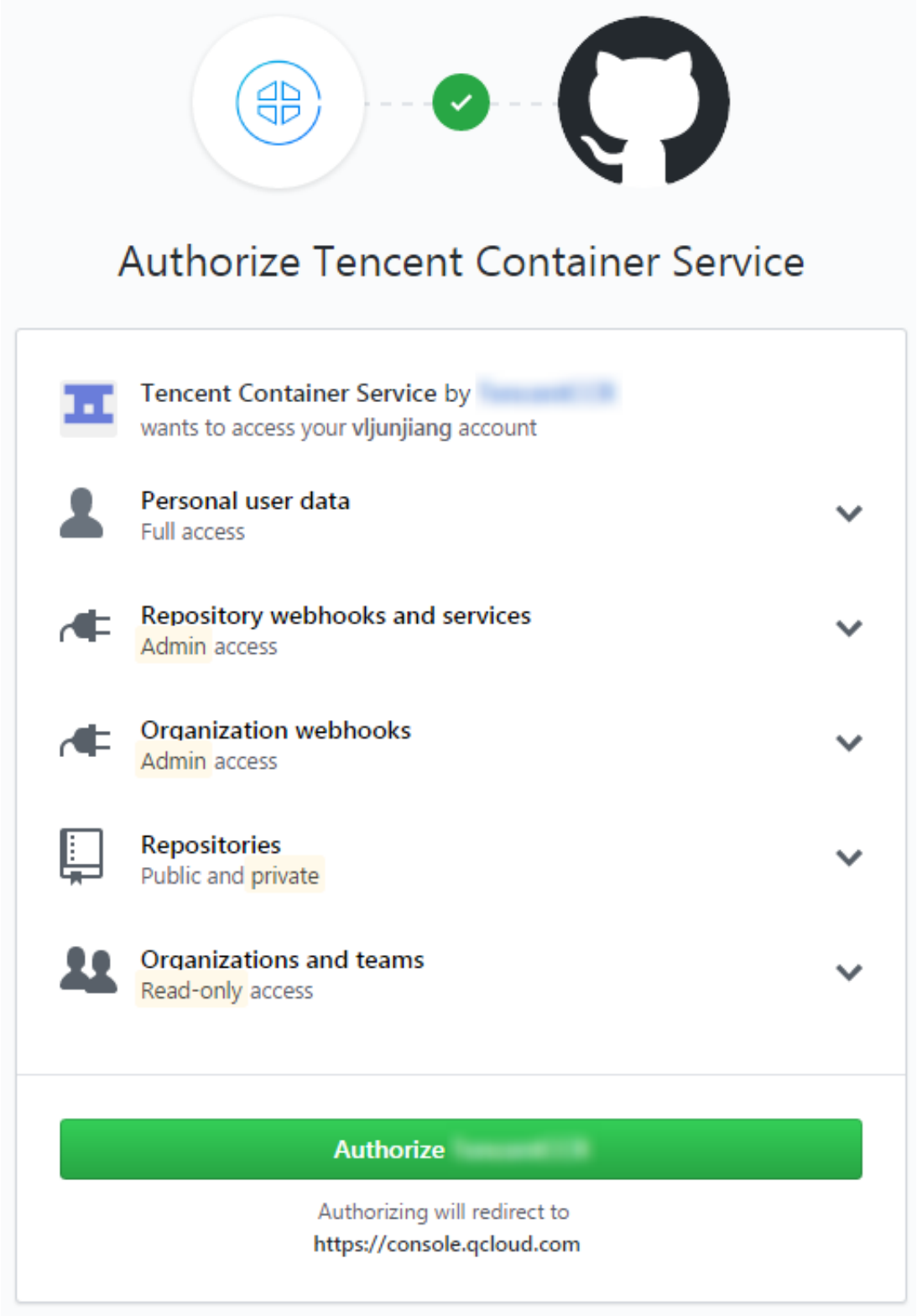
3. 进入代码源授权选择页面。

Github 授权

如果您的代码仓库位于 Github，那么请单击【授权同步 Github 代码源】。

跳转至 Github 网站，Github 会提示 APP

需要访问用户的代码仓库、个人信息等数据，单击【authorize】完成 Github 代码仓库的授权。

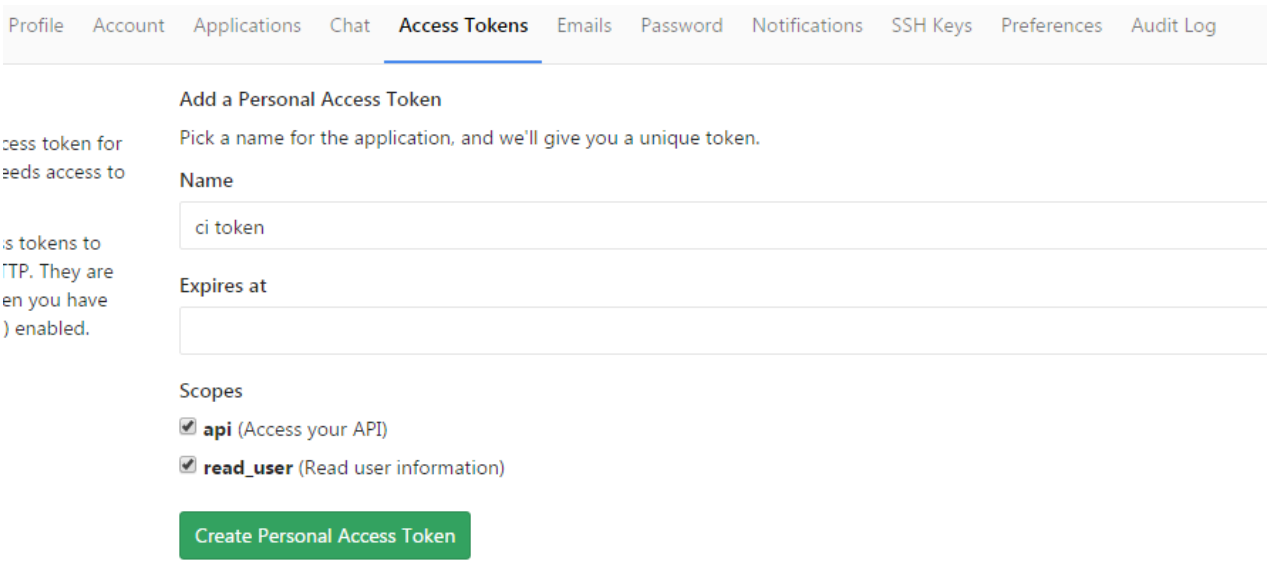


Gitlab 授权

如果您的代码仓库位于自建的 Gitlab 服务器，或者官方的 Gitlab 托管服务器，那么请单击【授权同步 Gitlab 代码源】。

跳转至 Gitlab 代码源授权 页面。

- 服务地址：Gitlab 服务器地址。必须包含 http 或者 https 协议，并且公网必须能够访问该地址。
- 用户名：您在 Gitlab 上注册的用户名。在 Gitlab 页面的 Profile -> Name 可以查看您的用户名。
- 私有Token：必须是 Access Token，如果您没有 Access Token，在 Gitlab 上可以创建一个 Access Token：



注意：

每个用户可以同时授权 Github 和 Gitlab 帐号，但是 Github 和 Gitlab

帐号分别只能授权一个帐号，如果需要更改 Github 或者 Gitlab 帐号，则需要先注册原来的帐号。

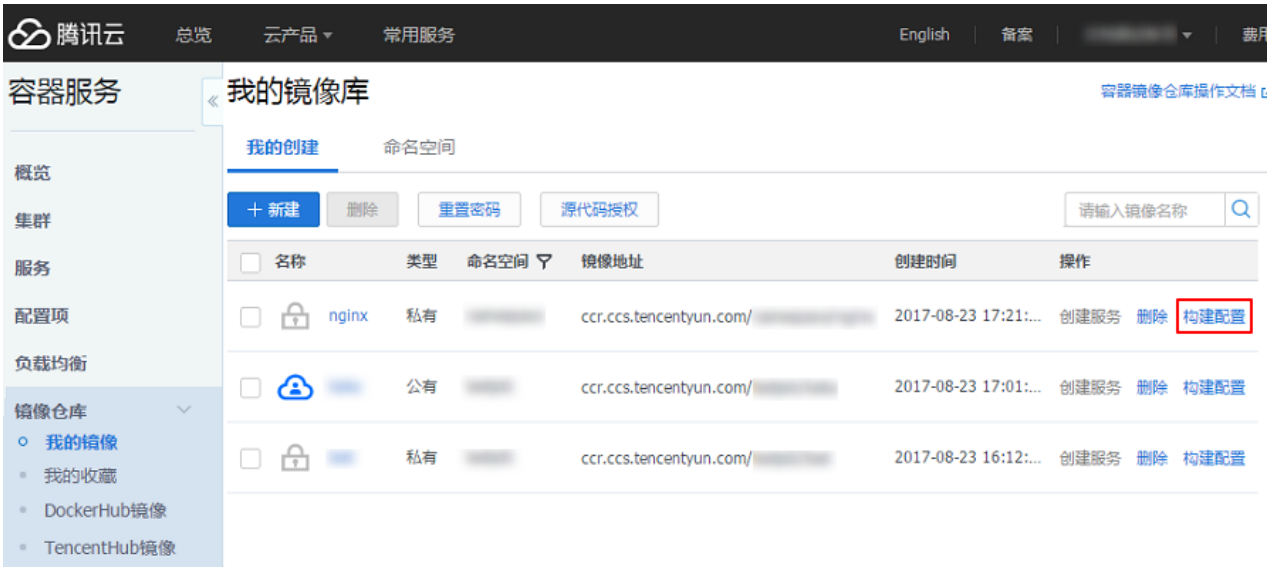
构建规则设置

前提条件

如果您想通过 Git 仓库来构建容器镜像，那么请先完成 源码仓库授权 操作。

操作步骤

1. 登录 腾讯云容器服务控制台 。
2. 单击左侧导航栏中的【镜像仓库】，在下拉列表中单击【我的镜像】，进入我的镜像库页面，单击右侧【构建配置】。



3. 进入构建规则页面，配置相关信息：

Github 构建规则

- 代码源： 选择 Github。
- Organization：选择您的 Organization，通常为您的 Github 账户，如果您属于多个组织，那么挑选其中一个。
- Repository：选择您需要构建容器镜像的仓库。
- 触发方式：复选模式，支持当 push 代码到某个分支或者新的 Tag 时，自动触发容器镜像构建。您也可以都不选到镜像仓库详情页的镜像构建页手动构建。
- 版本命名规则：即容器镜像 Tag 命名规则，镜像 Tag 名支持格式化，可以包含分支名 / 仓库 Tag 名。

  i：更新时间：镜像构建时间。

ii：commit 号：分支 / Tag 最近的 commit 号。

注意：
如果在某个分支或者Tag 上进行自动构建，且版本命名规则包含了分支 / 标签，那么分支或者 Tag 名不能包含特殊字符，例如不能包含 /，$ 等特殊字符，可以包含大小写字符，下划线（_），连接符（-）。

- Dockerfile路径：Dockerfile 在仓库中的相对路径。默认为空，可以不填，表示 Dockerfile 位于项目的根目录下，且文件名必须为 Dockerfile，以大写 D 开头。如果 Dockerfile 位于其它目录，例如位于仓库的 build 目录下，文件名为 Dockerfile，那么 Dockerfile 路径为：

  build/Dockerfile

  。

Gitlab构建规则

- 代码源：选择 Gitlab。
- Group：选择一个 Gitlab 的Group。
- Repository：选择您需要构建容器镜像的仓库。
- 触发方式：复选模式，支持当 push 代码到某个分支或者新的 Tag
  时，自动触发容器镜像构建。您也可以都不选到镜像仓库详情页的镜像构建页手动构建。
- 版本命名规则：即容器镜像 Tag 命名规则，镜像 Tag 名支持格式化，可以包含分支名 / 仓库 Tag

名。

i：更新时间：镜像构建时间。

ii：commit 号：分支 / Tag 最近的 commit 号。

注意：

如果在某个分支或者 Tag 上进行自动构建，且版本命名规则包含了分支 / 标签，那么分支或者 Tag 名不能包含特殊字符，例如不能包含 /，$ 等特殊字符，可以包含大小写字符，下划线（_），连接符（-）。

- Dockerfile路径：Dockerfile 在仓库中的相对路径。默认为空，可以不填，表示 Dockerfile 位于项目的根目录下，且文件名必须为 Dockerfile，以大写 D 开头。如果 Dockerfile 位于其它目录，例如位于仓库的 build 目录下，文件名为 Dockerfile，那么 Dockerfile 路径为：

  build/Dockerfile

  。

自动构建

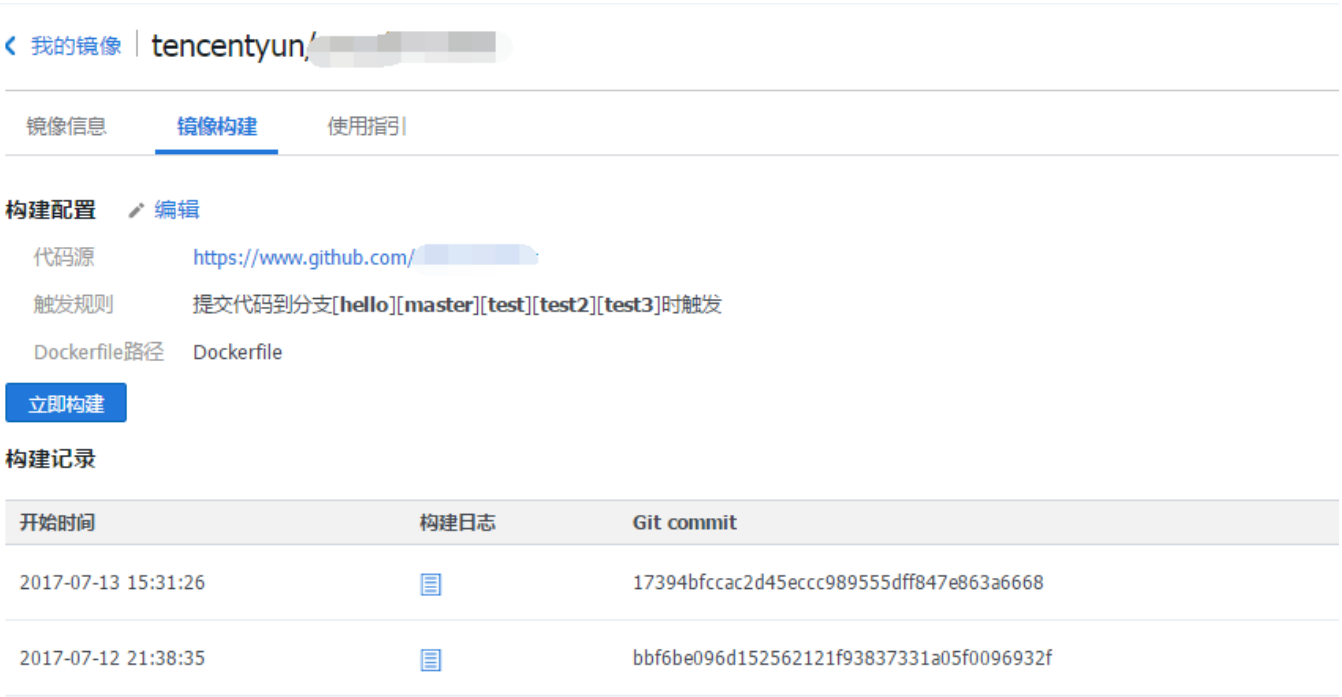
如果您完成了构建规则的配置，并且触发方式选择了添加新 Tag
时触发或者提交代码到分支时触发，那么当您提交一个新的分支或者 push 代码到指定仓库时，会自动触发容器镜像的构建，整个构建过程在腾讯云的容器平台上进行，构建完成后，按照您定义的版本命名规则，会生成新的镜像，并上传到腾讯云容器镜像仓库。

镜像构建

镜像构建页

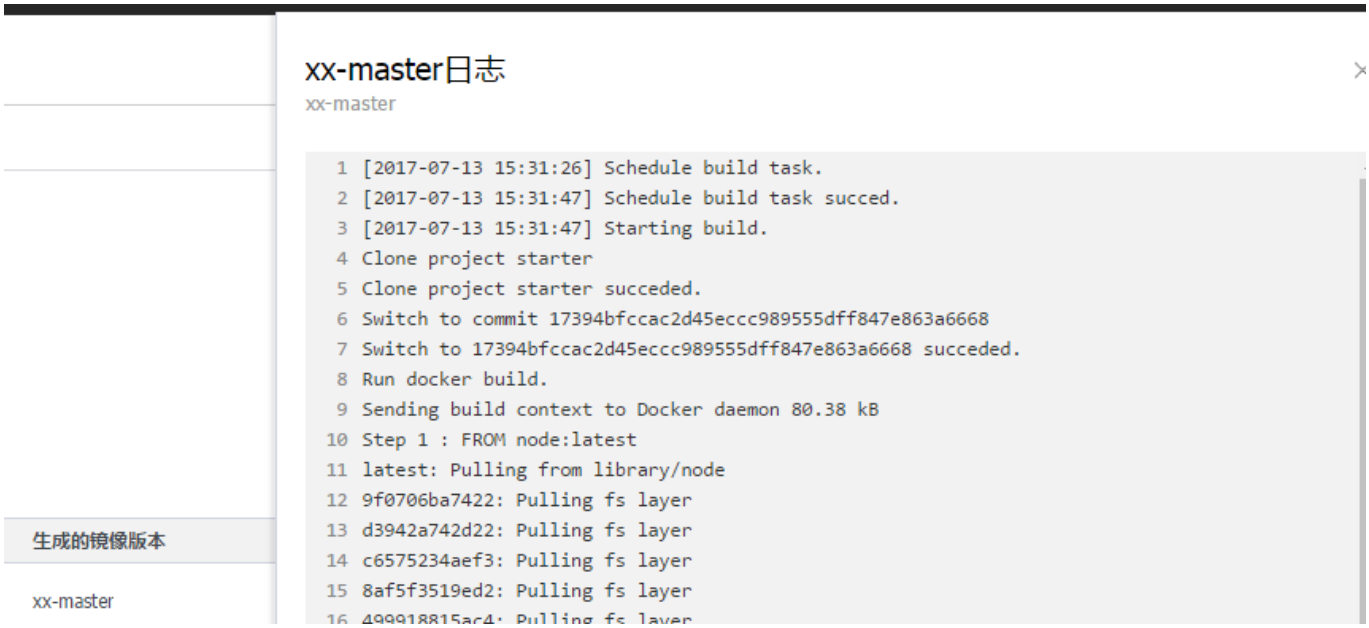
1. 首先打开镜像仓库列表

2. 点击任意一个镜像仓库，进入该仓库的详情页，我们可以查看该镜像仓库的构建规则，以及查看构建历史记录。



查看镜像构建记录

点击构建日志按钮，可以查看各个构建记录的log



立即构建

您可以对该仓库手动立即发起一次构建。

Dockerfile手动构建

使用Dockerfile手动构建容器镜像

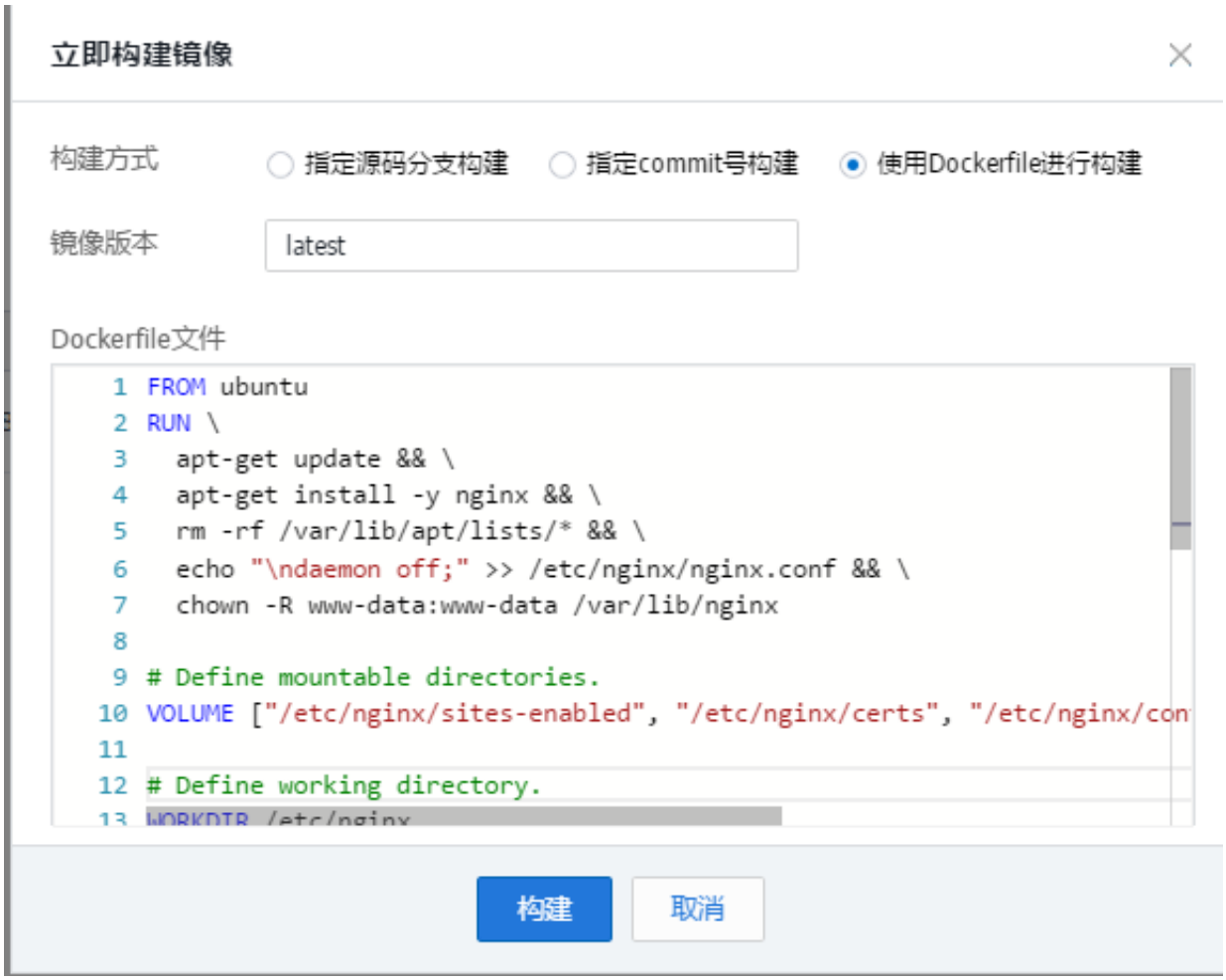
使用Dockerfile构建

如果您不想通过git代码仓库来构建容器镜像，我们提供一种手动上传Dockerfile来构建容器镜像的方式。通过这种方式，您需要准备一个Dockerfile文件，该文件指定了依赖的基础镜像和依赖的其他资源，这些资源必须能通过公网访问。

进入镜像构建

通过镜像仓库列表，我们点击某个仓库，进入详情页，切换到镜像构建Tab，我们可以查看镜像的构建历史记录。

Dockerfile构建



查看构建log

在构建记录里，可以看到使用Dockerfile构建的log

latest日志

latest

```
22 be232718519c: Download complete
23 75c416ea735c: Verifying Checksum
24 75c416ea735c: Download complete
25 75c416ea735c: Pull complete
26 c6ff40b6d658: Pull complete
27 a7050fc1f338: Pull complete
28 f0ffb5cf6ba9: Pull complete
29 be232718519c: Pull complete
30 Digest: sha256:a0ee7647e24c8494f1cf6b94f1a3cd127f423268293c25d924fbe18fd82db5a4
31 Status: Downloaded newer image for ubuntu:latest
32  ---> d355ed3537e9
33 Step 2 : RUN apt-get update && apt-get install -y nginx && rm -rf /var/lib/apt/lists/* &&
34  ---> [Warning] Your kernel does not support swap limit capabilities, memory limited witho
35  ---> Running in c0fe273eed5e
36 Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
37 Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
38 Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
39 Get:4 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
40 Get:5 http://security.ubuntu.com/ubuntu xenial-security/universe Sources [41.5 kB]
41 Get:6 http://archive.ubuntu.com/ubuntu xenial/universe Sources [9802 kB]
42 Get:7 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [373 kB]
43 Get:8 http://security.ubuntu.com/ubuntu xenial-security/restricted amd64 Packages [12.8 kB
44 Get:9 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 Packages [177 kB]
45 Get:10 http://security.ubuntu.com/ubuntu xenial-security/multiverse amd64 Packages [2931 B
46 Get:11 http://archive.ubuntu.com/ubuntu xenial/main amd64 Packages [1558 kB]
47 Get:12 http://archive.ubuntu.com/ubuntu xenial/restricted amd64 Packages [14.1 kB]
48 Get:13 http://archive.ubuntu.com/ubuntu xenial/universe amd64 Packages [9827 kB]
49 Get:14 http://archive.ubuntu.com/ubuntu xenial/multiverse amd64 Packages [176 kB]
50 Get:15 http://archive.ubuntu.com/ubuntu xenial-updates/universe Sources [205 kB]
51 Get:16 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages [730 kB]
52 Get:17 http://archive.ubuntu.com/ubuntu xenial-updates/restricted amd64 Packages [13.2 kB]
53 Get:18 http://archive.ubuntu.com/ubuntu xenial-updates/universe amd64 Packages [637 kB]
```

源码构建Dockerfile路径设置

源码构建功能 Dockerfile路径 与 构建目录 问题指南

腾讯云容器服务提供了镜像自动构建能力，在源码构建配置中，可以指定 Dockerfile路径 与 构建目录 。

源码构建功能 Dockerfile路径 与 构建目录 该怎么填？

答案: 填写以项目为根路径的相对路径

如果没有填写，系统有默认值：

- Dockerfile路径 默认值： 代码仓库根目录下的 Dockerfile（

  Dockerfile

  ）
- 构建目录 默认值： 代码仓库根目录（

  ./

  ）

源码构建功能使用 Dockerfile路径 和 构建目录 细节?

源码构建功能首先clone用户指定的仓库，切换到相应的分支(branch)或标签(tag)，然后在代码仓库根目录执行

docker build -f $DOCKERFILE_PATH $WORKDIR

构建出容器镜像。

Dockerfile文件中的源路径应该怎么写？

对于

COPY

，

ADD

等涉及源路径的命令，源路径应该写成相对于 构建目录 的 相对路径。

触发器

触发器概述

镜像仓库触发器帮助用户在镜像构建完毕后，自动执行服务更新、webhook、消息推送等触发动作。通过触发器可以和持续集成结合实现持续部署。

镜像仓库触发器包含如下四个属性：

- 触发器名称：创建的触发器的名称。
- 镜像仓库：指定触发器绑定的镜像仓库，一个镜像仓库目前最多支持 10 个触发器。
- 触发条件：通过该属性可设置只有符合特定 Tag（镜像版本）格式的镜像被提交时，才执行触发动作。
- 触发动作：目前支持触发容器服务更新动作，后续将支持更多触发动作，如 webhook、消息推送等。

触发条件

腾讯云容器服务的镜像仓库目前支持三种 Tag 触发表达式，通过 Tag 表达式来设置触发条件：

- 全部触发：镜像仓库内，有新的 Tag 生成或 Tag 发生更新时，触发动作。
- 指定 Tag 触发：输入多个 Tag 名称，Tag 名称之间以分号隔开。镜像仓库内，有指定 Tag 生成或更新时，触发动作。
- 正则触发：指定 Tag 的正则表达式。镜像仓库内，有符合正则表达式的 Tag 生成或更新时，触发动作。

触发动作

目前支持的触发动作是服务更新，在配置触发动作时需要设置容器服务所在地域、所属集群、Namespace、服务及对应容器镜像等参数。
当满足触发条件时，通过配置的参数，该服务下的指定容器镜像将更新。

触发记录

仓库触发器每次执行触发动作时，都会产生触发记录。触发记录中包含触发器名称、触发条件、触发动作、触发结果、触发时间等信息。

触发器的基本操作

如何使用镜像仓库触发器

使用仓库触发器的步骤分为如下三步：

1. 选择具体镜像仓库创建触发器，配置触发表达式和服务更新参数。
2. 通过腾讯云 CI 或者 docker push 镜像到镜像仓库，确认提交的镜像是否满足触发表达式的条件。
3. 查看触发器日志，检查触发动作是否执行成功。

创建镜像仓库触发器

1. 登录 腾讯云容器服务控制台 。
2. 单击左侧导航栏的【镜像仓库】，在镜像仓库下拉列表中单击【我的镜像】。在我的镜像库页面单击镜像的 名称（如图中 test）。

3. 单击【触发器】>【添加触发器】。

4. 设置触发器属性。
   - 触发器名称：创建的触发器名称。英文字母开头，2 - 64 个字符以内 。
   - 触发条件：分为三种触发条件。

     i：全部触发：镜像仓库内，有新的 Tag（镜像版本）生成，或 Tag 发生更新时，触发动作。

     ii：指定 Tag 触发：镜像仓库内，有指定 Tag 生成或更新时，触发动作。

     iii：正则触发：镜像仓库内，有符合正则表达式的 Tag 生成或更新时，触发动作。
   - 触发动作：更新容器的镜像。
   - 选择服务/镜像

     ：单击【请选择容器镜像】，在下拉列表中选择地域、集群、Namespace、服务、容器镜像属性。

5. 单击【保存】。完成触发器的创建。

修改镜像仓库触发器

1. 登录 腾讯云容器服务控制台 。
2. 单击左侧导航栏的【镜像仓库】，在镜像仓库下拉列表中单击【我的镜像】。在我的镜像库页面单击镜像的 名称（如图中 test）。

3. 单击【触发器】进入触发器列表页，单击触发器显示栏右侧修改图标。

删除镜像仓库触发器

1. 登录 腾讯云容器服务控制台 。
2. 单击左侧导航栏的【镜像仓库】，在镜像仓库下拉列表中单击【我的镜像】。在我的镜像库页面单击镜像的 名称（如图中 test）。

3. 单击【触发器】进入触发器列表页，单击触发器显示栏右侧删除图标。

查看触发日志

1. 登录 腾讯云容器服务控制台 。
2. 单击左侧导航栏的【镜像仓库】，在镜像仓库下拉列表中单击【我的镜像】。在我的镜像库页面单击镜像的 名称（如图中 test）。

3. 单击【触发器】进入触发器列表页，即可浏览触发日志。

触发记录

触发器名称	触发器条件	触发动作	触发结果	触发时间
all-gz	全部触发	更新容器的镜像	成功	2017-08-22 19:50:40
all-sh	全部触发	更新容器的镜像	成功	2017-08-22 19:50:40
all-bj	全部触发	更新容器的镜像	成功	2017-08-22 19:50:40
all-sg	全部触发	更新容器的镜像	成功	2017-08-22 19:50:40
all-gz	全部触发	更新容器的镜像	成功	2017-08-22 19:09:48
all	正则触发（^[v]）	更新容器的镜像	成功	2017-08-22 18:50:01