docker.docu

Michael Klöckner

February 20, 2014

Contents

1 Overview

1.1 Purpose

This paper documents the Proof of Concept Project: **Enable a Jenkins build server to publish release artifacts as Docker images to a Docker registry. Start service on a new EC2 node by fetching the artifact from the registry.**

1.2 Author

- Author: Michael Klöckner, Weberstr. 39, 60318 Frankfurt am Main,
- Email:mkl@im7.de
- Phone: +49 69 9866 1103

1.3 docker version

We installed docker version 0.8 in 2014/02.

2 docker

This chapter deals with installation issues and some basic docker commands. It is mainly take from the Docker documentation.

2.1 TODO explain docker teminology: Image, Container, Repository, Registry, Index.

2.2 Installation

2.2.1 Kernel options

Docker needs a 64-Bit Linux distribution, a recent kernel $>$ 3.8 and LXC installed. Either you use a system with the appropriate kernel installed, or you update the kernel by hand as described in kernel compilation. The kernel needs to have compiled all options concerning virtual NICs, especially BRIDGED NICs, all NAT options and all net (NF) options. Download the kernel source, untar it, change into the directory and configure it prop- erly. To compile the kernel as a debian package named **fora-kernel-3.13.3** to be installed later together with it's header follow these instructions:

```
make-kpkg clean
make-kpkg --append-to-version "-flora-kernel-3.13.3" --revision "1" \
--initrd kernel_image kernel_headers
```

The package is to be found one directory upwards and can be installed using

```
dpkg -i ../linux-headers-3.13.3-flora-kernel-3.13.3_1.2_amd64.deb \
../linux-image-3.13.3-flora-kernel-3.13.3_1.2_amd64.deb/.
```

2.2.2 Installation by hand

First add the Docker repository key to your local keychain.

```
sudo apt-key adv --keyserver keyserver.ubuntu.com \
--recv-keys 36A1D7869245C8950F966E92D8576A8BA88D21E9
```

Add the Docker repository to your apt sources list, update and install the lxc-docker package.

```
sudo sh -c "echo deb http://get.docker.io/ubuntu docker main\
> /etc/apt/sources.list.d/docker.list"
sudo apt-get update
sudo apt-get install lxc-docker
```

Now verify that the installation has worked by downloading the ubuntu image and launching a container. `sudo docker run -i -t ubuntu /bin/bash`. Type exit to exit.

2.2.3 Installation by script

Docker.io provides an installation script to be called: `curl -s https://get.docker.io/ubuntu/ | sudo sh` Now verify that the installation has worked by downloading the ubuntu image and launching a container. `sudo docker run -i -t ubuntu /bin/bash` Type exit to exit.

2.2.4 Installing on AWS

Docker.io provides an installation guide for Amazon Web Services EC2.

- Choose an image:
  - Launch the Create Instance Wizard menu on your AWS Console.
  - Click the Select button for a 64Bit Ubuntu image. For example: Ubuntu Server 12.04.3 LTS.
  - For testing you can use the default (possibly free) t1.micro instance (more info on pricing).
  - Click the Next: Configure Instance Details button at the bottom right.
- Tell CloudInit to install Docker:
  - When you're on the Configure Instance Details step, expand the Advanced Details section.
  - Under User data, select As text
  - Enter =#include https://get.docker.io = into the instance User Data. CloudInit is part of the Ubuntu image you chose; it will bootstrap Docker by running the shell script located at this URL.
- After a few more standard choices where defaults are probably ok, your AWS Ubuntu instance with Docker should be running!

If this is your first AWS instance, you may need to set up your Security Group to allow SSH. By default all incoming ports to your new instance will be blocked by the AWS Security Group, so you might just get timeouts when you try to connect. Installing with get.docker.io (as above) will create

a service named lxc- docker. It will also set up a docker group and you may want to add the ubuntu user to it so that you don't have to use sudo for every Docker command.

2.3 Play with docker

We describe a view docker commands.

2.3.1 Check your Docker installation.

```
# Check that you have a working install
docker info
```

2.3.2 Download a pre-built image

```
# Download an ubuntu image
sudo docker pull ubuntu
```

2.3.3 Run an interactive shell

```
# Run an interactive shell in the ubuntu image,
# allocate a tty, attach stdin and stdout
# To detach the tty without exiting the shell,
# use the escape sequence Ctrl-p + Ctrl-q
sudo docker run -i -t ubuntu /bin/bash
```

2.3.4 Bind to a port

The Docker client can use -H to connect to a custom port. -H accepts host and port assignment in the following format:

- tcp://[host][:port] =
- unix://path =
- host[:port] or :port =

```
# Run docker in daemon mode
sudo <path to>/docker -H 0.0.0.0:5555 -d &
# Download an ubuntu image
sudo docker -H :5555 pull ubuntu
```

2.3.5 Starting a long run

```
# Start a very useful long-running process
JOB=$(sudo docker run -d ubuntu /bin/sh -c "while true; \
do echo Hello world; sleep 1; done")
# Collect the output of the job so far
sudo docker logs $JOB
# Kill the job
sudo docker kill $JOB
```

2.3.6 Bind a service on a TCP port

```
# Bind port 4444 of this container, and tell netcat to listen on it
JOB=$(sudo docker run -d -p 4444 ubuntu:12.10 /bin/nc -l 4444)

# Which public port is NATed to my container?
PORT=$(sudo docker port $JOB 4444 | awk -F: '{ print $2 }')

# Connect to the public port
echo hello world | nc 127.0.0.1 $PORT

# Verify that the network connection worked
echo "Daemon received: $(sudo docker logs $JOB)"
```

2.3.7 Committing (saving) a container state

Save your containers state to a container image, so the state can be re-used.

When you commit your container only the differences between the image the container was created from and the current state of the container will be stored (as a diff). See which images you already have using the docker images command.

```
# Commit your container to a new named image
sudo docker commit <container_id> <some_name>

# List your containers
sudo docker images
```

2.3.8 Committing a Container to a Named Image

When you make changes to an existing image, those changes get saved to a container's file system. You can then promote that container to become an

image by making a commit. In addition to converting the container to an image, this is also your opportunity to name the image, specifically a name that includes your user name from the Central Docker Index (as you did a login above) and a meaningful name for the image.

```
# format is "sudo docker commit <container_id> <username>/<imagename>"
$ sudo docker commit $CONTAINER_ID myname/kickassapp
```

2.3.9 Pushing an image to its repository

In order to push an image to its repository you need to have committed your container to a named image (see above). Now you can commit this image to the repository designated by its name or tag.

```
# format is "docker push <username>/<repo_name>"
$ sudo docker push myname/kickassapp
```

2.3.10 Private Repositories

Right now (version 0.6), private repositories are only possible by hosting your private registry. To push or pull to a repository on your own registry, you must prefix the tag with the address of the registry's host, like this:

```
# Tag to create a repository with the full registry location.
# The location (e.g. localhost.localdomain:5000) becomes
# a permanent part of the repository name
sudo docker tag 0u812deadbeef localhost.localdomain:5000/repo_name
# Push the new repository to its home location on localhost
sudo docker push localhost.localdomain:5000/repo_name
```

Once a repository has your registry's host name as part of the tag, you can push and pull it like any other repository, but it will not be searchable (or indexed at all) in the Central Index, and there will be no user name checking performed. Your registry will function completely independently from the Central Index.

2.3.11 Export a container

To export a container to a tar file just type:

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL
mkl/debian          7.4                 11ed3d47ec89        About an hour ago   117.8 M
mkl/debian          latest              11ed3d47ec89        About an hour ago   117.8 M
mkl/debian          wheezy              11ed3d47ec89        About an hour ago   117.8 M
ubuntu              13.10               9f676bd305a4        2 weeks ago         182.1 M
ubuntu              saucy               9f676bd305a4        2 weeks ago         182.1 M

$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
ac3a595c294c        mkl/debian:7.4      /bin/bash           58 minutes ago      Exit 1
f7528d270208        mkl/debian:7.4      echo success        About an hour ago   Exit 0
6a569d77e974        ubuntu:12.04        /bin/bash           16 hours ago        Exit 0

$ docker export ac3a595c294c  > exampleimage.tar
```

2.3.12 Import a container

At this time, the URL must start with http and point to a single file archive (.tar, .tar.gz, .tgz, .bzip, .tar.xz, or .txz) containing a root filesystem. If you would like to import from a local directory or archive, you can use the - parameter to take the data from stdin. To import from a remote url type:

```
$ sudo docker import http://example.com/exampleimage.tar
```

To import from a local file type:

```
$ cat exampleimage.tar | sudo docker import - exampleimagelocal:new
```

Note the sudo in this example – you must preserve the ownership of the files (especially root ownership) during the archiving with tar. If you are not root (or the sudo command) when you tar, then the ownerships might not get preserved.

2.3.13 Authentication file

The authentication is stored in a json file, .dockercfg located in your home directory. It supports multiple registry urls.

docker login will create the "https://index.docker.io/v1/" key.

docker login https://my-registry.com will create the "https://my-registry.com" key.

For example:

```
{
     "https://index.docker.io/v1/": {
             "auth": "xXxXxXxXxXx=",
             "email": "email@example.com"
     },
     "https://my-registry.com": {
             "auth": "XxXxXxXxXxX=",
             "email": "email@my-registry.com"
     }
}
```

The auth field represents base64(<username>:<password>)

2.4 Build your own base image

Docker.io provides a way to create a base image. The base image heavily depends on the distribution, the host is running. The example script mkimage-debootstrap.sh creates a debian base image.

2.4.1 Download the script

```
$ wget https://raw.github.com/dotcloud/docker/master/contrib/mkimage-debootstrap.sh
$ chmod +x mkimage-debootstrap.sh
```

This downloads the build-script for a debian docker base image.

2.4.2 Build the base image

```
$ ./mkimage-debootstrap.sh flora/debian wheezy
$ docker images -a
```

This creates a new docker base image for debain wheezy and puts it into ropsitory *flora/debian*, where *flora* is the username and *debian* the repo name.

2.5 Container and Images

As docker is under heavy developement, the file system storing docker related informatian changes rapidly. The main directory to look for docker relevant bits and bytes is *var/lib/docker*.

2.5.1 Configuration

- The daemon's config file is placed in *etc/default/docker*.
- Images, containers and their configurations are placed under *var/lib/docker*.

2.5.2 LXC configuration

Using the Linux Container package http://linuxcontainers.org/, docker configures each container by setting lxc options in *var/lib/docker/container/GUID/config.lxc*. Here GUID is the full blown container id as given by `docker ps -a -no-trunc`

2.5.3 Contianer Root File System

The corresponding root file system is stored in *var/lib/docker/devicemapper/mnt/GUID*. Here GUID is the full blown container id as given by `docker ps -a -no-trunc`

3 Installing a *Scala/Java* WebApp

As a proof of concept, we install a *Scala* WebApp with *Lift*. We need *Java* version > 6 and we use *Lift* as the framework.

3.1 Installing *Java* and *Lift*

3.1.1 The necessary debian packages

We need *jdk* at least version 6, *wegt* and *zip*:

```
$ apt-get install -y openjdk-7-jre
$ apt-get install -y openjdk-7-jdk
$ apt-get install -y wget
$ apt-get install -y zip
```

This installs Java 7 and my take a minute.

3.1.2 TODO check if we need apache packages?

3.1.3 Scala WebApp

We download and configure a sample *Scala* WebApp and generate the War-file.

```
$ cd /opt
$ wget https://github.com/Lift/Lift_26_sbt/archive/master.zip
$ unzip master.zip
$ cd lift_26_sbt-master/scala_210/lift_basic/
$ ./sbt  compile
$ ./sbt  package
```

/Lift/ web framework will download *sbt*, *Scala* and the necessary dependencies and compile the War-File **/opt/lift_26_sbt-master/scala_210/lift_basic/target/scala-2.10/lift-2.6-starter-template$_2$.10-0.0.3.war**. By typing

```
$ ./sbt
> start
```

we should be able to see the WebApp at http://localhost:8080. To exit just type `exit`. The source of this WebApp is under **/opt/lift_26_sbt-master/scala_210/lift_basic/src/main/webapp/**. To prove the concept, we will later just change *index.html*.

3.2 Installing *tomcat7*

We use *tomcat* as the **Apache Tomcat Servlet/JSP** engine to serve our *Scala* WebApp, installing it by typing:

```
$ apt-get install tomcat7
```

Tomcat serves servlets at http://localhost:8080. The debian package starts the service automatically at boot time via *etc/init.d/tomcat7* script.

3.3 Deploying the WebApp to *tomcat7*

Lift uses *sbt* to compile the project and output a WAR- or JAR-file, which we want to copy into *tomcat7*'s webapp directory **/var/lib/tomcat7/webapps/**. We recompile the package and deploy it statically into *tomcat*.

```
$ cd /opt/lift_26_sbt-master/scala_210/lift_basic/
$  ./sbt  package
$ cp  target/scala-2.10/lift-2.6-starter-template_2.10-0.0.3.war \
    /var/lib/tomcat7/webapps/lift.war
$ service tomcat7 restart
```

This copies the war-file and restarts *tomcat7*. To see the WebApp direct your browser to http://localhost:8080/lift_basic/. There is no need to restart *tomcat* manually, as the *autoDeploy* attribute is set to "true" in file **/etc/tomcat7/server.xml**. *tomcat* even unpacks war-files if attribute *unpackWARs* is set to "true".

4 Installing Jenkinsx

How to install a jenkins server

5 Configure Jenkins to publish a container into the registry

Each time the WebApp has changed in git, Jenkins builds a new container, consisting of three parts:

1. Deploying the WebApp-Files into the latest container.
2. Commit the newly build container and tag it properly.
3. Start the newly taged container.