# Accelerating adoption of OME Remote Objects, a microscopy image repository

Sammy Nguyen

## Abstract

To share microscope imaging data, metadata must also be shared to give full context around an image. The Open Microscopy Environment has developed a centralized repository, OME Remote Objects (OMERO), to facilitate sharing of microscopy data, but this software can be difficult and time-consuming to set-up without extensive computer system administration knowledge. To speed up adoption of OMERO, I explored two container technologies, Docker and Singularity, to vastly simplify the deployment of OMERO. Both technologies enable research groups to quickly instantiate OMERO instances, saving hours of work, though Singularity's simplicity make it the preferred choice for initial setups. In the future, further work can be done to make advanced features easier to configure and to improve training on the actual use of OMERO.

## Introduction

An important problem in science is the method by which researchers share their data. Making data available to others greatly increases the reproducibility of science, engendering greater trust in the work others have done. With microscope imaging data in particular, it is not sufficient to only share the raw images themselves -- without context on what the images contain and the conditions under which they were obtained, the image metadata, researchers viewing the shared data cannot accurately evaluate the results of an experiment. Even if the metadata is provided, it can often be in proprietary formats that cannot be viewed without the appropriate tools.

To solve these problems in microscopy imaging, the Open Microscopy Environment (OME), a consortium of research groups across the US and Europe, created a data model to store imaging metadata and a database system for organizing and sharing microscopy images. With well over a decade of experience working to address these problems around sharing microscopy data, new adopters of OME's tools can have confidence that many potential use cases have been considered for their tools.[1]

However, potential adopters may have difficulty approaching the migration to OME's tools. In particular, OME's database system, OME Remote Objects, or OMERO, can be quite difficult to set up, due to its extensive installation and configuration requirements. To enable research groups to transition to the use of OMERO, this work will present an overview of OME's tools to demonstrate their value and two methods for a simple deployment of OMERO that require minimal up-front knowledge of computer system administration.

### *OME Data Model*

To facilitate the sharing of microscopy data, OME has developed a data model, the OME Data Model, that captures metadata associated with a microscope image. The data model defines many semantic types that concretely categorize much of the metadata that give context to a microscope image. Figure 1 shows a hierarchical organization of some of these semantic types, with higher level concepts on the left, and descendant types on the right. Descendant data types describe more granular details about the parent data type.

The data model provides organizational data types that allow researchers to easily keep track of information applicable to all images. These organizational data types describe information around the members of the lab (`Experimenter` and `ExperimenterGroup`), equipment shared in the lab (`Instrument`), and projects a lab works on (`Experiment` and `Project`). Note that these can have descendent data types -- for example, `Instrument` is designed as a wrapper around the various instrument component types, including `Microscope` and `LightSource`.[2]

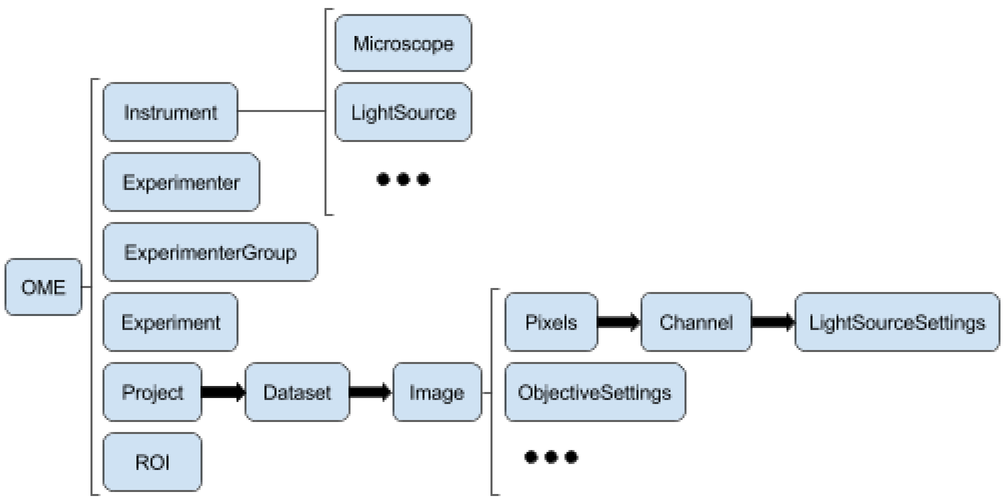

Figure 1) A simplified version of the OME Data Model. Various high level concepts appear on the left, with more granular detail appearing in descendent elements moving to the right.

The bulk of the complexity in the OME Data Model comes with the metadata directly associated with an image. In microscopy images, there are five dimensions to consider: height, width, depth, time, and channel. Of particular interest is channel, because channels are used to distinguish different features in the image, and intensity of a pixel value in a particular channel can hold biological significance. Thus, much of the metadata in an image relates to the information around how a researcher generated the channel image. Other metadata captured include details such as microscope settings, environmental readings, and regions of interest (ROIs) within an image. For custom use cases between different research groups, the data model also supports free-form text descriptions on most data types, as well as more structured annotations.

To make the model machine-readable, OME implemented its data model in Extensible Markup Language (XML) as OME-XML, allowing for validation of data stored in the OME-XML format and easy interchange of files between different software products. Because of the tightly defined data model, users can be sure that all software products will interpret the data in the OME-XML format in the same way. XML parsers and other tools to work with general XML files are readily available, so users can take advantage of these tools to manage their OME-XML. Another benefit of the XML format is that it also allows users to open OME-XML files with a simple text editor as a last resort.

## *OME-TIFF*

Taking metadata storage one step further, OME realized that image pixel data and metadata could be contained in a single file. TIFF (Tagged Image File Format) files have headers which can contain metadata about the file. In photography, this typically would include information such as the camera used, its settings, or the date the photo was taken. In OME-TIFF, the header instead contains OME-XML. Additionally, OME-TIFF can take advantage of the capability of TIFF to store multiple images in a single file. This is especially useful in microscopy because a single experiment often requires capturing images of a plate at different depths, channels, and timepoints. However, frames can still be stored in separate images if desired. Keeping frames in separate files means metadata can still be recovered if some images in the dataset are lost.[3]

## *OMERO*

By leveraging the OME Data Model, OME created OMERO, a server-based system used to manage and share imaging datasets. OMERO serves as a central database, using the OME Data Model as the underlying schema for the database, that users can connect to over the Internet to interact with their datasets. Most users interact with the system through either a desktop interface, OMERO.insight, or a web application, OMERO.web. Using these interfaces, image datasets can be uploaded, shared with other users, explored with a 5D image viewer, and commented or annotated for future reference. With the correct configuration, datasets can also be made publicly available, useful for data associated with published work.

Additional methods of connecting to OMERO allow users to minimize changes to their workflows. With programs like ImageJ[4] that support plugins, users can perform tasks as normal in the original program, and then save their work to the OMERO database through an extension in that program. OMERO also provides a command-line interface and application programming interfaces (APIs) in Python, Java, and MATLAB. These non-graphical interfaces can be used to batch process large amounts of data and access more advanced features of OMERO not exposed in OMERO.insight or OMERO.web.[5]

## *The OME community*

The OME Data Model and OME-TIFF have broad use across the scientific community. Equipment manufacturers Carl Zeiss, GE, Leica, and others, and commercial software products such as Huygens and Imaris work with OME to support OME-TIFF and to help OME develop converters of their proprietary file formats to OME-TIFF.[6] Several open source software program including some used in the Covert lab, like Micro-Manager[7], ImageJ, and CellProfiler[8], also support OME-TIFF.

OMERO has also been implemented within many groups worldwide. As of January 2012, about 3,200 OMERO installations, with about 500 in active use, were known to the OME. High profile groups using OMERO include the Allen Institute for Cell Science[9], Harvard Medical School[9], and Lawrence Berkeley National Laboratory[10], all of which have presented at annual user meetings in recent years. Additional community support comes in the form of forums and mailing lists users can join, and a variety of training manuals and user guides that can be consulted for assistance. With this broad third-party backing, potential adopters can be certain that the OME Data Model, OME-TIFF, and OMERO will be maintained for some time to come.

## Methods

OMERO is difficult to deploy for those unfamiliar with system administration tasks due to its many constituent parts. Three major pain points exist in the deployment process. First, OMERO has many prerequisite libraries and services that must be installed onto the operating system. Second, the system requires extensive configuration. OMERO itself and a PostgreSQL[11] database must be configured correctly. If users wish to access OMERO through the web interface, nginx[12], a reverse proxy server, must also be configured to retrieve static HTML, images, and JavaScript and act as an intermediary between the user and the OMERO.web server service. Third, each of component of OMERO must be coordinated to start in the correct order, otherwise the entire system could fail to start correctly. Simplification of this setup process would enable research groups to adopt OMERO more quickly.

To implement a single-command basic deployment of OMERO, I investigated the use of container technologies to eliminate the need for deep system administration knowledge. At a high level, containers are lightweight virtual machines (VMs), allowing users to essentially run a full operating system (OS) on top of a host operating system. Within a container, any required programs or libraries can be installed with no impact to the host OS. Containers can also be pre-built with a script-like file, and shared with others once they are built. To use a new tool, if a containerized version of the tool exists, users can simply download a container image and start working without having to navigate a complex setup process.

### *Docker*

I investigated Docker, one of the most commonly used container technologies in industry today, for the initial implementation of OMERO. Docker containers provide many of the same isolation features as a regular VM, separating subsystems like processes and networking inside the container from those outside the container, while still allowing specified files from the host to be accessible from inside the container. Docker containers also can be constrained to use a limited amount of the host's resources, so that applications running inside a container do not monopolize all of the system's computing power. Another key feature of Docker is the concept of image layers, which allows for "image inheritance." This feature allows users to build small base images and extend them as they install other dependencies on top of the base images. Sharing of base images, in addition to reducing build time, also reduces the space taken by each container, as the data in a single image layer is shared between containers using that layer.[13]

Best practices for running Docker containers hold that individual components of a system should be isolated into different containers. This modularity allows maintainers to easily configure and upgrade a single component of the system while minimizing changes to the other parts.[14] However, container modularity also requires somewhat complex interconnection between containers, both for sharing files and for direct communication through ports. To simplify this orchestration, Docker provides a tool called Docker Compose for administrators to configure how multiple containers will run in concert.[15]

Container images for major OSs and commonly used programs are also easily shared and downloaded. Docker runs a service, Docker Hub, where users can upload images for others to use. Docker Hub also provides several automation tools to allow image maintainers to easily create updated images if the build instructions or dependencies are changed. If desired, private registries of images can also be kept using Docker Registry. Images can be quickly downloaded by executing a `docker pull` command in a shell, or by building an image that inherits from an existing image in Docker Hub[16], or in an accessible private registry.[17]

### *Singularity*

The computing clusters at Stanford available to the Covert lab, Sherlock and covertlab.stanford.edu, are currently running versions of Linux that do not meet the prerequisites for running Docker. Even though it is not possible to support Docker, administrators of the Sherlock high performance computing (HPC) cluster realize the utility of a container system for running applications. To support this need, Sherlock currently supports a new container system called Singularity.[18] Built by Gregory Kurtzer, a software developer at Lawrence Berkeley, and other open source contributors, Singularity focuses on enabling the mobility of applications and workflows to different system environments, what is referred to as the “mobility of compute.”

Unlike Docker, Singularity does not attempt to replicate all of the isolation features of a VM, because these features are not essential to the portability of applications. For instance, the network inside a Singularity container is the same as the host network, while the network inside a Docker container is isolated from the host network. Singularity’s architecture also means that resources dedicated to running a Singularity container cannot be constrained, unlike with Docker containers. However, this missing feature is not a concern in HPC environments, because they usually handle resource allocation as part of their job submission processes. Users within a Singularity container are also handled differently -- the host username launching the container will be the username inside the container, and escalation to root from within the container is prohibited.

To build an image, Singularity also can run a script-like file similar to Docker. However, Singularity’s architectural differences with Docker mean that there are three main constraints that image builders need to be aware of. First, images must be built as the root user, since installs require root permissions. Second, broad permissions may need to be granted on certain root-owned files or directories, depending on an installed application’s needs. Since containers are typically run as non-root, a running process within the container may need permissions to access root-owned files. Finally, image builders should also note that image build times are much longer and image sizes are much larger than with Docker, due to the lack of layering and caching in Singularity’s image build architecture. Because of this, it is no longer practical to build separate images for each running process, like a Docker user might do.

## Results

### *Docker implementation*

An existing implementation of OMERO running on Docker containers is available on GitHub, developed by Hadrien Mary, a researcher at McGill University, and further improved by myself.[19]

Figure 2a depicts the architecture of the Docker images used to run OMERO. A layer containing dependencies and the OMERO binaries, based off a public `ubuntu:14.04` image from Docker Hub, forms the base of the images for `omero-data`, `omero-init-db`, `omero-server`, and `omero-web`. A public `postgres:9.4` image (not shown) is used for the required PostgreSQL server, `omero-pg`. Note that `omero-web` deviates from Docker best practices by having both the OMERO.web service and the nginx proxy server run within the container, orchestrated by a supervisor[20] service. This deviation is required because OMERO.web generates static files served by the nginx proxy server, creating a tight coupling between the two services.

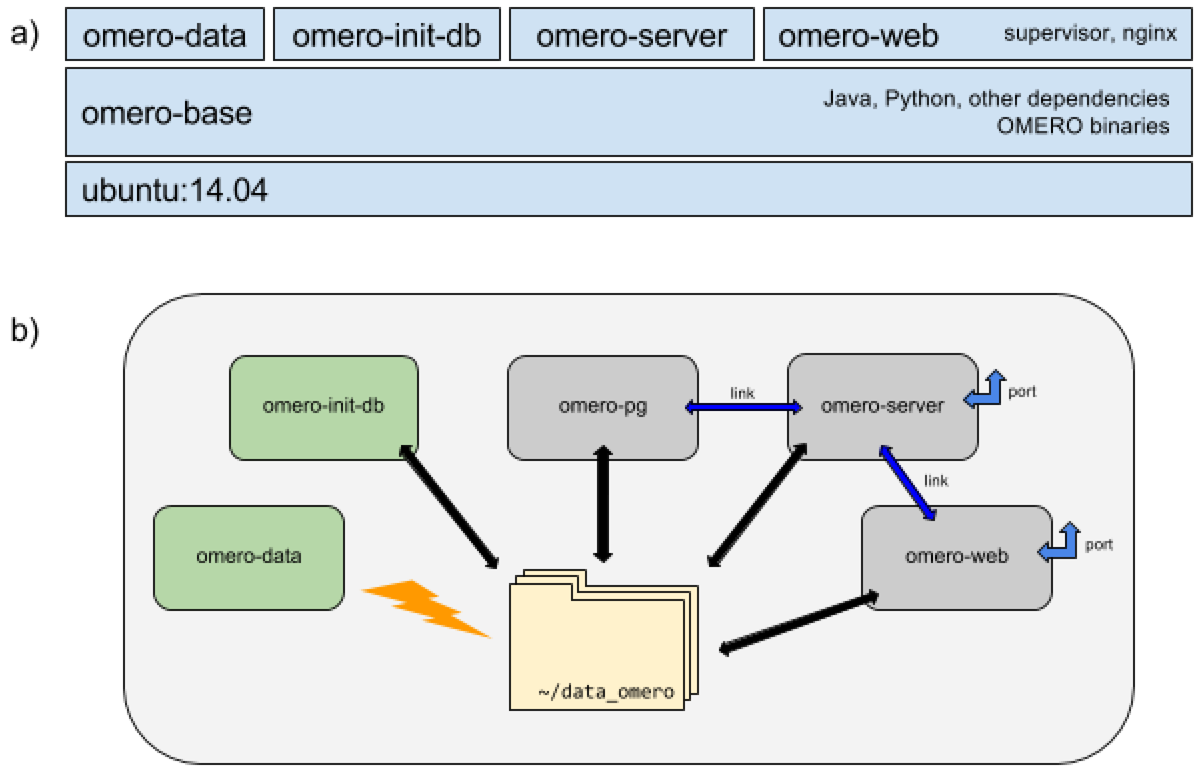

Figure 2) Containerizing OMERO with Docker. (a) OMERO images are based on a public Ubuntu image and a base OMERO image with all dependencies installed. Main OMERO images then have slight customizations of the base OMERO image to perform their specific tasks, and additional dependencies as needed. (b) Containers' setup is coordinated using Docker Compose. Containers start in clockwise order, starting with `omero-data`, which creates folders on the host if necessary. Each of the other containers mounts folders from the host to read and write data. Other required configuration with Docker Compose is the links to allow for communication between containers and the opened ports to allow communication with outside users.

To run OMERO in Docker, I orchestrated the launch of the containers using Docker Compose (Figure 2b). Two transiently running containers are launched first: `omero-data`, which creates a shared directory on the host where OMERO can store its data, and `omero-init-db`, which creates a database initialization script (stored in the aforementioned shared directory) if it notices this is the very first run of OMERO. Once those containers have completed their tasks, the three application containers `omero-pg`, `omero-server`, and `omero-web` can then launch. Using Docker Compose, I specified links between the three application containers to allow for required communication between the services, as well as connection endpoints to allow end users to use the OMERO instance. This work and further documentation is available at https://github.com/smnguyen/docker-omero/tree/dev. Although this method of deployment cannot be used on the Covert lab's current resources, this work can still be useful to those able to run Docker containers.

*Singularity implementation*

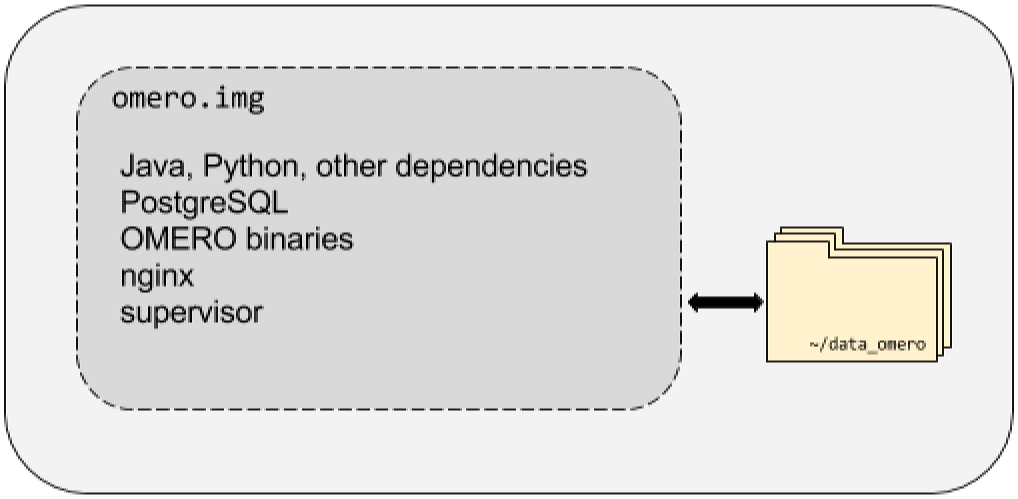

Figure 3) Containerizing OMERO with Singularity. With Singularity, all components required to run OMERO are installed within a single container. Directories on the host are mounted onto the container to allow easy viewing of written data and logs. Communication with processes inside the container does not need to be configured and can occur unimpeded by Singularity itself.

Taking the discussed constraints on Singularity images into account, I constructed a single Singularity image instead of several images to run the OMERO software (Figure 3). The Singularity image contains an installation of the main services OMERO server, PostgreSQL, and nginx, and supervisor to start each of those services simultaneously. Initialization scripts, start scripts, and configuration files are kept on the host and mounted onto the image to actually run OMERO. This allows users unfamiliar with Singularity to easily configure OMERO, simply by modifying files on the host. Like with the Docker setup, additional directories are created on the host and mounted on the image so that OMERO can store image files, write to the database, and keep logs of user actions and errors, while allowing for easy backup of the system state by administrators. This work and additional documentation can be found at https://github.com/smnguyen/singularity-omero.

*Utility scripts*

To aid in the adoption of OMERO, I implemented two utility scripts to aid in moving existing data sets onto OMERO. First, I created a Fiji[21] script that converts sets of PNG image files into OME-TIFF files. Microscopy data in the Covert lab is organized by the researcher at the top level, the date of capture at the second level, and the microscope position at the lowest level. Within the lowest level, images are stored for each channel, time point, and depth at that position -- this means that images in a single position folder can be converted into a single OME-TIFF file. To perform this conversion, my script simply takes a directory as an input parameter and saves an OME-TIFF file with the combined images from the input directory. The resulting OME-TIFF files can then be uploaded, viewed, and annotated in OMERO.

The second tool I built adds ROIs to images already existing in OMERO. A common analysis step for microscopy images is the segmentation and tracking of cells in time-lapse images. To perform this analysis, the Covert lab has a Python tool, covertrack, that applies various algorithms to images to segment and track cells, outputting image masks with the results of the analysis.[22] I developed a Jupyter[23] notebook file that, when given an image in OMERO and a directory with image masks, converts the masks to ROIs as expected by the OME Data Model and uploads the ROIs to OMERO. Each ROI corresponds to a single cell tracked across all the frames in which the cell is visible. Using these scripts, potential users can easily migrate their data into OMERO and begin taking advantage of its many capabilities.

## Discussion

### *Evaluation of container technologies*

Overall, container technologies greatly reduce the setup time required to get started with OMERO. Although determining how to build containers for an application correctly does take a significant amount of up-front effort, this investment means that many people can quickly benefit from an image builder's work. Lab groups wishing to adopt OMERO could save nearly 150 hours of work, the time used to build and test the images, by using the images from this work instead of learning how to install and configure OMERO on their own.

Research groups will need to evaluate which container technology is best for them. The use of Docker to containerize OMERO offers many of the benefits of Docker. One of Docker's more valuable features -- its speed in building images -- is aimed more for image developers. The caching and layering of images Docker provides allows builders to make small changes to image build scripts and rebuild the image very quickly. Docker's orchestration tools would also allow OMERO to scale out to serve more users very easily. OMERO is capable of running in a distributed fashion, with a single "master" process and several "worker" processes, and this distribution would be well supported by Docker's toolset. Additionally, its widespread application means a large community is available for quick support and advice.

Despite Docker's benefits, some of its features and requirements make it unsuitable for use with the Covert lab in particular. With Docker's power comes some complexity that arises from several of its key features. This complexity can make it more difficult for users new to Docker to maintain existing images. Another drawback to Docker is that it requires a recent version of Linux in order to support some of Docker's more advanced features, and systems available for the lab's use do not meet this requirement.

Although Singularity is currently the only option available for containerizing OMERO, its simplicity would still make it the container technology of choice for an initial deployment of OMERO. Because Singularity has much less isolation than Docker, users can setup Singularity containers with much less chance of configuring an option incorrectly. A Singularity container's all-in-one setup means that users wanting to configure OMERO do not need to concern themselves with how the container technology itself works. Although it may not be able to scale as well as Docker, this lack of scalability is not a concern in early stages or in labs with few users.

*Future work*

Three enhancements could be made to further simplify any research group's transition to OMERO. First, on the administrative side, one critical task is to perform data backups on a regular schedule, to ensure that there is a safe fallback state if the worst happens. Automating this process for the containerized OMERO installations would increase users' trust in the system's reliability. Second, additional useful features of OMERO have not yet been made easily configured. OMERO.dropbox allows users to add files to a folder that OMERO monitors -- when a new file is added to that folder, it is automatically uploaded to OMERO. OMERO.web supports the installation of plugins, such as OMERO.figure, that add functionality not present in the default configuration. Making these features simple to set-up would increase the appeal of OMERO to many groups. A final major enhancement would be to build an additional Docker or Singularity image with OMERO's API libraries installed. The OMERO API bindings are a major method of interacting with OMERO, but their dependencies can be somewhat complex to setup, though instructions are available on OME's webpage. Eliminating this setup step by distributing images with a pre-built development environment would make researchers more willing to consider adopting OMERO.

Other useful work to investigate would include better training materials for teaching new users about the OME Data Model and OMERO. The two utilities to convert files to OME-TIFF and upload ROIs serve as a good start, but many more useful features exist within OMERO.insight, OMERO.web, and the OMERO APIs. Although OME provides some documentation around basic features, more guided tutorials around advanced features would provide immense value to the community.

## References

1. Swedlow, Jason R. "The Open Microscopy Environment: A collaborative data modeling and software development project for biological image informatics." Imaging cellular and molecular biological functions. Springer Berlin Heidelberg, 2007. 71-92.
2. Goldberg, Ilya G., et al. "The Open Microscopy Environment (OME) Data Model and XML file: open tools for informatics and quantitative analysis in biological imaging." Genome biology 6.5 (2005): 1.
3. Linkert, Melissa, et al. "Metadata matters: access to image data in the real world." The Journal of cell biology 189.5 (2010): 777-782.
4. Collins, Tony J. "ImageJ for microscopy." Biotechniques 43.1 Suppl (2007): 25-30.
5. Allan, Chris, et al. "OMERO: flexible, model-driven data management for experimental biology." Nature methods 9.3 (2012): 245-253.
6. https://www.openmicroscopy.org/site/about/partners
7. Edelstein, Arthur, et al. "Computer control of microscopes using µManager." Current protocols in molecular biology (2010): 14-20.
8. Carpenter, Anne E., et al. "CellProfiler: image analysis software for identifying and quantifying cell phenotypes." Genome biology 7.10 (2006): R100.
9. https://www.openmicroscopy.org/site/community/minutes/meetings/11th-annual-users-meeting-2016
10. https://www.openmicroscopy.org/site/community/minutes/meetings/9th-annual-users-meeting-june-2014

11. Momjian, Bruce. PostgreSQL: introduction and concepts. Vol. 192. New York: Addison-Wesley, 2001.
12. Reese, Will. "Nginx: the high-performance web server and reverse proxy." Linux Journal 2008.173 (2008): 2.
13. https://docs.docker.com/engine/understanding-docker/#/the-underlying-technology
14. https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/
15. https://docs.docker.com/compose/overview/
16. https://docs.docker.com/docker-hub/
17. https://docs.docker.com/registry/
18. Kurtzer, G. M. (2016). Singularity 2.1.2 - Linux application and environment containers for science [Data set]. Zenodo. http://doi.org/10.5281/zenodo.60736
19. https://github.com/hadim/docker-omero
20. http://supervisord.org/
21. Schindelin, Johannes, et al. "Fiji: an open-source platform for biological-image analysis." Nature methods 9.7 (2012): 676-682.
22. https://github.com/CovertLab/covertrack
23. Pérez, Fernando, and Brian E. Granger. "IPython: a system for interactive scientific computing." Computing in Science & Engineering 9.3 (2007): 21-29.