Astronomy and Computing 13 (2015) 50–57

Contents lists available at ScienceDirect

Astronomy and Computing

journal homepage: www.elsevier.com/locate/ascom

Full length article

Toyz: A framework for scientific analysis of large datasets and astronomical images[☆]

F. Moolekamp[*], E. Mamajek

Department of Physics & Astronomy, University of Rochester, Rochester, NY, 14627-0171, USA

ARTICLE INFO

Article history:
Received 30 June 2015
Accepted 1 October 2015
Available online 26 October 2015

Keywords:
Big data
Visualization
Python
HTML5
Web application

ABSTRACT

As the size of images and data products derived from astronomical data continues to increase, new tools are needed to visualize and interact with that data in a meaningful way. Motivated by our own astronomical images taken with the Dark Energy Camera (DECam) we present *Toyz*, an open source Python package for viewing and analyzing images and data stored on a remote server or cluster. Users connect to the *Toyz* web application via a web browser, making it a convenient tool for students to visualize and interact with astronomical data without having to install any software on their local machines. In addition it provides researchers with an easy-to-use tool that allows them to browse the files on a server and quickly view very large images (>2 Gb) taken with DECam and other cameras with a large FOV and create their own visualization tools that can be added on as extensions to the default Toyz framework.

1. Introduction

In the past, large scientific datasets were used mainly by large collaborations while independent researchers worked with much more manageable volumes of data. Over the past few years we have been entering a new paradigm where very large sets of data are available to (and at times even generated by) much smaller groups. This abundance of data has highlighted a shortage of scientific tools to store, organize, analyze, and visualize that data. Fortunately this problem overlaps with the needs of the industrial community at large and in the past decade there has been a lot of work by traditional scientists, data scientists, and software engineers to develop software to aid researchers in dealing with this new (and rewarding) problem.

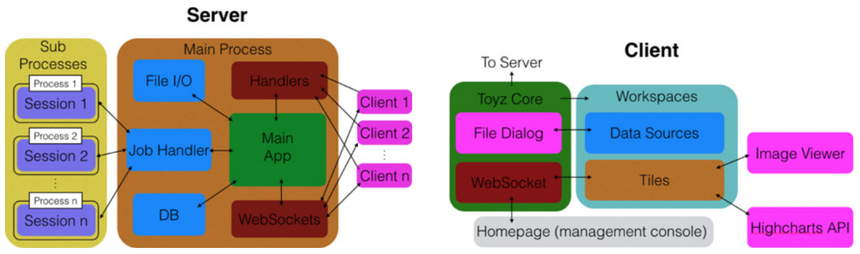
Unfortunately much of the current work in astronomy is often on the fringe of what is possible and has been done before, meaning the types of data we work with poses new challenges, which in turn create a need for new tools (Merenyi, 2014; Gopu et al., 2014; Lins et al., 2013; Loebman et al., 2014; Federl et al., 2012, 2011). Ideally these new tools should be built on existing frameworks that are under active development by software engineers to minimize the effort from research scientists while taking advantage of the latest technologies and updates to existing codes. The Python language has become a fertile ground for rapid software development and with the creation of a vast array of modules for scientific image and data processing like *numpy* (van der Walt et al., 2011), *scipy* (Jones et al., 2001), *pandas* (McKinney, 2010) and *scikit-image* (van der Walt et al., 2014); machine learning modules like *scikit-learn* (Pedregosa et al., 2011); statistics and modeling packages like *scikits-statsmodels*, *pymc* and *emcee* (Foreman-Mackey et al., 2013), and what has become the de facto astronomy python project *astropy* (Astropy Collaboration et al., 2013) and its affiliated packages.

While many of the tools listed above are useful for astronomers, data scientists, and software engineers; there is a great divergence when it comes to tools for visualization. Much of the interactivity and visualization work done in the realm of data science and software development tends to be focused on web frameworks like *jQuery Ui*, *Highcharts*, *D3.js* and even more advanced libraries using webGL like *PhiloGL*, *pathGL* and many others; or R libraries like *ggplot2* (Wickham, 2009). Contrast this with astronomy where programs like *ds9* (Joye and Mandel, 2003) that are used primarily by astronomers with few updates and changes over the past decade. Several recent python packages have been created to help bridge the gap between professional visualization tools and those available in astronomy: *GLUE* (Beaumont et al., 2014) provides a rich GUI for interacting with datasets and images and *Ginga* is one of the most advanced frameworks for viewing and interacting with FITS images.

The disadvantage of using any of the visualization tools in astronomy mentioned above is that to run efficiently all of them

☆ This code is registered at the ASCL with the code entry ascl:1507.006.
* Corresponding author.
E-mail address: fmooleka@pas.rochester.edu (F. Moolekamp).

http://dx.doi.org/10.1016/j.ascom.2015.10.001
2213-1337/© 2015 Elsevier B.V. All rights reserved.



(a) Server architecture. (b) Client architecture.

Fig. 1. Basic architecture of the python web application (server) and HTML5 client. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

must be run on a local machine with data stored locally. With new instruments like the Dark Energy Camera (DECam) that create 2 Gb images (0.5 Gb compressed) and over 1Tb of data products per night (Valdes and Gruendl, 2014; Flaugher et al., 2012), it is no longer feasible to store an entire observing run (or even a single night) on a laptop or PC. Recognizing the need for a server side image viewer several groups have been independently developing web applications to serve images from a remote server to a client with only a web browser installed including *VisiOmatic* (Bertin et al., 2015), *Data Labs* (Fitzpatrick et al., 2014), and now *Toyz*. *VisiOmatic* is an open source web application running on an Apache web server with an IIPImage (Pillay, 2014) server to display large images in a browser using a so called “slippy map” implementation (similar to Google maps). In addition to viewing images, the *VisiOmatic* client also enables users to interact with the image including marking point sources and plotting slices of the data. One of the few drawbacks to *VisiOmatic* is its dependency on libraries and applications not usually run by astronomers including a web server (that must be configured). It also provides no native support for a file dialog (as web browsers do not include a tool to browse a remote file directory), causing users to write one themselves or create a webpage on the server for *each* image they want to view.

When we first began to analyze our own DECam images, which took up over 1 Tb of disk space on our server, we realized that in order to view the images and analyze the catalogs we created with them that we would need a new tool, preferably one that could run on the server storing the data and allow a platform independent way for users to connect to the data and interact with it. This was our initial motivation for creating a new python package called *Toyz*, which seeks to combine the best of all of the software discussed so far: the remote image viewing of *VisiOmatic*, the interactivity of *GLUE* and *Ginga*, the astronomical tools of *astropy*, and the convenience of doing it all in a single framework built on existing Python and HTML5 software maintained by computer scientists. Because *Toyz* is a framework, not an application, it is designed to be easily customized by end users for their specific scientific needs but easy enough to use that a class of undergraduates could use it for analyzing their data without having to install any software on their home computers. One of the guiding principles of *Toyz* is that an undergraduate student should be able to install *Toyz* and begin analyzing data on his/her first day!

In this paper we highlight the various functions of *Toyz*. Section 2 describes the core *Toyz* package that allows users to view images and interactive plots in their browsers, Section 3 describes the affiliated package *Astro-Toyz* that incorporates astronomy specific tools including WCS and interactive tools for the image viewer, and Section 4 describes future plans for integrating *Toyz* with other software packages.

2. Toyz

Toyz can be thought of as a platform-independent tool for visualizing and interacting with large images or catalogs of data. Instead of trying to create a one-size-fits-all application, *Toyz* is designed to be an open source framework that scientists can customize to fit their own research needs.

A graphical representation of the server is shown in Fig. 1(a). The web application at the heart of *Toyz* is built on the Tornado web framework (Darnell, 2015), a python library originally written by FriendFeed as the backend for their social media website. User authentication is done via HTML handlers built into Tornado while most other communications between the server and client are done via WebSockets (Hickson, 2011): a bi-directional protocol that uses an HTML handshake to setup an open communication between the server and the client without the need for constant polling by the client to get the status of a job. Similar technology is used for a variety of websites and web applications including Jupyter (formerly iPython) notebooks (Ragan-Kelley et al., 2014). A separate module handling file I/O provides an API to load data from a variety of formats (see Section 2.2). The file I/O module is written to allow users to create affiliated packages or extensions that allow users to create custom classes for loading additional data types not currently supported by *Toyz* with minimal coding.

Each time a new connection is made to the server a new process called a *session* is spawned using python’s multiprocessing module. All of the variables and methods defined in a session will be stored until the user closes the browser and disconnects from the server. This environment is completely separate from the web application (which handles connections to and from the server) where the state of a user’s variables are stored for the duration of the connection. Because each session is a separate process, *Toyz* is able to take advantage of all of the processing cores on a server, meaning that if the number of processors scales with the number of simultaneous users, large classes and groups should be able to access the same *Toyz* server with little reduction in performance. To communicate with an associated client each session has a single websocket that connects to a single browser window on remote client, which can send jobs to be run on the server in the form of a JSON object specifying the python module, function and function parameters. All of the jobs sent from a client to the server are verified for authenticity and put in a queue to be run for the correct session. Once a job is completed, a response in the form of a JSON object is sent to the browser that at a minimum contains a status key (indicating whether or not the job completed successfully or encountered an error) and often additional keyword arguments generated by the function.

On the client side all communications are pushed through a single function that maintains information about the current session (a graphical representation of the client is shown in Fig. 1(b)). When initialized the user can choose how errors and warnings that might occur while running a job are handled as well as what actions to take when various types of responses are returned. A file dialog is also initialized that allows users to browse the directory tree on the server, functionality that is not incorporated into web browsers for obvious security reasons. The default homepage when a user logs onto the server is a management console that allows one to