Introduction to Docker

Shuo Zhao

Docker is an technology to help us package, ship and run our applications with maximal ease and minimal overhead in different environments. It achieves this purpose by utilizing features of the Linux kernel to run some processes in isolated environment and bundling all of its dependencies into layered lightweight image files. Its popularity mainly stems from the high performance of the native execution of the machine instructions, and from the small size of the image files.

1. Dependency issue.

The deployment of an application could be really complicated process, during which dependency issue frequently arise and require much effort to get fixed. Dependency issue happens when people try to run their applications in a different environments, for example, the development laptops of collaborating developers, production servers, stage servers etc. These machines might run different operating systems, have different user permissions and use different versions of libraries, all of which might cause problems during this shipment [1].

To solve this dependency issue, currently there are two main tools: container-based technologies, like Docker, and virtual machine. Virtual machine could let us run applications in completely isolated environment, where developers can install the exact versions of all the dependences. But it also has its own serious drawbacks. A virtual machine is significantly less efficient than a real machine due to its huge overhead of interpreting and dispatching the machine instructions to be executed in the guest machine to the host machine. The virtual machine is also extremely large since the

file system of an entire machine is included and need to be shipped around with our applications. This large size could make its building and shipping very slow. And it is apparently very inefficient to ship an entire machine snapshot of several gigabytes for the purpose of deploying an app of size of only maybe a few of megabytes.

2. Architecture, workflow and advantages of docker

2.1 Docker's architecture

Docker is an open source, container-based project. Docker could package applications in image files that contain everything that is needed to run: system libraries, code, system tools, runtime. This guarantees that the applications will run portably in the same way, no matter what environment it is running in. While capable of achieving the main purpose of virtual machines, one big difference between virtual machines and Docker containers is that containers do not has an entirely isolated guest operating system integrated into it. All containers shares the docker engine and host operating system kernel. It just appears to each of the containers that they are in isolation [2].

2.2 Why is docker better than VM for application deployment?

By using virtual machine, we have a snapshot of an entire machine as a part of our application. Actually in a lot of different virtual machines, most of the files in the snapshots actually duplicate. In contrast, in Docker, the machine image are not a single opaque entity. Images can be layered together. So we can easily base our image file on an existing image file that is commonly used, then we do not have duplicate the files in the base, and the shipped image file can be very lightweight containing only what is added relative to this base image. For example, if we have a Java application of 10 megabyte size. A virtual machine might contain more than 10 gigabytes size. But when we are using docker, we can base the image

on a popular Docker image for Java applications of less than one gigabyte, which in turn is based on a stripped image of a bare minimum Ubuntu system of size of maybe 9 gigabytes. In this way, we can normally only ship the image of 10 megabytes for our app, the images for the base system and the Java environment can be shared among all containers in the environment to be deployed.

Another advantage Docker enjoys is the efficiency. By using virtual machine, all machine instructions of our virtual machines are actually given to the virtual machine supervisor rather than being executed naitively on the CPU. The dispatch of these instruction to the host machine by the virtual machine supervisor could incur significant overhead. Docker takes a different approach. In docker, the process just appears to be in isolation, all CPU instructions are native and all kernel system calls are actually given directly to the kernel (only kernel) on the host machine. So the overhead of virtual machine could be avoided and hence operations are fast.

2.3 Docker component

To run docker, we need a docker daemon, which is the docker engine and does all the heavy lifting of building and running containers, to run on the host operating system. We also need a docker user interface (normally the command-line interface, although GUI exists) to talk with docker engine. Then we can spin up many well-isolated docker containers inside docker engine. For a linux laptop/server, that’s all we need. Since Docker is based on technologies unique to the Linux kernel, for all other operating systems, like Windows or Mac OS X, we all also need an extra layer of a virtual machine, which runs only a bare minimal Linux system boot2docker, whose only purpose is to run the Docker engine. Although this brings in the overhead and cost of virtual machine to some extent, when we deploy our apps to productive Linux servers, Docker will just run directly.

2.4 Workflow of docker.

To work with Docker, we need to start by images. Images are kind of a template description of the file system, which can be used to instantiate its instance containers. One good thing about images is that it is a layered structure. When we add our own settings and applications, we add one extra layer onto it. And then we can make deployment to other servers/laptops already having the image file that our image is based, the server/laptops do not have to download the whole image again. So only the image for the app itself actually needs to be shipped. In this way, the deployment will be very lightweight and fast.

Once we have the images ready, we could spin up containers as many as we want. Since container is very lightweight and well isolated, we could spin up many containers simultaneously. And this scaling up or tearing down applications and services could happen in near real time. Take our social network as an example, we could dynamically change number of our running containers depending on the workload and use nginx to relocate requests to our containers. In this way, we could have multiple parallel applications running at the same time. What's more, it is very dense and economic.

And once we made some changes to a container, we could commit these changes back to the image and save image on Registry publicly or privately. Registry stores a huge collection of existing images where developers could upload and download images. And a lot of professionally-manufactured and carefully-tested image files can be found for common development tasks. They can be used as the base image of our apps [3,4].

3. Summary

Due to its lightweightness, portability, and efficiency, for the purpose of deploying our applications, Docker would surely be a better choice than

virtual machine. Virtual machine could not be replaced for a lot of other purposes, like operating system kernel development, but only for the purpose of deployment of applications, Docker would greatly simplify and accelerate this process of shipping, testing and deploying applications. With docker, developers could separate their applications from infrastructure and treat infrastructure as executioner of containers, which is a great separation of concerns, and enjoy a faster and easier delivery cycles of applications.

4. Reference

[1] https://en.wikipedia.org/wiki/Dependency_hell
[2] https://en.wikipedia.org/wiki/Docker_(software)
[3] https://docs.docker.com/
[4] https://github.com/docker/docker