ImageX 3.0: A Full Stack Imaging Archive Solution

Michael D. Young, Arvind Gopu, and Raymond W. Perigo

Indiana University, 2709 E 10th St, Bloomington, IN 47408, USA

ABSTRACT

Over the past several years we have faced the need to develop a number of solutions to address the challenge of archiving large-format scientific imaging data and seamlessly visualizing that data—irrespective of the image format—on a web browser. ImageX is a ground-up rewrite and synthesis of our solutions to this issue, with a goal of reducing the workload required to transition from simply storing vast amounts of scientific imaging data on disk to securely archiving and sharing that data with the world. The components that make up the ImageX service stack include a secure and scalable back-end data service optimized for providing imaging data, a pre-processor to harvest metadata and intelligently scale and store the imaging data, and a flexible and embeddable front-end visualization web application. Our latest version of the software suite called ImageX 3.0 has been designed to meet the needs of a single user running locally on their own personal computer or scaled up to provide support for the image storage and visualization needs of a modern observatory with the intention of providing a 'Push button' solution to a fully deployed solution. Each ImageX 3.0 component is provided as a Docker container, and can be rapidly and seamlessly deployed to meet demand. In this paper, we describe the ImageX architecture while demonstrating many of its features, including intelligent image scaling with adaptive histograms, load-balancing, and administrative tools. On the user-facing side we demonstrate how the ImageX 3.0 viewer can be embedded into the content of any web application, and explore the astronomy-specific features and plugins we've written into it. The ImageX service stack is fully open-sourced, and is built upon widely-supported industry standards (Node.js, Angular, etc.). Apart from being deployed as a standalone service stack, ImageX components are currently in use or expected to be deployed on: (1) the ODI-PPA portal serving astronomical images taken at the WIYN Observatory in near real-time; (2) the web portal serving microscopy images taken at the IU Electron Microscopy Center; (3) the RADY-SCA portal supporting radiology and medical imaging as well as neuroscience researchers at IU.

Keywords: ImageX, astronomy, imaging, JavaScript, portal, archive, search, metadata

1. INTRODUCTION

It is not uncommon for large amounts of scientific imaging data to remain inaccessible to the wider communities that may have interest in validating the results of previous publications or exploring new avenues of research previously overlooked. In some cases observational data is never utilized by those who collected it. In addition, funding agencies and scientific journals are putting in place data sharing requirements which may be challenging to researchers whose expertise does not lie within the domain of archival and portal systems. While large observatories and survey projects can afford to make necessary investments into the hardware and software infrastructure to distribute their data, there remain many without such resources. It is these needs that have driven the evolution of the ImageX software suite toward its 3rd major iteration. ImageX 3.0 enables those who possess valuable troves of archival images or are actively taking observational data to harvest, process, and securely share those scientific images with the world via a 'Push button' solution to a fully deployed solution.

Overall, ImageX represents an evolution and a synthesis of the conceptual and technical knowledge we have gained while developing previous archival and portal systems, including the One-Degree Imager Portal, Pipeline, and Archive (ODI-PPA[*]) [1], the IU Electron Microscopy Center Portal (EMC-SCA[†]) [2], the Spectral Archive (SpArc[‡]) [3], along with several others either still in active development or for the private use of our

Please direct any correspondence to M.D.Y. (youngmd@iu.edu)

[*]https://portal.odi.iu.edu

[†]https://portal.emcenter.iu.edu

[‡]https://sparc.sca.iu.edu

collaborators. The lessons learned from these systems include—among many others—how to structure and secure a distributed architecture, how to harvest and store metadata, and methods for presenting large scale imaging data over standard Internet connections. In two previous publications we have discussed the technology behind the ImageX Viewer and Data Service [4] and the overall design of the ImageX suite of tools [5] including its underlying components. In this paper we describe the logical progression of our efforts toward *ImageX 3.0*. We discuss in further detail our use of Docker in the development and deployment of ImageX (Section 2), how different methods of data ingestion make ImageX suitable for a variety of use cases (Section 3), and various improvements to the user interface and administrative features of ImageX (Section 4).

2. DOCKER

Our goal with ImageX 3.0 is to make a set of tools that are easily deployable in a wide variety of environments. The portability of Docker[6] containers makes them ideally suited for helping us to achieve this goal. In Ref. 7 there is a more detailed discussion of the creation of Docker images and the use of a private Docker network to secure services. The information in that paper is presented in the context of migrating a legacy project to Docker (ODI-PPA), but the methods used are applicable to any use-case, including how ImageX 3.0 components are built. An application under active development offers a different set of challenges compared to maintaining a legacy code-base, and here we will describe how we utilize Docker Compose to enable real-time development without the need to rebuild Docker images with each change. We will also discuss other advantages to using Docker Compose as opposed to managing the component containers of ImageX on an individual basis.

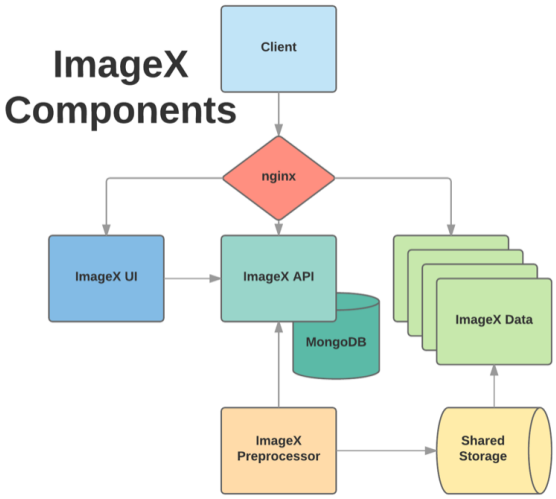

Figure 1. The components that make up the ImageX. See Ref. 5 for more details.

2.1 Docker Compose

Docker Compose is a tool designed to aid in the definition and running of multi-container applications. Within a single `docker-compose.yml` all of the containers for a given application can be defined, including their mount points, network configuration, and container dependencies. In the case of ImageX we have at least five containers running at any given time: a standard MongoDB instance, the user interface (UI) server, the application programming interface (API) server, the Data Service, and one or more data pre-processor applications (See Ref. 5 for more details). At runtime the API service requires that MongoDB be available, and both the UI and pre-processor services require that the API be available. With Docker Compose we can specify these dependencies, and at runtime the containers are initiated in the proper order. Additionally if we need to restart a service that has dependencies, Docker Compose will ensure that the dependent services are also restarted.

To illustrate how Docker Compose is useful for active development we will use the ImageX API service as an example. The API is a node.js application, and in the `Dockerfile` for the API service a number of node

modules are specified to be installed using `npm` during the build process inside the Docker Image specifically to the application directory. At runtime, using Docker Compose we can then mount the application directory on the host into the container, overwriting the application folder within the container, but leaving the installed node modules intact. This allows any changes made to the API on the host to be reflected inside the container without the time-consuming task of rebuilding the image. Issuing the restart command to the API via Docker Compose will ensure that the changes are reflected in the running instance and dependent services are properly restarted as well. In production mode it is a simple task to remove the volume mount command from the `docker-compose.yml` configuration and make the service completely self-contained, the way Docker containers are intended to be. An example `docker-compose.yml` file is included in the main code repository, and each component repository (linked from the main repository) includes the `Dockerfile` used to build the Docker image for that component.

3. DEPLOYMENT & DATA INGESTION

We have attempted to make the deployment of the ImageX 3.0 stack as streamlined a process as possible. Assuming the target system already has Docker, Docker Compose, and nginx installed, the deployment procedure is to clone the base repository[§], edit a few configuration files, and produce a public and private key-pair. After instantiating the docker containers using Docker Compose the site should be fully operational, albeit lacking entirely in data. The site administrator can then begin the process of data ingestion, the methods of which are described in subsequent sections of this paper. For a more detailed set of deployment instructions please see the `README` file on the ImageX github repository.

The ImageX 3.0 service stack offers flexibility in how data is ingested (i.e. have its metadata harvested and image tiles generated) into the system. There are three built-in methods of ingestion that can be used in conjunction or exclusively as a particular project requires. A supported scientific image identified for ingestion—regardless of the method used to make the data available to the system—is processed in several steps. The ingestion pipeline begins with a format validation: in the case of astronomical images this is a python script that verifies that the file is in the `.fits` format (`.fz` compression is also supported), and that the file header is both present and meets the format standard. The file header is then extracted into a key-value python dictionary and injected into the portal's MongoDB database via a `REST` call, along with some additional values (timestamp, file size, owner, location, etc.). After metadata extraction is complete the imaging data is extracted, scaled, converted, and tiled into image pyramids. See Ref. 4 for a more in-depth discussion of the tiling procedure. Below we discuss the three different ingestion methods and how they support various modes of operation and use cases for the ImageX 3.0 stack.

3.1 Automated Data Ingestion

The automated method of ingestion is an instance of the ImageX Preprocessor (Section 2) docker image. When initiated in Automated mode, a Python application called `Watcher` monitors a specified directory for the arrival of new files. When a new file is detected `Watcher` creates a monitoring thread which checks the file at regular intervals. When the file has remained unchanged for a set duration it is submitted to an ingestion queue which feeds files into the ingestion pipeline for further processing. The reason for this monitoring and delay on file changes is to ensure that images that are being actively read out from an instrument or arriving via data transfer are not marked for ingestion until the file is complete. This method of ingestion is ideal for supporting active observing operations, such as monitoring the output from automated instruments.

3.2 Batch Data Ingestion

The batch method of ingestion is also an instance of the ImageX Preprocessor docker image. When launching in batch mode, a root directory is specified. The directory structure is then recursively scanned for supported files and passed to the ingestion pipeline. When the ingestion queue is empty the batch ingestion script will exit and the docker container will stop itself. Multiple batch ingestion containers can be run in parallel without interference. The batch ingestion can optionally set a standardized scaling based on the first ingested image and apply that same scaling for all images in a dataset, useful for ensuring that large scale mosaics have consistent background and brightness levels. Batch ingestion is best suited for processing archival datasets.

[§]https://github.com/IUSCA/imagex

3.3 User Data Uploads

If user uploads are enabled by the administrator (Section 4.2) then—subject to quota and maximum file sizes limits—users will be able to upload science images directly through the portal. Each uploaded file is placed in a temporary directory assigned to that user. A variant of the automated ingestion system verifies that the file is safe, that the user is allowed to upload files and is within quota, and then passes the file into the ingestion pipeline, with markers to indicate the data's provenance. The user upload feature is intended to be used by small teams of collaborators with a need to share data collected on a variety of instruments with each other.

4. USER INTERFACE

The design and architecture of the ImageX user interface and image viewer are described in more detail in Ref. 5. Here we will discuss several features and improvements that have been added to the user-facing elements of ImageX since that paper was submitted.

4.1 Metadata-adaptive Search Interface

ImageX 3.0 is intended to function with images collected from any number of scientific instruments. This goal is hampered by the unfortunate reality of the severe heterogeneity of imaging metadata. Even among optical astronomers there is little consistency in how basic metadata—such as the exposure time or the telescope pointing coordinates—is to be recorded in the image header. We considered two ways to address this issue. The first was to add a homogenizing layer to the metadata ingestion process. This would involve identifying all possible variants of a given keyword and storing their values under a single preferred keyword. For example, the observation time-stamp could have been recorded in the header as any of `DATE`, `DATETIME`, `DATE-OBS`, `TIME-OBS`, `CAL-OBS`, `OBSDATE`, `MJD`, `MJD-OBS` or many others. Cataloging these variations for every potentially interesting keyword would have required an extensive amount of work, and would need to be continuously maintained. There is also the added concern that we would be imposing our own ideas of which keywords in the header are "interesting", ultimately limiting both the flexibility and usefulness of ImageX.

Instead we opted to pursue a different method of addressing the issue of metadata heterogeneity. ImageX 3.0 provides a search builder interface tied to a set of search element prototypes which can be customized by the site administrator to produce a search interface optimal and applicable to the data present. These prototypes are mapped to a specified keyword and include text, numerical range, date range, drop-down selection, and coordinate cone search UI elements. Further customizations include alternate keywords, how the element will be labeled in the interface, minimum and maximum values, and auto-complete support. Through the search builder interface the site administrator can also choose to temporarily enable or disable each search element. Figure 2 displays the search builder form and the rendering of customized search elements within the search interface.

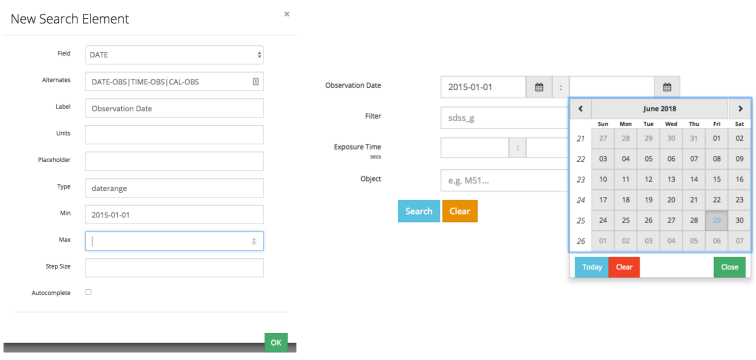

Figure 2. Left: The search builder form used to define the search element, including alternate keywords. Right: The search element rendered within the user interface.

Most relevant to the issue of metadata confusion is the support for alternate keywords. In the above example for the observation time-stamp a site administrator could declare a date range search element mapped to the

keyword of `DATE`, and then set alternate keywords of `DATETIME` and `DATE-OBS`. A search element entry is inserted into the database, and when the search page is rendered a single date selection element is shown with the specified options. When a search request is submitted to the API, it is first synthesized into a query with a logical OR with the primary and alternate keywords, producing all relevant results. This method does require that the site administrator manually enter the alternate keywords, however it has the advantage in that it does not require that any metadata be re-ingested or updated, and it leaves the decision of what metadata keywords are relevant in the appropriate hands.

4.2 Administrative Controls

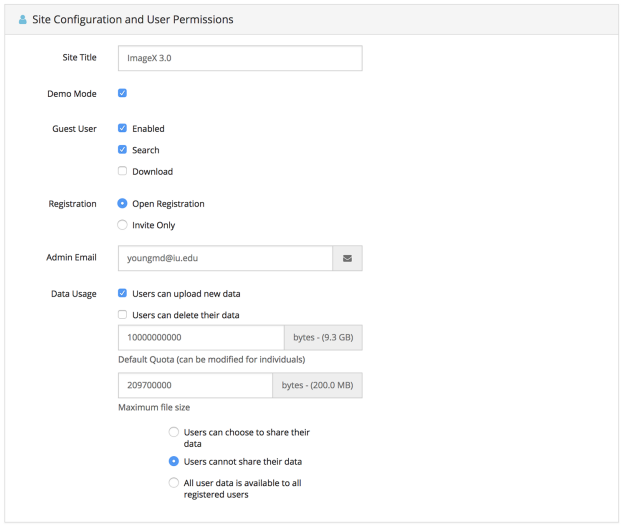

Figure 3. The administrative control panel allowing for ImageX Portal customization.

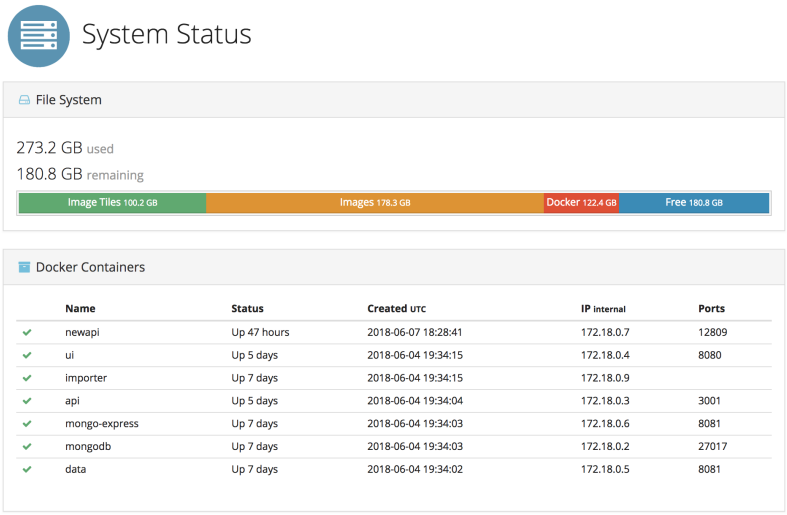

Figure 4. The system monitor tool, showing status of each docker container + file system usage information.

ImageX 3.0 is intended as a push-button method of securely distributing scientific data, and those who choose to deploy it must have control over how it will be utilized, with little or no computational or software service operational expertise. To that end we have created a set of default configuration options that dictate basic site

functionality and exposed those options via an administrative control panel in the UI. This interface can be used to allow or deny guest login and downloads, whether or not users can upload data, if they are allowed to share or remove their data, and what file size and quota restrictions will be enforced. Administrators can also choose to allow open site registrations, or restrict access only to invited users. Additional site configurations are supported, and Figure 3 illustrates the full set of controls given to ImageX site administrators.

Other administrative tools include a user control panel which displays active user accounts and their data usage, and a monitoring tool which tabulates the disk usage of docker containers, scientific images, and image tile pyramids. The current status of each docker container (Section 2) within the ImageX stack is also made available (See Figure 4).

4.3 Image Alignment

One of the issues we faced in previous iterations of ImageX was proper image alignment within the Viewer. The coordinate system of the OpenSeaDragon [8] (OSD) library that the ImageX Viewer uses is based upon the top-left corner of the first placed image, and each image's placement is also defined by the top-left corner. To place subsequent images we first generated a global WCS based on the header of the first placed image, then identified the physical (`RA`,`DEC`) coordinates of the top-left corner of the new image and converted that into the image coordinates of the original image. This gave us a roughly accurate `x,y` offset to place the top-left corner of the new image. This method does not account for rotation and sky curvature however, and large scale mosaics quickly showed significant deviations from optimal placement.

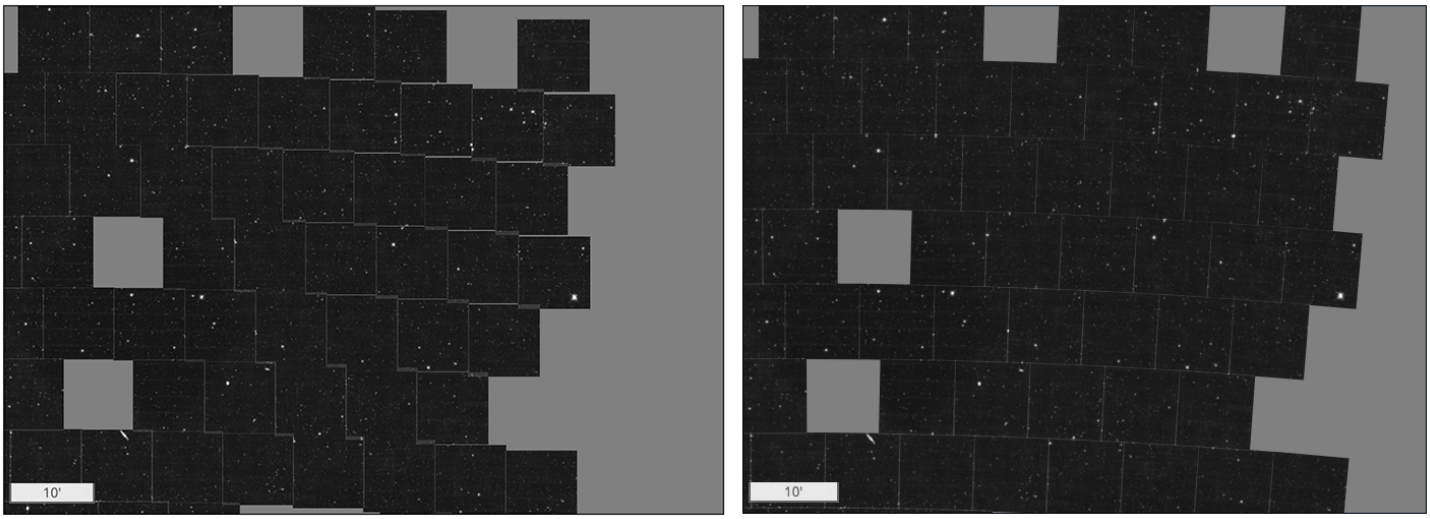

Figure 5. Example of our improved WCS alignment using a subset of the Dark Energy Survey field images and spanning multiple degrees. The gaps in the shown coverage are intentional. Left: No rotation Right: Rotation enabled.

To correct for this, we made use of OSD's support for image rotation about the central point. Consider the case of Images `A` and `B`. We first place the top-left corner of `A` at the origin and assign it a rotation angle of 0. To place Image `B` we first construct a WCS based on the header of `A`, then find the central `RA` and `DEC` of `B`. After converting that central point to the image coordinates of `A` we calculate the offset for an uncorrected top-left corner that would allow the central point to be placed correctly. To determine the rotation we find the uncorrected and corrected image coordinates of each corner of `B`, then determine the angle of a vector from the central point to each. The difference in these angles is our rotation value for that corner, and we use the mean result from all four corners as our final rotation value. Appendix A shows the JavaScript code used within the ImageX 3.0 Viewer module to align multiple images, and Figure 5 illustrates the use of our improved alignment code to successfully display a mosaic of Dark Energy Survey [9] images. At several degrees distance the alignment is accurate within the precision allowed by the WCS solution found in the initial image.

5. CONCLUSIONS & FUTURE WORK

Although ImageX was developed with astronomical data in mind it is by no means our intent that it be limited to only that particular field of research. We have developed proof of concepts that demonstrate that the ImageX set of tools can be applied to the data formats used in electron microscopy (`.dm3`,`.dm4`), Geographical Information Systems (GIS, `GeoTIFF`), and radiology (`DICOM`). In the future we will seek to partner with researchers and archivists working in these fields and expand the feature set and adaptability of ImageX.

We are working towards widening that scope of ImageX Viewer functionality by implementing a Jupyter [10] plugin to enable real-time visualization of the remote processing of large-format images. We have also begun to investigate the feasibility of adding support for HEALPix [11] coordinates and the Hierarchical Progress Survey (HiPS [12]) tile formats, enabling ImageX to be deployed against a large number of existing astronomical datasets.

ImageX represents a distillation of the techniques and lessons learned over several years of working to ensure that scientific data products are securely stored and available to researchers around the world. The code of ImageX is open source and freely available and we welcome input and contributions from the community.

APPENDIX A. WCS ALIGNMENT CODE

```
if($scope.wcs == null) {
    //no wcs found yet, place at origin with no rotation
    $scope.wcs = new WCS.Mapper(image.header);
    var center = $scope.wcs.pixelToCoordinate([image.width / 2.0, image.height / 2.0]);
} else {
    var rad2deg = 180.0/Math.PI;
    var tmp_wcs = new WCS.Mapper(image.header);
    var center = tmp_wcs.pixelToCoordinate([image.width / 2.0, image.height / 2.0]);

    //center position on master wcs
    var center_xy = $scope.wcs.coordinateToPixel(center.ra, center.dec);

    //center shift from image coords to master coords
    var center_offsets = {x: (image.width / 2.0) - center_xy.x, y: (image.height / 2.0) - center_xy.y};
    var c0_real = $scope.wcs.coordinateToPixel(image.ra0, image.dec0);
    var c =  $scope.wcs.coordinateToPixel(center.ra, center.dec);
    var c0 =  $scope.wcs.coordinateToPixel(center.ra, center.dec);

    var rots = [];
    image.corners.forEach(function(cor){
        //this is where the corner should be on "master" wcs
        var cor0 = $scope.wcs.coordinateToPixel(cor[0], cor[1]);

        //corner location on local wcs
        var _c = tmp_wcs.coordinateToPixel(cor[0], cor[1]);
        var xoff = _c.x - image.width / 2.0;
        var yoff = _c.y - image.height / 2.0;

        //this is where the corner actually gets placed
        var cor1 = {x: center_xy.x + xoff, y: center_xy.y + yoff}

        var opp1 = center_xy.y - cor1.y;
        var adj1 = center_xy.x - cor1.x;
        var opp2 = center_xy.y - cor0.y;
        var adj2 = center_xy.x - cor0.x;
```

```
36
37          //rotation angle is difference between placed and true corner
38          var rot = Math.atan(opp1 / adj1) - Math.atan(opp2 / adj2);
39          rots.push(rot * rad2deg);
40      });
41    //calculate mean rotation value for all corners
42      rot = rots.reduce((a,b) => (a+b)) / rots.length;
43  }
```

Listing 1. JavaScript code within the UI AngularJS application, used to determine image rotation relative to a master WCS for the ImageX Viewer

ACKNOWLEDGMENTS

We want to express our profound gratitude to all the open source developers who have contributed to Docker, Docker Compose, nginx, OpenSeaDragon, and the many other open source libraries and applications that we have utilized in this project.

REFERENCES

[1] 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," (2014).

[2] Morgan, D., Gopu, A., Young, M. D., and Hayashi, S., "EMC-SCA: Organizing, Managing and Searching Large Collections of Images: A New Resource To Handle High-Throughput Imaging," (2014).

[3] Pilachowski, C., Hinkle, K., Brokaw, H., Young, M. D., and Gopu, A., "Preservation of 20 Years of Spectrographic Data ," (2016).

[4] Hayashi, S., Gopu, A., Kotulla, R., and Young, M. D., "ImageX: New and Improved Image Explorer for Astronomical Images and Beyond.," (2016).

[5] Young, M. D., Gopu, A., Perigo, R. W., and Hayashi, S., "Image Archives Made Easy with Docker and ImageX," *TBD* **TBD** (2017).

[6] Merkel, D., "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.* **2014** (Mar. 2014).

[7] Perigo, R. W., Gopu, A., Young, M. D., and Bao, Y., "Toward sustainable deployment of distributed services on the cloud: dockerized ODI-PPA on Jetstream," *Proc. SPIE* **In progress** (2018).

[8] "OpenSeaDragon ." https://openseadragon.github.io/.

[9] Abbott, T. M. C., Abdalla, F. B., Allam, S., Amara, A., Annis, J., Asorey, J., and et al., "The Dark Energy Survey Data Release 1," *ArXiv e-prints* (Jan. 2018).

[10] "Project Jupyter." http://jupyter.org/.

[11] Górski, K. M., Hivon, E., Banday, A. J., Wandelt, B. D., Hansen, F. K., Reinecke, M., and Bartelmann, M., "HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere," *ApJ* **622**, 759–771 (Apr. 2005).

[12] Fernique, P., Allen, M. G., Boch, T., Oberto, A., Pineau, F.-X., Durand, D., Bot, C., Cambrésy, L., Derriere, S., Genova, F., and Bonnarel, F., "Hierarchical progressive surveys. Multi-resolution HEALPix data structures for astronomical images, catalogues, and 3-dimensional data cubes," *A&A* **578**, A114 (June 2015).