# 容器镜像服务

# 最佳实践

**文档版本** 03

**发布日期** 2022-05-05

**华为技术有限公司**

## 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。

# 目 录

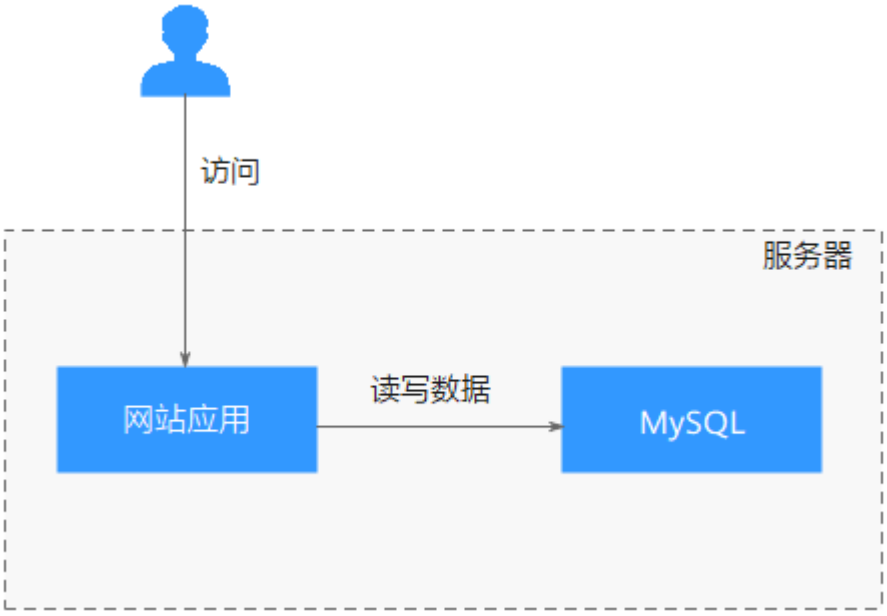


# 1 编写高效的 Dockerfile

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
```

当然这样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）**
- **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、映射端口和数据卷**
- **使用EXPOSE暴露端口**
- **使用VOLUME管理数据卷**
- **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

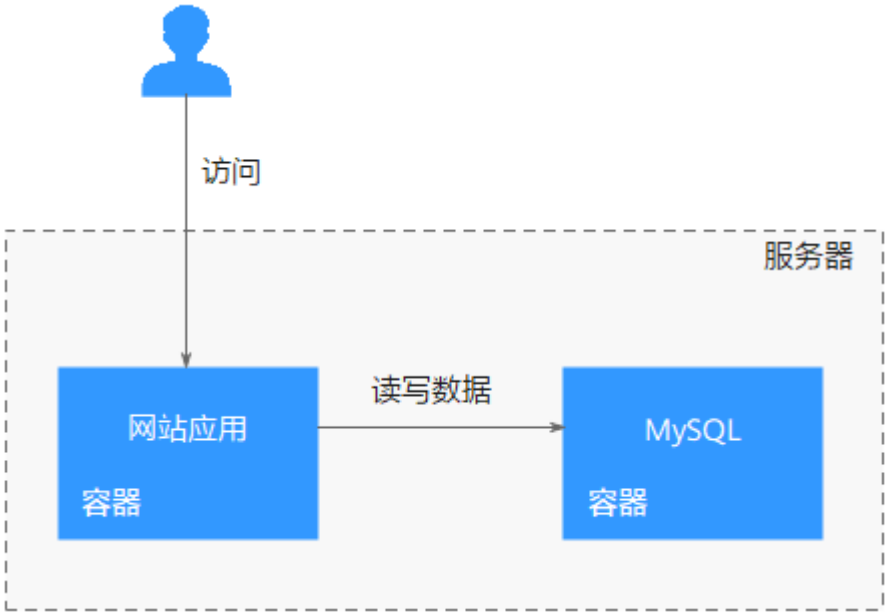
从技术角度讲，Docker容器中可以运行多个进程，您可以将数据库、前端、后端、ssh等都运行在同一个Docker容器中。但是，这样跟未使用容器前没有太大区别，且这样容器的构建时间非常长（一处修改就要构建全部），镜像体积大，横向扩展时非常浪费资源（不同的应用需要运行的容器数并不相同）。

通常所说的容器化改造是对应用整体微服务进行架构改造，改造后，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，若存在安全漏洞，一旦被攻击，所有功能的权限都可能会被窃取。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。
- 提升稳定性：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。

因此，上述企业门户网站可以进行如下改造，Web应用和MySQL运行在不同容器中。




MySQL运行在独立的镜像中，这样的好处就是，我们可以对它们分别进行修改，且不会牵一发而动全身。如下面这个例子所示，我们可以删除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。

如果基础镜像中的某个包过时了，但你不知道具体是哪一个包，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。

apt-get upgrade会使得镜像构建过程非常不稳定，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。因此通常我们会删掉apt-get upgrade。

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，如果我们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

假设我们更新了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。因此最好将它们删除，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，我们选择了ubuntu作为基础镜像。但是我们只需要运行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"]
```

## 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，我们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，这显然不是我们希望看到的。因此我们可以先拷贝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，我们应该在Dockerfile中设置映射端口和数据卷。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。我们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，我们并不需要.git目录等内容。

示例如下：

```
.git/
```

# 2 跨云 Harbor 同步镜像至华为云 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。跨云Harbor同步镜像至SWR存在两种场景：

1. Harbor可以通过公网访问SWR，配置方法参见**公网访问场景**。
2. 通过专线打通Harbor到VPC间的网络，使用VPC终端节点访问SWR，配置方法参见**专线打通场景**。

## 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。

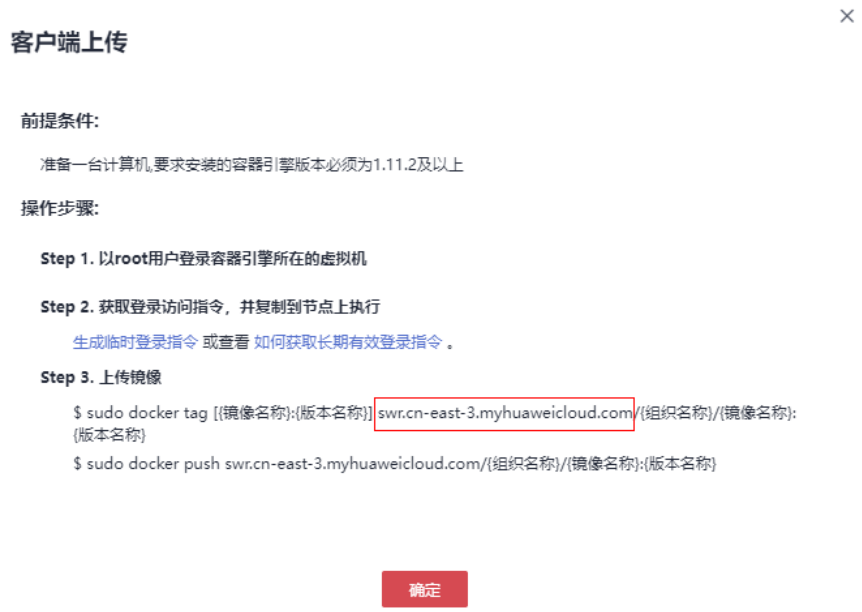
1. 新建目标。


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-east-3.myhuaweicloud.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。

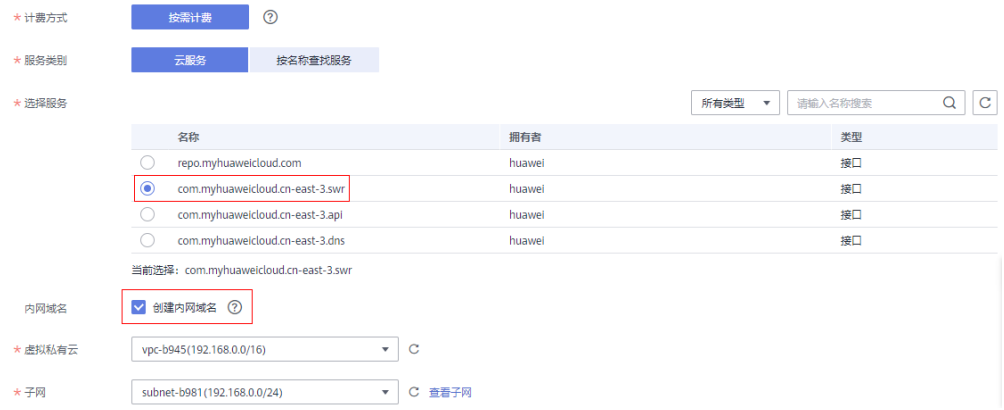
- 触发模式：选择“手动”。
- 带宽：设置执行该条同步规则时的最大网络带宽，“-1”表示无限制。

**步骤3** 创建完成后，选中后单击“复制”即可完成同步。


**----结束**

## 专线打通场景

**步骤1** 配置VPC终端节点。




**步骤2** 获取VPC内网访问IP及域名（华为云VPC内默认会自动加域名解析规则，不需要配置hosts；非华为云节点需要配置hosts），可以在终端节点详情页的“内网域名”中查询。

**步骤3** 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VPC终端节点中的内网域名**，必须以https开头，同时Harbor所在的容器内要配置域名映射。
- 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：必须**取消勾选**。

**步骤4** 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3**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5** 创建完成后，选中后单击“复制”即可完成同步。


**----结束**