云托管 CloudBase Run

快速入门

产品文档

腾讯云

【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。

文档目录

快速入门
构建并部署 Node.js 应用

最近更新时间：2022-07-12 10:42:25

步骤1：编写基础应用

1. 创建名为 helloworld 的新目录，并转到此目录中：

```
mkdir helloworld
cd helloworld
```

2. 创建一个包含以下内容的 package.json 文件：

```
{
"name": "helloworld",
"description": "Simple hello world sample in Node",
"version": "1.0.0",
"main": "index.js",
"scripts": {
"start": "node index.js"
},
"author": "Tencent CloudBase",
"license": "Apache-2.0",
"dependencies": {
"express": "^4.17.1"
}
}
```

3. 在同一目录中，创建一个 index.js 文件，并将以下代码行复制到其中：

```
const express = require("express");
const app = express();

app.get("/", (req, res) => {
res.send(`Hello World!`);
});
```

```
const port = 80;
app.listen(port, () => {
console.log(`helloworld: listening on port ${port}`);
});
```

> **说明**
> 此代码会创建一个基本的 Web 服务器，侦听 80 端口。

步骤2：将应用容器化

1. 在项目根目录下，创建一个名为 Dockerfile 的文件，内容如下：

```
FROM alpine:3.13

# 容器默认时区为UTC，如需使用上海时间请启用以下时区设置命令
# RUN apk add tzdata && cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime && echo As
ia/Shanghai > /etc/timezone

# 安装依赖包，如需其他依赖包，请到alpine依赖包管理(https://pkgs.alpinelinux.org/packages?
name=php8*imagick*&branch=v3.13)查找。
RUN apk add --update --no-cache nodejs npm

# # 指定工作目录
WORKDIR /app

# 拷贝包管理文件
COPY package*.json /app

# npm 源，选用国内镜像源以提高下载速度
RUN npm config set registry https://mirrors.cloud.tencent.com/npm/
# RUN npm config set registry https://registry.npm.taobao.org/

# npm 安装依赖
RUN npm install

# 将当前目录（dockerfile所在目录）下所有文件都拷贝到工作目录下（.gitignore中的文件除外）
COPY . /app
```

```
# 执行启动命令.
# 写多行独立的CMD命令是错误写法！只有最后一行CMD命令会被执行，之前的都会被忽略，导致业务报错。
# 请参考[Docker官方文档之CMD命令](https://docs.docker.com/engine/reference/builder/#cmd)
CMD ["npm", "start"]
```

2. 添加一个 .dockerignore 文件，以从容器映像中排除文件：

```
Dockerfile
.dockerignore
node_modules
npm-debug.log
```

步骤3（可选）：本地构建镜像

1. 如果您本地已经安装了 Docker，可以运行以下命令，在本地构建 Docker 镜像：

```
docker build -t helloworld
```

2. 构建成功后，运行 docker images，可以看到构建出的镜像，随后您可以将此镜像上传至您的镜像仓库。

```
REPOSITORY TAG IMAGE ID CREATED SIZE
helloworld latest 1c8dfb88c823 8 seconds ago 146MB
```

步骤4：部署到云托管

详情请参见 部署服务 与 版本配置说明。

构建并部署 PHP 应用

最近更新时间：2022-07-12 10:42:36

步骤1：编写基础应用

1. 创建名为 helloworld-php 的新目录，并转到此目录中：

```
mkdir helloworld-php
cd helloworld-php
```

2. 创建名为 index.php 的文件，并将以下代码粘贴到其中：

```
<?php
echo sprintf("Hello World!");
```

> **说明**
> 此代码会对所有请求响应“Hello World”，HTTP 处理由容器中的 Apache Web 服务器进行。

步骤2：将应用容器化

1. 在项目根目录下，创建一个名为 Dockerfile 的文件，内容如下：

```
# 使用官方 PHP 7.3 镜像.
# https://hub.docker.com/_/php
FROM php:7.3-apache

# 将本地代码复制到容器内
COPY index.php /var/www/html/

# Apache 配置文件内使用 80 端口
RUN sed -i 's/80/80/g' /etc/apache2/sites-available/000-default.conf /etc/apache2/ports.co
nf

# 将 PHP 配置为开发环境
# 如果您需要配置为生产环境，可以运行以下命令
# RUN mv "$PHP_INI_DIR/php.ini-production" "$PHP_INI_DIR/php.ini"
```

```
# 参考：https://hub.docker.com/_/php#configuration
RUN mv "$PHP_INI_DIR/php.ini-development" "$PHP_INI_DIR/php.ini"
```

2. 添加一个 .dockerignore 文件，以从容器映像中排除文件：

```
Dockerfile
README.md
vendor
```

步骤3（可选）：本地构建镜像

1. 如果您本地已经安装了 Docker，可以运行以下命令，在本地构建 Docker 镜像：

```
docker build -t helloworld-php
```

2. 构建成功后，运行 docker images，可以看到构建出的镜像，随后您可以将此镜像上传至您的镜像仓库。

```
REPOSITORY TAG IMAGE ID CREATED SIZE
helloworld-php latest 1c8dfb88c823 8 seconds ago 411MB
```

步骤4：部署到云托管

详情请参见 部署服务 与 版本配置说明。

构建并部署 Java 应用

最近更新时间：2022-07-12 10:42:46

步骤1：编写基础应用（使用 spring boot 框架）

1. 首先我们创建一个 Spring Boot 应用。使用 curl 和 unzip 命令新建一个空 Web 项目：

```
curl https://start.spring.io/starter.zip \
-d dependencies=web \
-d javaVersion=1.8 \
-d bootVersion=2.3.3.RELEASE \
-d name=helloworld \
-d artifactId=helloworld \
-d baseDir=helloworld \
-o helloworld.zip
unzip helloworld.zip
cd helloworld
```

上述命令将创建一个 Spring Boot 项目。

> **说明**
> 如需在 Windows 系统上使用上述 curl 命令，您需要以下命令行之一：
> - WSL（推荐）
> - cygwin
>
> 或者选择性地使用 Spring Initializr（预加载配置）生成项目。

2. 将 src/main/java/com/example/helloworld/HelloworldApplication.java 内容更新如下：

```
package com.example.helloworld;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
public class HelloworldApplication {
```

```
@RestController
class HelloworldController {

@GetMapping("/")
String hello() {
return "Hello World!";
}
}

public static void main(String[] args) {
SpringApplication.run(HelloworldApplication.class, args);
}
}
```

3. 在 src/main/resources/application.properties 中，将服务器端口设置成 80：

```
server.port=80
```

> **说明**
> 以上代码会创建一个基本的 Web 服务器，并监听 80 端口。

步骤2：将应用容器化

1. 在项目根目录下，创建一个名为 Dockerfile 的文件，内容如下：

```
# 选择构建用基础镜像。如需更换，请到[dockerhub官方仓库](https://hub.docker.com/_/java?tab=tags)自行选择后替换。
FROM maven:3.6.0-jdk-8-slim as build

# 指定构建过程中的工作目录
WORKDIR /app

# 将src目录下所有文件，拷贝到工作目录中src目录下（.gitignore/.dockerignore中文件除外）
COPY src /app/src

# 将pom.xml文件，拷贝到工作目录下
COPY settings.xml pom.xml /app/
```

```
# 执行代码编译命令
# 自定义settings.xml, 选用国内镜像源以提高下载速度
RUN mvn -s /app/settings.xml -f /app/pom.xml clean package

# 选择运行时基础镜像
FROM alpine:3.13

# 容器默认时区为UTC，如需使用上海时间请启用以下时区设置命令
# RUN apk add tzdata && cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime && echo As
ia/Shanghai > /etc/timezone

# 安装依赖包，如需其他依赖包，请到alpine依赖包管理(https://pkgs.alpinelinux.org/packages?
name=php8*imagick*&branch=v3.13)查找。
# 选用国内镜像源以提高下载速度
RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.tencent.com/g' /etc/apk/repositories \
&& apk add --update --no-cache openjdk8-jre-base \
&& rm -f /var/cache/apk/*

# 指定运行时的工作目录
WORKDIR /app

# 将构建产物jar包拷贝到运行时目录中
COPY --from=build /app/target/*.jar .

# 暴露端口
# 此处端口必须与部署时填写的端口一致，否则会部署失败。
EXPOSE 80

# 执行启动命令.
# 写多行独立的CMD命令是错误写法！只有最后一行CMD命令会被执行，之前的都会被忽略，导致业
务报错。
# 请参考[Docker官方文档之CMD命令](https://docs.docker.com/engine/reference/builder/#c
md)
CMD ["java", "-jar", "/app/springboot-1.0.jar"]
```

2. 添加一个 .dockerignore 文件，以从容器映像中排除文件：

```
Dockerfile
.dockerignore
target/
```

步骤3（可选）：本地构建镜像

1. 如果您本地已经安装了 Docker，可以运行以下命令，在本地构建 Docker 镜像：

```
docker build -t helloworld-java
```

2. 构建成功后，运行 docker images，可以看到构建出的镜像，随后您可以将此镜像上传至您的镜像仓库。

```
REPOSITORY TAG IMAGE ID CREATED SIZE
helloworld-java latest c994813b495b 8 seconds ago 271MB
```

步骤4：部署到云托管

详情请参见 部署服务 与 版本配置说明。

构建并部署 Python 应用

最近更新时间：2022-07-12 10:42:56

步骤1：编写基础应用（使用 flask 框架）

1. 创建名为 helloworld-python 的新目录，并转到此目录中：

```
mkdir helloworld-python
cd helloworld-python
```

2. 创建名为 main.py 的文件，并将以下代码粘贴到其中：

```
import os

from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
return 'Hello World!'

if __name__ == "__main__":
app.run(debug=True, host='0.0.0.0', port=80)
```

> 说明
> 以上代码会创建一个基本的 Web 服务器，并监听 80 端口。

步骤2：将应用容器化

1. 在项目根目录下，创建一个名为 Dockerfile 的文件，内容如下：

```
# 选择基础镜像。如需更换，请到[dockerhub官方仓库](https://hub.docker.com/_/python?tab=tags)自行选择后替换。
# 已知alpine镜像与pytorch有兼容性问题会导致构建失败，如需使用pytorch请务必按需更换基础镜像。
FROM alpine:3.13
```

```
# 容器默认时区为UTC，如需使用上海时间请启用以下时区设置命令
# RUN apk add tzdata && cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime && echo Asia/Shanghai > /etc/timezone

# 安装依赖包，如需其他依赖包，请到alpine依赖包管理(https://pkgs.alpinelinux.org/packages?name=php8*imagick*&branch=v3.13)查找。
# 选用国内镜像源以提高下载速度
RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.tencent.com/g' /etc/apk/repositories \
# 安装python3
&& apk add --update --no-cache python3 py3-pip \
&& rm -rf /var/cache/apk/*

# 拷贝当前项目到/app目录下（.dockerignore中文件除外）
COPY . /app

# 设定当前的工作目录
WORKDIR /app

# 安装依赖到指定的/install文件夹
# 选用国内镜像源以提高下载速度
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 scipy 等数学包失败，可使用 apk add py3-scipy 进行， 参考安装 https://pkgs.alpinelinux.org/packages?name=py3-scipy&branch=v3.13
&& pip install --user -r requirements.txt

# 暴露端口。
# 此处端口必须与部署时填写的端口一致，否则会部署失败。
EXPOSE 80

# 执行启动命令
# 写多行独立的CMD命令是错误写法！只有最后一行CMD命令会被执行，之前的都会被忽略，导致业务报错。
# 请参考[Docker官方文档之CMD命令](https://docs.docker.com/engine/reference/builder/#cmd)
CMD ["python3", "run.py", "0.0.0.0", "80"]
```

2. 添加一个 .dockerignore 文件，以从容器映像中排除文件：

```
Dockerfile
README.md
*.pyc
*.pyo
*.pyd
__pycache__
.pytest_cache
```

步骤3（可选）：本地构建镜像

1. 如果您本地已经安装了 Docker，可以运行以下命令，在本地构建 Docker 镜像：

```
docker build -t helloworld-python
```

2. 构建成功后，运行 docker images，可以看到构建出的镜像，随后您可以将此镜像上传至您的镜像仓库。

```
REPOSITORY TAG IMAGE ID CREATED SIZE
helloworld-python latest 1c8dfb88c823 8 seconds ago 123MB
```

步骤4：部署到云托管

详情请参见 部署服务 与 版本配置说明。

构建并部署 C#（.NET）应用

最近更新时间：2022-07-12 10:43:03

步骤1：编写基础应用

1. 安装 .NET Core SDK 3.1。在 Console 中，使用 dotnet 命令新建一个空 Web 项目：

```
dotnet new web -o helloworld-csharp
cd helloworld-csharp
```

2. 更新 Program.cs 中的 CreateHostBuilder 定义，侦听 80 端口：

```
using System;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

namespace helloworld_csharp
{
public class Program
{
public static void Main(string[] args)
{
CreateHostBuilder(args).Build().Run();
}

public static IHostBuilder CreateHostBuilder(string[] args)
{
string port = "80";
string url = String.Concat("http://0.0.0.0:", port);

return Host.CreateDefaultBuilder(args)
.ConfigureWebHostDefaults(webBuilder =>
{
webBuilder.UseStartup<Startup>().UseUrls(url);
});
}
}
}
```

3. 将 Startup.cs 的内容更新为如下：

```
using System;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace helloworld_csharp
{
public class Startup
{
// This method gets called by the runtime. Use this method to add services to the contain
er.
// For more information on how to configure your application, visit https://go.microsoft.co
m/fwlink/?LinkID=398940
public void ConfigureServices(IServiceCollection services)
{
}

// This method gets called by the runtime. Use this method to configure the HTTP request
pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
if (env.IsDevelopment())
{
app.UseDeveloperExceptionPage();
}

app.UseRouting();

app.UseEndpoints(endpoints =>
{
endpoints.MapGet("/", async context =>
{
await context.Response.WriteAsync("Hello World!\n");
});
});
}
```

```
}
}
```

> **说明**
> 以上代码会创建一个基本的 Web 服务器，并监听 80 端口。

步骤2：将应用容器化

1. 在项目根目录下，创建一个名为 Dockerfile 的文件，内容如下：

```
FROM mcr.microsoft.com/dotnet/sdk:6.0-alpine AS build
RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.tencent.com/g' /etc/apk/repositories
WORKDIR /source

# copy csproj and restore as distinct layers
COPY *.sln .
COPY aspnetapp/*.csproj ./aspnetapp/
RUN dotnet restore -r linux-musl-x64 /p:PublishReadyToRun=true

# copy everything else and build app
COPY aspnetapp/. ./aspnetapp/
WORKDIR /source/aspnetapp
RUN dotnet publish -c release -o /app -r linux-musl-x64 --self-contained true --no-restore /
p:PublishTrimmed=true /p:PublishReadyToRun=true /p:PublishSingleFile=true

# final stage/image
FROM mcr.microsoft.com/dotnet/runtime-deps:6.0-alpine-amd64

# 容器默认时区为UTC，如需使用上海时间请启用以下时区设置命令
# RUN apk add tzdata && cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime && echo As
ia/Shanghai > /etc/timezone

RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.tencent.com/g' /etc/apk/repositories
WORKDIR /app
COPY --from=build /app ./

# See: https://github.com/dotnet/announcements/issues/20
# Uncomment to enable globalization APIs (or delete)
```

```
# ENV \
# DOTNET_SYSTEM_GLOBALIZATION_INVARIANT=false \
# LC_ALL=en_US.UTF-8 \
# LANG=en_US.UTF-8
# RUN apk add --no-cache icu-libs

ENTRYPOINT ["./aspnetapp"]
```

2. 添加一个 .dockerignore 文件，以从容器映像中排除文件：

```
**/obj/
**/bin/
```

步骤3（可选）：本地构建镜像

1. 如果您本地已经安装了 Docker，可以运行以下命令，在本地构建 Docker 镜像：

```
docker build -t helloworld-csharp
```

2. 构建成功后，运行 docker images，可以看到构建出的镜像，随后您可以将此镜像上传至您的镜像仓库。

```
REPOSITORY TAG IMAGE ID CREATED SIZE
helloworld-csharp latest 1c8dfb88c823 8 seconds ago 105MB
```

步骤4：部署到云托管

详情请参见 部署服务 与 版本配置说明。

构建并部署 Go 应用

最近更新时间：2022-07-12 10:43:14

步骤1：编写基础应用

1. 创建名为 helloworld 的新目录，并转到此目录中：

```
mkdir helloworld
cd helloworld
```

2. 创建一个包含以下内容的 go.mod 文件：

```
module helloworld
go 1.13
```

3. 在同一目录中，创建一个 main.go 文件，并将以下代码行复制到其中：

```
package main

import (
"fmt"
"log"
"net/http"
)

func main() {
http.HandleFunc("/", handler)
port := "80"
if err := http.ListenAndServe(":"+port, nil); err != nil {
log.Fatal(err)
}
}

func handler(w http.ResponseWriter, r *http.Request) {
fmt.Fprintf(w, "Hello World!\n")
}
```

> 说明
> 此代码会创建一个基本的 Web 服务器，侦听 80 端口。

步骤2：将应用容器化

1. 在项目根目录下，创建一个名为 Dockerfile 的文件，内容如下：

```
# 选择构建用基础镜像（选择原则：在包含所有用到的依赖前提下尽可能体积小）。如需更换，请到[d
ockerhub官方仓库](https://hub.docker.com/_/golang?tab=tags)自行选择后替换。
FROM golang:1.17.1-alpine3.14 as builder

# 指定构建过程中的工作目录
WORKDIR /app

# 将当前目录（dockerfile所在目录）下所有文件都拷贝到工作目录下（.dockerignore中文件除外）
COPY . /app/

# 执行代码编译命令。操作系统参数为linux，编译后的二进制产物命名为main，并存放在当前目录
下。
RUN GOOS=linux go build -o main .

# 选用运行时所用基础镜像（GO语言选择原则：尽量体积小、包含基础linux内容的基础镜像）
FROM alpine:3.13

# 容器默认时区为UTC，如需使用上海时间请启用以下时区设置命令
# RUN apk add tzdata && cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime && echo As
ia/Shanghai > /etc/timezone

# 指定运行时的工作目录
WORKDIR /app

# 将构建产物/app/main拷贝到运行时的工作目录中
COPY --from=builder /app/main /app/index.html /app/

# 执行启动命令
# 写多行独立的CMD命令是错误写法！只有最后一行CMD命令会被执行，之前的都会被忽略，导致业
务报错。
# 请参考[Docker官方文档之CMD命令](https://docs.docker.com/engine/reference/builder/#c
```

```
md)
CMD ["/app/main"]
```

2. 添加一个 .dockerignore 文件，以从容器映像中排除文件：

```
vendor/
.dockerignore
.gcloudignore
.gitignore
```

步骤3（可选）：本地构建镜像

1. 如果您本地已经安装了 Docker，可以运行以下命令，在本地构建 Docker 镜像：

```
docker build -t helloworld .
```

2. 构建成功后，运行 docker images，可以看到构建出的镜像，随后您可以将此镜像上传至您的镜像仓库。

```
REPOSITORY TAG IMAGE ID CREATED SIZE
helloworld latest 6948f1ebee94 8 seconds ago 82.7MB
```

步骤4：部署到云托管

详情请参见 部署服务 与 版本配置说明。