Docker quicktip #3 - ONBUILD

06 Feb 2014

Docker 0.8 came out today, with it a slew of fantastic enhancements. Today we'll be looking at one of them: ONBUILD.

ONBUILD is a new instruction for the Dockerfile. It is for use when creating a base image and you want to defer instructions to child images. For example:

# Dockerfile

FROM busybox

ONBUILD RUN echo "You won't see me until later"

docker build -t me/no\_echo\_here .

Uploading context 2.56 kB

Uploading context

Step 0 : FROM busybox

Pulling repository busybox

769b9341d937: Download complete

511136ea3c5a: Download complete

bf747efa0e2f: Download complete

48e5f45168b9: Download complete

---&gt; 769b9341d937

Step 1 : ONBUILD RUN echo "You won't see me until later"

---&gt; Running in 6bf1e8f65f00

---&gt; f864c417cc99

Successfully built f864c417cc9

Here the ONBUILD instruction is read, not run, but stored for later use.

Here is the later use:

# Dockerfile

FROM me/no\_echo\_here

docker build -t me/echo\_here .

Uploading context 2.56 kB

Uploading context

Step 0 : FROM cpuguy83/no\_echo\_here

# Executing 1 build triggers

Step onbuild-0 : RUN echo "You won't see me until later"

---&gt; Running in ebfede7e39c8

You won't see me until later

---&gt; ca6f025712d4

---&gt; ca6f025712d4

Successfully built ca6f025712d4

The ONBUILD instruction only gets run when building the cpuguy83/echo\_here image.

ONBUILD gets run just after the FROM and before any other instructions in a child image.

You can also have multiple ONBUILD instructions.

Why would you want this? It turns out it's pretty darn awesome, and powerful. I have a demo github repo setup for this: [Docker ONBUILD Demo](https://github.com/cpuguy83/docker-onbuild_demo)

Before diving into this, I just want to say I've probably used ONBUILD a bit excessively here in order to get the point across for what ONBUILD does and what it can do, it's up to you how to use it in your projects.

# Dockerfile

FROM ubuntu:12.04

RUN apt-get update -qq &amp;&amp; apt-get install -y ca-certificates sudo curl git-core

RUN rm /bin/sh &amp;&amp; ln -s /bin/bash /bin/sh

RUN locale-gen en\_US.UTF-8

ENV LANG en\_US.UTF-8

ENV LANGUAGE en\_US.UTF-8

ENV LC\_ALL en\_US.UTF-8

RUN curl -L [https://get.rvm.io](https://get.rvm.io/) | bash -s stable

ENV PATH /usr/local/rvm/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

RUN /bin/bash -l -c rvm requirements

RUN source /usr/local/rvm/scripts/rvm &amp;&amp; rvm install ruby

RUN rvm all do gem install bundler

ONBUILD ADD . /opt/rails\_demo

ONBUILD WORKDIR /opt/rails\_demo

ONBUILD RUN rvm all do bundle install

ONBUILD CMD rvm all do bundle exec rails server

This Dockerfile is doing some initial setup of a base image. Installs Ruby and bundler. Pretty typical stuff. At the end are the ONBUILD instructions.

ONBUILD ADD . /opt/rails\_demo Tells any child image to add everything in the current directory to /opt/rails*demo. Remember, this only gets run from a child image, that is when another image uses this one as a base (or FROM). And it just so happens if you look in the repo I have a skeleton rails app in rails*demo that has it's own Dockerfile in it, we'll take a look at this later.

ONBUILD WORKDIR /opt/rails\_demo Tells any child image to set the working directory to /opt/rails\_demo, which is where we told ADD to put any project files

ONBUILD RUN rvm all do bundle install Tells any child image to have bundler install all gem dependencies, because we are assuming a Rails app here.

ONBUILD CMD rvm all do bundle exec rails server Tells any child image to set the CMD to start the rails server

Ok, so let's see this image build, go ahead and do this for yourself so you can see the output.

git clone git@github.com:cpuguy83/docker-onbuild\_demo.git

cd docker-onbuild\_demo

docker build -t cpuguy83/onbuild\_demo .

Step 0 : FROM ubuntu:12.04

---&gt; 9cd978db300e

Step 1 : RUN apt-get update -qq &amp;&amp; apt-get install -y ca-certificates sudo curl git-core

---&gt; Running in b32a089b7d2d

# output supressed

ldconfig deferred processing now taking place

---&gt; d3fdefaed447

Step 2 : RUN rm /bin/sh &amp;&amp; ln -s /bin/bash /bin/sh

---&gt; Running in f218cafc54d7

---&gt; 21a59f8613e1

Step 3 : RUN locale-gen en\_US.UTF-8

---&gt; Running in 0fcd7672ddd5

Generating locales...

done

Generation complete.

---&gt; aa1074531047

Step 4 : ENV LANG en\_US.UTF-8

---&gt; Running in dcf936d57f38

---&gt; b9326a787f78

Step 5 : ENV LANGUAGE en\_US.UTF-8

---&gt; Running in 2133c36335f5

---&gt; 3382c53f7f40

Step 6 : ENV LC\_ALL en\_US.UTF-8

---&gt; Running in 83f353aba4c8

---&gt; f849fc6bd0cd

Step 7 : RUN curl -L [https://get.rvm.io](https://get.rvm.io/) | bash -s stable

---&gt; Running in b53cc257d59c

# output supressed

---&gt; 482a9f7ac656

Step 8 : ENV PATH /usr/local/rvm/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

---&gt; Running in c4666b639c70

---&gt; b5d5c3e25730

Step 9 : RUN /bin/bash -l -c rvm requirements

---&gt; Running in 91469dbc25a6

# output supressed

Step 10 : RUN source /usr/local/rvm/scripts/rvm &amp;&amp; rvm install ruby

---&gt; Running in cb4cdfcda68f

# output supressed

Step 11 : RUN rvm all do gem install bundler

---&gt; Running in 9571104b3b65

Successfully installed bundler-1.5.3

Parsing documentation for bundler-1.5.3

Installing ri documentation for bundler-1.5.3

Done installing documentation for bundler after 3 seconds

1 gem installed

---&gt; e2ea33486d62

Step 12 : ONBUILD ADD . /opt/rails\_demo

---&gt; Running in 5bef85f266a4

---&gt; 4082e2a71c7e

Step 13 : ONBUILD WORKDIR /opt/rails\_demo

---&gt; Running in be1a06c7f9ab

---&gt; 23bec71dce21

Step 14 : ONBUILD RUN rvm all do bundle install

---&gt; Running in 991da8dc7f61

---&gt; 1547bef18de8

Step 15 : ONBUILD CMD rvm all do bundle exec rails server

---&gt; Running in c49139e13a0c

---&gt; 23c388fb84c1

Successfully built 23c388fb84c1

Now let's take a look at that Dockerfile in the rails\_demo project:

# Dockerfile

FROM cpuguy83/onbuild\_demo

WAT?? This Dockerfile is a grand total of one line. It's only one line because we setup everything in the base image. The only pre-req is that the Dockerfile is built from within the Rails project tree. When we build this image, the ONBUILD commands from cpuguy83/onbuild\_demo will be inserted just after the FROM instruction here.

*Remember, this aggressive use of ONBUILD may not be optimal for your project and is for demo purposes... not to say it's not ok :)*

So let's run this:

cd rails\_demo

docker build -t cpuguy83/rails\_demo .

Step onbuild-0 : ADD . /opt/rails\_demo

---&gt; 11c1369a8926

Step onbuild-1 : WORKDIR /opt/rails\_demo

---&gt; Running in 82def1878360

---&gt; 39f8280cdca6

Step onbuild-2 : RUN rvm all do bundle install

---&gt; Running in 514d5fc643f1

# output supressed

Step onbuild-3 : CMD rvm all do bundle exec rails server

---&gt; Running in df4a2646e4d9

---&gt; b78c1813bd44

---&gt; b78c1813bd44

Successfully built b78c1813bd44

Then we can run the rails\_demo image and have the rails server fire right up

docker run -i -t cpuguy83/rails\_demo

=&gt; Booting WEBrick

=&gt; Rails 3.2.14 application starting in development on [http://0.0.0.0:3000](http://0.0.0.0:3000/)

=&gt; Call with -d to detach

=&gt; Ctrl-C to shutdown server

[2014-02-06 11:53:20] INFO WEBrick 1.3.1

[2014-02-06 11:53:20] INFO ruby 2.1.0 (2013-12-25) [x86\_64-linux]

[2014-02-06 11:53:20] INFO WEBrick::HTTPServer#start: pid=193 port=3000

TLDR; ONBUILD... awesome. Use it to defer build instructions to images built from a base image. Use it to more easily build images from a common base but differ in some way, such as different git branches, or different projects entirely.

With great power comes great responsibility.