云托管 CloudBase Run

开发指南

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。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默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

开发指南
服务开发说明
优化容器镜像

开发指南
服务开发说明

最近更新时间：2024-05-22 17:15:31

本文介绍开发云托管服务时需要了解的一些事项。

编程语言支持

您可以使用任何语言、任何框架编写服务，甚至直接使用现有的容器镜像。快速入门文档中提供了以多种主流语言编写的示例，但暂不在示例范围内的语言也均可支持。

代码要求

侦听 HTTP 请求

服务必须侦听 HTTP 请求，云托管暂不支持 tcp/udp/mqtt 等其他协议。

不支持 Docker Compose

云托管暂不支持 Docker Compose。

不支持 Windows

云托管暂不支持 Windows 操作系统的容器。

无状态服务

服务必须是无状态服务，不能依赖永久性本地状态。这是为了能够进行水平的自动扩缩容。

- **无状态服务（stateless service）**指的是对单次请求的处理，不依赖其他请求。处理一次请求所需的全部信息必须都包含在这个请求里，或者可以从数据库、文件存储等外部获取，实例本身不存储任何信息。
- **有状态服务（stateful service）**则相反，它会在自身保存一些数据，先后的请求是有关联的。

避免后台活动

代码中请不要在请求处理程序范围之外启动后台线程或例程，并确保在传送响应之前完成所有异步操作。强行运行后台线程可能会导致异常。例如一个 HTTP 服务，在收到一个接口后，直接调用 sh 命令，做系统层面的动作。如系统文件更新，指向，重启等，这都是不建议的，会引发不稳定导致回收。
再展开异步任务的例子来进一步说明：

1. 当服务最小实例为0（低成本模式），触发执行一个异步任务，时间达2小时。这个时间段中如果没有任何访问请求的话，半小时后会直接缩容到0，回收实例，异步任务会被终止。
2. 当服务负载出现高峰，触发了自动扩容，有2个甚至更多实例存在。这时候如果有异步任务2小时，但恰巧2小时内的流量访问又不足以一直支撑2个或多个容器提供服务，服务负载回落时，实例数减少。**因为默认服务无状**

态，即每个实例都是完全等同的，缩容时无法指定具体销毁哪个容器，此时异步任务如果刚好在缩容的那个容器中，也会被终止。

异步任务处理

确实需要用云托管服务处理异步任务，且无法在传送响应之前完成所有异步操作的话，需要注意以下几点：

- 必须保证实例不会回收，意味着无法采用低成本模式，服务最小实例数不能设置为0；
- 可以约束任务，或者是通过守护请求来支撑住，避免任务受缩容影响；
- 可以通过放弃自动扩缩容能力，将服务最小和最大实例数都设置为同样的值（例如均设置为1，锁死服务无论如何只有1个实例），保障异步任务运行。

定时任务处理

确实需要用云托管服务处理定时任务，且无法在传送响应之前完成所有异步操作的话，需要注意以下几点：

- 必须保证实例不会回收，因为定时任务无法通过外部请求维持，意味着无法采用低成本模式，服务最小实例数不能设置为0；
- 必须放弃自动扩缩容能力，将服务最小和最大实例数都设置1，锁死服务无论如何只有1个实例，保障定时任务运行。

代码容器化

基于容器部署服务，您需要提供一个容器镜像，或者提供代码由云托管在线构建镜像。容器镜像是一种封装格式，其中包含您的代码、代码软件包、所需的全部二进制依赖项、要使用的操作系统以及运行服务所需的其他任何内容。因此，镜像一旦构建完成，以上所有信息就都被固化，云托管不会感知镜像具体内容，容器实例运行过程中镜像内容也不会变化。

> **说明：**
> 如您需要修改任何内容，则需要重新构建镜像。

通常，您可以使用名为 Dockerfile 的文件来声明如何构建镜像。快速入门文档中提供的示例，也包含多种主流语言的 Dockerfile 参考。

> **说明：**
> Dockerfile 背景知识：查看 Dockerfile 官方文档 了解语法，并查看 编写 Dockerfile 的最佳做法 中的提示了解如何将这些语法整合在一起。

Dockerfile 通常从基础镜像（例如 `FROM golang:1.11` ）开始。您可以在 Docker Hub 上找到由操作系统和语言作者维护的基础镜像。

谨慎使用依赖项

如果您使用具有依赖项库的动态语言，例如导入 Node.js 模块，那么在冷启动期间加载这些模块会增加延迟时间。您可以通过以下方式缩短启动延迟时间：

- 最大限度地减少依赖项的数量和大小，以构建精简服务。
- 惰性加载不常用的代码（如果您所用的语言支持此方式）。
- 使用代码加载优化技术。

控制镜像大小

镜像较大会造成以下影响：

- 增加安全漏洞。
- 降低镜像的构建速度，增加构建时长，甚至引发构建超时。
- 降低服务的部署速度。

> **说明：**
> 镜像的大小不会影响请求处理时间，但过大的镜像会消耗更多构建时间、部署时间，也会消耗更多资源，从而产生更多费用。

具体实现镜像优化，请参见 优化容器镜像 。

优化容器镜像

最近更新时间：2024-08-20 15:49:01

云托管服务是通过 Docker 镜像来启动的，也就是说您需要将您自己的业务项目代码打包成 Docker 镜像，才能在云托管中正常运行。

业务项目代码打包成 Docker 镜像

如果有一台没有任何系统的服务器，您准备通过什么步骤，让自己的代码项目在这一台服务器中运行?
我们以一个 nodejs 例子举例，用服务器的操作步骤进行描述：

```
1. 安装一个alpine系统，版本随意
2. 然后在alpine系统上，下载nodejs并安装
3. 把我们的代码复制到服务器的一个目录中
4. npm安装一下项目所需要的依赖
5. 使用启动命令将项目运行起来
```

相应的，java、php、python 的整体步骤也是差不多的，都是**安装系统 > 安装语言支持 > 复制代码（或运行文件）> 安装依赖 > 启动项目**这一个过程。那么 Docker 构建的过程，其实也是类似的，只不过我们需要用一个标准的描述格式，把上面的步骤描述一下，这个格式文本叫 `Dockerfile` 。
例如上述 nodejs 例子，用 Dockerfile 描述如下：

```
# 安装一个alpine系统，版本随意
FROM alpine:3.15

# 然后在alpine系统上，下载nodejs并安装
RUN apk add --update --no-cache nodejs npm

# 把我们的代码复制到服务器的一个目录中
COPY . .

# npm安装一下项目所需要的依赖
RUN npm install

# 使用启动命令将项目运行起来
CMD ["node", "index.js"]
```

通过这个简单的例子，您就会发现，其实我们编写 Dockerfile，主要是为了标准化项目部署过程。Docker 根据您的描述，就可以不折不扣的执行，然后打包成一个镜像，这个镜像就包含了您项目以及项目所依赖的全部底层，就可以在任意一处建立容器来运行了，不需要再上去运维。

您可以在网上找到**您的语言框架+Dockerfile** 搜索关键字的经验条目，绝大多数语言框架都有覆盖到，当您找到 Dockerfile 时，就可以通过阅读步骤，来了解它的构建过程，不会因为人与人自然语言描述的模糊而造成理解偏差。

如何提升镜像的构建效率

当您还没有对 Dockerfile 和构建流程有更深刻认知时，您的项目 Dockerfile 一般是从网上和各模板中直接复制下来的。但有的时候这些 Dockerfile 可能并没有完全契合您的项目，甚至在构建的时候效率会非常低。
所以您可以尝试改造它们，来提升自己的构建效率，在这里我们总结了一些常见的优化点，希望您可以有所收获。

1. 变化的放后面，高效利用缓存

每次构建时，Docker 都会充分利用之前构建的成果物，如果没有变化，则直接使用而不是重新构建。在构建的过程中遵循以下规则：

1. 根据 `FROM` 命令指定的基本镜像，将每一条指令与从该基本镜像派生的所有子镜像进行比较，查看是否有使用完全相同的指令构建的，如果没有则缓存无效。
2. 对于 ADD 和 COPY 指令，检查镜像中文件的内容，并为每个文件计算一个校验标识。在这些校验标识中通常不考虑文件的最后修改时间和最后访问时间，将校验标识与现有镜像中的进行比较，如果文件中的任何内容（例如内容和元数据）发生了更改，则缓存将无效。
3. 除了 ADD 和 COPY 命令外，缓存检查不会查看容器中的文件来确定缓存是否匹配。
4. 缓存无效后，所有后续命令都会重新构建。

举个例子：

```
FROM alpine:3.15
RUN apk add --update --no-cache nodejs npm
COPY . .
RUN npm install
CMD ["node", "/index.js"]
```

我们变化的只是项目代码，所以打包时 COPY 命令之前的直接使用缓存，但以上还不是更好的，因为我们每次还必须要安装依赖 `npm install`，而绝大多数情况下，我们根本不会变更项目依赖，所以就可以改进一下，如下：

```
FROM alpine:3.15
RUN apk add --update --no-cache nodejs npm
COPY ./package*.json .
RUN npm install
COPY . .
CMD ["node", "/app/index.js"]
```

我们先把 `npm install` 必要的 `package*.json` 文件复制进来，然后安装依赖，完成后再把其它的文件复制进来。这样做的好处是，只要您不变更 `package*.json` 文件，则不用执行 `npm install`，而只是复制最后的

文件，打包时间基本都在1秒左右。

2. 减少层数，合并命令

Dockerfile 中的所有命令最终都会形成一个 layer 层，减少层级从一定程度上会减少最终镜像的大小。
举个例子：

```
FROM alpine:3.13

# ...
RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.tencent.com/g'
/etc/apk/repositories
RUN apk add --update --no-cache php7 php7-json php7-ctype php7-exif
php7-pdo php7-pdo_mysql php7-fpm nginx
RUN rm -f /var/cache/apk/*

# ...
RUN cp /app/conf/nginx.conf /etc/nginx/conf.d/default.conf
RUN cp /app/conf/fpm.conf /etc/php7/php-fpm.d/www.conf
RUN cp /app/conf/php.ini /etc/php7/php.ini
RUN mkdir -p /run/nginx
RUN chmod -R 777 /app/runtime
RUN mv /usr/sbin/php-fpm7 /usr/sbin/php-fpm
```

上述这些命令可以直接合并一下，这样会显著的减少层数。

```
FROM alpine:3.13

# ...
RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.tencent.com/g'
/etc/apk/repositories && apk add --update --no-cache php7 php7-json
php7-ctype php7-exif php7-pdo php7-pdo_mysql php7-fpm nginx && rm -f
/var/cache/apk/*

# ...
RUN cp /app/conf/nginx.conf /etc/nginx/conf.d/default.conf \
    && cp /app/conf/fpm.conf /etc/php7/php-fpm.d/www.conf \
    && cp /app/conf/php.ini /etc/php7/php.ini \
    && mkdir -p /run/nginx \
    && chmod -R 777 /app/runtime \
    && mv /usr/sbin/php-fpm7 /usr/sbin/php-fpm
```

版权所有：腾讯云计算（北京）有限责任公司

但需要注意的是，不要滥合并，有时候一些命令不稳定，可能会影响其它命令的有效运行，导致最终效率反而不是很好，建议合并相关的同一类操作，例如安装依赖，文件操作。

3. 更换源站

Docker 基础镜像的下载，构建执行时，各种系统软件的安装下载，语言依赖的下载都需要通过网络加载，所以可以在安装前更换一下源站。
云托管线上构建的 Docker 镜像源已经是腾讯加速源了，所以不需要自己设置。

```
{
   "registry-mirrors": [
       "https://mirror.ccs.tencentyun.com"
  ]
}
```

Dockerfile 中，您可以在安装前指定源，以下列举一些常见的：

```
# alpine-apk更换腾讯加速源
RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.tencent.com/g'
/etc/apk/repositories

# NPM更换腾讯加速源
RUN npm config set registry https://mirrors.tencent.com/npm/

# Python更换腾讯加速源
RUN pip config set global.index-url
http://mirrors.cloud.tencent.com/pypi/simple && pip config set
global.trusted-host mirrors.cloud.tencent.com
```

开源部分完成了，接下来就进行节流了，尽可能只保留最小可运行的内容，不要在生产包中安装调试之类的内容。

4. 选一个好的基础镜像

一个好的基础镜像将对您最终项目镜像产生重要影响。一般来说，我们尽可能选择一个既符合项目要求的，又非常小的镜像。您可以先从 Docker 官方镜像仓库中找一个合适的镜像。
例如我们想需要一个 node.js 环境，您可以自己安装。

```
FROM alpine:3.15

RUN apk add --update --no-cache nodejs npm

# 55M
```

也可以直接引用已经封好的：

```
FROM node:alpine3.15

# 171.8M，还有更小的node镜像
```

不过已经封装好的，虽然不用您再安装配置了，但体积也是比较大的，有时候它依赖的基础镜像还会不一样，所以看自己的需要来用。
如果有些环境的安装配置极为复杂，您很难做的既优雅又明白，这个时候不用考虑了，除非您对镜像有偏执的要求，用一个已经封装好的基础镜像，开局会非常轻松。
另外我们也会在构建环节缓存常见的一些镜像，所以不用太担心镜像太大的下载问题，除非您选了一些冷门的，或者偏门的镜像。
不过非常大的镜像对于拉起实例也是不友好的（因为有项目镜像下载过程），所以也希望各位开发者尽可能的提供小而美的构建步骤。

5. jar 包的优化

当您做 springBoot 项目时，构建物 jar 包对于 Docker 镜像来说，元数据改变了，就需要重新打包上传，其实比较难受的。
这种情况下，可以使用 `spring-boot-jarmode-layertools` 试一试，它将 jar 包拆成若干个层，直接对应到 Docker 的镜像缓存中。

实际例子

以下是一个真实的例子：

```
FROM alpine:3.14

RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.tencent.com/g'
/etc/apk/repositories \
    && apk add --update --no-cache python3 py3-pip \
    && rm -rf /var/cache/apk/*

ENV TZ Asia/Shanghai
RUN apk add tzdata && cp /usr/share/zoneinfo/${TZ} /etc/localtime &&
echo ${TZ} > /etc/timezone

RUN apk add ca-certificates

COPY . /app

WORKDIR /app
```

```
RUN pip config set global.index-url
http://mirrors.cloud.tencent.com/pypi/simple \
    && pip config set global.trusted-host mirrors.cloud.tencent.com \
    && pip install --upgrade pip \
    && pip install --user -r requirements.txt

EXPOSE 80

CMD ["python3", "run.py", "0.0.0.0", "80"]
```

上述构建过程中会出现一些错误，偶发性比较多，并且构建时间每次都很长。我们在看了一下后，对 Dockerfile 进行了以下优化：

```
FROM python:3.9.12-alpine3.14

ENV TZ Asia/Shanghai

RUN apk add tzdata && cp /usr/share/zoneinfo/${TZ} /etc/localtime &&
echo ${TZ} > /etc/timezone \
    && apk add ca-certificates

WORKDIR /app

COPY requirements.txt .

RUN pip config set global.index-url
http://mirrors.cloud.tencent.com/pypi/simple \
    && pip config set global.trusted-host mirrors.cloud.tencent.com \
    && pip install --upgrade pip \
    && pip install --user -r requirements.txt

COPY . .

EXPOSE 80

CMD ["python3", "run.py", "0.0.0.0", "80"]
```

您可以根据自己掌握的知识，对上述修改做梳理，希望有助于您的 Dockerfile 优化过程。
当然构建只是第一步，项目镜像最终大小，以及镜像在实例中启动速度（包含您的项目运行过程），也最终决定了云托管的服务创建速度，和冷启动速度。

版权所有：腾讯云计算（北京）有限责任公司