# DOCKER 101: AN INTRODUCTION

*Part of our series on Docker. Check out How to Introduce Docker Containers in Enterprise, Docker Production Deployment Security Considerations, and Top 4 Docker Management Tips*

I've been working with Vagrant over the past few months, and recently started investigating Docker hands-on in my lab. Both container technologies are impressive and useful and are changing the face of application delivery. But…well, we still have some distance to travel.

For this article I consider both Vagrant and Docker to be similar technologies, although they operate at different levels of virtualization. These technologies, based on pre-packaging large units of pre-configured software, are excellent at reducing friction and increasing speed to market.

Think about all the non-value-add time IT professionals spend installing and configuring software. Jane installs Apache in San Francisco, while Joe does so in Chicago, and thousands of their colleagues around the world do the same every day. In the aggregate, this easily consumes billions of dollars in economic activity—all repetitive and ripe for streamlining.

**What Docker Can (and Can't) Do**

Instead of painstakingly configuring software from scratch, you can just download a container. It has the software you want (including all the dependencies) configured and ready to go—literally, after the first download, the container will run in milliseconds. (Vagrant is slower to start up, but still much quicker than installing from scratch.) Sure, a Docker container might be 150 MB, and a Vagrant image three times that, but with today's infrastructure, who cares? Downloading and storing a proven

package is far cheaper than wrestling with configuration.

But with this reduced friction comes risk. When you are painstakingly installing software on your own server, you install what you need while also developing (one hopes) some understanding of what you've done. Not so with containers. A container is built by someone else using standard installation techniques, but who knows what they put on there? Is all the licensing being handled correctly? What about security vulnerabilities?

Take the Oracle Java Development Kit, for example. If you download this (currently free) software from Oracle, you have to indicate your acceptance of Oracle licensing terms; even if you are doing this in an automated fashion, your script must indicate that you have accepted Oracle's licensing.

However, I was recently able to download a Docker image with the Oracle JDK, with no Oracle licensing apparent—not on the page documenting the image, nor as an acceptance prompt.

**Nine-Figure True-Ups? Ouch.**

Software licensing is serious business. Not everything is free and open source, and in some cases previously free software changes its terms. The license audit activities of major software houses generate billions of dollars in revenue, at very attractive margins. Tales of nine-figure true-ups exist.

Similarly, with security, some containers are "official," meaning that the container is supported by the publisher of the primary software on it. However, many published containers (such as we see on the Docker Hub) are published by individuals who found it useful to build them at a point in time. No one takes responsibility for maintenance, e.g. applying patches to remedy vulnerabilities.

**How Do We Protect Ourselves?**

There is no magic to container technology. I can log into a container, which is nothing more than a Linux environment, and call on the usual tools to figure out what is there. I can patch things myself. In theory, software discovery should work, but we still have the questions of throughput and governance. With less friction, how can we protect the enterprise from security and license risks? New software is appearing and disappearing at an accelerating rate, and it's at least possible, if not likely, that this increases risk.

Some may say that "all you have to do is look at the Dockerfile," or similar artifacts. However, there is (as far as I can see) nothing enforcing the use of any such explicit build conventions. Will an Oracle licensing auditor, motivated by the prospect of a $10 million true-up, be satisfied with such evidence?

A partial answer may be found in the package repository approach, which could be extended to approved images. Although this replaces technical friction with process friction and may not be well received by Agile teams within the enterprise. Open source discovery and analysis services are another tool in the toolbox.

At this time, Docker and Vagrant are bigger in the developer community than the enterprise world, and Docker's official position is that it is not ready for production. However, there are already rumors of its use beyond development. I suspect there will be much further discussion of the security and licensing issues raised by containerization.

For more information about provisioning Docker and other dynamic environments, check out this cloud management blog.