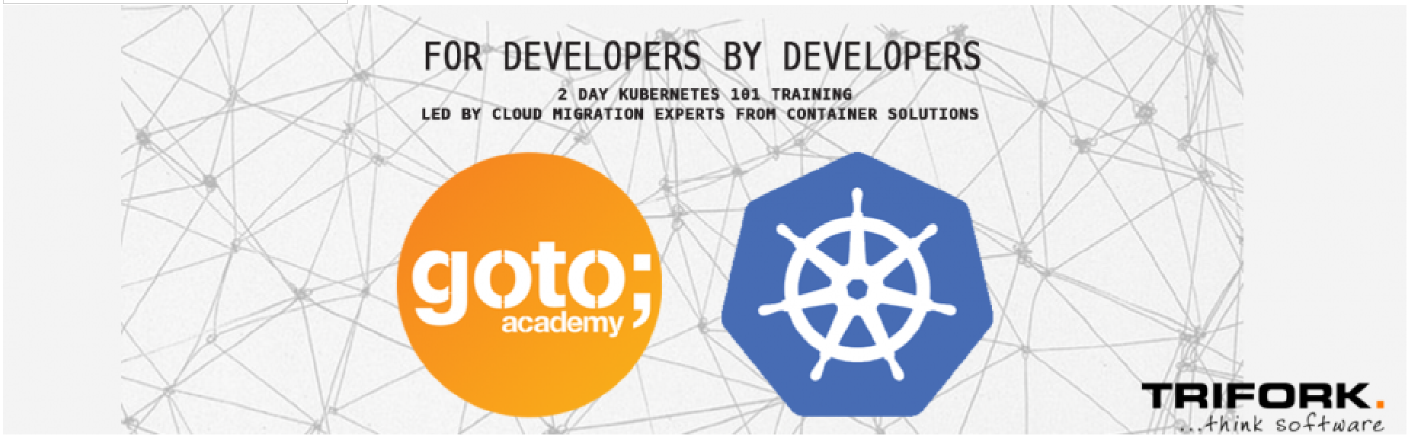
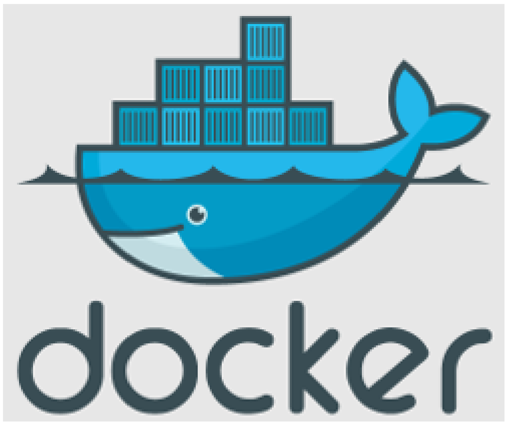
Trifork Blog


FOR DEVELOPERS BY DEVELOPERS
2 DAY KUBERNETES 101 TRAINING
LED BY CLOUD MIGRATION EXPERTS FROM CONTAINER SOLUTIONS
goto; academy
TRIFORK.
...think software

Using Docker to efficiently create multiple tomcat instances

August 15th, 2013 by Quinten Krijger
|



In my previous blog article I gave a short introduction into Docker ("an open-source engine that automates the deployment of any application as a lightweight, portable, self-sufficient container that will run virtually anywhere"). In this article we'll check out how to create an image for Tomcat 7 and the Java 7 JDK as dependency.

So, let's go ahead and do some 'coding'. First, you need to install docker. Instructions can be found here. I already mentioned you need a Linux system with a modern kernel, so if you happen to be a Mac or Windows user, there are instructions on linked pages on how to use Vagrant to easily setup a virtual machine (VM) to use. For now we'll work locally, but once you start installing servers you might find the Chef project to install docker useful as well.

As a first step after installation, let's pick the first example from the Docker getting started page and create an Ubuntu 12.04 container, with completely separated processes, its own file system and its own network interface (but with network connection via the host), and have it print "hello world". Do this by running

```
docker run ubuntu /bin/echo hello world
```

Cool huh? You probably just ran something on a different OS than that of your own machine or (in case you're on Windows/Mac) the VM in which Docker is running! In this command `ubuntu` defines the image (found automatically as it is one of the standard images supplied by Docker). The `run` command creates an instance of the image (a container), feeding it `/bin/echo hello world` as the command to execute.

The behavior that you just tested is exactly the behavior that I had, or some other fellow doing this blog; no more "but it works on my machine…". It probably took some time due to Ubuntu 12.04 being downloaded. However, that is on the first run only; after that we're talking milliseconds. You can run

```
docker images
```

to see that the Ubuntu 12.04 has been downloaded to your system, aliased as `precise` and `latest`. So, you could have used `docker run ubuntu:precise /bin/echo hello world` (notice the use of the tag) as well. Run

```
docker ps -a -s
```

to see the containers. `-a` makes sure you see already closed containers (useful, since echoing hello world doesn't take that long), and `-s` gives you the container file size as well. Containers as simple as these are some kBs big only (even when running multiple concurrently), because they both use the same image and share those resources. That is a major advantage over VMs, but I already mentioned that in the introductory blog.

Step 2. Create an image with the Java JDK 7. I chose to do the Oracle version with the added complexity of scripting the automatic license acceptance. Actually, we could go about this two ways.

1. Run the Ubuntu image interactively (using `docker run ubuntu ps -i -t /bin/bash`) and do whatever we want in the shell. When done, we could exit the shell, thereby closing the container, and commit the result into a new image using `docker commit quintenk/jdk7-oracle`. This method has the disadvantage of not being transparant or reproduceable.
2. Use Dockers Dockerfile Builder: we create a simple script file called "Dockerfile" and have Docker run and commit each command to a new image, allowing us to give a human readable name to the final result. Note that while we end up with 8 images (which you can see later by using the `-a` option with the `docker images` command), Docker only saves a diff for each file, so most images use a few kBs of disk space. Now all build steps are completely transparant and reproduceable. You can also find the following Dockerfiles on my github repository.

```
FROM ubuntu:precise
MAINTAINER Quinten Krijger < qkrijger [at] gmail {dot} com>
# make sure the package repository is up to date
RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" > /etc/apt/sources.list

RUN apt-get update && apt-get -y install python-software-properties
RUN add-apt-repository ppa:webupd8team/java
RUN apt-get update && apt-get -y upgrade

# automatically accept oracle license
RUN echo oracle-java7-installer shared/accepted-oracle-license-v1-1 select true | /usr/bin/debconf-set-selections
# and install java 7 oracle jdk
RUN apt-get -y install oracle-java7-installer && apt-get clean
RUN update-alternatives --display java
RUN echo "JAVA_HOME=/usr/lib/jvm/java-7-oracle" >> /etc/environment
```

The `FROM` statement is the mandatory first line to let Docker know what to build this image from. `MAINTAINER` is of course optional. The `RUN` commands are executed when building the image: for each line, a container is initialized and the command executed after which the result is committed. To actually build the image, you need this in a file called "Dockerfile" (e.g. by cloning my repository) and go to the file location. Then run

```
docker build -t quintenk/jdk7-oracle .
```

Notice that Ubuntu was not downloaded again. As an experiment to see how developing a Dockerfile goes in practice you could alter the Dockerfile and change one of the latter lines. If you then build again (maybe without the `-t quintenk/jdk7-oracle` option) you will find that the caching mechanism is worth gold. Try stuff like that when testing Ansible installation scripts on a local VM and you'll see the difference!

Step 3. Now that we have an image with Java installed, we can create other images based on it. We'll try Tomcat 7, but if you make other images as well on the same base, they will share the JDK image resources! For Tomcat I created the following Dockerfile (also in the same git repository):

```
FROM quintenk/jdk7-oracle
MAINTAINER Quinten Krijger "qkrijger@gmail.com"

RUN apt-get -y install tomcat7
RUN echo "JAVA_HOME=/usr/lib/jvm/java-7-oracle" >> /etc/default/tomcat7
EXPOSE 8080
CMD service tomcat7 start && tail -f /var/lib/tomcat7/logs/catalina.out
```

(edit: note the comment by Anton on using 'tail -F' rather than '-f') You'll notice the `EXPORT` command in here, which exposes Tomcats port to the container. The `CMD` is the command that will be executed when a container is started. So, when we build the image and run it as a daemon using

```
docker run -d quintenk/tomcat7
```

it starts Tomcat as a service and then tails the log file. The last step is needed because you need a running foreground process in order for the container to keep running. If you omit it, the container will start Tomcat and shut down immediately after. Now run `docker ps` to see

```
ID       d02ed25688be
IMAGE    quintenk/tomcat7:latest
COMMAND  /bin/sh -c service t
PORTS    49153 -> 8080
```

So, 8080 on the container is exposed to the host port 49153, and you can see Tomcat running at http://localhost:49153. Run another container, and you'll probably get it under port 49154. This way, we can now start lots of Tomcat containers, all completely isolated and show the results in the same browser.

As a next step, and an actual use case, I would install different versions of the same web application, create images from that and run them at the same time. You can then demonstrate to your customer what you have produced in the last few weeks :). However, I've blogged enough for today. If you do give it a spin, you don't have to expose 8080 again, but I believe you do need to give a new `CMD`. If you don't specify a `CMD` you could run the container in interactive mode with shell, and start tomcat from the command line – maybe less clean, but useful to understand that it is an option.

Note that if you would just like the images and aren't interested in building them yourself, you can download them from the Docker index, where I published them. To do this, simply run

```
docker pull quintenk/tomcat7
```

From there, you could install you own web application using the techniques described in this article.

Have fun!

Tags: configuration, Deployment, docker, Java, jdk, linux, LXC, tomcat Posted in: DevOps

9 Responses

1. August 16, 2013 at 21:15 by Murali Allada
   |

   Thanks for sharing Quinten!

   Have you been able to restart tomcat on a running container from the host machine? I'm trying to update an environment variable in the container, and I need to restart tomcat for my app to pick it up.

   - August 19, 2013 at 16:36 by Quinten Krijger
     |

     @Murali Allada
     Well, the described method really would be for an application that is ready to roll, I guess. If you need to set a variable, you might want to consider creating a new image, based on tomcat7, but with you application and altered settings.

An alternative would be to start tomcat7 using `docker run -i -t quintenk/tomcat7 /bin/bash`. You can then manually start an restart tomcat using `service tomcat7 (re)start` and do anything you would like. You can also commit your changes to a new images, but note that your image history will the contain the step "/bin/bash", which is less than insightful into what you actually did (you'll hate yourself for doing it when you pick up the project later :))

2. August 16, 2013 at 21:24 by Anton
   |

   Hi, just wanted to let you know that running tail -f will not follow the file after it rotates, however using tail -F does. If tomcat rotates the file once every night your instance will die at that time. However running tail -F will follow the file even if it's rotated or renamed.

   Cheers!

3. August 19, 2013 at 16:41 by Quinten Krijger
   |

   @Anton, thanks! So, if anyone bases an image on this one and uses the tail trick, -F is the better option. I will not update the image, since it is in the CMD which will be overridden in a depending image anyway. I'll add a note in the blog though.

4. October 25, 2013 at 08:40 by nirmata | Netflix OSS, meet Docker! - nirmata
   |

   […] running in Docker (v0.6.3) containers. So, first we created the base image for all our services by installing jdk 7 and tomcat 7. For test environment, we wanted to make sure that the base container image can be use for any […]

5. January 15, 2014 at 18:51 by Gary
   |

   Curious. if you were fronting tomcat with Apache would you host Apache in the container with tomcat and therefore have many Apache's or would you host Apache separately and use virtual hosts to connect to the individual tomcat containers.. and how about a mySQL instance.. hmmm

6. January 22, 2014 at 00:04 by Greg Stephens
   |

   @Gary I'm hosting mysql and apache in the same container although I eventually plan on hosting mysql in it's on separate container.

7. June 1, 2014 at 03:02 by David Medinets
   |

   I needed to use the -P argument. In order to get 8080 exposed on the host. "docker run -d -P medined/tomcat7" is the command that I ran.

8. November 26, 2014 at 15:11 by Larry
   |

   On the latest ubuntu docker image (image id 5506de2b643b), the tomcat init script does not return successfully, although it does start tomcat. Looks like start-stop-daemon used by the init script incorrectly returns false when Tomcat is running and makes the init script think the startup was unsuccessful.

« Ansible from the Trenches
Lessons learned how to do Scrum in a fixed price project »

-

-
- goto; conference

**- GOTO Chicago
April 24-27, 2018**

**- GOTO Amsterdam
June 18 - 20, 2018**

**- GOTO Berlin
Oct 3 - Nov 2, 2018**

**- GOTO Copenhagen
Nov 19 - 21, 2018**

About this blog

The content on this blog is developed mostly by our own developers. It contains a selection of best practices, trends and technical information on different open source technologies. Our blog is where we can share our experiences with technologies we are experimenting with, advancing with and more often than not indulging in. In true open source community style, we don't hold back and we share code snippets, screen shots and provide insight wherever we can. We particularly enjoy setting up a dialogue with our readers and often there is also a thread of question and answers in the blogs too. Our blogs are not however all about just the technical successes but also our less successful efforts too ;-)

trifork

In many of our blogs, you'll find we endeavour to take you on a journey of our thought process. This means some of our blogs can often become epics and have a whole series associated with them. We think that most of our readers appreciate reading about our experiences on new products, releases, conferences and technologies and to be honest its what makes us tick and what we thrive on most. Every now and then we also invite partners, industry leaders and others around us as guest writers or we share some interesting conversations we have had with them. Suggestions, input and feedback is always welcome, just drop us a note and we'd be happy to hear from you.