转载：http://segmentfault.com/a/1190000000366923

几个月以前，红帽（Red Hat）宣布了在 Docker 技术上和 dotCloud 建立合作关系。在那时候，我并没有时间去学习关于 Docker 的知识，所以在今天，趁着这个 30 天的挑战，我决定去学习一下 Docker 究竟是怎样的。这篇博文并不是说以后怎么在 OpenShift 上用 Docker 的。请阅读由 Mike McGrath 撰写的 "关于 OpenShift 和 Docker 的技术思考"。也可以看看这个 Stackoverflow 的问题，了解一下 Docker 和 OpenShift 的差别。

什么是 Docker？

Docker 提供了一个可以运行你的应用程序的封套(envelope)，或者说容器。它原本是 dotCloud 启动的一个业余项目，并在前些时候开源了。它吸引了大量的关注和讨论，导致 dotCloud 把它重命名到 Docker Inc。它最初是用 Go 语言编写的，它就相当于是加在 LXC（LinuX Containers，linux 容器）上的管道，允许开发者在更高层次的概念上工作。

Docker 扩展了 Linux 容器（Linux Containers），或着说 LXC，通过一个高层次的 API 为进程单独提供了一个轻量级的虚拟环境。Docker 利用了 LXC， cgroups 和 Linux 自己的内核。和传统的虚拟机不同的是，一个 Docker 容器并不包含一个单独的操作系统，而是基于已有的基础设施中操作系统提供的功能来运行的。这里有一个 Stackoverflow 的答案，里面非常详细清晰地描述了所有 Docker 不同于纯粹的 LXC 的功能特性

Docker 会像一个可移植的容器引擎那样工作。它把应用程序及所有程序的依赖环境打包到一个虚拟容器中，这个虚拟容器可以运行在任何一种 Linux 服务器上。这大大地提高了程序运行的灵活性和可移植性，无论需不需要许可、是在公共云还是私密云、是不是裸机环境等等。

Docker 由下面这些组成：
1. Docker 服务器守护程序（server daemon），用于管理所有的容器。
2. Docker 命令行客户端，用于控制服务器守护程序。
3. Docker 镜像：查找和浏览 docker 容器镜像。它也访问这里得到：
https://index.docker.io/

我为什么要关心这些？

Docker 之所以有用，是因为把代码从一个机器迁移到另一个机器经常是困难的。它尝试去使得软件迁移的过程变得更加可信和自动化。Docker 容器可以移植到所有支持运行 Docker 的操作系统上。

可以看这篇文章了解更多：how the Fedora Project is embracing Docker

但是我已经在使用虚拟机（VMs）了

到现在为止，要把程序可靠地移植的唯一选择是虚拟机（Virtual Machines，VMs）。虚拟机现在已经很常见了，但虚拟机是非常低级，它提供的是完整的操作系统环境。虚拟机的问题是，迁移的时候太大了。它们包含了大量类似硬件驱动、虚拟处理器、网络接口等等并不需要的信息。 虚拟机也需要比较长时间的启动，同时也会消耗大量的内存、CPU 资源。

Docker 相比起来就非常轻量级了。运行起来就和一个常规程序差不多。这个容器不仅仅运行快，创建一个镜像和制作文件系统快照也很快。它可以在 EC2, RackSpace VMs 那样的虚拟环境中运行。事实上，在 Mac 和 Windows 系统上使用 Docker 的更好方式是使用 Vagrant。Docker 的初衷其实是发挥类似 VM 的作用，但它启动得更快和需要更少的资源。

它就像 Vagrant 一样吗？

我遇到的一个疑问是，我应该用 Vagrant 还是 Docker 去为我的下一个项目创建沙箱环境？答案再一次是一样的。

Docker 比起 Vagrant 来说，运行起来会更加省资源。Vagrant 提供的环境其实是基于 Virtual Box 提供的虚拟机。可以阅读 Stackoverflow 的这个回答了解更多。

噢，不是！另一个应用程序打包系统

当第一次读到 Docker 打包应用程序时，我困惑了。我们为什么需要再多一个应用打包系统（packaging system）？我早已经把我的 Java 程序打包成 JAR 或 WAR 了。在花了些时间阅读了关于 Docker 的资料后，我明白了 Docker 应用包（application package）的含义。Docker 就是虚拟机和你的像 WAR 或 JAR 那样的应用包之间的桥梁。一方面来说，虚拟机是非常重量级的（耗资源），因为移植时要附带些不需要的东西。另一方面来说，应用代码包（the application code packages）是非常的轻量的，并没有附带足够可靠地运行起来的信息。Docker 很好地平衡了这两方面。

在 Docker 中，应用程序包(application package)意味着一个包含了应用程序代码和所需部署环境的包。例如，在 Java 中我们一般把我们的 Web 应用程序打包在一个 WAR 文件中。这个 WAR 文件是一个非常简约的软件包，它仅仅包含了应用程序的代码。但应用程序需要特定部署的环境去高效地运行起来。有时候部署的环境和开发时的环境是不同的。例如开发者使用 Java 7 开发程序，但部署时的环境是在 OpenJDK Java 6 中；又或

者是在 Mac 上开发的，但在 RHEL 上部署。情况也有可能是：有一些系统库（system libraries）在开发环境和模拟环境(staging environment)中，在不同的应用程序上有不同的效果。Docker 通过不仅仅打包应用程序，也打包应用程序的依赖环境来解决这个问题。

开始使用 Docker

在 Fedora 机器上使用这篇博文中的指令安装 Docker

```
$ vagrant up

$ vagrant ssh
```

然后安装 Docker Fedora 镜像：

```
$ sudo docker pull mattdm/fedora
```

上面的命令会从 https://index.docker.io/ 上下载 Docker Fedora 镜像。
安装了 Docker Fedora 镜像后，我们可以使用下面命令列出所有的镜像：

```
$ sudo docker images



REPOSITORY                    TAG                 IMAGE ID            CR
EATED            SIZE

shekhargulati/node_image_007   latest              e12b3054d981
 50 minutes ago      470.3 MB (virtual 601.8 MB)



mattdm/fedora                         12.04               8dbd9e392a96
      7 months ago        131.5 MB (virtual 131.5 MB)
```

上面列表中第一个镜像就是我以前创建的。它打包了 NodeJS 及 Express Fremework。第二个镜像就是存储的 Docker Fedora 镜像了。

现在，我们在 Docker 容器内运行一个脚本：

```
$ sudo docker run -t -i -p 3000 mattdm/fedora /bin/bash
```

在运行完上面的命令后，我们就在 Docker 的容器里面了。我们可以通过 `ls` 命令列出所有的命令。

现在我们创建下面的目录结构 `/home/shekhar/dev`：

```
$ mkdir -p home/shekhar/dev

$ cd home/shekhar/dev
```

现在，我会安装 NodeJS。运行下面的命令去在 Fedora Docker 镜像上安装 Node：

```
$ sudo yum install npm
```

接着，我们安装 Express 框架：

```
$ npm install express -g
```

Express 框架安装后，我们创建一个新的 Express 程序，然后运行它：

```
$ express myapp

$ cd myapp

$ npm install

$ node app.js
```

上面会在 `3000` 端口启动 NodeJS Express 程序。

现在打开另一个命令行标签，列出所有的 Docker 进程：

```
$ sudo docker ps

CONTAINER ID        IMAGE                               COMMAND
    CREATED             STATUS              PORTS                     NA
MES

4a5715a915e5        mattdm/fedora   /bin/bash           5 minutes ago
     Up 5 minutes        0.0.0.0:49157->3000/tcp   red_duck
```

你会注意到，`3000` 端口和本机上的 `49157` 绑定了。你可以通过下面所示的 `curl` 命令测试 Express 应用：

```
$ curl 0.0.0.0:49157

<!DOCTYPE html><html><head><title>Express</title><link rel="styleshe
et" href="/stylesheets/style.css"></head><body><h1>Express</h1><p>We
lcome to Express</p></body></html>
```

现在 commit 镜像，然后 push 到 Docker 镜像注册表（registry）。在你做这步之前，你必须通过 https://index.docker.io/account/signup/ 去注册一个 Docker 注册表。

```
$ sudo docker commit 4a5715a915e5 shekhargulati/node_image_007

$ sudo docker push shekhargulati/node_image_007
```

请使用你自己的用户名和镜像名。

所以，我的第一个镜像已经上传到 Docker 注册表上面了： https://index.docker.io/u/shekhargulati/node_image_007/

你可以使用 `pull` 命令下载这个镜像：

```
$ docker pull shekhargulati/node_image_007
```