

StarDock: Shipping Customized Computing Environments to the Data

Michael D. Young^a, Soichi Hayashi^a, and Arvind Gopu^a

^aIndiana University, 3709 E. 10th St, Bloomington IN, USA;

ABSTRACT

Surging data volumes make it increasingly unfeasible to transfer astronomical datasets to the local systems of individual scientists. Centralized pipelines offer some relief, but lack flexibility to fulfill the needs of all users. We have developed a system that leverages the Docker container application virtualization software. Along with a suite of commonly used astronomy applications, users can configure a container with their own custom software and analysis tools. Our StarDock system will move the users container to the data, and expose the requested dataset, allowing our users to safely and securely process their data without needlessly transferring hundreds of gigabytes.

Keywords: Docker, container, ODI-PPA, scalable compute archive

1. INTRODUCTION

There is a growing need in astronomy to allow observers and archival users alike to run their preferred software “close to the data”. The combination of large-format CCD imagers and rapid readout times produces data volumes too large for the typical user to transport, store, reduce, and analyze the astronomical data obtained. In recent years there have been attempts to address this issue by developing one-size-fits-all pipelines to filter the data stream down into a manageable set of data products^{1,2}. While this approach has some merit, these pipelines tend to have limited flexibility and cannot possibly satisfy the various research requirements of all the potential consumers of a telescope’s observational data.

As a compromise we have developed StarDock, a system that allows users to construct customized computing environments that can run on shared hardware, without polluting said shared hardware with niche software, or suffering delays waiting for system admins to deliver specific components. This is accomplished through the use of Docker containers to create a predictable, stable, and customizable computing environment for the users. With StarDock, users can create a space where they can use their preferred software and methods to analyze large volumes of data without the overhead of local storage or time-consuming and unnecessary data transfers.

In this work we will describe the design and structure of StarDock, trace a typical StarDock workflow, and address the security concerns raised by allowing outside users to execute arbitrary code. It is our hope that StarDock can serve as a model to show that the centralization of computing and storage resources does not necessarily require that users abandon their preferred methods of astronomical data processing and analysis.

2. STARDOCK COMPONENTS

StarDock makes use of the SCA Workflow (in preparation), which defines a set of resources, services, and tasks and dictates the flow of information between different services based on the requirements of a given workflow. For StarDock, the workflow consists of a Data Service which stages the astronomical data, a Build service which constructs the customized computing environment in Docker, and a Run service, which initiates the Docker container and makes the requested data accessible to the user. Each of these services are discussed in more detail in the following sections, and the workflow is illustrated in Figure 1.

Further author information: (Send correspondence to M.D.Y)

M.D.Y: E-mail: youngmd@iu.edu, Telephone: 1 812 606 8940

S.H: E-mail: hayashis@iu.edu

A.G: E-mail: agopu@iu.edu

Software and Cyberinfrastructure for Astronomy III, edited by Gianluca Chiozzi,
Juan C. Guzman, Proc. of SPIE Vol. 9913, 991318 · © 2016 SPIE
CCC code: 0277-786X/16/\$18 · doi: 10.1117/12.2233081

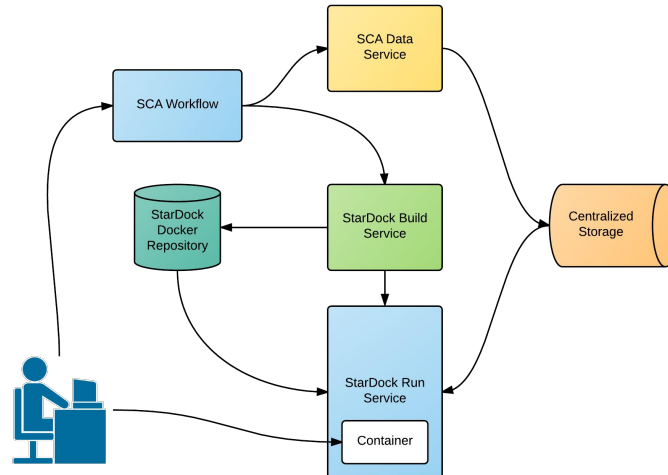


Figure 1. StarDock Workflow. The user initiates the workflow, which transfers data from tape using the SCA Data Service, constructs the Docker image using the StarDock Build Service, and executes the container using the StarDock Run Service.

2.1 SCA Data Service

The SCA Data Service was created separately from StarDock, and is an example of the sort of the reusable microservice that the SCA Workflow was created for. Briefly, the SCA Data Service receives a list of files and confirms the requesting user is allowed access to the requested data, which are then staged from the secure tape archive at IU and are moved to the Data Capacitor, a shared Lustre-based file system accessible from the compute resources at IU. A more complete description of the Data Service has been previously published as a portion of the ODI Pipeline Portal and Archive.³ After staging the requested files to disk the Data Service returns as its output the network-accessible path to the staged data.

2.2 Build Service

The Build Service receives a set of configuration options from the SCA Workflow. Based on these options, it constructs a customized DockerFile. The DockerFile begins with a base operating system image, typically some version of the Ubuntu Linux distribution. This is then supplemented by Docker commands which specify the installation and configuration of an SSH server, as well as defining a default account which will be the user's point of entry into the system.

The Build Service then iterates through the user's chosen suite of astronomical applications, adding in turn each of the required steps to install and configure the software within the Docker image. The current set of pre-configured applications includes such standard astronomical software as IRAF, DS9, astropy, Montage, SExtractor and SWarp. It also includes the QuickReduce pipeline, a set of python routines customized to reduce and calibrate ODI-PPA observations.

The Build Service then appends any custom Docker commands the user may have entered. These commands allow the user to specify the installation of software specific to their needs. There is some potential for abuse or mis-configuration here, and our efforts to mitigate those risks are detailed in Section ???. An example DockerFile constructed by the Build Service is shown in Appendix A.

After constructing the DockerFile, the Build Service calls the Docker engine to construct an image based on that DockerFile. After a successful build, the new custom image is transferred to a private StarDock repository of Docker images. The unique ID assigned to the custom image is returned to the SCA workflow as the product of the Build Service.

2.3 Run Service

The Run Service is responsible for instantiating the Docker image as a container. It has both the Data Service and Build Service as dependencies, and so receives the output from both. In the case of the Data Service, the information required is the path to the staged input dataset. From the Build Service the Run Service obtains the ID of the Docker image constructed.

With the above dependencies handled, the Run Service constructs the Docker run command, which launches the container using the specified image. The Docker run command also specifies the directories to be mounted as volumes accessible from within the container. The staging input directory specified by the Data Service is mounted as a read-only volume in the user's home directory, and a read/write temporary directory is specified to store the user's output. The final portion of the run command is the execution of the SSH daemon, which waits for the user to connect and initiate a shell. The run command also specifies that the Docker engine should forward an unknown port on the host to any exposed ports in the container (22 in our case for SSH access).

After entering a running state, the Run Service responds to status inquiries from the SCA Workflow. These status inquiries are answered with the host name, the port that can be used for SSH access, and the status of the container.

Upon the completion of the workflow—either user-indicated or based on scheduling concerns—the Run Service receives a stop command, which results in the stoppage of the container. After confirming that the container is stopped, the output directory is compressed and stored for usage in further workflows, as well as being made available for direct download by the user.

2.4 User Interface

The user interface of StarDock is constructed within the framework of the SCA Workflow. The form for initiating a StarDock container (Figure 2) begins with the user specifying which set of data the container is meant to access. The collections of data are defined through other services, such as the ODI-PPA archive search interface.


After selecting the input dataset, the user has the opportunity to choose from a number of pre-configured software options by simply checking a box. After that there is a field for the user to enter custom Docker commands to install their own software. Finally the user must enter (and confirm) a password to restrict access to their StarDock container.

After submission, the user is presented with the Workflow status page (Figure 3) where they can monitor the process of the different StarDock service components. Each portion of the workflow can be stopped, paused, or restarted as necessary. The parameters and paths of each service component are accessible for debugging purposes.

After the Run Service enters a "Running" state, new interface components are revealed. A WebSSH terminal window provides immediate access to the container (Figure 4), and the user is given an SSH command they can execute to login via the terminal on their system. An output tab displays the contents of the mounted read/write output directory, and users can use this listing to directly download any products produced within their container.

3. SECURITY CONCERNS

Whenever outside users are allowed access to an organization's resources, security concerns must be addressed. In the case of StarDock (and Docker containers in general) the layers of abstraction between the user and the hardware can be made to act as bulwarks against destructive or malicious actors. By restricting user access to the Docker container through SSH to a single non-root account, the user is blocked from accessing any critical components within the container itself. If the user were able to elevate their privileges, they would still be restricted to the container, which has only a single read-only mounted volume and a temporary read/write mounted volume, so there is no danger of a user obtaining unauthorized access to proprietary scientific data. Outgoing traffic from the container and the physical host are both monitored for potential misuse.



StarDock

Dataset

Messier 51 (27 files, 7.3GB)

Select your applications

IRAF QuickReduce Pipeline PyRAF
 DS9 SExtractor GAIA
 astropy SWarp IDL
 Topcat Montage R/RStudio

Customize your container

Enter Docker RUN Commands

```

RUN apt-get install -y vim wget
RUN wget http://heasarc.gsfc.nasa.gov/FTP/software/fitsio/c/cfitsio3390.tar.gz
RUN tar -xzf cfitsio3390.tar.gz
RUN cd cfitsio3390
RUN /configure
RUN make
  
```

Password

Password (Confirm)

Build

Figure 2. The StarDock Workflow User Interface. Users select the dataset to process, choose applications, insert custom Docker commands, and set passwords for access to the container.

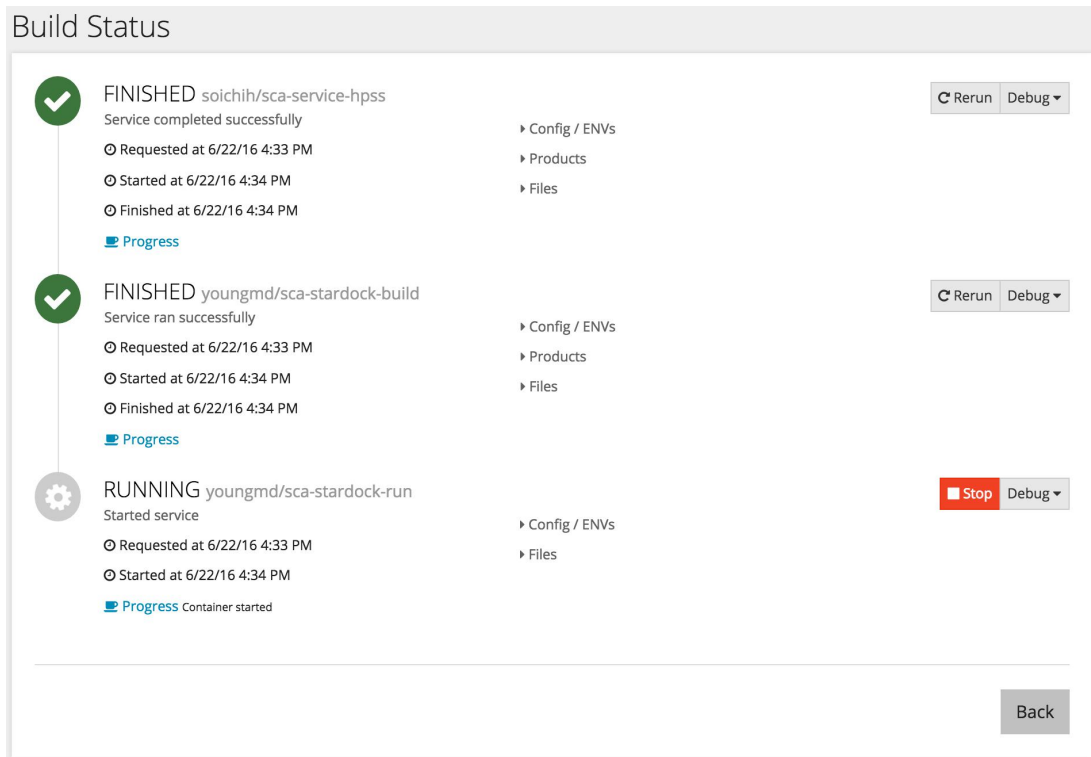


Figure 3. The StarDock Workflow Status page. In this example the SCA Data Service and StarDock Build Service have finished, and the StarDock Run service is in a running state. This status page allows access to configuration, debugging, environmental settings, and relevant system paths.

4. CONCLUSION AND FUTURE WORK

At the current time StarDock exists as a prototype, with the intention of becoming a production service within the next year. In the meantime we intend to make several improvements to StarDock to reach that stage. To reduce storage overhead we intend to use heuristics to identify similar containers and reduce the size and number of images stored in the repository. We would also like to make it possible for users to pause, store, and resume their containers. Policies must be enacted to define acceptable usage and additional computational nodes should be added as SCA Workflow resources.

The ultimate goal of StarDock and similar services that may arise is to free astronomers from worrying about where their data is, or what software is available to process it. Enabling researchers to use a remote system running next to the data as easily as their own desktop or laptop computer will have a significant impact on the astronomical community. The number of resources wasted in unnecessarily transferring data or constructing redundant computer clusters in every department must be reduced, and StarDock is a potential path towards achieving that goal.

APPENDIX A. EXAMPLE DOCKERFILE

This is an example of the type of DockerFile that would be constructed by the Build Service. In this example the software packages DS9 and IRAF are installed. The header determines the base image, which in this case is Ubuntu 16.04. The header also installs the SSH server and creates the *docker* account that the user will use to login to the system. This is followed by the commands that install the requested software, then the custom run commands entered by the user in the interface. Finally the DockerFile is appended with user access controls that assign ownership and reset the login passwords. For this example the specified passwords have been hidden.

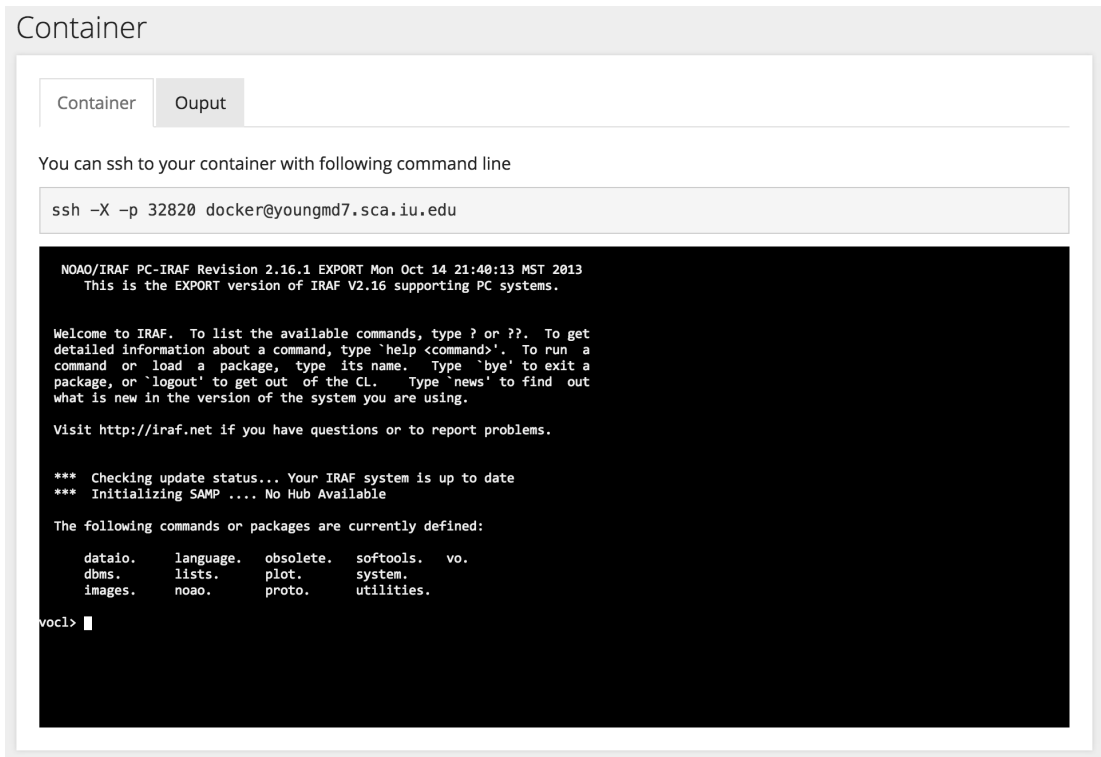


Figure 4. After the container is initialized by the StarDock Run service, the workflow interface initiates a web shell for immediate access to the container.

```
FROM ubuntu:16.04

MAINTAINER Michael Young <youngmd@iu.edu>

# install ssh
RUN apt-get update && apt-get install -y openssh-server

# add the docker user
RUN useradd -m docker
RUN usermod -s /bin/bash docker
RUN usermod -aG sudo docker
ENV HOME /home/docker

# set up ssh on port 22
RUN mkdir /var/run/sshd
EXPOSE 22
#### END STARDOCK HEADER

### IRAF BUILD
RUN apt-get update && apt-get install --no-install-recommends \\\
--fix-missing -y wget csh
#setup directories for iraf
RUN mkdir /iraf
RUN mkdir /iraf/iraf
```

```

#download and extract iraf
RUN wget ftp://iraf.noao.edu/iraf/v216/PCIX/iraf.linux.x86_64.tar.gz
RUN tar -xzf iraf.linux.x86_64.tar.gz -C /iraf/iraf
#use csh to install iraf
RUN csh
RUN cd /iraf/iraf && ./install --system
RUN exit
#cleanup iraf install
RUN rm iraf.linux.x86_64.tar.gz
# add iraf environmental entries to .bashrc
RUN echo "\nexport IRAFARCH=linux64\nexport IRAF=/iraf/iraf\n" >> \
/home/docker/.bashrc
#### END IRAF BUILD

### DS9 BUILD
RUN apt-get install -y saods9
### END DS9 BUILD

#### CUSTOM
RUN apt-get install -y vim emacs
### END CUSTOM

### STARDOCK FOOTER
RUN echo "docker:*****" | chpasswd
RUN chown -R docker:docker /home/docker
#### END STARDOCK FOOTER

```

REFERENCES

- [1] Young, M. D., Kotulla, R., Gopu, A., and Liu, W., "Integrating the ODI-PPA scientific gateway with the QuickReduce pipeline for on-demand processing," in [*Software and Cyberinfrastructure for Astronomy III*], *SPIE* **9152**, 91522U (July 2014).
- [2] Gopu, A., Kotulla, R., Young, M. D., Hayashi, S., Harbeck, D., Liu, W., and Henschel, R., "Rapid Large Scale Reprocessing of the ODI Archive using the QuickReduce Pipeline," in [*Astronomical Data Analysis Software and Systems XXIV (ADASS XXIV)*], Taylor, A. R. and Rosolowsky, E., eds., *Astronomical Society of the Pacific Conference Series* **495**, 53 (Sept. 2015).
- [3] Gopu, A., Hayashi, S., Young, M. D., Harbeck, D. R., Boroson, T., Liu, W., Kotulla, R., Shaw, R., Henschel, R., Rajagopal, J., Stobie, E., Knezek, P., Martin, R. P., and Archbold, K., "ODI - Portal, Pipeline, and Archive (ODI-PPA): a web-based astronomical compute archive, visualization, and analysis service," in [*Software and Cyberinfrastructure for Astronomy III*], *SPIE* **9152**, 91520E (July 2014).