An array-tomography deployable software system

## overview

This document is a rough outline on how to package and bind together various software components essential for Array Tomography (AT) activities.

This document only details software employed *after* raw image data is acquired.

A very rough overview of the steps from raw image data to say, a cut out sub-volume from a larger, reconstructed and assembled volume, is

1. Ingest image tile specs into Render.
2. Median Filter, Flatfield Correction, Deconvolution
3. Stitching (tile by tile). Record transforms in Render
4. Alignment (one channel) and Registration (subsequent channels). Record transforms in render.

See workflow, scripts diagram in Appendix.

### Key Software Components

* Render Server (with backend Mongo database and webserver frontend)
* RenderPython and RenderPython Apps (thin Python API for the Render Server)
* Many standalone Python scripts that puts it all together.
* NDViz, ImageJ, TomViz, Vaa3D, VTK (data viewers)

## Docker

Setting up a system capable of bringing raw image data to high quality assembled volumes is not a trivial task. However, software components and applications can be captured as they evolve using *Docker images*.

The diagram below shows a preliminary design of involved software apps for the ATDeployable.

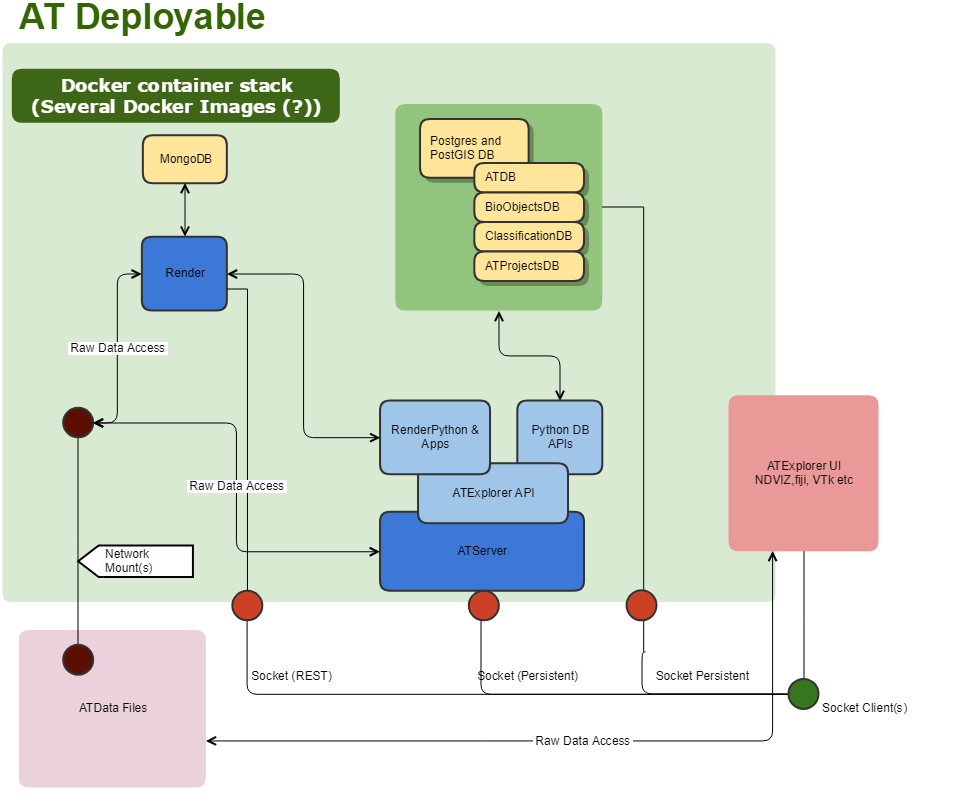


Figure : Overview of the ATDeployable

## How to get it started?

Currently the ATExplorer UI, which is running on a Windows machine, is performing remote execution of RenderPython apps (that lives in a Docker container) over a SSH connection to a remote Ubuntu host.   
  
A (remote) command for appending an affine transformation to a render stack may look like below, for example:

docker exec renderapps\_develop python -m renderapps.stack.apply\_global\_affine\_to\_stack --render.host ibs-forrestc-ux1 --render.port 80 --render.owner NM\_SST\_2018 --render.project M362218\_c6a\_Tlx3\_NMSST11199 --render.client\_scripts /var/www/render/render-ws-java-client/src/main/scripts --input\_stack TEST\_Totte\_Rotated\_RENAMED\_AFF --output\_stack TEST\_Totte\_Rotated\_RENAMED\_AFF\_AFF --transformId Affine\_2018\_Aug\_08\_12:23:39 --pool\_size 1 --M00 0.984808 --M01 -0.173648 --M10 0.173648 --M11 0.984808 --B0 0.000000 --B1 0.000000

The above require a username and password on the remote machine on which to execute the command.

Having a dedicated server (the ATServer in the diagram above) will simplify execution of lower level code for clients, by using a more refined API, and will be served over a conventional network socket.

### Step 1 (Forrest + TK)

* Outline the steps for setting up an initial Docker image **on Linux** (atbigdawg?) that provides a Render service instance at a configurable port, say Port 9000, as well as the ability to execute Renderpython apps. Call the container *ATDeployable-devel.*
* Have a network mount to a dedicated device on one of the nases.
* Don’t do the Postgres DB initially.

Perhaps this is as simple as following the instructions here:

<https://github.com/saalfeldlab/render/blob/master/docs/src/site/markdown/render-ws-docker.md>

If so, I will start there, and consult with you on the way.

### Step 2 (TK + Forrest)

Get the same setup as above, working on Windows, but network mount will be to the local machine.

### Next steps

* Get work on the ATServer component started. This will eventually require a container containing a (Linux) C++ compiler development environment to be setup. The Clang C++ compiler will be used.
* Ability to run X or VNC to the host, for debugging purposes.
* Add Postgres databases

# Appendix

Docker file documentation

## Python scripts overview

