

[登录](#) [注册](#)

- [Java开源](#)
- [JS脚本](#)
- [OPEN家园](#)
- [OPEN文档](#)
- [OPEN资讯](#)
- [OPEN论坛](#)
- [Github日报](#)
- [OPEN代码](#)^{NEW}

**OPEN经验库**[经验搜索](#)**推荐:**[所有分类](#) > [服务器软件](#)

基于Nginx、Node.js和Redis的Docker工作流

您的评价:[收藏该经验](#)

本文是一篇实践性很强的文章。作者通过一个完整的示例讲述了构建一个基于Nginx、Node.js、Redis的应用服务的Docker流程。推荐所有Docker使用者阅读，并根据文章实践。

在我的前一篇 [文章](#) 中，我已经介绍了关于容器和Docker是如何影响PaaS、微服务和云计算的。如果你刚刚接触Docker和容器，我强烈建议你先读一读我之前的 [文章](#)。作为之前文章的一个延续，在本文中我仍会讲述一些Docker工作流实例的内容。你可以在 [GitHub上找到所有的代码示例](#)。

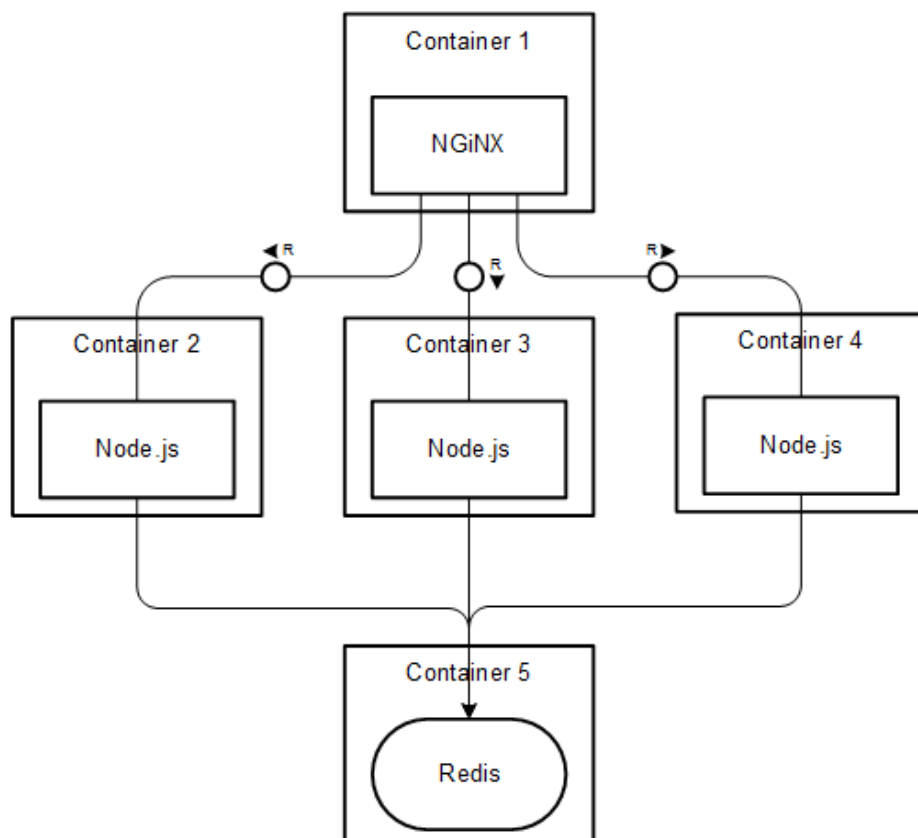
在这个例子中，我有一个非常简单的Node.js应用，它实现了一个递增的计数器并且将数据存储到Redis上。为了保证应用的高可扩展的能力，我会独立运行Redis和Node应用。

让我们先谈谈容器，特别是Docker容器。我们要为每个服务/进程分配一个容器！

- 1个Redis容器
- 3个Node容器
- 1个Nginx容器

因此，整体架构看上去是这样的：





我可以用Docker命令来构建容器, 但为了更加简单, 我推荐使用Dockerfile。我也用Docker Compose去编排应用连接容器。

首先, 我先介绍下如何定义容器。

-----华丽的分割线-----
修改日期: 2015年4月3日

有多种方法来配置一个Dockerfile和镜像层次。一个方法, 将启动一个基于操作系统的镜像, 如Ubuntu, 并建立自己的应用和在这之上的依赖项。另一个可能是最理想的方法是为你的具体使用而使用一个预建的镜像。[Docker Hub Registry](#)有许多用于构建流行应用和其依赖的预建镜像, 这些可以直接用。

我会修改例子来演示不同的使用情况。我将演示为Redis容器使用一个预建镜像, 为Nginx容器使用一个预建的自定义配置的镜像和一个构建在Ubuntu镜像上的Node容器。

Redis 容器

让我们使用官方的 [Redis镜像](#)从Docker Hub上的Redis容器。它预打包了Redis服务的安装, 运行在默认端口6379。所以你只要默认配置ok就不需要修改任何配置, 直接创建并运行Redis容器镜像:

```
docker run -d --name redis -p 6379:6379 redis
```

如果你想从基于Ubuntu的镜像构建Redis镜像, Dockerfile会是这样:

```
# Set the base image to Ubuntu
FROM ubuntu

# File Author / Maintainer
MAINTAINER Anand Mani Sankar

# Update the repository and install Redis Server
RUN apt-get update && apt-get install -y redis-server
```

```
# Expose Redis port 6379
EXPOSE      6379

# Run Redis Server
ENTRYPOINT  ["/usr/bin/redis-server"]
```

Node 容器

让我们来看看Node应用。我不想做太多的解释。我做的是在每个请求使用Redis的INCR的递增的一个视图计数器。我使用 [node-redis](#) 模块连同 [hiredis](#) 从而获得更好的性能。(Yeah, 超高性能的视图计数器不会受损！)

```
var express = require('express'),
    http = require('http'),
    redis = require('redis');

var app = express();

console.log(process.env.REDIS_PORT_6379_TCP_ADDR + ':' + process.env.REDIS_PORT_6379_TCP_PORT);

// APPROACH 1: Using environment variables created by Docker
// var client = redis.createClient(
//   process.env.REDIS_PORT_6379_TCP_PORT,
//   process.env.REDIS_PORT_6379_TCP_ADDR
// );

// APPROACH 2: Using host entries created by Docker in /etc/hosts (RECOMMENDED)
var client = redis.createClient('6379', 'redis');

app.get('/', function(req, res, next) {
  client.incr('counter', function(err, counter) {
    if(err) return next(err);
    res.send('This page has been viewed ' + counter + ' times!');
  });
});

http.createServer(app).listen(process.env.PORT || 8080, function() {
  console.log('Listening on port ' + (process.env.PORT || 8080));
});
```

你可能已经注意到用于地址和端口的Redis服务的环境变量, 这些环境变量是在容器连接时由Docker定义, 以方便容器间通讯。

-----华丽的分割线-----
修改日期: 2015年3月31日

Docker, 除了创建环境变量, 还会更新/etc/hosts文件中的主机记录。事实上, Docker官方推荐使用/etc/hosts文件来替代环境变量, 因为如果源容器重启的时候, 环境变量并不会自动更新。

接下来我们将使用另一种方法来构建Node容器。我们从一个基本的Ubuntu镜像开始, 并逐步在上面添加Node和它的依赖项。

Node Dockerfile:

```
# Set the base image to Ubuntu
FROM      ubuntu

# File Author / Maintainer
MAINTAINER Anand Mani Sankar

# Update the repository
RUN apt-get update
```



```
# Install Node.js and other dependencies
RUN apt-get -y install curl
RUN curl -sL https://deb.nodesource.com/setup | sudo bash -
RUN apt-get -y install python build-essential nodejs

# Install nodemon
RUN npm install -g nodemon

# Bundle app source
COPY . /src

# Install app dependencies
RUN cd /src; npm install

# Expose port
EXPOSE 8080

# Run app using nodemon
CMD ["nodemon", "/src/index.js"]
```

上面的Dockerfile解释如下：

- 从Docker Hub拉取Ubuntu基础镜像
- 使用apt-get安装Node.js以及依赖
- 使用npm安装nodemon
- 从host目录复制应用源码到容器内src
- 运行npm install安装Node应用依赖
- 端口8080从容器抛出，使用nodemon运行应用

使用Dockerfile构建一个Docker镜像：

```
docker build -t msanand/node .
```

从自定义镜像中创建一个Node容器并连接Redis容器：

```
docker run -d --name node -p 8080 --link redis:redis msanand/node
```

由于我计划在3个node服务器之间做负载均衡，我会创建3个容器 — node1、node2和node3。

请注意，Redis容器将会连接到Node容器，所以Node容器可以通过Docker创建的主机记录或者环境变量定义的IP地址和端口来与Redis容器交互。

有了这一点，我有一个Node应用显示一个视图计数器并将数据保存在Redis。让我们来看看如何使用Nginx来做负载均衡。

NGINX容器

Nginx的核心是它的配置：一个conf文件。我使用一个简单的Nginx配置文件定义3个upstream server：

```
worker_processes 4;

events { worker_connections 1024; }

http {

    upstream node-app {
        least_conn;
        server node1:8080 weight=10 max_fails=3 fail_timeout=30s;
        server node2:8080 weight=10 max_fails=3 fail_timeout=30s;
        server node3:8080 weight=10 max_fails=3 fail_timeout=30s;
    }
}
```



```
server {
    listen 80;

    location / {
        proxy_pass http://node-app;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

我已经注册一个node-appupstream server, 它会负载均衡3个服务:node1、node2、node3。我还通过一个全等加权least_conn负载平衡策略指定了每个服务器上的活动连接数的负载均衡。或者, 你可以使用一个基于负载均衡方法的罗宾循环(round robin)或IP哈希或键哈希。Nginx监听80端口, 它基于负载均衡策略代理请求到上游服务器node-app。如果要了解更多的Nginx的配置我会另外讨论。

为了构建Nginx容器, 我计划从Docker Hub上使用正式的 [Nignx镜像](#)。我将使用一个Dockerfile以及我自定义的Nginx conf文件配置Nignx。

该Dockerfile是最小的一使用Nginx镜像和副本自定义Nginx的配置:

```
# Set nginx base image
FROM nginx

# File Author / Maintainer
MAINTAINER Anand Mani Sankar

# Copy custom configuration file from the current directory
COPY nginx.conf /etc/nginx/nginx.conf
```

构建Docker镜像:

```
docker build -t msanand/nginx .
```

从镜像中创建一个Nginx容器, 并连接到Node容器:

```
docker run -d --name nginx -p 80:80 --link node:node msanand/nginx
```

最后, 我们有个Nginx服务器负载均衡3个Node服务器。回过来谈Node服务器, 他们每一个运行在自己的容器中!

如果我们要创建一个基本的Ubuntu镜像定制Nginx的镜像, Dockerfile会是这个样子:

```
# Set the base image to Ubuntu
FROM ubuntu

# File Author / Maintainer
MAINTAINER Anand Mani Sankar

# Install Nginx

# Add application repository URL to the default sources
# RUN echo "deb http://archive.ubuntu.com/ubuntu/ raring main universe" >> /etc/apt/sources.list

# Update the repository
RUN apt-get update

# Install necessary tools
RUN apt-get install -y nano wget dialog net-tools

# Download and Install Nginx
RUN apt-get install -y nginx

# Remove the default Nginx configuration file
RUN rm -v /etc/nginx/nginx.conf
```

```
# Copy a configuration file from the current directory
ADD nginx.conf /etc/nginx/

# Append "daemon off;" to the configuration file
RUN echo "daemon off;" >> /etc/nginx/nginx.conf

# Expose ports
EXPOSE 80

# Set the default command to execute when creating a new container
CMD service nginx start
```

Dockerfile(daemon off)配置了Nginx不以守护进程运行,这也是必须的,因为Docker容器本身就是无状态的,只有当他们所承载的进程运行的时候,容器才有存在的意义。所以把Nginx当成后台进程运行根本不可能。相反,把Nginx作为一个服务运行可以确保容器的正常运行。官方Nginx镜像默认配置也是这样的。

Docker Compose编排应用

Compose是一个使用Docker定义和运行复杂应用的工具。

使用单独的命令来构建镜像并运行和连接容器非常繁琐和复杂,特别是你要运行多个容器的时候。

Docker Compose让你在一个文件中定义多容器应用并用一个命令使应用程序运行起来。

我已经定义一个Docker Compose YAML文件,如下:

```
nginx:
  build: ./nginx
  links:
    - node1:node1
    - node2:node2
    - node3:node3
  ports:
    - "80:80"
node1:
  build: ./node
  links:
    - redis
  ports:
    - "8080"
  volumes:
    - node:/src
node2:
  build: ./node
  links:
    - redis
  ports:
    - "8080"
  volumes:
    - node:/src
node3:
  build: ./node
  links:
    - redis
  ports:
    - "8080"
  volumes:
    - node:/src
redis:
  image: redis
  ports:
    - "6379"
```



YAML文件定义每个容器的名字,指向一个用于构建的Dockerfile。此外,它包含所述容器连接并通过它们暴

露端口。由于Redis容器使用Redis官方镜像, 所以不必构建。

只需要一个命令, Docker Compose就可以构建所需镜像, 并导出所需端口, 然后通过YAML中的定义运行和连接容器。所有你需要做的就是运行docker-compose up, 然后你的5个容器应用就会启动并运行。输入你的主机URL和80端口, 你就可以看到你的视图计数器!

Docker Compose的一个显著特点就是具有动态扩展容器的能力。使用命令docker-compose scale node=5可以扩展容器的数量来运行一个服务。如果你已有一个基于微服务架构的Docker, 你可以轻松地扩展, 并动态地根据负载分配具体服务。理想情况下, 我宁愿定义一个node服务并使用Docker Compose来扩展它。但我还没有想出一个方法来动态地调整Nginx的配置。如果你有这方面的想法, 请在本文后回复。

但这里有个需要注意的是, Docker Compose还不能用于生产环境。文档建议使用在开发环境中, 而不是生产环境。但也有其它的容器编排引擎如我之前的文章 [讨论的Kubernetes](#)。

持续集成和部署

我在我的 [GitHub仓库](#)中配置了2个hook服务(译者注: 作者指的是GitHub Webhook)。

- CircleCI—用于持续集成(以及部署)
- Docker Hub -用于Docker构建(continuous Docker builds)

CircleCI YAML配置文件看这儿:


```
machine:
  services:
    - docker

dependencies:
  override:
    - sudo pip install -U docker-compose

test:
  override:
    - docker-compose run -d --no-deps node1
    - cd node; mocha
```

YAML配置文件使用CircleCI提供的Docker服务, 并安装Docker Compose依赖, 创建了Node容器(未连接到Redis容器)。它使用Mocha(译者注: Mocha是一个基于Node.js和浏览器的集合各种特性的JavaScript测试框架, 并且可以让异步测试也变的简单和有趣。Mocha的测试是连续的, 在正确的测试条件中遇到未捕获的异常时, 会给出灵活且准确的报告。Mocha托管在Github上)在Node应用上触发测试, 这确保了GitHub上每个提交都会对应一个测试。

🏠 / msanand / docker-workflow ⚙️ Project Settings

Build	Revision	Branch	Author	Log	Started at	Length	Status
#3	2ec7ae7	master	Anand M.S	Updated circle yaml	4 minutes ago	02:38	Fixed
#2	b0f1989	master	Anand M.S	Mocha test	6 minutes ago	00:35	
#1	492e946	master	Anand Mani Sankar	Updated Nginx conf	1 hour ago	00:06	No Tests

每次提交都会触发我的 [Docker Hub Repository](#)进行一次构建。这确保在Docker Hub中通过持续部署到生产环境的最终镜像总是可用的。生产环境能在任何时间从Docker Hub和从容器中编排的应用中能拉到最终镜像。

以上是我的一个基于Nginx、Node.js和Redis的Docker流程实例。如果你有任何建议和更好的方法, 请发表评

论。

原文链接: [A sample Docker workflow with Nginx, Node.js and Redis](#) (翻译: 吴锦晟 校对: 李颖杰)

来自: <http://dockerone.com/article/291>

相关资讯 — 更多

- [我的碎碎念: Docker入门指南](#)
- [使用 Docker Swarm 对 Docker 进行规模扩展](#)
- [云PaaS向容器生态系统的演变之路](#)
- [来自官方映像的 6 个 Dockerfile 技巧](#)
- [从容器和Kubernetes技术看现代云计算的发展轨迹](#)
- [IBM Bluemix开启云开发时代: 基于Cloud Foundry开源项目的PaaS服务](#)
- [解读2014之云计算篇: 有一种态度叫做“拥抱”\(下\)](#)
- [一个很有借鉴价值的编程故事](#)
- [《JavaScript快速全栈开发》作者Azat Mardanov: 现在是拥抱Node技术栈的最佳时机\(图灵访谈\)](#)
- [Docker周报: Docker获C轮4000万美元融资](#)
- [Docker周报: Microsoft发布全新的容器技术](#)
- [2015年十大热门开源Docker开发工具分类盘点](#)
- [去IOE真相: 亚马逊BAT们, 对传统IT的深刻颠覆](#)
- [Docker周报第18期](#)
- [Docker 传奇之 dotCloud](#)
- [为什么我要用 Node.js? 案例逐一介绍](#)
- [十大基于Docker的开发工具](#)
- [丢进“开源垃圾场”的法拉利? Joyent 开源云技术挑战OpenStack](#)
- [Bowery为什么放弃Node.js, 转向Go?](#)
- [GitHub上整理的一些工具](#)

相关文档 — 更多

- [田嘉林-打造基于docker的paas平台.pdf](#)
- [Docker & Docker Hub \(version 2\).pdf](#)
- [Spdy 简介文档.pdf](#)
- [Docker手册-中文版.pdf](#)
- [Docker在生产环境的挑战和应对.pdf](#)
- [RelateIQ超过1年的Docker使用经验.pptx](#)
- [The Docker Way.pdf](#)
- [docker中文版.pdf](#)
- [马全一-构建 Docker 的开发环境.pdf](#)
- [Docker 从入门到实践.pdf](#)
- [Docker & Docker Hub \(version3\).pptx](#)
- [学习docker的一个小总结waitfish.pdf](#)
- [Docker 实战培训.ppt](#)
- [Docker vs. Rocket: 技术方案差异性剖析.pdf](#)
- [Musings on Mesos: Docker, Kubernetes, and Beyond.pdf](#)
- [从Docker容器漏洞谈Docker安全_岑义涛.pdf](#)
- [刘永峰-Docker时代公有云面临的挑战和机遇.pdf](#)
- [The Docker Book.pdf](#)
- [介绍Docker Swarm + Mesos.pdf](#)
- [为什么要用 Go 开发 Docker.pdf](#)

[联系我们](#) - [问题反馈](#)

2005-2015 OPEN-OPEN, all rights reserved.

