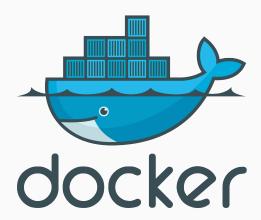
Rocker: Using R on Docker

A Hands-On Introduction

Dirk Eddelbuettel Pre-Conference Tutorial useR! 2015 June 30, 2015

Introduction



Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging tool, and Docker Hub, a cloud service for sharing applications and automating workflows. Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments. As a result, IT can ship faster and run the same app, unchanged, on laptops, data center VMs, and any cloud.

Text from http://docker.com as of May 16, 2015.

Ok, seriously, what is Docker?

- · Docker is a very lightweight abstraction
- · Using recent Linux kernel features lets us to run code in
 - · cheap (to launch) and
 - · easy (to build) units
- These units are containers
- We can share containers across OSs.
- · It may change how you build, test and deploy R (packages).

A number of recent-ish Linux kernel features:

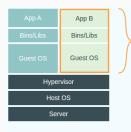
- · Process namespaces
- · Layered filesystems
- · Clever use of git etc

What is Docker NOT?

Do not think of Docker as

- · another virtual machine, or
- · intrinsically more secure

What is Docker NOT?



Virtual Machines

Each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.



Docker

The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.

Source: https://www.docker.com/whatisdocker/, retrieved 17 May 2015

Installation On Linux

- · Binaries for Linux provided by
 - · Docker itself, and
 - different Linux distributions

- · You may want to add yourself to the docker group:
 - · docker without sudo

- · Other good documentation:
 - · first steps with docker

Installation on Linux: Debian/Ubuntu

As easy as

\$ sudo apt-get install docker.io

as docker is the name of GUI package under Debian.

Docker itself also provides a repository for more current packages than the distribution.

Installation On Windows or OS X

You want

boot2docker for Windows

which installs in one step a very compact virtual machine and thin Linux layer so that Docker can run.

Installation On Windows or OS X

You want

- · boot2docker for Windows
- boot2docker for OS X

which installs in one step a very compact virtual machine and thin Linux layer so that Docker can run.

First steps: Getting images

After installation, run

docker pull r-base

(or ubuntu or debian) to pull a set of pre-built initial images.

Images

For Debian (and/or other distro are language stacks)

edd@max:~\$ docker images debian REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE debian 8.0 41b730702607 2 weeks ago 125.1 MB debian latest 41b730702607 2 weeks ago 125.1 MB debian 41b730702607 2 weeks ago 125.1 MB debian jessie 41b730702607 2 weeks ago 125.1 MB debian experimental b1a2377eafc8 3 weeks ago 125.3 MB debian rc-buggy f526decb8eaf 3 weeks ago 125.3 MB debian 7.8 b3d362b23ec1 3 weeks ago 84.98 MB edd@max:~\$

Images for R

On my server a few weeks ago:

```
edd@max:~$ docker images r-base
REPOSTTORY
           TAG
                   TMAGE, TD
                                 CREATED
                                             VIRTUAL SIZE
r-base
     3.2.0
                   0a773551ab9c
                                2 weeks ago
                                             528.5 MB
                   0a773551ab9c
r-base
          latest
                                 2 weeks ago
                                             528.5 MB
r-base 3.1.3
                   8a0dd612a7f5
                                 6 weeks ago
                                             516.7 MB
r-base 3.1.2
                   c04052b3ddd2
                                 9 weeks ago
                                             483.5 MB
edd@max:~$
```

This is from the Rocker repository (more on that below)

Key Commands

docker build

- · Standard command to build a new container
- · Assumes Dockerfile in current/given directory or on stdin
- · Resulting image can be 'tagged'

docker pull

- · Retrieves containers from the hub
- · Easiest way to get started.
- Containers can be altered / adapted / extended.
- · Note that it requires standard 64-bit Docker

docker images

- · lists the available images (containers)
- · gives snapshot and name:tag as well as uuid
- · useful to retrieve uuid for commit (see below) or rmi

docker run

- · principal way to run a container
- · useful options are
 - · --rm to remove artifacts after container terminated
 - · -ti for terminal mode and interactive mode
 - -v outer:inner to share the outer path or directory at mount point inner

 other options to run contains in the background (useful for e.g. databases or other networked / web-based services)

docker ps

- \cdot useful to get id of running processes in to commit, to stop
- · with ps -a more processes shown
- with ps -v only numeric is shown (useful for subsequent commands requiring numeric id)

docker stop containerid

- · sometimes we need to halt a running containers
- · docker ps can show the ids

docker kill \$(docker ps -q)

- \cdot kill all running instances by querying for just the numeric id
- · which is then fed into docker kill

docker rmi imageid

- · cleans up and removes images
- · (where images are frozen immutable snapshots of containers)
- · force option available
- · found helper script (see below) useful

docker rm containerid

- · cleans up and removes containers
- (which is a running, or stopped, stateful instantiation of an image)
- · sometimes only 'visible' layers removed
- · found helper script (see below) useful

docker commit processid

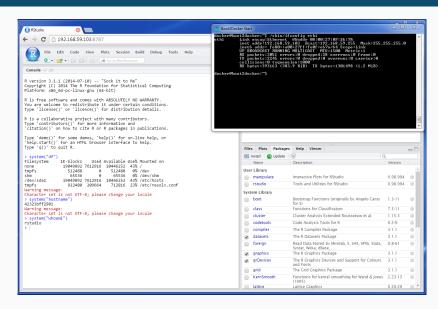
- · useful for saving state of currently running container
- · use docker ps to get process id
- \cdot use -t name:newtag to add a new tag
- · commits offer rollback as well

Helper script clean_docker.sh

```
#!/bin/bash
imgs=$(docker images | awk '/<none>/ { print $3 }')
if [ "${imgs}" != "" ]; then
   echo docker rmi ${imgs}
   docker rmi ${imgs}
else
  echo "No images to remove"
fi
procs=$(docker ps -a -q --no-trunc)
if [ "${procs}" != "" ]; then
   echo docker rm ${procs}
   docker rm ${procs}
else
   echo "No processes to purge"
fi
```

Examples

Running RStudio Server



Running RStudio Server

- As Docker excels at containerizing applications, it is particularly suitable for headless / networked app
- RStudio Server is one such application, and our Rocker project provides an image
- Via RStudio Server (plus whichever domain-specific R packages), R becomes a true cross-platform appliance
- Docker allows local use where RStudio Server would otherwise required a networked Linux server

Running RStudio Server

Basic command as easy as

docker run -d -p 8787:8787 rocker/r-studio

to launch in daemon mode and with an exported port

See earlier lightning talk for intro, and Rocker wiki for details

Running a R debugging session

- Running a current version of R-devel via Rocker r-devel container or the related Rocker drd container
- · A front-end script check.r offers one-command help
- Advanced use with Address Sanitizer (ASAN) and/or Undefined Behaviour Sanitizer (UBSAN) is described in an earlier blog post

ASAN/UBSAN Debugging

ASAN Errors

- \cdot CRAN is now using recent g++ / clang++ features for
 - ASAN ("Address Sanitizer")
 - · UBSAN ("Undefined Behaviour Sanitizer")
- These allow us to "instrument" R with compiler-dependent run-time diagnostics
- Problem: Needs R sources, recent compilers, knowledge of building R from source
- · Solution: Docker!
- · sanitizers CRAN package triggers 'true positives' validating toolchain setups so that errors can be replicated & fixed.
- · See my sanitizers page and my worked UBSAN example

ASAN Errors

```
#include <R.h>
#include <Rdefines.h>
extern "C" {
    // https://code.google.com/p/address-sanitizer/wiki/ExampleHeapOutOfBounds
    SEXP heapAddressSanitize(SEXP xs) {
        int *array = new int[100], x, y, *pres;
        SEXP res;
        array[0] = 0;
        x = INTEGER_VALUE(xs);
        y = array[x + 100];
                                       // BOOM
        delete [] array;
        PROTECT(res = NEW_INTEGER(1));
        pres = INTEGER_POINTER(res);
        pres[0] = y;
        UNPROTECT(1);
        return res;
                                                                          33/51
```

ASAN Errors

for details.

```
edd@max;~/git$ docker run --rm -ti -v $(pwd);/mnt rocker/r-devel-san RD CMD check /mnt/sanitizers 0.1.0.
* using log directory '//sanitizers.Rcheck'
* using R Under development (unstable) (2015-06-17 r68530)
Γ...1
* checking tests ...
 Running 'testHeapAddressSanitize.R'
ERROR
Running the tests in 'tests/testHeapAddressSanitize.R' failed.
Last 13 lines of output:
   Freed heap region:
                          fd
   Stack left redzone:
                           f1
    Stack mid redzone:
                        f2
   Stack right redzone: f3
   Stack partial redzone: f4
   Stack after return:
                           f5
   Stack use after scope: f8
   Global redzone:
                           f9
   Global init order: f6
   Poisoned by user:
                           f7
   Contiguous container OOB:fc
   ASan internal:
                            fρ
  ==266==ARORTING
* checking PDF version of manual ... OK
* DONE
Status: 1 ERROR
See
  '//sanitizers.Rcheck/00check.log'
```

UBSAN Errors

- · For UBSAN we use a different Docker image
- It includes a wrapper script check.r which makes deployment very easy.

UBSAN Errors

```
#include <R.h>
#include <Rdefines.h>
extern "C" {
   // with thanks to Greg Jefferis (https://qithub.com/eddelbuettel/docker-debi
   // call with a sufficiently large x such as 31
   SEXP intOverflow(SEXP xs) {
       int x, y, *pres;
       SEXP res:
       x = INTEGER_VALUE(xs);
       y = (1 \ll x) - 1; // BOOM -- (signed) int overflow
       PROTECT(res = NEW_INTEGER(1)); // Allocating storage space
       pres = INTEGER POINTER(res); // pointer to SEXP object
       pres[0] = y;
       UNPROTECT(1);
       return res;
```

UBSAN Errors

See

for details.

'/mnt/sanitizers.Rcheck/00check.log'

```
edd@max:~/git$ docker run --rm -ti -v $(pwd):/mnt rocker/r-devel-ubsan-clang check.r -s /mnt sanitizers
* using log directory '/mnt/sanitizers.Rcheck'
Γ...1
* checking tests ...
 Running 'testHeapAddressSanitize.R'
 Running 'testIntOverflowSanitize.R'
ERROR
Running the tests in 'tests/testIntOverflowSanitize.R' failed.
Last 13 lines of output:
 R is a collaborative project with many contributors.
 Type 'contributors()' for more information and
  'citation()' on how to cite R or R packages in publications.
 Type 'demo()' for some demos, 'help()' for on-line help, or
  'help.start()' for an HTML browser interface to help.
 Type 'q()' to quit R.
 >
 > library(sanitizers)
 > intOverflowSanitize(31)
 int_overflow.cpp:17:23: runtime error: signed integer overflow: -2147483648 - 1 cannot be represented in
* checking PDF version of manual ... OK
* DONE
Status: 1 ERROR
```

37/51

Building

Bootstrap

Building a container from scratch can be as easy as

```
#!/bin/bash
## need to bootstrap a Debian system
sudo apt-get install debootstrap
## use the 'testing' distro named 'stretch'
sudo debootstrap stretch /tmp/rootfsStretch
## tar this up, and import it
sudo tar czf /tmp/stretch base32 rootfs.tgz \
      -C /tmp/rootfsStretch .
cat /tmp/stretch_base32_rootfs.tgz | \
    docker import - debian:testing
```

With thanks to Matt Whiteley's blog post on using i386 use.

Dockerfiles

Building the Rocker base image is then as easy as

```
#!/bin/bash

cd ~/git/rocker/r-base
docker build -t rocker/r-base .
docker tag rocker/r-base r-base
```

The tag command ensures we can refer to this also as just r-base.

Dockerfiles

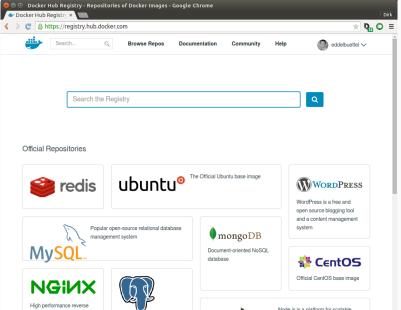
And now that we have r-base:

```
#!/bin/bash

cd ~/git/rocker/r-devel
docker build -t rocker/r-devel .
```

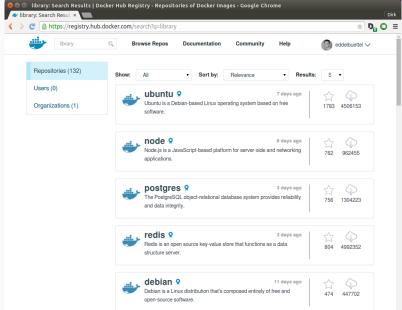
But note that this is not needed for standard Docker use where can just docker pull.

Hub



42/51

Hub



43/51

Hub

- · For standard (64-bit) Docker, the 'Hub' provides access to lots of different contains
- The Hub provides 'official' containers for languages, distributions or applications
- · Our r-base container is now the official R container
- · But any registered user can create his or her set

GitHub and Hub integration

- · Easy and smooth GitHub and Docker Hub integration
- Create a GitHub repo with a Dockerfile, link it to a Hub repository
- Each commit can tickle and updated build, dependency graphs can be declared
- \cdot Carl uses this e.g. for a r-base -> r-studio -> hadleyverse -> ropensci toolchain

Next

Swarms / Orchestration

Lots of new things we didn't get to today:

- · docker-machine (formerly fig)
- · docker-compose
- · docker-swarm

The End

More Info

- · Various Docker tutorials on the web, starting with trylt demo
- · Docker documentation
- · For R on Docker, our Rocker.Org has
 - · GitHub repository containing,
 - · the rocker repo and more, and then the
 - Docker Hub page

Thanks

- · Carl Boettiger for all the joint work on Rocker
- · the folks at Docker for making rocker/r-base the official R
- Matt Whiteley for the blog post allowing me to locally patch the Debian package and get started on i386

Collophon

Made using

- · TeXlive 20141024
- · Beamer with mtheme
- · Pandoc 1.12.4.2
- · R 3.2.1
- · rmarkdown 0.7
- · Emacs 24.4
- · Ubuntu 15.04