

# 51 | 多容器部署：如何利用 Docker Compose 快速搭建本地爬虫环境？

2023-02-07 郑建勋 来自北京

《Go进阶·分布式爬虫实战》

[课程介绍 >](#)



讲述：郑建勋

时长 07:11 大小 6.57M



你好，我是郑建勋。

这节课，我们一起来学习如何使用 Docker Compose 来部署多个容器。

## 什么是 Docker Compose？

那什么是 Docker Compose 呢？

一句话解释，Docker Compose 一般用于开发环境，负责部署和管理多个容器。

现代的应用程序通常由众多微服务组成，就拿我们的爬虫服务来说，它包含了 Master、Worker、etcd、MySQL，未来还可能包含前端服务、日志采集服务、鉴权服务等等。部署和管理许多像这样的微服务可能很困难，而 Docker Compose 就可以解决这一问题。

领资料



Docker Compose 并不是简单地将多个容器脚本和 Docker 命令排列在一起，它会让你在单个声明式配置文件（例如 `docker-compose.yaml`）中描述整个应用程序，并使用简单的命令进行部署。部署应用程序之后，你可以用一组简单的命令来管理应用程序的整个生命周期。

Docker Compose 的前身是 Fig。Fig 由 Orchard 公司创建，它是管理多容器的最佳方式。Fig 是一个位于 Docker 之上的 Python 工具，它可以让你在单个 YAML 文件中定义整个容器服务。然后，使用 `fig` 命令行工具就可以部署和管理应用程序的生命周期。Fig 通过读取 YAML 文件和 Docker API 来部署和管理应用程序。

后来，Docker Inc 公司收购了 Orchard 并将 Fig 重新命名为 Docker Compose，而命令行工具也从 `fig` 重命名为了 `docker-compose`。Docker Compose 仍然是在 Docker 之上的外部工具，它从未完全集成到 Docker 引擎中，但却一直很受欢迎并被广泛使用。

Docker Compose 目前仍然是通过 Python 开发的工具。借助 Docker Compose，你可以在 YAML 文件中定义多个服务，并由 `docker-compose` 对文件完成解析，然后借助 Docker API 部署容器。2020 年 4 月，[Compose 规范](#) 正式发布，它的目的是定义一个多容器，平台无关应用程序的标准。Docker Compose 就是基于该规范实现的。

## Compose 的安装

下面我们来看看如何安装 Docker Compose。

安装 Docker Compose 最简单的方法是安装 Docker Desktop。之前，我们已经看到了如何通过简单的界面化的方式安装 Docker Desktop。Docker Desktop 中包括 Docker Compose、Docker Engine 以及 Docker CLI。要想通过其他方式安装，你也可以查看[官方安装文档](#)。

接下来，我们执行以下命令可以验证是否拥有了 Docker Compose。

```
1 » docker-compose --version
2 Docker Compose version v2.13.0
```

 复制代码

 领资料

## Compose 配置文件的编写

Compose 使用 YAML 和 JSON 格式的配置文件来定义多服务应用程序。其中默认的配置文件的名称是 `docker-compose.yml`。但是，你也可以使用 `-f` 标志来指定自定义的配置文件。

下面我们为爬虫项目书写第一个简单的 `docker-compose.yml` 文件，如下所示。

 复制代码

```
1 version: "3.9"
2 services:
3   worker:
4     build: .
5     command: ./crawler worker
6     ports:
7       - "8080:8080"
8     networks:
9       - counter-net
10    volumes:
11      - /tmp/app:/app
12    depends_on:
13      mysql:
14        condition: service_healthy
15  mysql:
16    image: mysql:5.7
17    # restart: always
18    environment:
19      MYSQL_DATABASE: 'crawler'
20      MYSQL_USER: 'myuser'
21      MYSQL_PASSWORD: 'mypassword'
22      # Password for root access
23      MYSQL_ROOT_PASSWORD: '123456'
24      # docker-compose默认时区UTC
25      TZ: 'Asia/Shanghai'
26    ports:
27      - '3326:3306'
28    expose:
29      # Opens port 3306 on the container
30      - '3306'
31      # Where our data will be persisted
32    volumes:
33      - /tmp/data:/var/lib/mysql
34    networks:
35      counter-net:
36    healthcheck:
37      test: ["CMD", "mysqladmin" ,"ping", "-h", "localhost"]
38      interval: 5s
39      timeout: 5s
40      retries: 55
41  networks:
42    counter-net:
```

领资料

在这个例子中，`docker-compose.yml` 文件的根级别有 3 个指令。

- **version**

**version** 指令是 **Compose** 配置文件中必须要有的，它始终位于文件的第一行。**version** 定义了 **Compose** 文件格式的版本，我们这里使用的是最新的 3.9 版本。注意，**version** 并未定义 **Docker Compose** 和 **Docker** 的版本。

- **services**

**services** 指令用于定义应用程序需要部署的不同服务。这个例子中定义了两个服务，一个是我们爬虫项目的 **Worker**，另一个是 **Worker** 依赖的 **MySQL** 数据库。

- **networks**

**networks** 的作用是告诉 **Docker** 创建一个新网络。默认情况下，**Compose** 将创建桥接网络。但是，你可以使用 **driver** 属性来指定不同的网络类型。

除此之外，根级别配置中还可以设置其他指令，例如 **volumes**、**secrets**、**configs**。其中，**volumes** 用于将数据挂载到容器，这是持久化容器数据的最佳方式。**secrets** 主要用于 **swarm** 模式，可以管理敏感数据，安全传输数据（这些敏感数据不能直接存储在镜像或源码中，但在运行时又需要）。**configs** 也用于 **swarm** 模式，它可以管理非敏感数据，例如配置文件等。

更进一步地，让我们来看看 **services** 中定义的服务。在 **services** 中我们定义了两个服务 **Worker** 和 **MySQL**。**Compose** 会将每一个服务部署为一个容器，并且容器的名字会分别包含 **Worker** 与 **MySQL**。

在对 **Worker** 服务的配置中，各个配置的含义如下所示。

- **build** 用于构建镜像，其中 **build:** 告诉 **Docker** 使用当前目录中的 **Dockerfile** 构建一个新镜像，新构建的镜像将用于创建容器。
- **command**，它是容器启动后运行的应用程序命令，该命令可以覆盖 **Dockerfile** 中设置的 **CMD** 指令。
- **ports**，表示端口映射。在这里，"**SRC:DST**" 表示将宿主机的 **SRC** 端口映射到容器中的 **DST** 端口，访问宿主机 **SRC** 端口的请求将会被转发到容器对应的 **DST** 端口中。
- **networks**，它可以告诉 **Docker** 要将服务的容器附加到哪个网络中。
- **volumes**，它可以告诉 **Docker** 要将宿主机的目录挂载到容器内的哪个目录。





- `depends_on`，表示启动服务前需要首先启动的依赖服务。在本例中，启动 **Worker** 容器前必须先确保 **MySQL** 可正常提供服务。

而在对 **MySQL** 服务的定义中，各个配置的含义如下所示。

- `image`，用于指定当前容器启动的镜像版本，当前版本为 `mysql:5.7`。如果在本地查找不到镜像，就从 **Docker Hub** 中拉取。
- `environment`，它可以设置容器的环境变量。环境变量可用于指定当前 **MySQL** 容器的时区，并配置初始数据库名，根用户的密码等。
- `expose`，描述性信息，表明当前容器暴露的端口号。
- `networks`，用于指定容器的命名空间。**MySQL** 服务的 `networks` 应设置为和 **Worker** 服务相同的 `counter-net`，这样两个容器共用同一个网络命名空间，可以使用回环地址进行通信。
- `healthcheck`，用于检测服务的健康状况，在这里它和 `depends_on` 配合在一起可以确保 **MySQL** 服务状态健康后再启动 **Worker** 服务。

要使用 **Docker Compose** 启动应用程序，可以使用 `docker-compose up` 指令，它是启动 **Compose** 应用程序最常见的方式。`docker-compose up` 指令可以构建或拉取所有需要的镜像，创建所有需要的网络和存储卷，并启动所有的容器。

如下所示，我们输入 `docker-compose up`，程序启动后可能会打印冗长的启动日志，等待几秒钟之后，服务就启动好了。根据我们的配置，将首先启动 **MySQL** 服务，接着启动 **Worker** 服务。

```
1 » docker-compose up
2 [+] Running 2/0
3  :: Container crawler-mysql-1          Created
4  :: Container crawler-crawler-worker-1 Created
5 Attaching to crawler-crawler-worker-1, crawler-mysql-1
```

 复制代码

领资料

默认情况下，`docker-compose up` 将查找名称为 `docker-compose.yml` 的配置文件，如果你有自定义的配置文件，需要使用 `-f` 标志指定它。另外，使用 `-d` 标志可以在后台启动应用程序。

现在，应用程序已构建好并开始运行了，我们可以使用普通的 `docker` 命令来查看 `Compose` 创建的镜像、容器、网络。

如下所示，`docker images` 指令可以查看到我们最新构建好的 `Worker` 镜像。

复制代码

```
1 » docker images
2 REPOSITORY          TAG          IMAGE ID      CREATED      SIZE
3 crawler-crawler-worker  latest      1fec0f6fc04e  23 hours ago  41.3MB
```

`docker ps` 可以查看当前正在运行的容器，可以看到 `Worker` 与 `MySQL` 都已经正常启动了。

复制代码

```
1 » docker ps
2 CONTAINER ID        IMAGE          COMMAND                  CREATED
3 a43f4ed671fc        crawler-crawler-worker  "/./crawler worker"      2 minutes ago
4 2bd879656049        mysql:5.7      "docker-entrypoint.s..." 38 minutes ago
```

接着，我们执行 `docker network ls`，可以看到 `dokcek` 创建了一个新的网络 `crawler_counter-net`，它为桥接模式。

复制代码

```
1 » docker network ls
2 NETWORK ID          NAME              DRIVER      SCOPE
3 ef63428fb70e        bridge           bridge      local
4 71d238bd7e46        crawler_counter-net  bridge      local
5 1fa0c4c53670        host             host        local
6 04d433213cca        localnet         bridge      local
7 25c4683eb897        none            null        local
```

## Compose 生命周期管理



接下来，我们看看如何使用 `Docker Compose` 启动、停止和删除应用程序，实现对于多容器应用程序的管理。


当应用程序启动后，使用 `docker-compose ps` 命令可以查看当前应用程序的状态。和 `docker ps` 类似，你可以看到两个容器、容器正在运行的命令、当前运行的状态以及监听的网络端

口。

 复制代码

```
1 » docker-compose ps
2 NAME                                COMMAND                                SERVICE    STATUS
3 crawler-mysql-1                    "docker-entrypoint.s..."           mysql      running (healt
4 crawler-worker-1                    "./crawler worker"                   worker      running
```

使用 **docker-compose top** 可以列出每个服务（容器）内运行的进程，返回的 **PID** 号是从宿主机看到的 **PID** 号。

 复制代码

```
1 » docker-compose top
2 crawler-mysql-1
3  UID      PID      PPID      C    STIME   TTY      TIME          CMD
4  999      71494    71468     0    14:58   ?        00:00:00      mysqld
5
6 crawler-worker-1
7  UID      PID      PPID      C    STIME   TTY      TIME          CMD
8  root      71773    71746     0    14:58   ?        00:00:00      ./crawler worker
```

如果想要关闭应用程序，可以执行 **docker-compose down**，如下所示。

 复制代码

```
1 » docker-compose down
2 [+] Running 3/3
3  :: Container crawler-worker-1    Removed
4  :: Container crawler-mysql-1    Removed
5  :: Network crawler_counter-net   Removed
```

要注意的是，**docker-compose up** 构建或拉取的任何镜像都不会被删除，它们仍然存在于系统中，这意味着下次启动应用程序时会更快。同时我们还可以看到，当前挂载到宿主机的存储目录并不会随着 **docker-compose down** 而销毁。

 领资料

同样，使用 **docker-compose stop** 命令可以让应用程序暂停，但不会删除它。再次执行 **docker-compose ps**，可以看到应用程序的状态为 **exited**。

```
1 » docker-compose ps
2 NAME                                COMMAND                                SERVICE    STATUS
3 crawler-mysql-1                    "docker-entrypoint.s..."           mysql      exited (0)
4 crawler-worker-1                   "./crawler worker"                   worker     exited (0)
```

[复制代码](#)

因为 `docker-compose stop` 而暂停的容器，之后再执行 `docker-compose restart` 就可以重新启动。

```
1 » docker-compose restart
2 [+] Running 2/2
3  :: Container crawler-mysql-1   Started
4  :: Container crawler-worker-1 Started
```

[复制代码](#)

最后，整合了 Master，Worker，MySQL 和 etcd 服务的 Compose 配置文件如下所示。具体的你可以查看项目最新分支的 `docker-compose.yml` 文件。

```
1 version: "3.9"
2 services:
3   worker:
4     build: .
5     command: ./crawler worker --id=2 --http=:8080 --grpc=:9090
6     ports:
7       - "8080:8080"
8       - "9090:9090"
9     networks:
10      - counter-net
11     volumes:
12      - /tmp/app:/app
13     depends_on:
14       mysql:
15         condition: service_healthy
16       etcd:
17         condition: service_healthy
18   master:
19     build: .
20     command: ./crawler master --id=3 --http=:8082 --grpc=:9092
21     ports:
22       - "8082:8082"
23       - "9092:9092"
24     networks:
25      - counter-net
26     volumes:
27      - /tmp/app:/app
```

[复制代码](#)[领资料](#)



```
28     depends_on:
29         mysql:
30             condition: service_healthy
31         etcd:
32             condition: service_healthy
33 mysql:
34     image: mysql:5.7
35     # restart: always
36     environment:
37         MYSQL_DATABASE: 'crawler'
38         MYSQL_USER: 'myuser'
39         MYSQL_PASSWORD: 'mypassword'
40         # Password for root access
41         MYSQL_ROOT_PASSWORD: '123456'
42         # docker-compose默认时区UTC
43         TZ: 'Asia/Shanghai'
44     ports:
45         - '3326:3306'
46     expose:
47         # Opens port 3306 on the container
48         - '3306'
49         # Where our data will be persisted
50     volumes:
51         - /tmp/data:/var/lib/mysql
52     networks:
53         counter-net:
54     healthcheck:
55         test: ["CMD", "mysqladmin" ,"ping", "-h", "localhost"]
56         interval: 5s
57         timeout: 5s
58         retries: 55
59     etcd:
60         image: gcr.io/etcd-development/etcd:v3.5.6
61         volumes:
62             - /tmp/etcd:/etcd-data
63         ports:
64             - '2379:2379'
65             - '2380:2380'
66         expose:
67             - 2379
68             - 2380
69         networks:
70             counter-net:
71         environment:
72             - ETCDCTL_API=3
73         command:
74             - /usr/local/bin/etcd
75             - --data-dir=/etcd-data
76             - --name
77             - etcd
78             - --initial-advertise-peer-urls
79             - http://0.0.0.0:2380
```

```

80     - --listen-peer-urls
81     - http://0.0.0.0:2380
82     - --advertise-client-urls
83     - http://0.0.0.0:2379
84     - --listen-client-urls
85     - http://0.0.0.0:2379
86     - --initial-cluster
87     - etcd=http://0.0.0.0:2380
88     - --initial-cluster-state
89     - new
90     - --initial-cluster-token
91     - tkn
92   healthcheck:
93     test: ["CMD", "/usr/local/bin/etcdctl", "get", "--prefix", "/"]
94     interval: 5s
95     timeout: 5s
96     retries: 55
97
98   networks:
99     counter-net:

```

在这之后，我们就可以方便地测试最新的代码了。如下所示，调用 **Master** 添加资源接口之后，**Worker** 将能够正常地爬取网站。

 复制代码

```

1 » curl -H "content-type: application/json" -d '{"id":"zjx","name": "douban_book'
2 {"id":"go.micro.server.worker-2", "Address":"172.22.0.5:9090"}

```

## 总结

这节课，我们学习了如何使用 **Docker Compose** 部署和管理多容器应用程序。**Docker Compose** 是一个运行在 **Docker** 之上的 **Python** 应用程序。它允许你在单个声明式配置文件中描述多容器应用程序，并使用简单的命令进行管理。

**Docker Compose** 默认的配置文件中为当前目录下的 **docker-compose.yml** 文件。配置文件中可以书写丰富的自定义配置，以此控制容器的行为。这节课我们了解了其中最常用的一些，其他的参数你可以查阅参考文档。

要注意的是，编写配置参数时候需要配置参数的缩进。例如，描述服务的 **networks** 参数和根级别的 **networks** 参数的含义是截然不同的。在实践中我们一般会复制一个模版文件，并在此基础上将其改造为当前项目的配置。



Docker Compose 多是用在单主机的开发环境中。在更大规模的生产集群中，我们一般会使用 Kubernetes 等容器编排技术，这部分内容我们后续会介绍。


## 课后题



这节课的思考题如下。

你认为，执行 `docker-compose down` 关闭容器时，挂载到容器中的 `volume` 会被销毁吗？为什么要这样设计呢？

欢迎你在留言区与我交流讨论，我们下节课再见！

分享给需要的人，Ta购买本课程，你将得 20 元

 生成海报并分享

 赞 1     提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇    50 | 不可阻挡的容器化：Docker核心技术与原理

下一篇    52 | 容器海洋中的舵手：Kubernetes工作机制

领资料



学习推荐

全新汇总

# Go 面试必考 300+ 题

面试真题 | 进阶实战 | 专题视频 | 学习路线

限时免费

仅限 99 名

## 精选留言 (1)

写留言



Realm

2023-02-07 来自浙江

思考题:

docker-compose down时, 会自动删除原有容器以及虚拟网。但是其中定义的volumes会保留。

如果要down的同时清理干净, 就直接加参数--volumes.

这样做是为了保护用户数据, 下次启动容器可以直接用。



1

领资料