

# Update on the Jupyter-based Analysis Portal at BNL SDCC

Ofer Rind

Fall HEPiX, Barcelona, Spain

October 9, 2018

**70** YEARS OF  
**DISCOVERY**

A CENTURY OF SERVICE



# Data Analysis As A Service

- Jupyter Notebooks (iPython)
  - “Literate Computing”: Combines code, text, equations within a narrative
  - Easy to document, share, and reproduce results; create tutorials
  - Provide a flexible, standardized, platform independent interface through a web browser
  - Well-suited for *interactive* analysis
  - Can run with no local software installation
  - Many language extensions (kernels) and tools available



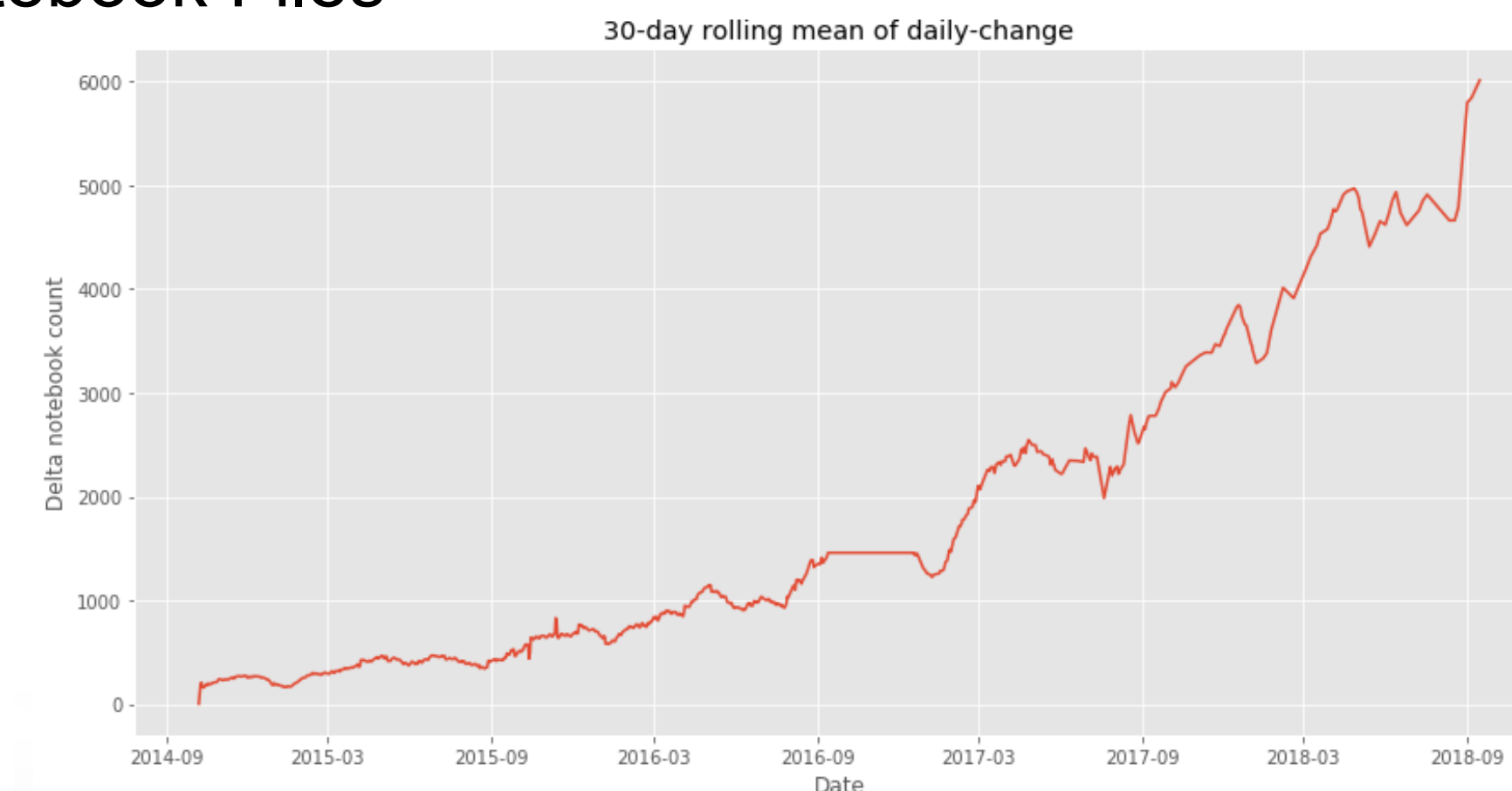
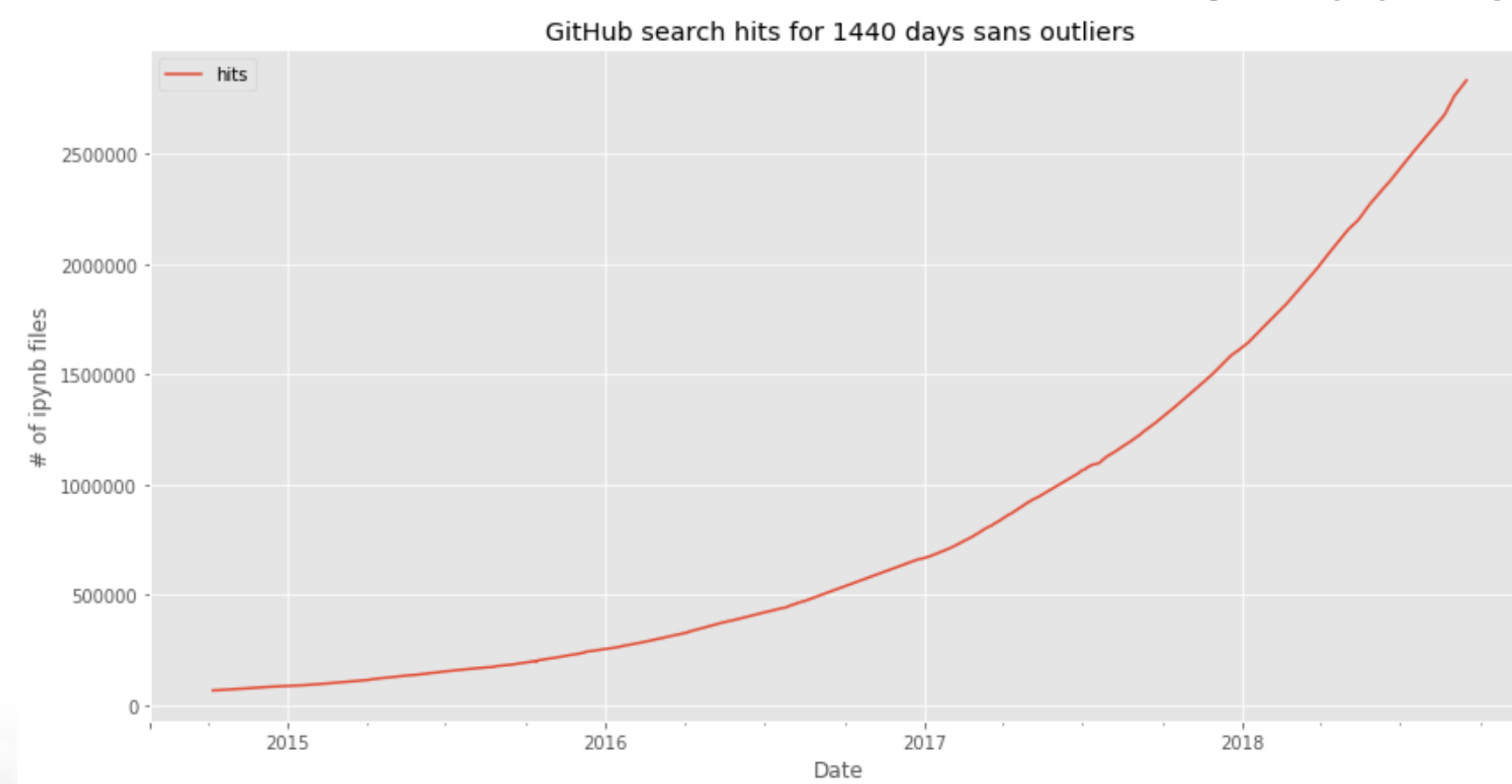
Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.

- From the facility point of view:  
Would like to layer this service atop existing resources, without building a new dedicated infrastructure, such as a specialized cluster.

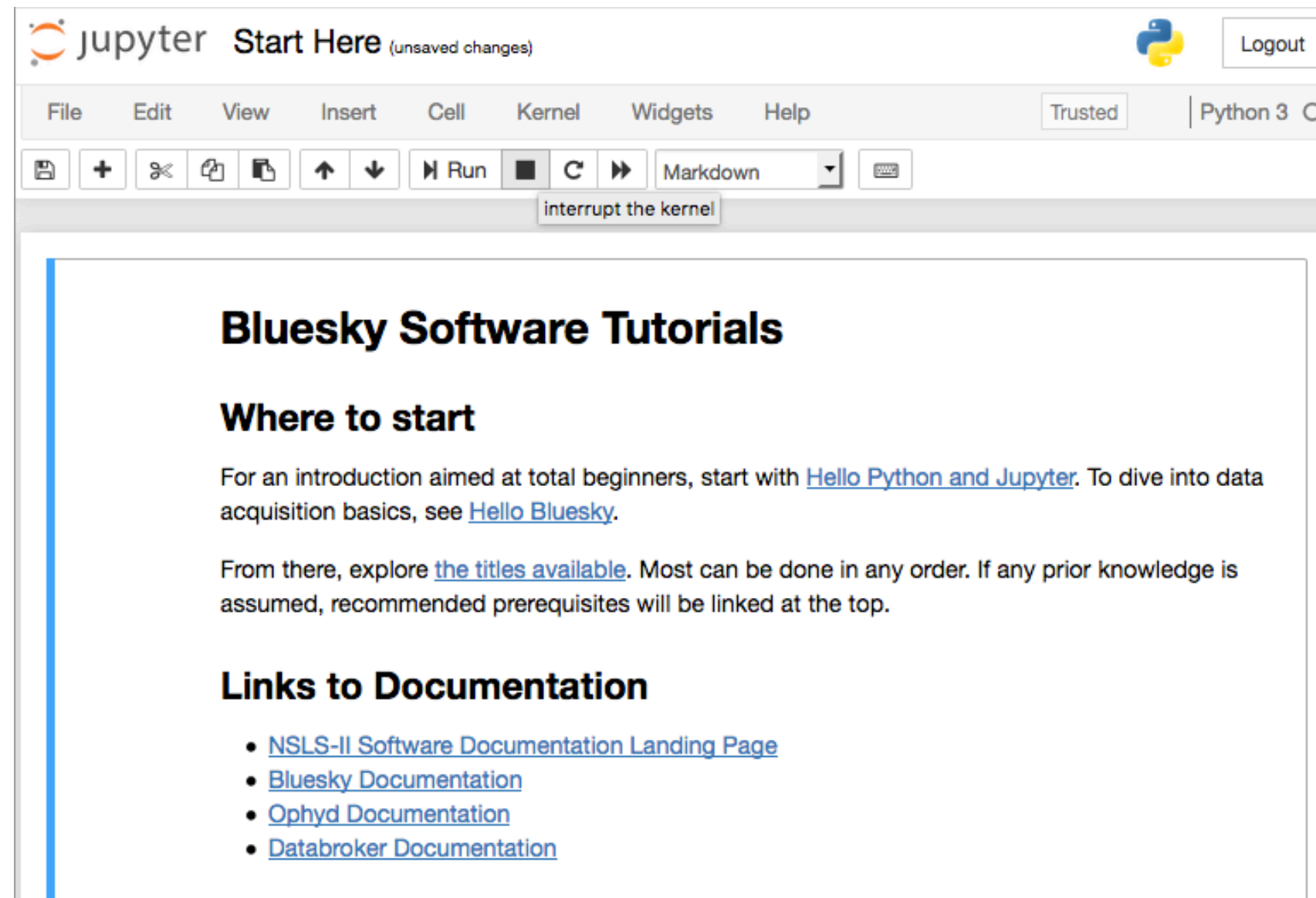
# Growing in Popularity...



## Github Notebook Files



# ...Across The Scientific Community As Well



jupyter Start Here (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

## Bluesky Software Tutorials

### Where to start

For an introduction aimed at total beginners, start with [Hello Python and Jupyter](#). To dive into data acquisition basics, see [Hello Bluesky](#).

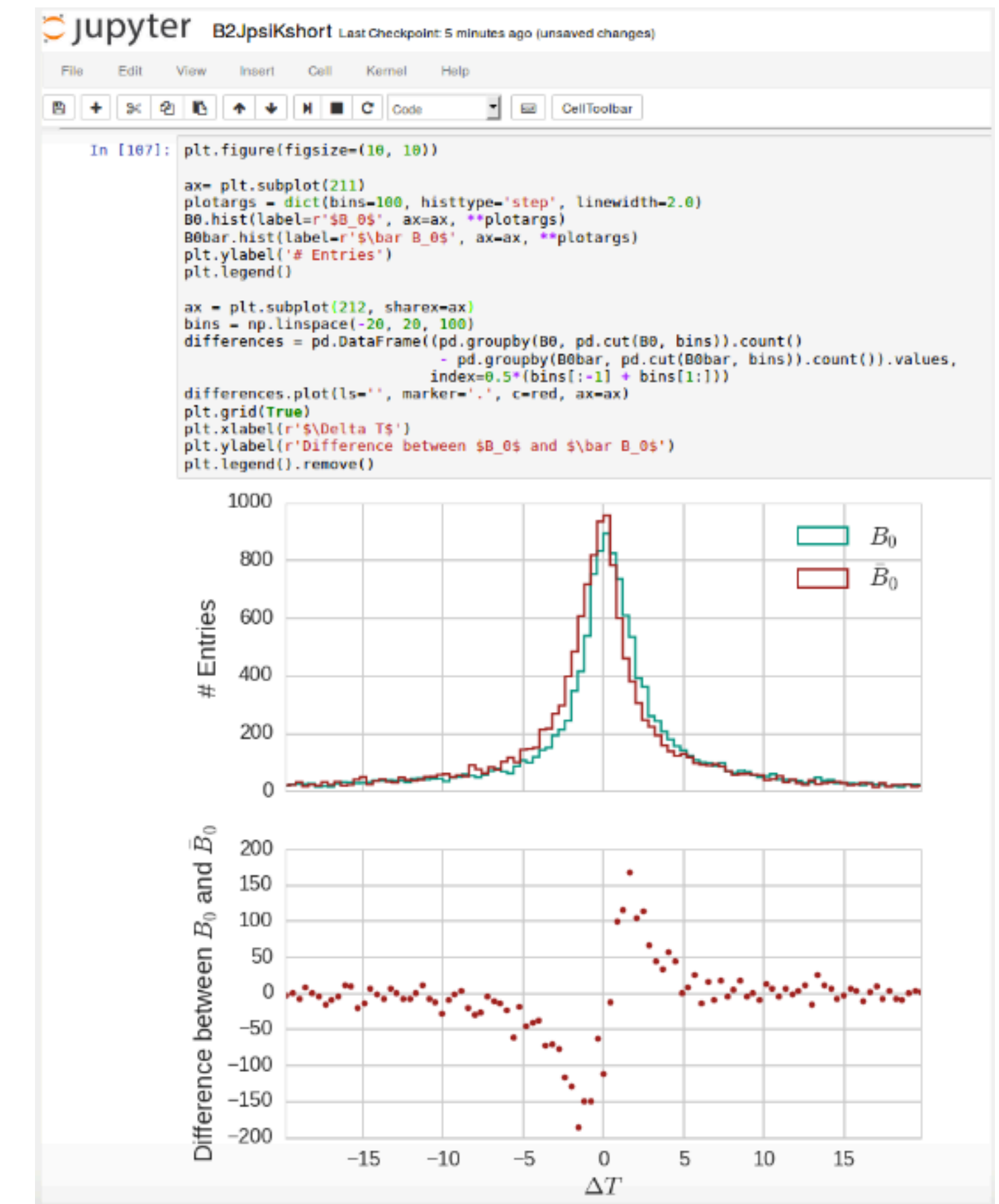
From there, explore [the titles available](#). Most can be done in any order. If any prior knowledge is assumed, recommended prerequisites will be linked at the top.

### Links to Documentation

- [NSLS-II Software Documentation Landing Page](#)
- [Bluesky Documentation](#)
- [Ophyd Documentation](#)
- [Databroker Documentation](#)



## SQR-018: Investigations into JupyterLab as a basis for the LSST Science Platform



```
In [107]: plt.figure(figsize=(10, 10))

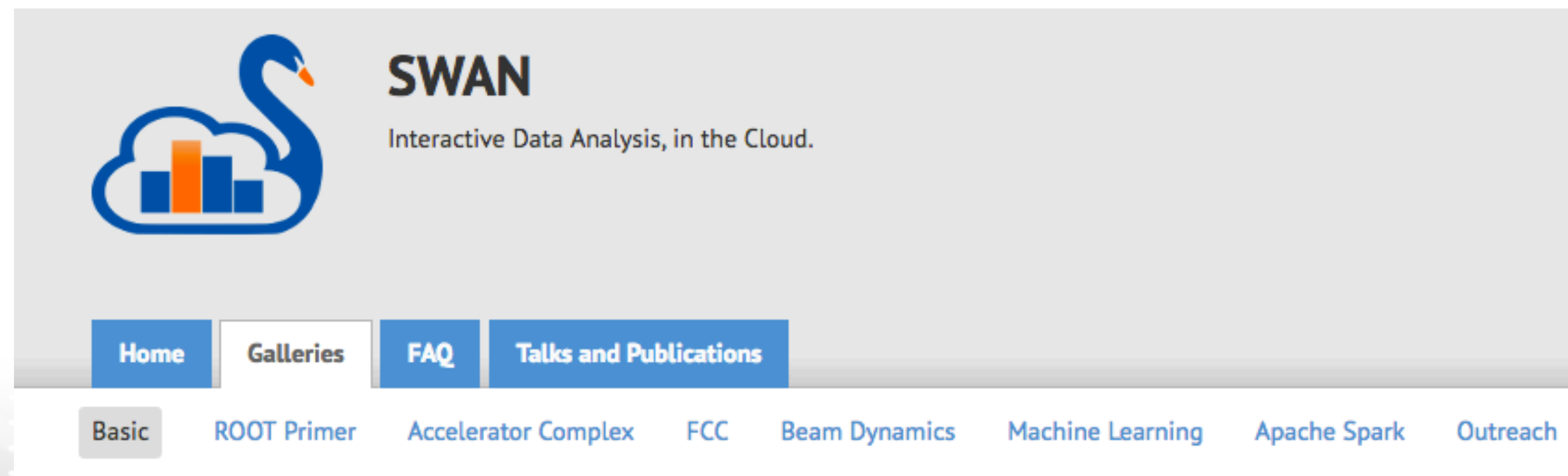
ax= plt.subplot(211)
plotargs = dict(bins=100, histtype='step', linewidth=2.0)
B0.hist(label=r'$B_0$', ax=ax, **plotargs)
B0bar.hist(label=r'$\bar{B}_0$', ax=ax, **plotargs)
plt.legend()

ax = plt.subplot(212, sharex=ax)
bins = np.linspace(-20, 20, 100)
differences = pd.DataFrame((pd.groupby(B0, pd.cut(B0, bins)).count()
                           - pd.groupby(B0bar, pd.cut(B0bar, bins)).count()).values,
                           index=0.5*(bins[:-1] + bins[1:]))
differences.plot(ls='.', marker='.', c='red', ax=ax)
plt.grid(True)
plt.xlabel(r'$\Delta T$')
plt.ylabel(r'Difference between $B_0$ and $\bar{B}_0$')
plt.legend().remove()
```

# Entries

Difference between  $B_0$  and  $\bar{B}_0$

$\Delta T$



## SWAN

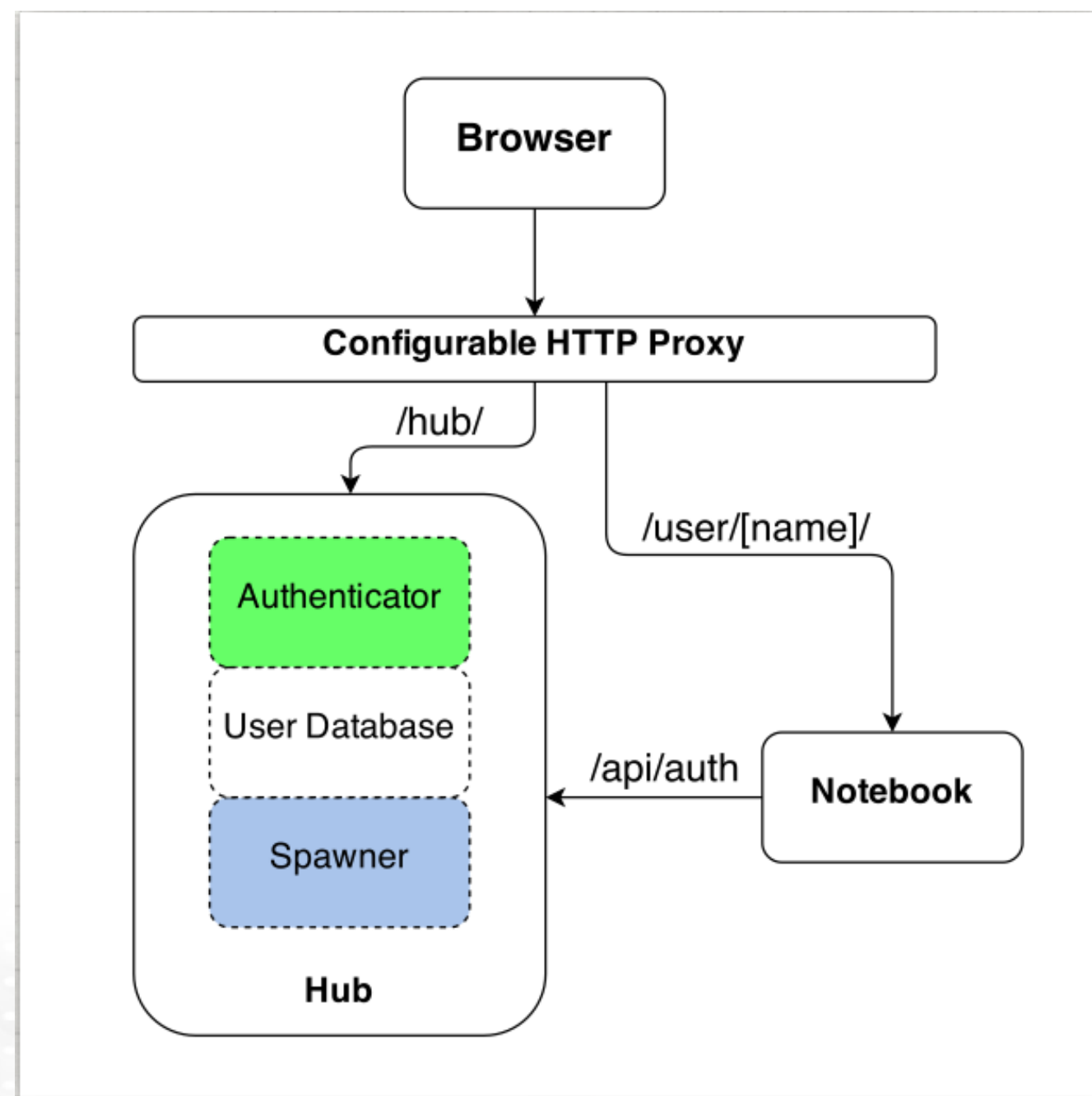
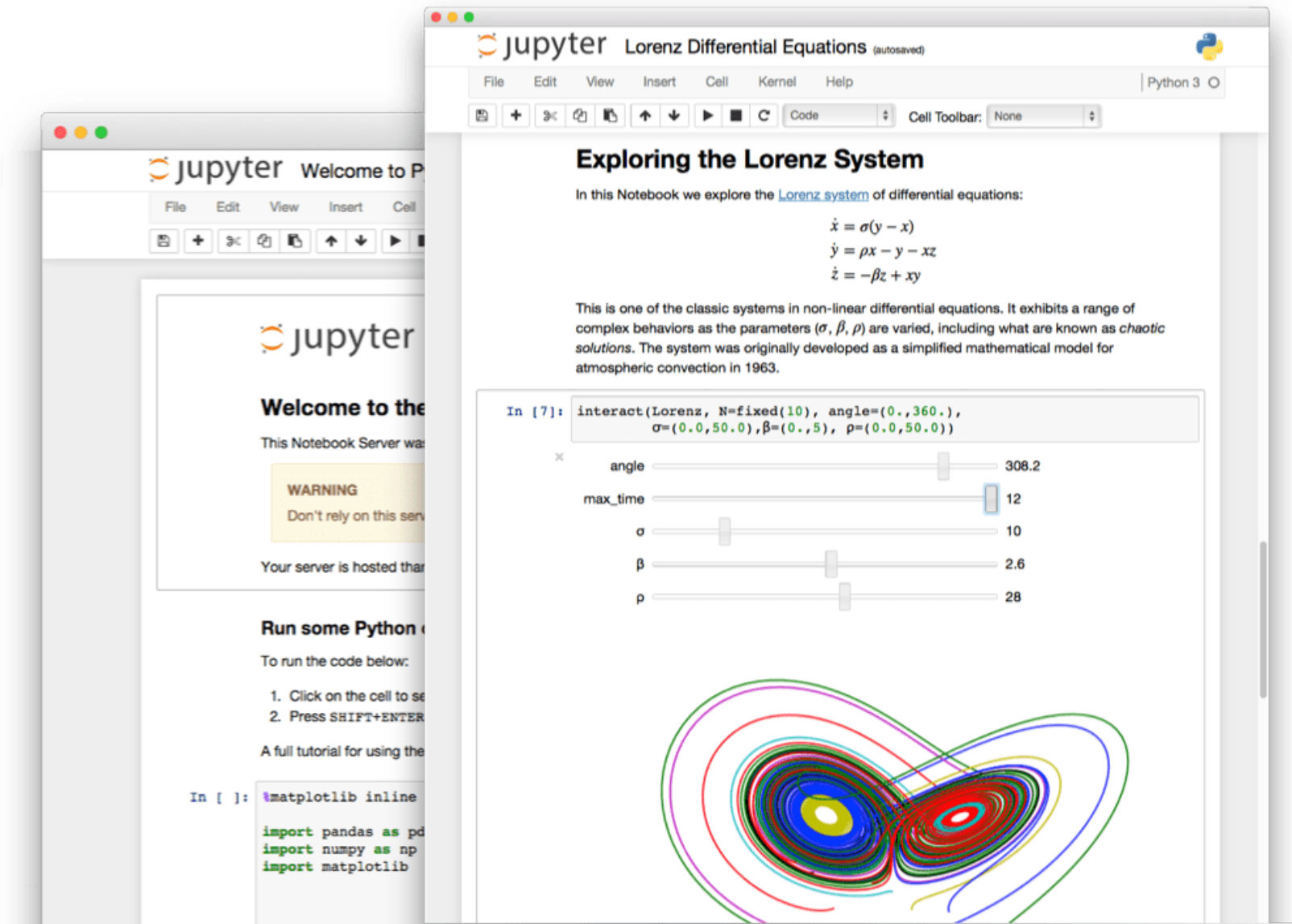
Interactive Data Analysis, in the Cloud.

Home Galleries FAQ Talks and Publications

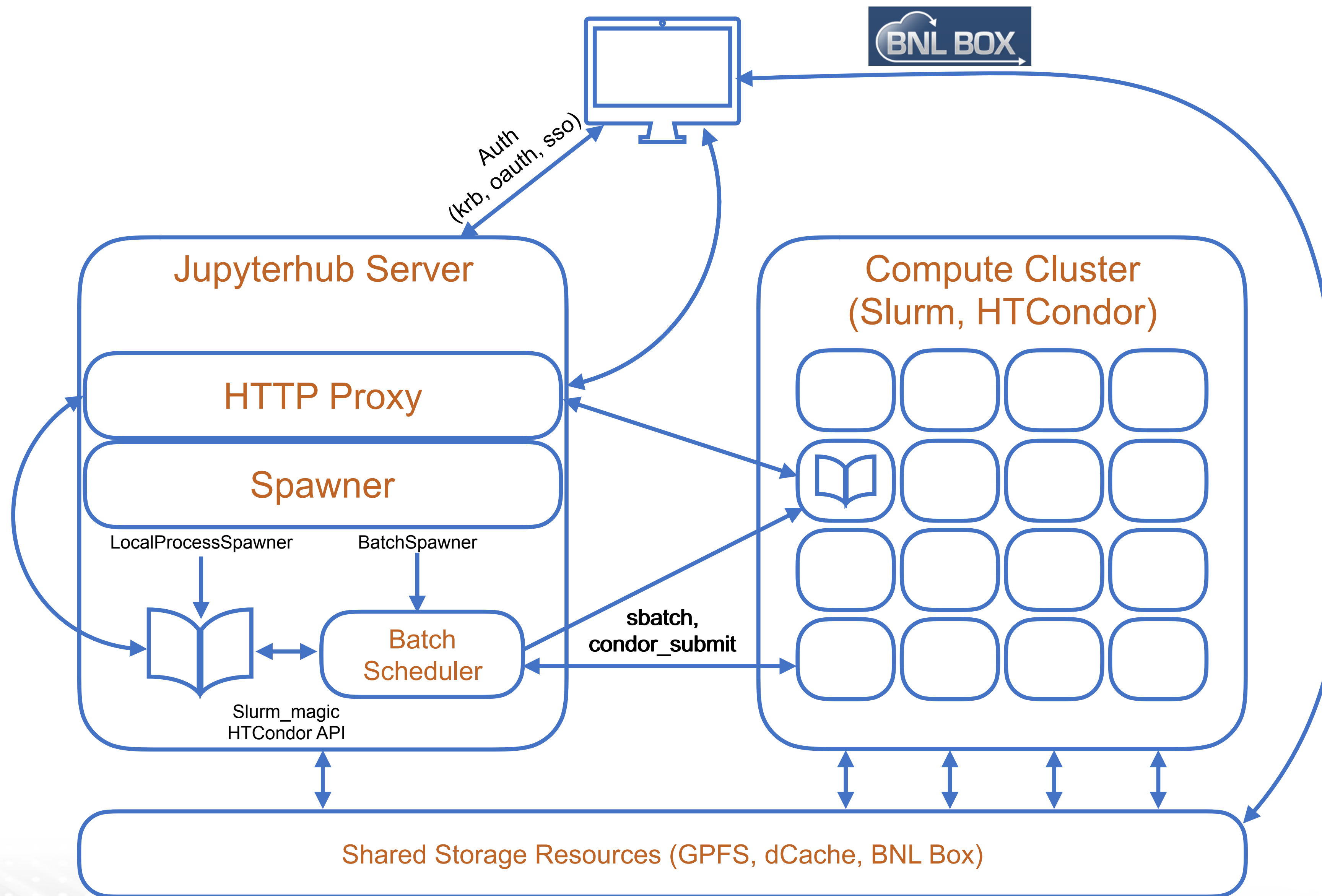
Basic ROOT Primer Accelerator Complex FCC Beam Dynamics Machine Learning Apache Spark Outreach

# Some terminology

- **Jupyter notebook:** web-based application suitable for capturing and documenting the entire computation process



- **Jupyterlab:** next-generation web-based user interface
- **Jupyterhub:** multi-user hub, spawns, manages, and proxies multiple instances of the single-user Jupyter notebook



# Current Setup at BNL

- Three Jupyterhub server portals deployed on RHEV
  - Jupyter01: Primary HTC development environment with access to HTCondor queue
  - Jupyter02: Primary HPC development environment with access to Institutional Cluster via Slurm
  - Jupyter03: OAuth/Keycloak test deployment
- Access behind firewall via ssh tunnel through gateway to Jupyterhub https proxy
- Kerberos auth to Jupyterhub server leveraging PAM stack (standard SDCC account)
- Three independent Python environments (mainly Anaconda)

# Recently Developed Features

- Custom Kernel Environments
- Containerized Environment
- Condor API
- OAuth/Keycloak



# User/Experiment Specific Computing Environments

- Define custom kernels
- Can support large experimental software stacks without containerized environment

```
-bash-4.2$ cd ~/.local/share/jupyter/kernels/ATLAS
-bash-4.2$ ls
kernel.json  logo-64x64.png  setup.sh
-bash-4.2$ cat kernel.json
{
  "argv": [
    "/usr/local/share/jupyter/kernels/ATLAS/setup.sh",
    "-f",
    "{connection_file}"
  ],
  "display_name": "ATLAS test",
  "language": "python"
}
-bash-4.2$ cat setup.sh
#!/usr/bin/env bash

export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
export ALRB_localConfigDir=$HOME/localConfig
source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh --quiet
source ${ATLAS_LOCAL_ROOT_BASE}/utilities/oldAliasSetup.sh root --rootVersion=6.08.06-HiggsComb-x86_64-slc6-gcc49-opt

# python will be in the anaconda2 directory
PYTHONPATH=${PYTHONPATH}:/u0b/software/anaconda2/condor/lib/python exec /u0b/software/anaconda2/bin/python -m ipykernel_launcher $@
```

ATLAS  
interactive  
analysis with  
custom kernel  
on locally  
spawned  
notebook with  
Condor access

## Running ATLAS GridScan Code

This is a demonstration of running ATLAS GridScan code provided by Viviana Cavaliere. This notebook is running within a custom ATLAS python 2 test kernel that sets up the local environment explicitly for this purpose. It issues the following commands to set up the environment:

```
export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
export ALRB_localConfigDir= $ {HOME}/localConfig
source $ {ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh --quiet
source $ {ATLAS_LOCAL_ROOT_BASE}/utilities/oldAliasSetup.sh root --rootVersion=6.08.06-HiggsComb-x86_64-slc6-gcc49-opt
```

```
[1]: %cd /usatlas/u/rind/GridScan
/direct/usatlas+u/rind/GridScan
```

The command below runs a python script that submits a set of jobs to the Condor queue. In this case, it was a set of nine jobs as defined by the user\_config.py script located in the cwd. When the jobs complete, they create a set of nine root files that are aggregated in the next step.

```
[2]: %run submitAllJobs.py "LL" "3000" "2" "Obs" "Test"
```

Welcome to JupyROOT 6.08/06

```
[10]: !condor_q -run
```

```
-- Schedd: jupyter01.sdcc.bnl.gov : <10.42.34.39:9618?... @ 09/27/18 14:36:13
ID      OWNER      SUBMITTED      RUN_TIME HOST(S)
12742.0  rind        9/27 14:31     0+00:02:13 slot1@spar0405.usatlas.bnl.go
12743.0  rind        9/27 14:31     0+00:02:13 slot1@spar0401.usatlas.bnl.go
12744.0  rind        9/27 14:31     0+00:02:13 slot1@spar0410.usatlas.bnl.go
12745.0  rind        9/27 14:31     0+00:02:13 slot1@spar0402.usatlas.bnl.go
12746.0  rind        9/27 14:31     0+00:00:21 slot1@spar0403.usatlas.bnl.go
```

```
[ ]: %cd Plotting
      %run collectScans.py "LLObs" "3000" "Grid2"
```

Now that we have collected the outputs into one TTree, we can examine the results.

```
[3]: f = ROOT.TFile("NewScans/LLObs_3000_Grid2.root")
      f.ls()
```

```
/direct/usatlas+u/rind/GridScan/Plotting
TFile**      NewScans/LLObs_3000_Grid2.root
TFile*       NewScans/LLObs_3000_Grid2.root
KEY: TTree   stats;1 runAsymptoticsCLs
```

```
[4]: f.stats.Scan("mass:x:y:CLb_med:CLb_obs:mu_hat_exp:mu_hat_obs:param0:param1:param2")
```

```
[4]: 9L
```

```
*****
*****
*   Row   *   mass *       x *       y *   CLb_med *   CLb_obs * mu_hat_ex * mu_hat_ob *   param0 *
* param1 *   param2 *
*****
*****
*     0 *   3000 * 0.2599999 *   0.25 *   0.5 * 0.2957884 *   0 * -0.018856 * 0.8739007 * 0.
4773877 *   1 *
*     1 *   3000 * 0.2599999 * 0.3499999 *   0.5 * 0.3028981 *   0 * -0.012117 * 1.4389778 * 0.
7858586 *   1 *
*     2 *   3000 * 0.2599999 * 0.4499999 *   0.5 * 0.4117247 *   0 * -0.009105 * 1.9597632 * 1.
0701090 *   1 *
*     3 *   3000 * 0.3600000 *   0.25 *   0.5 * 0.2982341 *   0 * -0.015742 * 1.0551071 * 0.
5763579 *   1 *
*     4 *   3000 * 0.3600000 * 0.3499999 *   0.5 * 0.3071902 *   0 * -0.009815 * 1.8467967 * 1.
0085470 *   1 *
*     5 *   3000 * 0.3600000 * 0.4499999 *   0.5 * 0.3143115 *   0 * -0.006793 * 2.6705367 * 1.
4581754 *   1 *
*     6 *   3000 * 0.4600000 *   0.25 *   0.5 * 0.2995282 *   0 * -0.014767 * 1.1563247 * 0.
6316376 *   1 *
*     7 *   3000 * 0.4600000 * 0.3499999 *   0.5 * 0.3095638 *   0 * -0.007941 * 2.0977754 * 1.
1455882 *   1 *
```

# Containerized Environment

- Start Jupyter kernel inside a Singularity container from within custom kernel environment
- Can facilitate notebook sharing
- Care needed to pass certain environment variables

```
-bash-4.2$ cat setup.sh
#!/usr/bin/env bash

RELEASE=/cvmfs/belle.cern.ch/sl7/releases/release-02-00-00
unset PYTHONPATH
export BELLE2_NO_TOOLS_CHECK=TRUE
[source /cvmfs/belle.cern.ch/sl7/tools/b2setup $RELEASE
[
[# python will be in the anaconda2 directory
[SINGULARITYENV_PATH=${PATH} SINGULARITYENV_LD_LIBRARY_PATH=${LD_LIBRARY_PATH} /usr/bin/singularity exec -B /direct
/u0b/hollowec/singularity/rhic_sl7_ext.simg /u0b/software/anaconda3/bin/python -m ipykernel_launcher $@
```

Belle-II  
Notebook  
running on  
shared cluster  
via HTCondor

jupyter B2T\_Basics\_2\_EventSimulation Last Checkpoint: 07/30/2018 (autosaved) Logout Control Panel

File Edit View Insert Cell Kernel Widgets Help Trusted Belle2-Sing test

Run Markdown

### Processing

Up to this point we didn't produced or processed any events! We only defined the BASF2 path to determine which modules are called in which order on each event

Now we start the processing of the BASF2 path. In a usual steering file this is done by calling `process(path)`. In this IPython notebook we use a small helper class which provides us a nice progress bar. Just wait a few seconds until all events are generated and the bar turns green.

```
In [*]: from ipython_tools import handler
        calculation = handler.process(path)
        calculation.show_path()
        calculation.start()
        calculation.wait_for_end()
        calculation.show_log()
```

- VariablesToExtraInfo\_J/psi:dqm\_mumu
- SoftwareTrigger
- RootOutput
- ProgressPython
- PrintCollections

50 % Remaining time: 3 minutes 9 seconds

If everything went well we should now have simulated our first set of events and can continue with the [next step](#)

# Interface to HTCondor

- API written by Will Strecker-Kellogg using HTCondor python bindings described at last HEPiX
- Starting to adapt real analysis code to make use of this API
- Development/testing of more advanced Condor-specific functionality still to come

# Interface to HTCondor

## Running ATLAS GridScan Code Using Condor API

This is a demonstration of running ATLAS GridScan code provided by Viviana Cavaliere also using a custom ATLAS python 2 test kernel.

```
In [1]: import job
%cd /usatlas/u/rind/GridScan

2018-10-08 09:48:05,917 [INFO] job - Init v 8.7.9

/direct/usatlas+u/rind/GridScan
```

The command below uses the Condor API to generate and submit ATLAS jdf's on the fly.  
When the jobs complete, they create a set of nine root files that are aggregated in the next step.

```
In [2]: for x in range(0,3):
        for y in range(0,3):
            j = job.Job({'Universe': 'vanilla', 'GetEnv': 'true', 'accounting_group': 'group_atlas.general', 'Executable':
'. /Output/Comb_LLObs_Grid2_3000_x'+str(x)+'_y0/Comb_LLObs_Grid2_3000_x'+str(x)+'_y0.sh', 'Output': './Output
/Comb_LLObs_Grid2_3000_x'+str(x)+'_y0/condor.stdout', 'Error': './Output/Comb_LLObs_Grid2_3000_x'+str(x)+'_y0
/condor.stderr', 'Log': './Output/Comb_LLObs_Grid2_3000_x'+str(x)+'_y0/condor.log', '+IsMediumJob': 'true'})
            j.submit()

2018-10-08 09:48:11,615 [DEBUG] job - Run query: []
2018-10-08 09:48:11,633 [DEBUG] job - Run query: []
2018-10-08 09:48:11,645 [DEBUG] job - Run query: []
2018-10-08 09:48:11,660 [DEBUG] job - Run query: []
2018-10-08 09:48:11,672 [DEBUG] job - Run query: []
2018-10-08 09:48:11,683 [DEBUG] job - Run query: []
2018-10-08 09:48:11,699 [DEBUG] job - Run query: []
2018-10-08 09:48:11,711 [DEBUG] job - Run query: []
2018-10-08 09:48:11,723 [DEBUG] job - Run query: []
```

# Interface to HTCondor

```
In [5]: !condor_q -wide
```

```
-- Schedd: jupyter01.sdcc.bnl.gov : <10.42.34.39:9618?... @ 10/08/18 09:52:27
  ID      OWNER      SUBMITTED  RUN_TIME ST PRI  SIZE CMD
12779.0   rind      10/8 09:48  0+00:02:13 R  0    0.0 Comb_LLObs_Grid2_3000_x0_y0.sh
12780.0   rind      10/8 09:48  0+00:02:12 R  0    0.0 Comb_LLObs_Grid2_3000_x0_y0.sh
12781.0   rind      10/8 09:48  0+00:02:12 R  0    0.0 Comb_LLObs_Grid2_3000_x1_y0.sh
12782.0   rind      10/8 09:48  0+00:02:12 R  0    0.0 Comb_LLObs_Grid2_3000_x1_y0.sh
12783.0   rind      10/8 09:48  0+00:00:08 R  0    0.0 Comb_LLObs_Grid2_3000_x1_y0.sh
12784.0   rind      10/8 09:48  0+00:00:00 I  0    0.0 Comb_LLObs_Grid2_3000_x2_y0.sh
12785.0   rind      10/8 09:48  0+00:00:00 I  0    0.0 Comb_LLObs_Grid2_3000_x2_y0.sh
12786.0   rind      10/8 09:48  0+00:00:00 I  0    0.0 Comb_LLObs_Grid2_3000_x2_y0.sh
```

```
Total for query: 8 jobs; 0 completed, 0 removed, 3 idle, 5 running, 0 held, 0 suspended
Total for rind: 8 jobs; 0 completed, 0 removed, 3 idle, 5 running, 0 held, 0 suspended
Total for all users: 9 jobs; 0 completed, 0 removed, 3 idle, 6 running, 0 held, 0 suspended
```

```
In [4]: %cat Output/Comb_LLObs_Grid2_3000_x2_y2/condor.submit
```

```
Universe = vanilla
GetEnv = true
accounting_group = group_atlas.general
Executable=./Output/Comb_LLObs_Grid2_3000_x2_y2/Comb_LLObs_Grid2_3000_x2_y2.sh
Input=/dev/null
Output=./Output/Comb_LLObs_Grid2_3000_x2_y2/condor.stdout
Error=./Output/Comb_LLObs_Grid2_3000_x2_y2/condor.stderr
Log=./Output/Comb_LLObs_Grid2_3000_x2_y2/condor.log
+IsMediumJob = true
Queue
```



# Interface to HTCondor

- Also looking at adapting [dask-jobqueue](https://dask-jobqueue.readthedocs.io/en/latest/) for HTCondor

## Dask-Jobqueue

<https://dask-jobqueue.readthedocs.io/en/latest/>

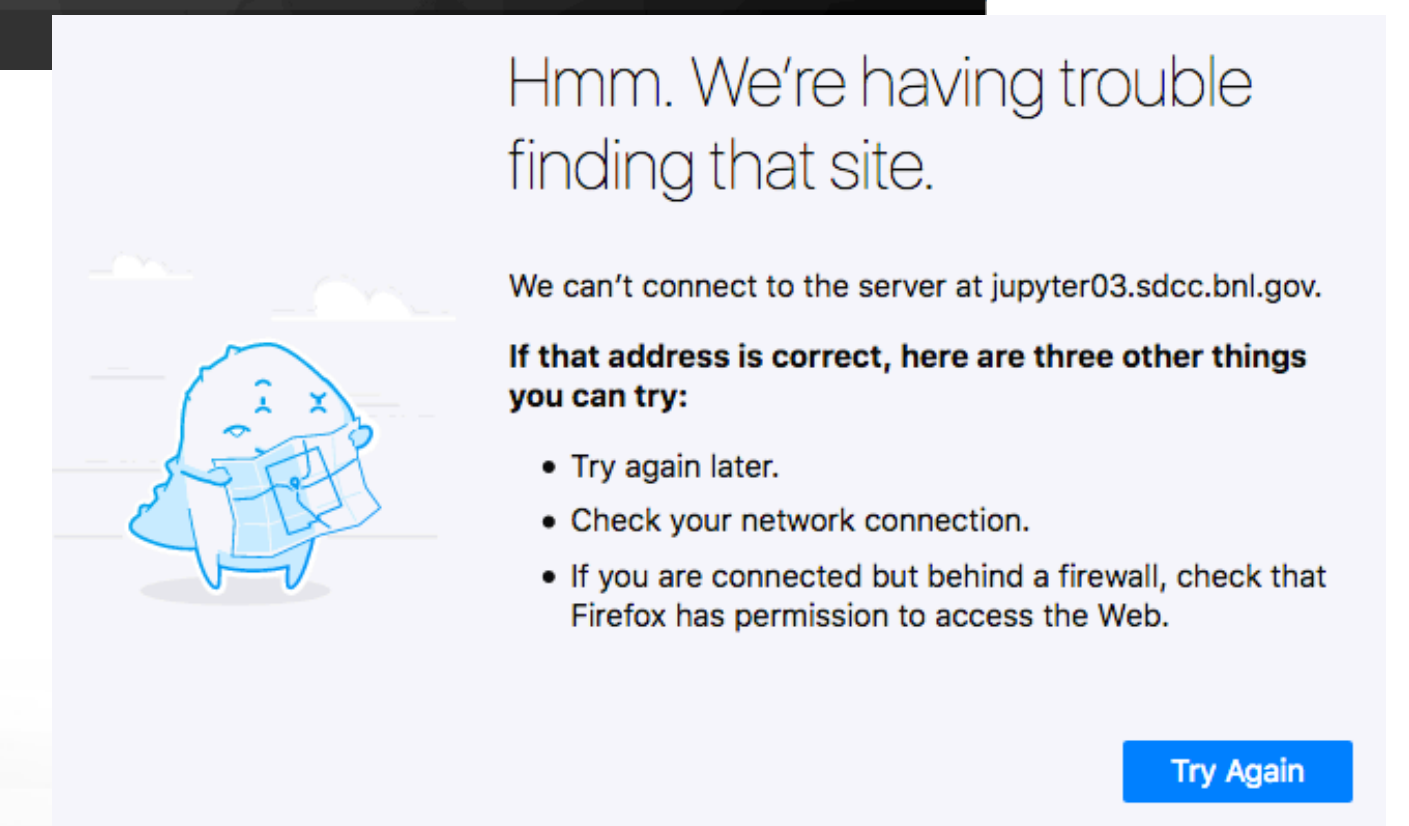
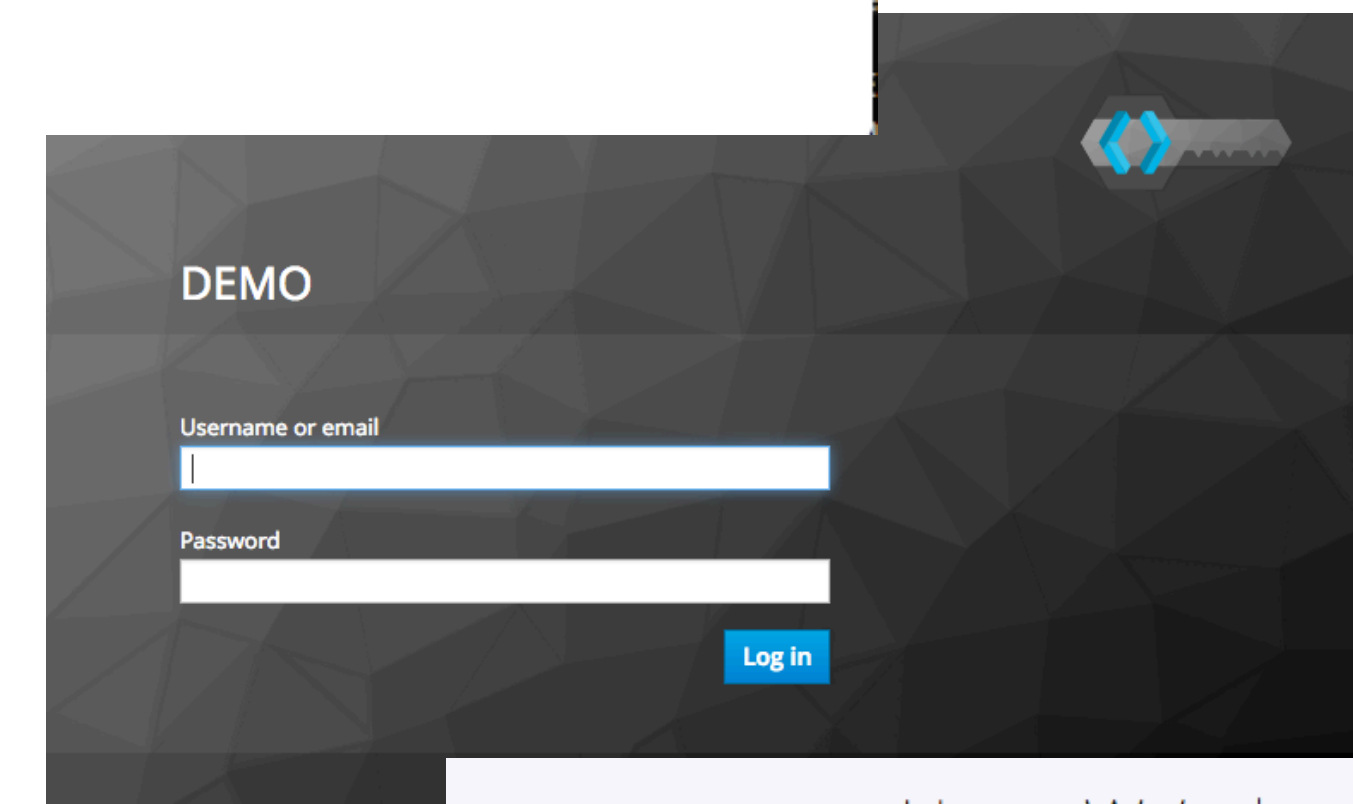
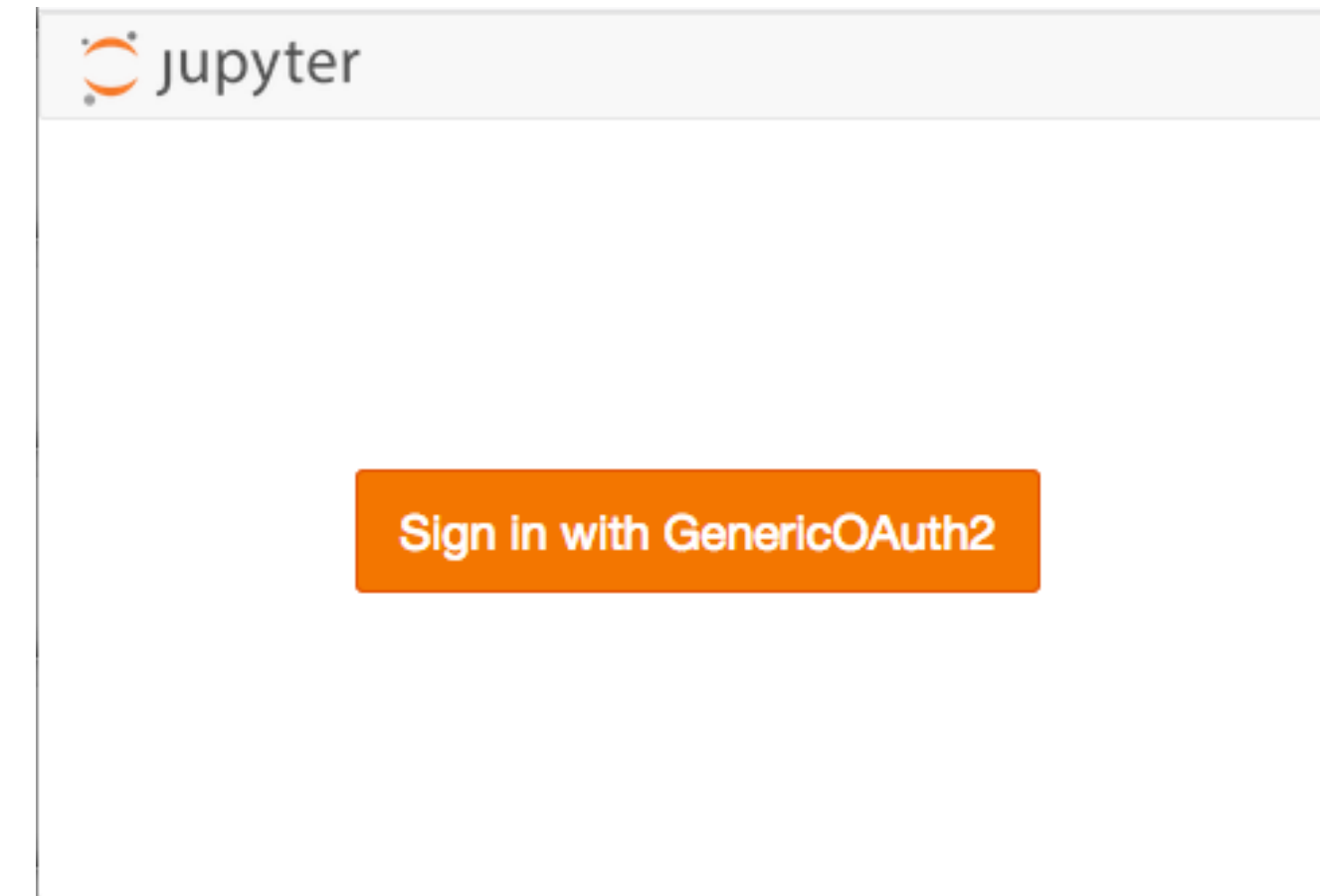
*Easily deploy Dask on job queuing systems like PBS, Slurm, MOAB, SGE, and LSF.*

The Dask-jobqueue project makes it easy to deploy Dask on common job queuing systems typically found in high performance supercomputers, academic research institutions, and other clusters. It provides a convenient interface that is accessible from interactive systems like Jupyter notebooks, or batch jobs.

- Tools for asynchronous communication with job clusters and dashboard for monitoring

# OAuth

- Current Jupyterhub access via SSH tunneling + KRB5 auth
- Bypassing gateway requires 2FA due to interactive login (shell) access
- OAuth/Keycloak implemented to enable this, along with SSO
  - Cannot be implemented through gateway tunnel without firewall mod
    - Callback not proxied - `https://jupyter03.sdcc.bnl.gov:8000/hub/oauth_callback?state=eyJzd...`  
`%3D%3D&session_state=0d6c6bd3-...&code=eyJhb...`
- Will be implementing as part of facility-wide project to migrate to IPA/SSO/2FA



# Conclusion & Discussion

- Gaining experience serving Jupyter notebooks across multiple environments, for multiple experiments, on multiple platforms
- Service now available! Expect to expand the user base over the next few months
  - If you build it, will they come?
  - Expecting to encounter the usual host of user support/management/scaling problems
- What is the current state around the HEP community - usage and monitoring?