容器单机编排实...

> 本节是第四部分“架构篇”的第八节，前面几节除了 Docker 核心组件与 Plugin 外，我还为你介绍了 Docker 的监控和日志。本节，我来为你介绍 Docker 容器的单机编排工具 docker-compose。

在之前章节中，我们大多都是启动一个独立的容器，并用它进行相关的实践。但在实际生产或者项目中使用 Docker 容器时，往往不是一个容器就能满足需求的。

比如对于一个常规的 Web 应用而言，前后端，数据库均需要独立的容器，这个时候便非常需要进行容器的“编排”了。

那么什么是容器的“编排”呢？通常情况下，我们将它理解为按照固定的规则，将容器的生命周期组织起来的行为，便称之为容器的编排。当然，有时候它还涉及到容器的启动顺序或是容器的互通性之类的。

在 Linux 上 systemd 做了一些类似的事情，可以控制服务单元的启动顺序及执行的命令之类的，但使用 systemd 毕竟有一些门槛， 为了满足用户对容器编排的需求，同时也为了降低复杂性，Docker 为我们提供了一个工具 docker-compose，可用于单个机器上的 Docker 容器编排。

docker-compose 介绍

docker-compose 是由 Docker 官方提供的工具，到目前为止最新的版本是 v1.25.1。

事实上 docker-compose 的最初版本 v0.0.1 是在 2013 年 12 月就发布了，当时它的名字叫 fig 到了 2014 年底才改成现在的 docker-compose，它一开始的目标是利用 Docker 创建轻量级的开发环境。

发展到现在，它已经是一个功能相当完备，满足绝大多数场景的单机 Docker 容器编排工具了。

但通过合理的配置，它也可以用于部署应用到远程的 Docker 中，或者和 Docker 的集群方案 Swarm 结合使用。

安装

docker-compose 是由 Python 编写的，安装起来也很方便，直接使用 Python 的包管理器 pip 进行安装即可。也可在 docker-compose 项目的 Release 页面，下载对应打包好的二进制文件，进行安装。

复制

```
(MoeLove) ➜  pip install -q -U docker-compose
(MoeLove) ➜  ~ docker-compose version
docker-compose version 1.25.1, build a82fef0
docker-py version: 4.1.0
CPython version: 3.7.0
OpenSSL version: OpenSSL 1.0.2o-fips  27 Mar 2018
```

它在安装过程中会自动将一些所需的依赖安装好，但同时也需要注意： **docker-compose v1.25.x 将会是最后一个支持 Python 2.x 系列的版本，之后版本需要使用 Python 3.x。**

使用

这里我们直接使用上上节《容器监控实践》 作为我们实践的目标，使用过程中再介绍 docker-compose 相关的重点知识。

前提

- 正确安装好 docker-compose 工具（课程内容使用了 v1.25.1）
- 正确安装并运行的 Docker（课程内容使用了 v19.03.5）

docker-compose 默认的配置文件为当前目录下的 docker-compose.yml，你要是把文件名后缀修改成 .yaml 也同样是可以工作的。

基本内容

每个 docker-compose.yml 都需要包含一个 version 的配置，这个 version 表示 docker-compose 配置文件格式的版本，不同的版本会影响到与 Docker 版本的兼容性。

docker-compose 文件的版本	Docker 的版本
3.7	18.06.0+
3.6	18.02.0+
3.5	17.12.0+
3.4	17.09.0+
3.3	17.06.0+
3.2	17.04.0+
3.1	1.13.1+
3.0	1.13.0+
2.4	17.12.0+

docker-compose 文件的版本	Docker 的版本
2.3	17.06.0+
2.2	1.13.0+
2.1	1.12.0+
2.0	1.10.0+
1.0	1.9.1.+

例如，如果你想要使用高版本的 docker-compose 文件格式，那么首先需要你的 docker-compose 版本支持，其次需要确认你使用的 docker-compose 配置版本或者你使用的特性是否与 Docker 版本兼容。

具体来看，比如我使用了最新版的 docker-compose (v1.25.1) 它是支持 `version: "3.7"` 的，而我运行了一个 17.03 版本的 Docker，使用如下内容的配置文件是否能正常启动呢？

复制

```
version: "3.7"
services:
  redis:
    image: redis
```

答案是可以。

上面的兼容性表格只是表示是否能支持**全部完整**的功能。

除去上面提到的 version 字段外，下一个字段就是 services 字段，它定义了我们要编排哪些容器，以及定义了它们之间的关系之类的。

了解了这两个基础内容，我们便开始实践吧。

实践

在写配置之前，我们先明确下需要哪些组件以及它们的关系。最简单的组件是：

- Prometheus server 用于执行监控和存储监控数据
- node-exporter 用于收集机器的相关指标

需要满足 Prometheus server 可以正常访问 node-exporter 组件监听的端口。

复制

```
version: "3.7"
services:
  prometheus:
    image: prom/prometheus:v2.15.1
    volumes:
      - ./prometheus/:/etc/prometheus/
      - prometheus_data:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--web.console.libraries=/usr/share/prometheus/console_libraries'
      - '--web.console.templates=/usr/share/prometheus/consoles'
    ports:
      - 9090:9090
    restart: always

  node-exporter:
    image: prom/node-exporter:v0.18.1
    volumes:
      - /proc:/host/proc:ro
      - /sys:/host/sys:ro
      - /:/rootfs:ro
    command:
      - '--path.procfs=/host/proc'
      - '--path.sysfs=/host/sys'
      - --collector.filesystem.ignored-mount-points
      - "^/(sys|proc|dev|host|etc|rootfs/var/lib/docker/containers|rootfs/var/lib/
    ports:
      - 9100:9100
    restart: always

volumes:
  prometheus_data: {}
```

在当前目录（我当前的目录名称为 c）下创建一个名为 prometheus 的目录，将配置文件放入其中：

复制

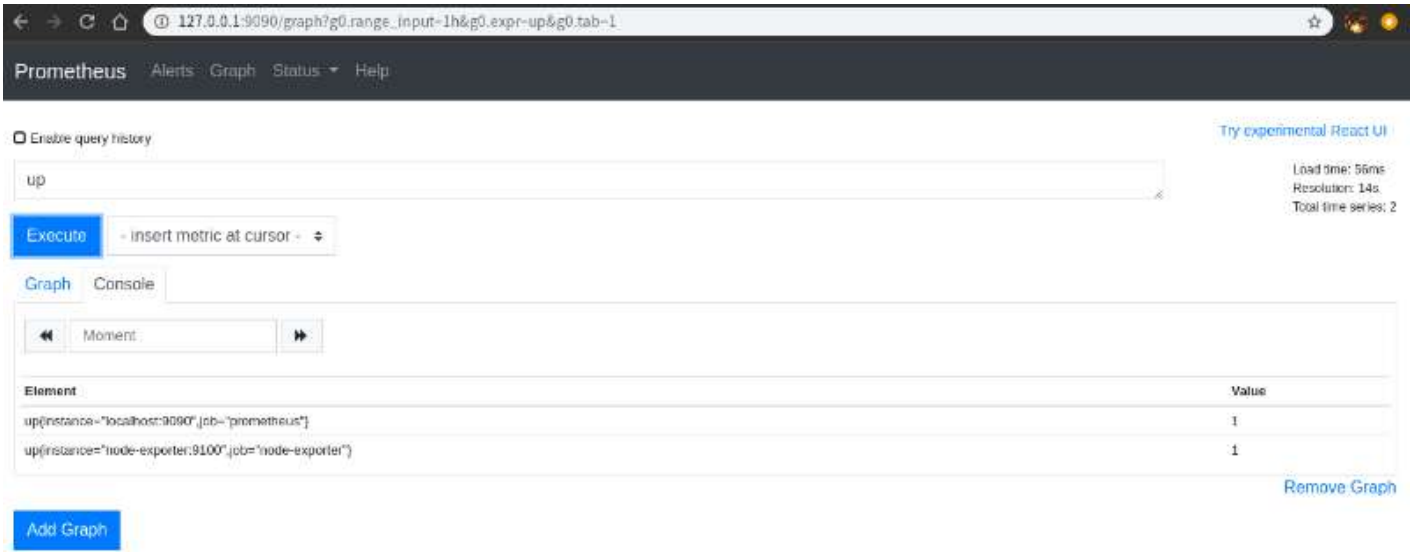
```
(MoeLove) ➜  c tree
.
├── docker-compose.yml
└── prometheus
    └── prometheus.yml

1 directory, 2 files
(MoeLove) ➜  c cat prometheus/prometheus.yml
global:
  scrape_interval:     15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node-exporter'

    static_configs:
      - targets: ['node-exporter:9100']
```

使用 docker-compose 启动相关容器：

复制

```
(MoeLove) ➜  c docker-compose up -d
Creating network "c_default" with the default driver
Creating c_node-exporter_1 ... done
Creating c_prometheus_1    ... done
(MoeLove) ➜  c docker-compose ps
      Name                     Command               State           Ports
-------------------------------------------------------------------------------------
c_node-exporter_1   /bin/node_exporter --path. ...   Up      0.0.0.0:9100->9100/tc
c_prometheus_1      /bin/prometheus --config.f ...   Up      0.0.0.0:9090->9090/tc
```

现在打开浏览器，访问 http://127.0.0.1:9090，即可看到 Prometheus 已正常启动。



我们对 docker-compose.yml 配置文件中的内容稍做解释：

- version 前面已经介绍过，表示 docker-compose 文件的版本；
- services 表示需要编排的容器。默认情况下 docker-compose 会为被编排的容器创建一个新的网络，并使用在 services 下定义的名称作为其 DNS 的名称（内部 DNS 相关的内容，在后续课程中会再行介绍）；
- volumes 中表示定义的存储卷（存储卷在课程第 5 部分会进行介绍）。

另外 `docker-compose up` 用于启动服务，加上 `-d` 参数表示将其启动在后台。

如果关掉启动在后台的服务，则可以使用 `docker-compose stop` 命令；使用 `docker-compose down` 会停止并删除容器，网络等相关的资源，加上 `-v` 参数则会删除存储卷。

复制

```
(MoeLove) ➜  c docker-compose stop
Stopping c_prometheus_1    ... done
Stopping c_node-exporter_1 ... done
(MoeLove) ➜  c docker-compose down
Removing c_prometheus_1    ... done
Removing c_node-exporter_1 ... done
Removing network c_default
```

总结

本节，我为你介绍了 docker-compose 相关的内容。对于在单机上进行容器编排，使用 docker-compose 是非常方便的，同时 docker-compose 也可以很方便的用在 CI 环境中。

docker-compose 提供的能力远不止本节中提到的内容，同样的对于单机上的容器编排也不止 docker-compose 一种方法。 在学习完后续课程的内容，也会对容器的编排有更深入的认识。

下一节，我将为你介绍 Docker 常见的问题定位和调试方法。