25 Docker 简介与消息中间件安装

更新时间：2019-07-15 11:19:13

> 衡量一个人的真正品格，是看他在知道没人看见的时候干些什么。
>
> ——孟德斯鸠

本章要用两篇文章来和大家聊一个新的话题，那就是消息中间件。在微服务中， Spring Cloud Bus （消息总线）通过轻量级消息代理连接各个微服务，在很多场景下我们对 Spring Cloud Bus 有非常迫切的需求，最简单的需求就是上篇文章我们提到的希望在配置文件更新时能够变得更加容易些，使用 Spring Cloud Bus 就可以实现这一需求。Spring Cloud Bus 的一个核心思想是通过分布式的启动器对 Spring Boot 应用进行扩展，也可以用来建立多个应用之间的通信频道。目前常见的实现方式是通过 AMQP 消息代理作为通道。

要学习 Spring Cloud Bus ，我们得先安装消息中间件，这里我们将引入目前流行的 Docker 容器，在 Docker 容器中来安装消息中间件。

Docker 简介

说起 Docker ，我想很多人可能都用过，即使没有用过也应该听说过。Docker 容器如日中天，这不是单纯的炒概念，而是因为 Docker 确确实实解决了开发与运维的痛点，因此 Docker 在企业开发中得到了非常广泛的使用，特别是在微服务中，由于服务数量增多，项目部署难度加大，因此在微服务架构中， Docker 也算是一个基础组件了。考虑到很多人可能对 Docker 的认识还不够全面，因此这里就先来带领大家学习一下 Docker 中一些常见的基本概念，使大家对于 Docker 有一个基本的认识。那么什么是 Docker 呢？

根据wikipedia中的介绍：

Docker是一个开放源代码软件项目，让应用程序布署在软件容器下的工作可以自动化进行，借此在Linux操作系统上，提供一个额外的软件抽象层，以及操作系统层虚拟化的自动管理机制。
Docker利用Linux核心中的资源分脱机制，例如cgroups，以及Linux核心名字空间（name space），来创建独立的软件容器（containers）。这可以在单一Linux实体下运作，避免启动一个虚拟机造成的额外负担。Linux核心对名字空间的支持完全隔离了工作环境中应用程序的视野，包括进程树、网络、用户ID与挂载文件系统，而核心的cgroup提供资源隔离，包括CPU、存储器、block I/O与网络。从0.9版本起，Dockers在使用抽象虚拟是经由libvirt的LXC与systemd - nspawn提供界面的基础上，开始包括libcontainer库做为以自己的方式开始直接使用由Linux核心提供的虚拟化的设施。
依据行业分析公司“451研究”：“Dockers是有能力打包应用程序及其虚拟容器，可以在任何Linux服务器上运行的依赖性工具，这有助于实现灵活性和便携性，应用程序在任何地方都可以运行，无论是公有云、私有云、单机等。”。

这里的介绍有点绕口，让我们来总结一下 Docker 的特点：

简化环境管理

传统的软件开发与发布环境复杂，配置繁琐。经常有读者在微信上问：我的代码开发环境可以运行，一旦部署到服务器上就运行不了了。这个问题很常见，也确实很烦人，但是问题总要解决，开发环境、测试环境、生产环境，每个环节都有可能出现这样那样的问题，如果能够在各个环境中实现一键部署，就会方便很多，例如一键安装 Linux 、一键安装 MySQL 、一键安装 Nginx 等，Docker 容器彻底解决了这个问题。

虚拟化更加轻量级

说到容器，说到虚拟化，很多人总会想到虚拟机，想到 VMware、VirtualBox 等工具。不同于这些虚拟技术，Docker 虚拟化更加轻量级，传统的虚拟机都是先虚拟出一个操作系统，然后在操作系统上完成各种各样的配置，这样并不能充分利用物理机的性能。 Docker 则是一种操作系统级别的虚拟技术，它运行在操作系统之上的用户空间，所有的容器都共用一个系统内核甚至公共库，容器引擎提供了进程级别的隔离，让每个容器都像运行在单独的系统之上，但是又能够共享很多底层资源。因此 Docker 更为轻量、快速和易于管理。

程序可移植

有了前面介绍的两个特点，程序可移植就是顺理成章的事情了。

Docker 和虚拟机

前面介绍了 Docker 与传统虚拟机的差异，通过下表再来详细了解下这种差异：

	Docker	虚拟机
相同点	1. 都可在不同的主机之间迁移 2. 都具备root权限 3. 都可以远程控制 4. 都有备份、回滚操作	
操作系统	在性能上有优势，可以轻易运行多个操作系统	可以安装任何系统，但是性能不及容器
原理	和宿主机共享内核，所有容器运行在容器引擎之上，容器并非一个完整的操作系统，所有容器共享操作系统，在进程级进行隔离	每一个虚拟机都建立在虚拟的硬件之上，提供指令级的虚拟，具备一个完整的操作系统
优点	高效、集中。一个硬件节点可以运行数以百计的容器，非常节省资源，QoS会尽量满足，但不保证一定满足。内核由提供者升级，服务由服务提供者管理	对操作系统具有绝对权限，对系统版本和系统升级具有完全的管理权限。具有一整套的的资源：CPU、RAM和磁盘。QoS是有保证的，每一个虚拟机就像一个真实的物理机一样，可以实现不同的操作系统运行在同一物理节点上。】
资源管理	弹性资源分配：资源可以在没有关闭容器的情况下添加，数据卷也无需重新分配大小	虚拟机需要重启，虚拟机里边的操作系统需要处理新加入的资源，如磁盘等，都需要重新分区
远程管理	根据操作系统的不同，可以通过shell或者远程桌面进行	远程控制由虚拟化平台提供，可以在虚拟机启动之前连接
缺点	对内核没有控制权限，只有容器的提供者具备升级权限。只有一个内核运行在物理节点上，几乎不能实现不同的操作系统混合。容器提供者一般仅提供少数的几个操作系统	每一台虚拟机都具有更大的负载，耗费更多的资源，用户需要全权维护和管理。一台物理机上能够运行的虚拟机非常有限
配置	快速，基本上是一键配置	配置时间长

	Docker	虚拟机
启动时间	秒级	分钟级
硬盘使用	MB	GB
性能	接近原生态	弱于原生态
系统支持数量	单机支持上千个	一般不多于几十个

Docker 核心组件

Docker 中有三大核心组件：

镜像

镜像是一个只读的静态模版，它保存了容器需要的环境和应用的执行代码，可以将镜像看成是容器的代码，当代码运行起来之后，就成了容器，镜像和容器的关系也类似于程序和进程的关系。

容器

容器是一个运行时环境，是镜像的一个运行状态，它是镜像执行的动态表现。

库

库是一个特定的用户存储镜像的目录，一个用户可以建立多个库来保存自己的镜像。

通过这样一个介绍，相信大家对于 Docker 会有一个更加全面的认识。

Docker 安装

Docker 支持主流操作系统，我们可以在 Windows、MacOS 以及 Linux 中安装 Docker。这里分别给大家来介绍下。

Windows 安装

Windows10 下 Docker 的安装还是比较容易的，以前老版本的 Windows 安装并不太容易，考虑到现在大家基本上都上 Windows10 了，我就来说说 Windows10 下 Docker 的安装，这个安装步骤很简单，首先下载 Docker：

Windows 版 Docker 下载地址

下载完成后，就是一普通的 Windows 可执行程序，双击开始安装，一路 Next 就安装好了，这个过程没有什么需要注意的，就按照普通应用程序来对待就行了。

MacOS 安装

MacOS 下的安装也比较容易，当然首先第一步还是下载软件：

Mac 版 Docker 下载地址

下载后是一个 .dmg 文件，双击安装即可，安装完成之后，会有一个 Docker 图标，需要使用时双击启动即可。

Linux 安装（推荐）

上面两种安装方式大家作为一个了解即可，大家以后在工作中，大部分还是在 Linux 下安装 Docker 并使用，这里我以 CentOS7 为例来向大家介绍 Docker 的安装。具体步骤如下：

CentOS7 中安装 Docker 其实也是非常容易的，就两个步骤，如下：

```
yum -y install docker
service docker start
```

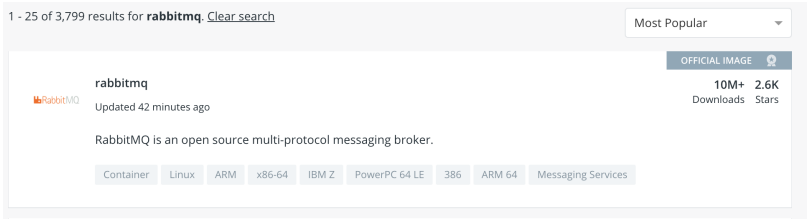
安装完成之后，启动 Docker ，执行如下命令查看 Docker 版本，如果能看到，说明就已经安装成功了：

```
sang-2:~ sang$ docker -v
Docker version 18.06.1-ce, build e68fc7a
sang-2:~ sang$
```

安装 RabbitMQ

Docker 安装成功之后，接下来我们来安装 RabbitMQ，RabbitMQ是一个实现了AMQP的开源消息中间件，使用高性能的Erlang编写。RabbitMQ具有可靠性、支持多种协议、高可用、支持消息集群以及多语言客户端等特点，在分布式系统中存储转发消息具有不错的性能表现。

如果读者不知道容器的安装命令是什么，可以直接去 Docker 镜像站搜索查找，Docker 镜像站是 https://hub.docker.com/ ，多内有网易等提供的镜像站，下载速度会快一些，地址是 https://c.163yun.com/hub ，不过网易这个镜像站必须登录之后才能访问。我这里以官方镜像站为例向读者演示 RabbitMQ 的安装，安装方式其实很容易，首先在官方镜像站上搜索 rabbitmq ，然后找到官方镜像，如下：



点进去，就能找到 RabbitMQ 的安装命令，如下:

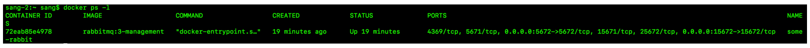
```
docker run -d --hostname my-rabbit --name some-rabbit -p 15672:15672 -p 5672:5672 rabbitmq:3-management
```

命令解释：

1. docker run 表示运行一个容器，如果这个容器本地镜像已经存在，则直接启动，如果这个容器本地景象不存在，则会自动先去 docker 仓库下载这个容器；
2. -d 表示让容器在后台运行；
3. –hostname 表示主机名；
4. –name 表示容器的名字；
5. -p 表示端口映射，因为我们将在宿主机中访问容器，宿主机中访问容器需要端口映射，这里配置了两个端口映射，分别是 15672 和 5672，其中 15672 是管理页面访问的端口，5672 则是消息通信端口；
6. 最后的 rabbitmq:3-management 则表示下载的镜像的名字。

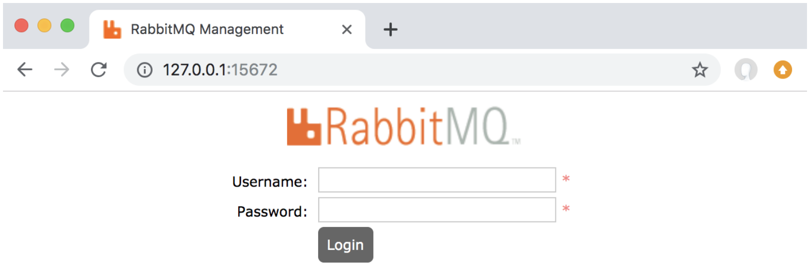
好了，当在命令行执行完如上命令，如果是第一次执行，我们会看到 Docker 会自动去镜像站下载相关镜像，如下：

执行完成之后，我们在命令行输入 `docker ps -l` 表示查看最近创建的容器，执行结果如下：



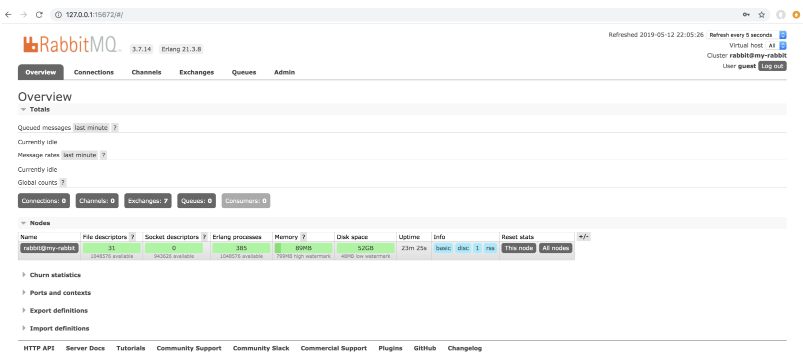
这就表示 RabbitMQ 已经安装成功了。

接下来，我们在浏览器中访问 RabbitMQ 的管理界面，如下：



我这里因为是直接在 Mac 安装的，所以访问的 IP 地址就是本机地址。如果是在 Linux 中安装 Docker 的话，那么访问地址就是 Linux 的 IP 地址，访问端口则是我们安装 RabbitMQ 时配置的默认端口。这里的默认用户名/密码都是 guest 。

登录成功的页面如下：



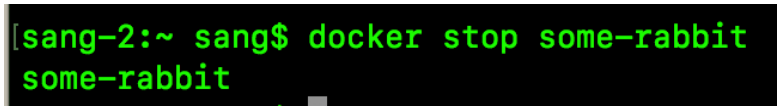
不知道读者中有没有人在 Linux 上直接安装过 RabbitMQ ？ 如果有的话，就能体会出使用 Docker 安装 RabbitMQ 是多么的清爽。

使用如下命令可以关闭 RabbitMQ：

```
docker stop some-rabbit
```

some-rabbit 是指容器的 name 。

执行结果如下：



表示容器已经成功关闭。

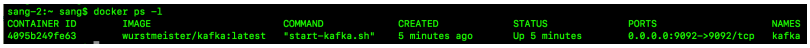
安装 Kafka

那么我们趁热打铁，顺便再来安装一下 Kafka。Kafka 我们一般采用一个第三方的镜像，Kafka 安装命令如下：

```
docker run -d --name zookeeper -p 2181 -t wurstmeister/zookeeper
docker run -d --name kafka --publish 9092:9092 --link zookeeper --env KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181 --env KAFKA_ADVERTISE
D_HOST_NAME=127.0.0.1 --env KAFKA_ADVERTISED_PORT=9092 wurstmeister/kafka:latest
```

这里的安装分为两步，首先是安装 zookeeper ，然后才是 Kafka，在 Kafka 启动时，使用容器连接的方式来使用 zookeeper 提供的服务。

执行完之后，我们的 Kafka 就算安装成功了，在命令行执行 `docker ps -l` 来查看刚刚创建的容器：



可以看到 Kafka 容器目前的状态是 Up，即正在运行。

接下来执行如下命令进入 Kafka 容器中：

```
docker exec -it kafka /bin/bash
```

执行结果如下：

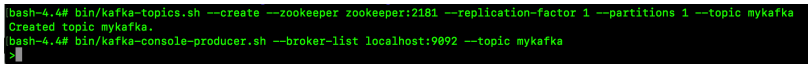
再执行如下命令，进入 Kafka 的安装目录：

```
cd /opt/kafka_2.12-2.2.0/
```

然后我们来创建一个主题和一个消息生产者：

```
bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic mykafka
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic mykafka
```

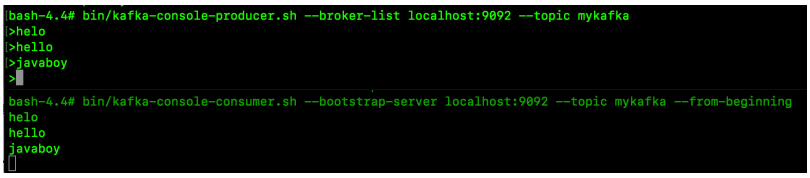
执行结果如下：



接下来打开一个新的窗口,并且进入到 Kafka 容器中，再创建一个消息消费者，命令如下：

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic mykafka --from-beginning
```

此时在消息生产者控制台输入消息，消息消费者就可以收到消息了，如下图：



小结

本文主要带领大家简单领略下 Docker 的魅力，同时安装好我们后文需要的中间件环境，使用 Docker 来搭建环境，确实能让我们省不少事情，使我们专注于业务的开发。限于篇幅，Docker 没有办法向大家介绍更多，有兴趣研究 Docker 的同学可以参考这篇文章 Docker 教程合集。

参考资料：

[1] 曾金龙，肖新华，刘清.Docker开发实践[M].北京：人民邮电出版社，2015.

24 Spring Cloud Config 服务化、动态刷新、重试

26 Spring Cloud Bus 整合 RabbitMQ 与 Kafka