**Technical Report on the installation of OSKAR using Docker**

Iain Emsley, … Fred , Ben …

In this report, we build the OSKAR software using Docker and Puppet configuration systems.

Why Docker? Why compare Puppet – ALMA ?

We chose the OSKAR simulation tool to build as it relies upon casacore and uses the CUDA library that requires access to the GPU card.

The software must be installed within 30 seconds to satisfy the constraint.

**Methodology**

We built the OSKAR software (oskar.oerc.ox.ac.uk) in two forms: from source and from binary files.

We then use Docker and Puppet to build the software on a Supermicro machine in the Oxford testbed (<https://confluence.ska-sdp.org/display/DATA/Testbed+Specifications)> using a Dell 1950 as the build machine.

These machines are linked by a 1 GB Ethernet cable.

In the first test, we built a series of Dockerfiles and ran this as a Dockerfile. We created one Docker file that builds the application from source and one that builds the application from the binaries in the Radio Astronomy ppa.

In the second experiment, we use a Jenkins Build System to build the nodes.

Our initial exploration is whether the OSKAR software can be built using the chosen tools.

We must also explore the reporting processes that the tools use. This will satisfy the requirements. We propose an experiment to read the data into a local collector and sent to the management node via a messaging system.

The second experiment combines the automated build system and the build tool.

**Performance Requirements**

The system must be scale to x (TBC) nodes.

The technology should provide less than x% (TBC) overhead to the installed system

The technology should deploy the software in less than 30 seconds (SKA\_SYS\_REQ 4)

**Functional Requirements**

The technology must implement the software and its associated configuration. (F.1.9, F.2.13)

The technology should allow for remote updating of the installed components. (F.20.1.1)

The system must allow for profiles or images for the machines to be built. (F.20.1, F.20.1.1, F.20.1.2)

The technology should be monitorable (F.15.2.4, F.15.3.3, F.15.2, F.15.2.3.2)

The system should have a repository of profiles and components that can be tested and updated in different environments.(F.15.1.8)

**Results**

From the initial tests, we observe that Dockerfiles can be used to build the system images repeatably, satisfying the functional requirements F.20.1, F.20.1.1, F.20.1.2.

The container must be run in privileged mode or have a GPU device explicitly assigned to it. By default, it does not have write access to the hardware.

**Discussion**

Our initial hypothesis for the experiment was that the images would have to be pre-built and then deployed onto the server in the best case to ensure that the performance constraint is met.

We observed that the constraint of 30 seconds for a process to be deployed onto a compute node may be consistently achieved from precompiling all the binaries and the system images. This creates a constraint in the networking between the Telescope Manager process that begins the change of modes to the build server and from the build server to the compute node or nodes being installed.

Saving the Dockerfile and scripting metadata helps the reproducibility of the science, especially over the long term of the experiment. This would suggest that the file with the parameters should be listed with the package version number. In [Boettiger], we see the arguments for the reproducibility, prevention of code rot, and documentation for how a system is built, or a version of the system.

*Parametrizable Files*

Docker files are not parametrizable. The COPY term on the language can be used for inheritance but it does not allow for the parameter via a function file.

*Monitoring*

There are three ways of monitoring Docker processes:

* Using the sysfs virtual file systems
* The remote API presented by libcontainer now part of https://www.opencontainers.org/
* The Docker stats command

There are options for monitoring Docker (<http://rancher.com/comparing-monitoring-options-for-docker-deployments/>). The Prometheus monitoring tool has now been replaced by Google's Container Advisor (cAdvisor) tool. The Sensu framework (<https://sensuapp.org/>) can also consume data.

It appears that cloud providers such as DataDog (https://www.datadoghq.com/blog/how-to-collect-docker-metrics/), use the sysfs options for monitoring the containers at intervals.

As the container is a Linux cgroup, we can use the tools linked to the kernel to monitor the usage. This would replicates the docker stats tool but writes the data in fashion that we would prefer. Once enabled, the memory.stat file allows us to view the memory statistics. CPU can be measured through cpuacct.stat and blkio provides the I/O metrics.

When using the sysfs option, Docker collects its containers into a namespace that can then be read. This requires the imonitoring to be able to map ithe Docker container ids to the human readable names. If we use the container id as the namespace within the compute node, the management node can record the container id when launching the container. Using this, the node can report on the statistics that it is tasked doing and the mapping overhead is held by the management systems.

The total memory can be calculated by reading the memory.usage\_in\_bytes file or combining the values from inactive\_anon, active\_anon,inactive\_file, active\_file.

We set up a collector on the node that streamed data to a management process [Brand].

*Open Standards*

Docker does appear to be using the Open Container Initiative, suggesting that the tool does use an emerging standard.

**Testing**

As a result of the work on building Dockerfiles to build Docker images that can then be run with the same build, we considered testing the files. This provides confidence that the file and Docker are installing the desired packages and that these can be run.

We consider the practice of Test Driven Development for Docker. This is an established practice in the Puppet workflow but the Docker practices are still relatively new.

Using the guidance from the rspec-docker Ruby library, we developed a light weight prototype testing library in Python using Behave and Python. The Behave library is used to write the tests in a common language that is then translated into machine readable tests.

Rspec is a Behaviour Driven Development testing tool that uses a version of the Cucumber language. This allows the test to be written in a way that is understandable, which does not determine the model to the expected behaviour.

Behave is a Python version of Rspec that uses the Cucumber language to link the features written in the domain common language to the feature tests. This provides user acceptance tests.

This allows us to test a built Docker file and to run user tests to provide “smoke tests” for the system. We then need to define a method of testing the running software on the image.

**Conclusion and Future Work**

Future work would include linking with the debtree tool (<http://collab-maint.alioth.debian.org/debtree/>) to verify that the directed acylcic graphs of configuration management such as Puppet or Docker container systems, are correctly created.

We see this as aiding the sustainability of the SKA in terms of science and engineering.

References

<https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes>

Di Tommaso P, Palumbo E, Chatzou M, Prieto P, Heuer ML, Notredame C. (2015) The impact of Docker containers on the performance of genomic pipelines. PeerJ 3:e1273 <https://doi.org/10.7717/peerj.1273>

Carl Boettiger. 2015. An introduction to Docker for reproducible research. SIGOPS Oper. Syst. Rev. 49, 1 (January 2015), 71-79. DOI=http://dx.doi.org/10.1145/2723872.2723882

Stodden, Victoria, The Scientific Method in Practice: Reproducibility in the Computational Sciences (February 9, 2010). MIT Sloan Research Paper No. 4773-10. Available at SSRN: <http://ssrn.com/abstract=1550193> or <http://dx.doi.org/10.2139/ssrn.1550193>

Aranguren, M. E., & Wilkinson, M. D. (2015). Enhanced reproducibility of SADI web service workflows with Galaxy and Docker. *GigaScience*, *4*, 59. <http://doi.org/10.1186/s13742-015-0092-3>

<https://docs.docker.com/engine/articles/runmetrics/>

The sysfs Filesystem, Patrick Mochel, (<https://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf>[),](https://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf),m) accessed 19 January 2016

Jim Brandt, David DeBonis, Ann Gentile, Jim Lujan, Cindy Martin, Dave Martinez, Stephen Olivier, Kevin Pedretti, Narate Taerat, Ron Velarde. Enabling Advanced Operational Analysis Through Multi-Subsystem Data Integration on Trinity, Proceedings of the 2015 Cray Users' Group Annual Technical Conference, Chicago, Illinois, April 2015