Use of Docker[☆] for deployment and testing of astronomy software

D. Morris, S. Voutsinas, N.C. Hambly and R.G. Mann

Institute for Astronomy, School of Physics and Astronomy, University of Edinburgh, Royal Observatory, Blackford Hill, EH9 3HJ, UK

Abstract

Lessons learned from using Docker for deployment and testing

Keywords:

1. Introduction

In common with many of the physical sciences astronomy has entered the era of Big Data (Mickaelian, 2015). Nowadays astronomical Data Centres aspire to not only serve subsets of large–scale datasets, but also to provide increasingly sophisticated services alongside their data holdings in order that their users can fully exploit them. The concept of the 'science archive' (e.g. Hambly et al., 2008, and refereces therein) emerged in the recent past in part fulfilment of the mantra 'ship the results, not the data' (e.g. Quinn et al., 2004, and other contributions within the same volume). Such archives generally provide Structured Query Language access to their catalogues which goes some way to confining processing/filtering, at least initially, to the server side.

Ultimately it is via provision to the end–user of significant computational resources, server–side within the Data Centre or distributed with the data in Grid or Cloud architectures, that the full potential of user driven server–side data analysis can be realised (e.g. Ball, 2013). This in turn requires an infrastructure capable of dealing with the questions and concerns involving scalability, security, portability and reproducability that arise when computational resources within the Data Centre are made available to the end user.

Docker[☆] is emerging as the technology of choice in such situations (Yu and Huang, 2015; Wang et al., 2015). Docker is an operating system level virtualization environment [1] that uses software containers to provide isolation between applications. The rapid adoption and subsequent evolution of Docker from the initial open source project

[☆] `https://www.docker.com`

Email address: `dmr,stv,nch,rgm@roe.ac.uk`

(D. Morris, S. Voutsinas, N.C. Hambly and R.G. Mann)

[1] `https://en.wikipedia.org/wiki/Operating-system-level_virtualization`

launched in 2013[2] by PaaS provider dotCloud[3], to the formation of the Open Container Initiative [4] in 2015[5] suggests that Docker met a real need within the software development community which was not being addressed by the existing tools.

The membership list for the Open Container Initiative (ref appendix A) mirrors the range of applications of the Linux kernel itself, from large scale cloud compute platforms (Amazon, Google and Joyent) and super computers (IBM and Fujitsu) to embedded IOT systems (Resin.IO).

The concepts and technologies for operating system level virtualization have existed for a number of years. Solaris Containers have been available as part of the Solaris operating system since 2005 [6] and has been used extensivley in large scale production environments. Even within Linux, the core technologies used by Docker containers, cgroups [7] and namespaces, were first added to the Linux kernel in 2007.

Although both the speed and simplicity of the Docker API have contributed to its adoption, it is the development of a standardized format and interface for describing and managing software containers that has been the *'unique selling point'* that differentiates Docker from its competitors, and has been the driving force behind the rapid adoption of Docker across such a wide range of different applications. [8] [9]

- At the end-user level, Docker enables users to describe, share and manage applications and services using a common interface by wrapping them in standardized containers.
- From a developer's perspective, Docker makes it easy to create standard containers for their software applications or services.
- From a system administrator's perspective, Docker makes easy to automate the deployment and mangement of business level services as a collection of standard containers.

1.1. Docker, DevOps and MicroServices

In the DevOps world, the software developer and system administrator roles are members of the same team, directly involved in developing, deploying and managing business level services.

Describing the deployment environment using Docker containers enables the team to treat system infrastructure as code, applying the same tools

[2] http://www.infoq.com/news/2013/03/Docker

[3] https://www.dotcloud.com/

[4] https://www.opencontainers.org/

[5] http://blog.docker.com/2015/06/open-container-project-foundation/

[6] https://en.wikipedia.org/wiki/Solaris_Containers

[7] https://en.wikipedia.org/wiki/Cgroups

[8] http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/

[9] http://www.americanbanker.com/news/bank-technology/why-tech-savvy-banks-are-gung-ho-about-container-softwa html

they use for managing the software source code, e.g. source control, automated testing etc. to the infrastructure configuration.

In particular, Docker has emerged as one of the key technologies for automating the continous delivery and continous deployment of MicroService based architectures.

> "in 2014, no one mentioned Docker ..
> in 2015, if they don't mention Docker,
> then they aren't paying attention"

[The State of the Art in Microservices by Adrian Cockcroft, Jan 2015] [10]

1.2. Reproducable science

In the science and research community, Dockers ability to describe a software deployment environment has the potential to improve the reproducability and the sharing of data analysis methods and techiniques. (Boettiger, 2014) describes how the ability to publish a detailed description of a software environment alongside a research paper enables other researchers to reproduce and build on the original work. (Nagler et al., 2015) describes work to develop containerized versions of software tools used to analyse data from particle accelerators. [11]

A simple example of how using Docker to wrap an application can make it more portable is our wrapper for the IVOATEX[12] document build system which has a number of OS–dependent installation instructions. Wrapping the toolkit in a Docker container and making it available on GitHub [13] and the Docker registry [14] makes it easy to deploy and run.

1.3. Compute resource services

In the case of submitting user code to a compute resource for execution within a data center, there are two roles in which Docker may be useful. Docker can be used internally to provide the virtualization layer for deploying and managing the execution environments for the submitted code. This scenario is already being evaluated by a number of groups, in particular Docker is one of the technologies being used to deliver a 'platform as a service' (PaaS) infrastructure for the European Space Agency's Gaia mission archive (O'Mullane, 2016; Ferreruela, 2016).

Alternatively, Docker could be used as part of the public service interface, providing the standard language for describing and packaging the software. In this scenario, the user would package their software in a container and then either submit the textual description, or the binary image of the container, to the service for execution. The advantage of this approach is that by wrapping the analysis software in a standard container it

[10] https://www.youtube.com/watch?v=pwpxq9-uw_0&t=160

[11] https://github.com/radiasoft/containers

[12] http://www.ivoa.net/documents/Notes/IVOATex

[13] https://github.com/ivoa/ivoatex

[14] https://hub.docker.com/r/ivoa/ivoatex/

makes it portable. The user can build and test their analysys software on their own platform before submitting it to the remote service for exection. The common standard for container runtime environment means that the user can be confident that their software will behave in the same manner when tested on a local platform or deployed on the remote service.

In this paper we focus on some specific aspects of the above during our development and deployment of the Firethorn infrastructure .... In Section ?? we provide details of ...; in Section ?? we discuss ...; and finally in Section ?? we conclude with a summary of our findings which we believe will be of wider benefit to the astronomical software development community.

2. Portability and Reproducibility

One of the key drivers for using Docker to deploy our systems was the level of portability and reproducibility it provides.

Within our development process our software is run on a number of different platforms, including the develoers desktop computer, the integration test systems and the live deployment system.

However much care was taken to control each environment they inevitably ended up being slightly different.

Even something as simple as the Java or Tomcat version used to run the software can be difficult to control reliably.

We could, in theory, mandate a specific version and configuration of the software stack used to develop, test and deploy our software.

In reality, unless a platform is created and managed exclusievly by an automated process, then manual configuration means that some level of discrepancy will creep in, often when it is least expected.

There are a number of different was of achieving this level of automation.

A common method of managing a large set of systems is to use an automated configuration management tool, such as Puppet [15] or Chef [16], to manage the system configuration based on information held in a centrally controlled template.

Another common practice is to use a continuous integration platform such as Jenkins [17] to automate the testing and deployment of the products from a software development project.

These two techniques are not exclusive, and it is not unusual to use an automated configuration management tool such as Puppet to manage the (physical or virtual) hardware platform, in combination with a continuous integration platform such as Jenkins to manage the integraton testing, and in some cases the live service deployment as well.

In our case, the limits on the number of resources available, both human and technological

[15] `https://puppetlabs.com/`

[16] `https://www.chef.io/chef/`

[17] `https://wiki.jenkins-ci.org/`

at the start of the project mean that whatever system we used needed to start small and evolve as the project developed.

We initially started out using manually configured virtual machines to host the test and live deployments, which gave us an initial level of isolation from the physical machine configuration.

For example, we could control the verion of the Java runtime and Tomcat web server deployed inside the virtual machines, without impacting other project's software and services running on the same physical hardware.

It also allowed us to run more than one set of our services on the same physical platform, while still being able to configure each set of services independently.

The first step in automating the deployment process was to automate the allocation of the virtual machines using a set of shell scripts which created new virtual machines from a set of preconfigured templates [18].

The templates handled the basic virtual machine configuration such as cpu and memory allocation, network configuration, disc space and operating system.

The command line tools enabled us to spin up a new virtual machine in about 20 seconds, from initial create command to running machine with a login prompt.

Replacing the manually configured virtual machines that were available at the start of the project with template based instances gave us a consistent set of platforms to work with.

Once the virtual machines were created, we used a set of shell scripts to automate the installation of the additional software packages needed to run our services.

For our Java webservices, this included installing and configuring specific versions of the Java runtime [19] and Apache Tomcat [20] web server.

The final step in the process was to deploy our webservices into Tomcat and configure them with the correct user accounts and passwords to access the local databases.

At the point when we started to look at using Docker, although the process of installing and configuring the software inside the virtual machines was scripted, it was not automated. The process still required a level of human interation, specifying the parameters for the scripts and running them.

We were beginning the process of evaluating the available configuration management and continuous integration tools when we included Docker in the list of tools to look at.

—-

Early in our adoption of Docker, we discovered one of its strengths to be the portability it provides. Often in scientific development, as well

[18] `https://github.com/Zarquan/ischnura-kvm`

[19] `http://openjdk.java.net/`

[20] `http://tomcat.apache.org/`

as in software development as a whole, projects consist of large teams whose members may have unique preferences in terms of development environments. This was the case during our own developments. We ran the same services in development and production servers apart from our laptops, each with their own specifications, characteristics and libraries, within an intricate and relatively inflexible network setup. In such complex systems, team collaboration in different environments raises issues often caused by different versions of libraries or software behaving differently in different platforms. These issues take time to track down, as we encountered a few times.

As we switched to Docker for running the full chain of our software services, we noticed fewer and fewer of these issues, i.e. missing dependencies, library mismatches; and we experienced more efficient collaboration and bug tracking, faster deployment in our testing and production servers, and automation of the virtualization, setup and configuration of our systems.

But even when looking at Docker from the perspective of simply doing science, the portability that it provides offers clear advantages and solves problems that scientists often encounter. Take as an example a scientist who has written a couple of Python classes and archived them as a zip somewhere. We can easily imagine someone in the future wanted to rerun this legacy piece of code from a completely different platform, environment and newer libraries, which may be incompatible with what the original scientist had written. Or even if the platform was fully compatible, running the code would require several manual processes such as downloading the zip file, extracting the files, reading documentation on how to set up and run the code, installing any dependencies with the right versions for each, etc. Contrast that process with how this could be done using Docker, which could be as simple as:

```
docker run --name container_name \
       scientist_id/my_python_code
```

3. Better control of support environment

It is often the case in software development that developers do not have full control of the server platform where their code will run. For example, a server, where the code is meant to be run, may already be set up with a specific version of java and tomcat, with no possibility to alter these versions due to other services running from this machine. With Docker, you can set up your containers to use whichever version of java, tomcat, or other libraries needed; and this chain will run in a virtual, isolated environment on the shared machine mentioned before. This, it provides the capability of having full control of the environment where your code will run, without the concern of what the underlying machine already has installed.

4. Container types

WebService containers

(Java + Tomcat) + webapp

(Apache + WebPy) + Python Built on top of the previously mentioned web service containers was our interface which was also built using docker containers. Without going into low level detail, the containers were built as an SQL proxy container that linked to our database for storing user queries, an apache container which sat on top of an ubuntu image, a base python and a python-libraries container and finally a webpy container.

Support services

JDBC connections ambasador abstraction

Metadata database

Test statistics database

Build tools

Maven Mercurial Docker

sqlsh

ivoatex

5. Tricks learned

mount unix sockets

docker in docker mount docker sock

ssh in docker mount ssh sock

ambasadors

socat proxy

issues with original ambasador[21]

original idea[22]

source code not published until later[23]

easy enough to roll our own

unix to http docker sock

JDBC proxy

JDBC over SSH proxy

rolling your own

docker makes it easy open source makes it easy to copy ideas start with binary blobs work back up the tree creating our own socat java + tomcat fedora

fine grained control over versions

6. Issues found

When using new technologies for the first time, in particular with newer ones, encountering issues is expected. These issues might be caused by a developer/scientist mistake made while overcoming the learning curve or by software bugs in the technology itself, which may have not been uncovered yet while adoption of the technology is still growing, and all possibly usages of it have not been visited yet. We document here some of these issues. More important than an analysis of the issues themselves is the understanding of the process undertaken to discover and solve them.

An important point to make here, before going into more detail, is in regard to the open-source

[21] https://hub.docker.com/r/svendowideit/ambassador/

[22] http://docs.docker.com/engine/articles/ambassador_pattern_linking/

[23] https://github.com/SvenDowideit/dockerfiles/blob/master/ambassador/Dockerfile

nature and culture of Docker and the Docker community. We will describe this in more detail in the following sections. The main takeaway from this was that both finding how to go about solving issues related to containers and figuring out how the preferred method of implementing a certain feature is easy enough as doing a search of the keywords related to what you need. This can be done by either using a generic search engine or visiting the sources where the main Docker community interaction takes place [24] [25] [26].

Because Docker is an open source solution, it has an active open source community behind it which enables users to find and fix issues more efficiently. An open source community means it is more likely that any issue you might find has already been encountered by someone else, and just as likely that it has been solved, officially (as part of a bug fix in Docker release) or unofficially (here is how I solved thisuser xyz). Contrast this with encountering issues using some proprietary technology with a more limited number of users, with a much slower pace of updating versions and bug-fixing. Compare it to building a workflow or software environment, sitting possibly on a Virtual Machine with a particular setup of interconnected technologies and/or services, which has for many years come with a significant overhead for finding solutions to problems encountered in this chain.

Another key point to note is how we benefited from Dockers support team as well as the number of early adopters. We decided to take up Docker at an early stage, which can be considered its bleeding-edge phase (Version 1.6), at which point it was more likely to discover issues. However, with the large team and strong technological support of its developers, as well as the significant number of early adopters, new releases to solve bugs or enhance usability and performance were issued frequently. Consequently, after some research, we realized that many of the issues we found, whether they could be considered bugs or usability improvements needed, were often fixed in subsequent releases, meaning that by updating our Docker version they would be solved.

7. Memory issues

As part of our development for the Firethorn project (link) we developed a testing suite written in Python. This suite included some long-running tests, which iterated a list of user submitted SQL queries that had been run through our systems in the past, running the same query via a direct route to SQL server as well as through the new Firethorn system and comparing the results. This list scaled up to several thousand queries, which meant that a single test pass for a given cata-

[24] https://forums.docker.com/c/general-discussions/general

[25] http://stackoverflow.com/questions/tagged/docker

[26] https://github.com/docker/docker/issues

logue could take several days to complete. The issue we encountered here was that the docker process was being killed after several hours of initializing the test, with out of memory error messages. An initial attempt at solving the problem was to set memory limits to all of our containers, which changed the symptoms and then caused one of our containers to fail with memory error messages. This also happened to be the main Tomcat service we had that served as the engine to the system. After a few iterations of attempting to run the chain with different configurations, the solution was found through community forums, when we discovered that several other people were encountering the same symptoms with similar setups. Specifically, the problem was due to a memory leak, caused by the logging setup in version 1.6 of Docker, where output sent to the system stdout was being stored in memory causing a continuous buffer growth. (https://github.com/docker/docker/issues/9139) The solution to this problem that we adopted was to send the container system output, and all other logs from our container processes, to log files in our host.

```
docker run \
       ...
       --volume "/var/logs/firethorn/: \
         /var/local/tomcat/logs" \
       ...
       "firethorn/firethorn:2.0"
```

We learned several valuable lessons through the process of researching how other developers managed these problems, for example, the approach to logging where the logs of a container are stored separately from the container itself, making it easier to debug and follow the system logs. In addition, we benefited from learning how and why limiting memory for each container was an important step when building each of our containers.

As we noticed shortly after the issue was raised in the Docker community, a fix was released as part of a new Docker release, 1.7. In addition Docker have since then released a new pluggable driver based framework for handling logging[27]

Original docker logging to JSON

Docker version host version docker in docker version

Docker registry fedora mis-tagged

storage drivers lvm btrfs

Conflict with libvirtd on RedHat/Fedora

8. Moving target

Memory issues fixed

Docker registry

Docker compose

Docker storage

Docker network

[27] https://docs.docker.com/engine/reference/logging/overview/

9. Conclusion

Acknowledgements

This work has been supported in part by grants from EC FP7 programmes ... and from the UK Science and Technology Facilities Council.

References

Ball, N.M., 2013. CANFAR+Skytree: A Cloud Computing and Data Mining System for Astronomy, in: Friedel, D.N. (Ed.), Astronomical Data Analysis Software and Systems XXII, p. 311. arXiv:1312.3997.

Boettiger, C., 2014. An introduction to Docker for reproducible research, with examples from the R environment. ArXiv e-prints arXiv:1410.0846.

Ferreruela, V., 2016. Gavip gaia avi portal, collaborative paas for data-intensive astronomical science, in: Lorente, N.P.F., Shortridge, K. (Eds.), ADASS XXV, ASP, San Francisco. p. TBD.

Hambly, N.C., Collins, R.S., Cross, N.J.G., Mann, R.G., Read, M.A., Sutorius, E.T.W., Bond, I., Bryant, J., Emerson, J.P., Lawrence, A., Rimoldini, L., Stewart, J.M., Williams, P.M., Adamson, A., Hirst, P., Dye, S., Warren, S.J., 2008. The WFCAM Science Archive. Mon. Not. R. Astron. Soc. 384, 637–662. doi:10.1111/j.1365-2966.2007.12700.x, arXiv:0711.3593.

Mickaelian, A.M., 2015. Astronomical Surveys and Big Data. ArXiv e-prints arXiv:1511.07322.

Nagler, R., Bruhwiler, D., Moeller, P., Webb, S., 2015. Sustainability and Reproducibility via Containerized Computing. ArXiv e-prints arXiv:1509.08789.

O'Mullane, W., 2016. Bringing the computing to the data, in: Lorente, N.P.F., Shortridge, K. (Eds.), ADASS XXV, ASP, San Francisco. p. TBD.

Quinn, P.J., Barnes, D.G., Csabai, I., Cui, C., Genova, F., Hanisch, B., Kembhavi, A., Kim, S.C., Lawrence, A., Malkov, O., Ohishi, M., Pasian, F., Schade, D., Voges, W., 2004. The International Virtual Observatory Alliance: recent technical developments and the road ahead, in: Quinn, P.J., Bridger, A. (Eds.), Optimizing Scientific Return for Astronomy through Information Technologies, pp. 137–145. doi:10.1117/12.551247.

Wang, X.Z., Zhang, H.M., Zhao, J.H., Lin, Q.H., Zhou, Y.C., Li, J.H., 2015. An Interactive Web-Based Analysis Framework for Remote Sensing Cloud Computing. ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences , 43–50doi:10.5194/isprsannals-II-4-W2-43-2015.

Yu, H.E., Huang, W., 2015. Building a Virtual HPC Cluster with Auto Scaling by the Docker. ArXiv e-prints arXiv:1509.08231.

Appendix A. Open Container Initiative membership

Cloud Services

- Cloud Services
  - Amazon web services - https://aws.amazon.co
  - Google - http://www.google.com/
  - Apcera - https://www.apcera.com/
  - EMC - http://www.emc.com/
  - Joyent - https://www.joyent.com/
  - Kyup - https://kyup.com/
  - Odin - http://www.odin.com/
  - Pivotal - http://pivotal.io/
  - Apprenda - https://apprenda.com/
  - IBM - http://www.ibm.com/

- Operating systems & software
  - Microsoft - http://www.microsoft.com/
  - Oracle - http://www.oracle.com/
  - CoreOS - https://coreos.com/
  - Redhat - http://www.redhat.com/en
  - Suse - https://www.suse.com/
- Container Software
  - Docker - https://www.docker.com/
  - ClusterHQ - https://clusterhq.com/
  - Kismatic - https://kismatic.io/
  - Portworx - http://portworx.com/
  - Rancher - http://rancher.com/
  - Univa - http://www.univa.com/
- Security
  - Polyverse - https://polyverse.io/
  - Scalock - https://www.scalock.com/
  - Twistlock - https://www.twistlock.com/
- Datacenter infrastructure
  - Nutanix- http://www.nutanix.com/
  - Datera - http://www.datera.io/
  - Mesosphere - https://mesosphere.com/
  - Weave - http://www.weave.works/
- Computing hardware
  - Intel - http://www.intel.com/
  - Dell - http://www.dell.com/
  - Fujitsu - http://www.fujitsu.com/
  - Hewlett Packard Enterprise - https://www.hpe
- Telecommunications hardware
  - Cisco - http://www.cisco.com/
  - Infoblox - https://www.infoblox.com/
  - Midokura - http://www.midokura.com/
  - Huawei - http://www.huawei.com/
- Telecommunications providers
  - AT&T - http://www.att.com/
  - Verizon Labs - www.verizonwireless.com/
- System Monitoring
  - Sysdig - http://www.sysdig.org/
- Finance
  - Goldman Sachs - http://www.goldmansachs.co
- Virtualization platforms
  - VMware - http://www.vmware.com/
- IOT Embedded Systems
  - Resin.io - https://resin.io/
- Social Media Platforms
  - Twitter - https://twitter.com/