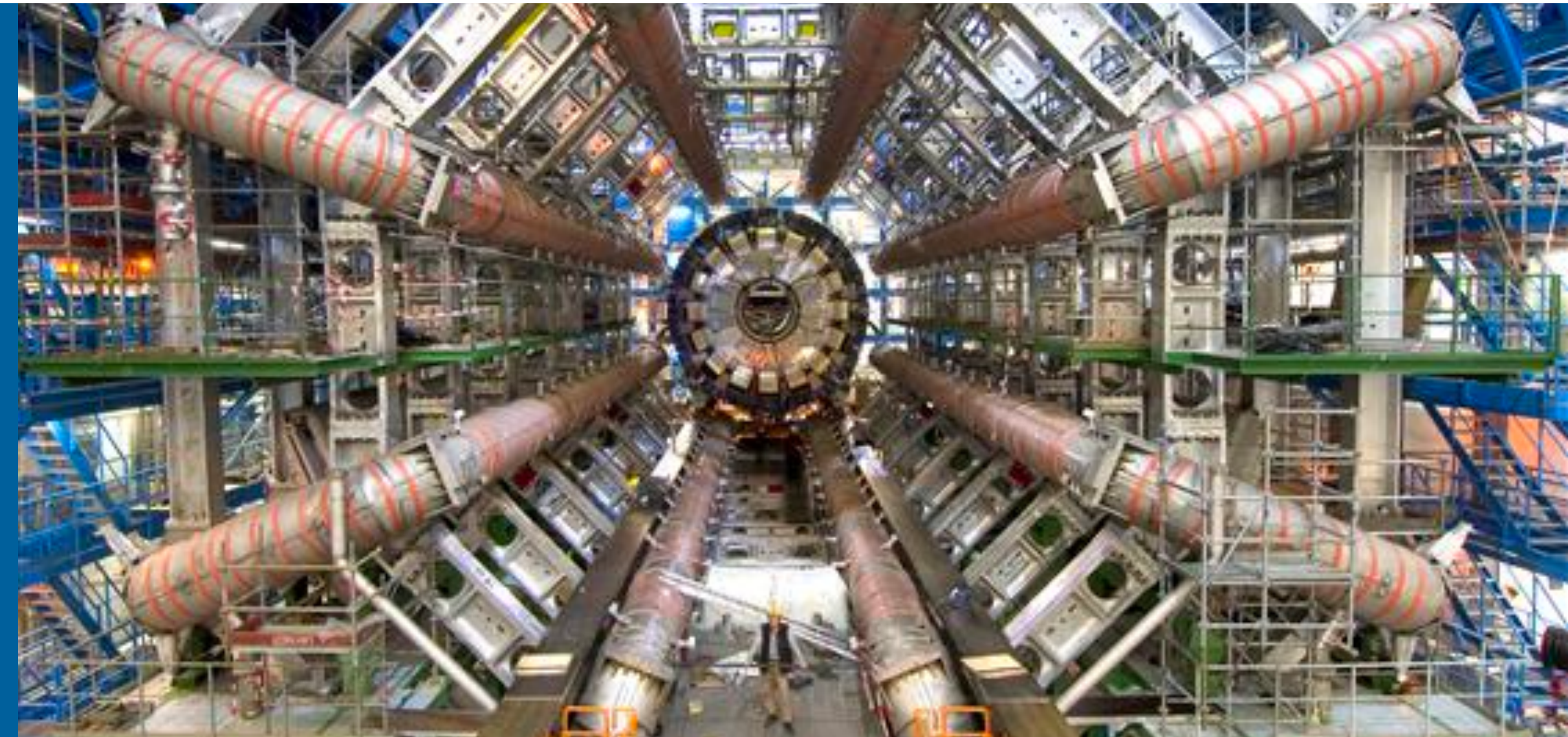


Jupyter for ATLAS experiment at BNL's SDCC



DOUG BENJAMIN
Argonne National Lab
High Energy Physics Division

*Slides from a Talk given
at CHEP 2019*

Integrating Interactive Jupyter Notebooks at the BNL SDCC

D. Allan, D. Benjamin*, M. Karasawa, K. Li, [O. Rind](#), W. Strecker-Kellogg
Brookhaven National Laboratory, *Argonne National Laboratory



BNL Scientific Data & Computing Center (SDCC)

- Located at Brookhaven National Laboratory on Long Island, NY — Largest component of the Computational Science Initiative (CSI)
- Serves an increasingly diverse, multi-disciplinary user community: RHIC Tier-0, US ATLAS Tier-1 and Tier-3, Belle-II Tier-1, Neutrino, Astro, LQCD, NSLS-II, CFN, sPHENIX....more than 2000 users from 20+ projects
- Large HTC infrastructure accessed via HTCondor (plus experiment-specific job management layers)
- Growing HPC infrastructure, currently with two production clusters accessed via Slurm
- Limited interactive resources accessed via ssh gateways



Two modes, Two workflows

- HPC & HTC (parallel vs interlinked, accelerator vs plain-cpu)
 - ▶ High-performance systems for GPUs / MPI / accelerators
 - ▶ High-throughput systems for big data parallel processing
- Batch & Interactive (working on code/GPUs vs submitting large workflows)
 - ▶ Job workflow management
 - ▶ Direct development & testing on better hardware

Traditional “Interactive SSH + Batch” paradigm places requirements on the users:

- Must be sufficiently motivated to learn and use batch systems
- Need to buy in to the workflow model: Develop, compile, move data, small-scale run on interactive nodes, full-scale processing on batch

Data Analysis As A Service



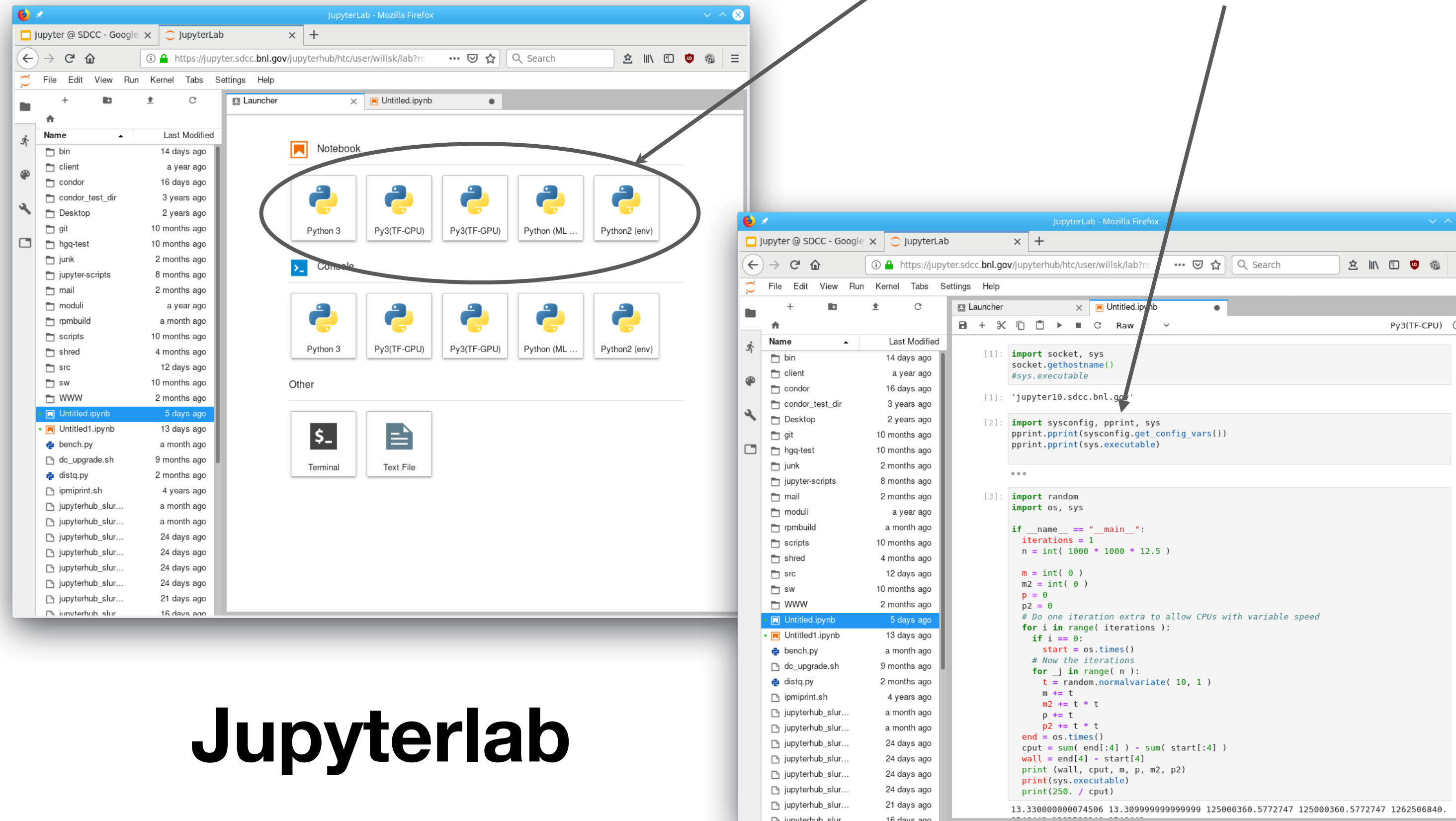
Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.

- New paradigm: **Jupyter Notebooks** (IPython)
 - ▶ Expanding the interactive toolset
 - ▶ “Literate Computing”: Combines code, text, equations within a narrative
 - ▶ Easy to document, share, and reproduce results; create tutorials...Lower barrier of entry, both for learning curve and user-base
 - ▶ Provides a flexible, standardized, platform independent interface through a web browser
 - ▶ Can run with no local software installation
 - ▶ Many language extensions (kernels) and tools available

Jupyter Service UI

Kernels

Notebook Documents

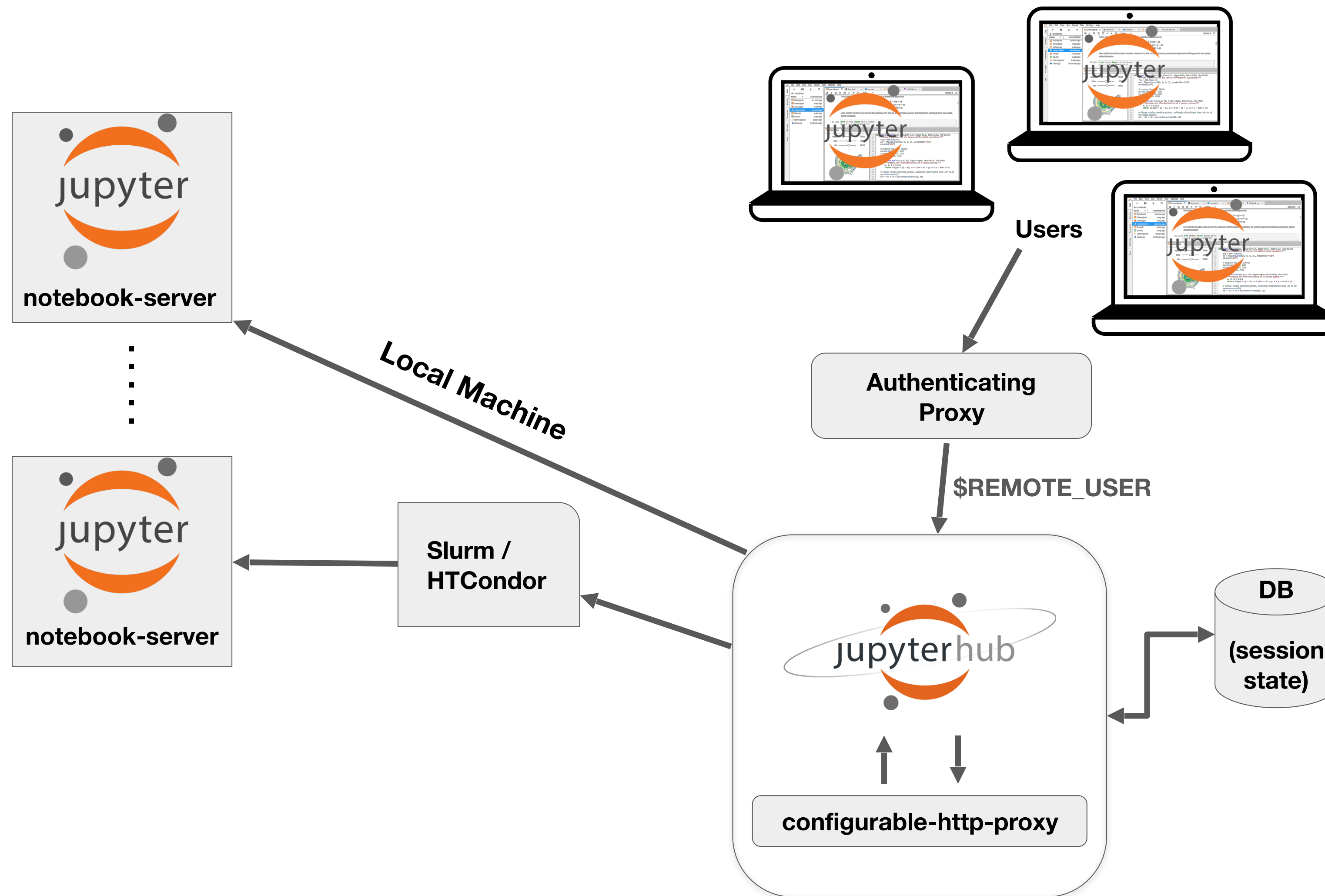


Jupyterlab

Production Architecture

- Goal: leverage already successful pre-existing resources, expertise, and infrastructure (batch) instead of rolling a new backend service
 - Allow users to leverage any type of computational resource they might need — implies enabling both HTC and HPC/GPU, e.g. upcoming ATLAS ML workflows
- Requirements
 - Expose to the world via unified interface <https://jupyter.sdcc.bnl.gov> — common solution for HTC and HPC resource access
 - Satisfy cybersecurity constraints
- Design
 - Insert authenticating proxy as frontend to decouple jupyterhub from cybersecurity requirements (e.g. MFA)
 - Scale notebooks via load-balancing as well as via batch systems
 - Automated deployment of multiple hub instances using Puppet
 - Enable access to GPU nodes in a user-friendly way
 - User-specific UI for Slurm spawner support

Jupyterhub Service Architecture




Frontend Proxy Interface

BROOKHAVEN NATIONAL LABORATORY | Scientific Data and Computing Center | SDCC JupyterHub

SDCC JupyterHub

The SDCC offers multiple JupyterHub instance and back-end combinations for different users and accounts. Choose the appropriate option from the instances displayed below.


[More information](#) [Questions and support](#)



SDCC **HTC**

Access to Condor queues and HTC computing resources via SDCC JupyterHub. Requires a valid SDCC account and corresponding experiment affiliation.

[Launch](#) [More info](#) SDCC HTC JH



SDCC **HPC**

Access to Slurm scheduling and GPU computing resources on the IC and KNL clusters via JupyterHub. Requires a valid SDCC account and computing resource allocation.

[Launch IC](#) [Launch KNL](#) [More info](#) SDCC HPC JH

- For Orchestration: a small cluster of directly-launched jupyter instances
 - ▶ HTTP-level Load-balanced from frontend proxy
 - ▶ One each on IC and HTCondor shared pool
- For Develop and Test: Use existing batch systems
 - ▶ HTCondor and Slurm support running a jupyterlab session as a batch job
 - ▶ Containers can enter at batch level to isolate external users or can be based on choice of environment
 - ▶ Best way to ensure exclusive, fair access to scarce resources (e.g. GPUs)
 - ▶ Open questions: Latency, Cleanup, Starvation

Using Jupyter tools to access local resources

Spawners

Jim Crist edited this page on Jul 16 · 26 revisions

Edit

New Page

Wiki › Spawners

Custom Spawners for JupyterHub

- **BatchSpawner** for spawning remote servers using batch systems (Torque, PBS, Slurm, etc)
- **DockerSpawner**, which actually has two different spawners in it:
 - **dockerspawner.DockerSpawner**, for spawning identical Docker containers for each user
 - **dockerspawner.SystemUserSpawner**, for spawning Docker containers with an environment and home directory for each user
- **FargateSpawner** for spawning Docker containers on AWS Fargate (ECS)
- **ImageSpawner** for allowing users to choose which Docker image to spawn.
- **KubeSpawner** for use with [Kubernetes](#).
- **MarathonSpawner** for spawning instances on Marathon.
- **UCRSpawner** for spawning GPU instances on Marathon.
- **RemoteSpawner** (Archived, no longer maintained)
- **SimpleSpawner**, for testing purposes.
- **SystemdSpawner**, spawns notebooks with the isolation / security benefits of containers without the complexity of image management.
- **SudoSpawner** uses sudo. SystemdSpawner is recommended instead of SudoSpawner, if it is available to you.
- **WrapSpawner** allows spawner class and options to be chosen at runtime. Includes ProfilesSpawner for doing this from a spawner options form. Formerly part of **BatchSpawner**.
- **YarnSpawner** for spawning instances on an Apache Hadoop/YARN cluster.

Pages 12



JupyterHub

- [Home](#)
- [Zero to JupyterHub with Kubernetes Guide \(newcomer friendly\)](#)
- [Authenticators](#)
- [Spawners](#)
- [Community Deployment Notes](#)
- [Gallery of JupyterHub Talks and Presentations](#)

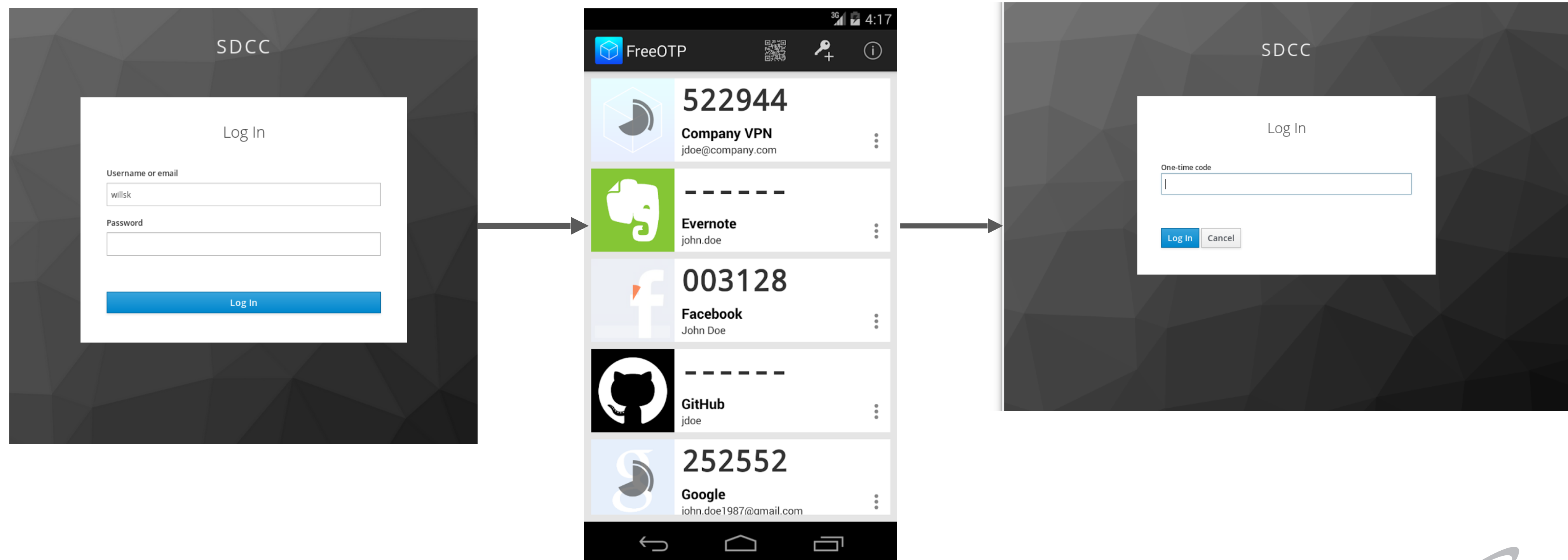
Clone this wiki locally

<https://github.com/jupyterh>



Multifactor Auth

- Using Keycloak MFA tokens
- Google Authenticator or FreeOTP app
- Easy setup by scanning QR code first time



Custom Slurm Spawner Interface

Spawner Options

Display only partitions/accounts to which user has access

Please choose your parameters to run on a node with a GPU or select to run locally on the submit node.

Select Partition: debug, harvester, scavenger, usatlas

Select Account: default

QOS: scavenger

GPU: any

Runtime (min): 30

Run Locally?

Spawn

~ or ~

Spawner Options

Account and Options defined by selected partition

Please choose your parameters to run on a node with a GPU or select to run locally on the submit node.

Select Partition: usatlas

Select Account: tier3

QOS: usatlas

GPU: any, Pascal, Tesla

Runtime (min): 720

Run Locally?

Spawn

Select here and will launch Local instead of Batch spawner

* For form spawner code see https://github.com/fubarwrangler/sdcc_jupyter

Adding containers to the mix

- Use of the batch spawn allows for the use of containers
- Singularity v3.4 is used at SDCC
 - Need to convert Docker images to Singularity images
- Load the images onto local shared file system
- Custom Slurm spawner interface is extendable to pickup container location from shared file system
- Should be straight forward to use EIC containers.

Challenges of Experiment Environments

- When you get a session (start a notebook-server), which environment?
 - Customization at the kernel level or via notebook-server container
- Whose problem is setting up the environments?
 - Work for a software librarian

```
-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 $@
```

Kernel Customization

```
-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 $@
```

Custom Container

Orchestration: Integrating Jupyter with Compute

- How to make it easier to use compute from Jupyter?
 - HTMap library from condor
 - Dask / IPyParallel / Parsl etc...
- Goal: abstract away the fact that you are using a batch system at all
 - Either through trivial substitutes
 - `map()` → `htmap()`
 - Or through cell "magics"
 - `%slurm` or equivalent
 - Or via nice pythonic decorators that submit to batch systems (e.g. Dask-jobqueue)

```
1  from condormap import condormap
2  import collections
3  import numpy
4
5
6  # Sample function
7  def logistic(r, len=10):
8      d = collections.deque(maxlen=len)
9      x = 0.4
10     for _ in xrange(5 * 10**7):
11         x = x * r * (1.0 - x)
12         d.append(x)
13     return list(d)
14
15
16  for k, d in condormap(logistic, numpy.arange(3.5, 3.6, 0.01), withdata=True):
17     print sorted(d)
18     t = set(round(x, 5) for x in d)
19     print k, "Mode ", len(t)
```

Conclusions

- US ATLAS worked with BNL SDCC to develop a Jupyter platform for Scientific analysis. That has grown beyond just HEP.
- The SDCC at BNL is deploying a Jupyterhub infrastructure enabling scientists from multiple disciplines to access our diverse HTC and HPC computing resources
- System designed to meet facility requirements with minimal impact on the backend
- Built-in support for experiment-based computing environment with a number of flexible access modes and workflows
- Continuing to develop new techniques for user collaboration

Additional missing enhancements for users

- Nice progress bar for a resource intensive shell would be nice to have.
- For example - CERN SWAN setup -

```
scope, filename, filesize, location
'ip', 'traceTimeentry')

#BNL_traces.show(10)
print 'number of traces - {}'.format(BNL_traces.count())

# write out to csv file
csv_BNL_traces='csv_BNL_traces_{0:4d}_{1:02d}_{2:02d}'.format(year,month,day)
print 'Writing csv output to directory - {}'.format(csv_BNL_traces)
BNL_traces.write.csv(csv_BNL_traces,mode='overwrite')

src_path = "/user/bdouglas/{}".format(csv_BNL_traces)
eos_dir = "/eos/atlas/user/b/bdouglas/BNL_MAS/csv_traces_BNL"
dst_path = os.path.join(eos_dir,csv_BNL_traces)
if not os.path.exists(dst_path) :
    os.mkdir(dst_path)
cmd = 'hdfs dfs -copyToLocal {0} {1} '.format(src_path,dst_path).split() # cmd must be an array of arguments
print cmd
files = subprocess.check_output(cmd).strip()
```

Job ID	Job Name	Status	Stages	Tasks	Submission Time	Duration
▶ 0	json	COMPLETED	1/1	83 / 83	2 minutes ago	26s
▶ 1	count	COMPLETED	2/2	84 / 84	2 minutes ago	1m:06s
▶ 2	csv	COMPLETED	1/1	83 / 83	a few seconds ago	23s

```
Processing - /user/rucio01/traces/traces.2019-10-10*
number of traces - 414196
Writing csv output to directory - csv_BNL_traces_2019_10_10
```

Extra Slides

Example: sPHENIX Test Beam



JUPYTER FAQ </> [Icons]

This study

This study extracts the dE/dx resolution from the sPHENIX 2019 beam test at FNAL by projecting the 16 layer prototype device to fully fledged 48-layer configuration envisioned for EIC.

Note the energy deposition from 120 GeV/proton is higher than MIP due to radiative rise that leads to slightly better dE/dx resolution due to ionization statistics.

Input

```
In [1]: const TString infile = "scan2/tpc_beam_ALL-0000.evt_TpcPrototypeGenFitT
// const TString description = "Position scan 2, #eta = 0, 3+ pad clust
// const TCut cut = "TPCTrack.nCluster>=14 && Sum$(ClusterY_Rotated>-.3
const TString description = "Scan 2, 120 GeV/c proton, #eta = 0, 2+ pad
const TCut cut = "TPCTrack.nCluster>=14 && Sum$(ClusterY_Rotated>-.3) =

// const TString infile = "eta_0.3/tpc_beam_ALL-0000.evt_TpcPrototypeGe
// const TString description = "120 GeV/c proton, #eta = 0.3, 2+ pad cl
// const TCut cut = "TPCTrack.nCluster>=12 && Sum$(ClusterY_Rotated>-.3
```

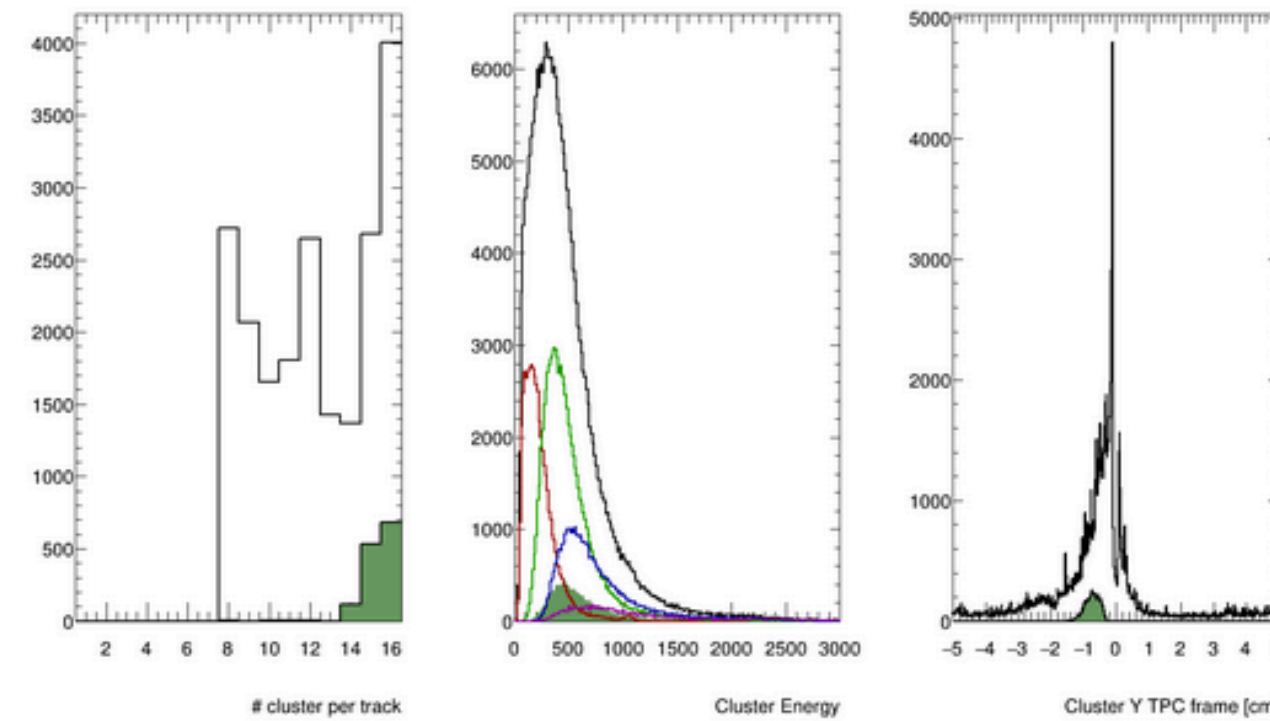
```
In [2]: %cpp -d
#include "sPhenixStyle.C"
#include "SaveCanvas.C"

TFile *_file0 = NULL;
TTree *T(nullptr);
```

```
In [3]: // gSystem->Load("libtpc2019.so");

SetsPhenixStyle();
TVirtualFitter::SetDefaultFitter("Minuit2");
gStyle->SetLegendTextSize(0);

sPhenixStyle: Applying nominal settings.
sPhenixStyle: ROOT6 mode
```



Info in <TCanvas::SaveSource>: C++ Macro file: scan2/tpc_beam_ALL-0000.

Note the cluster energy for 1-pad and 2-pad cluster (red, green) are significantly lower than that from 3/3+ pad clusters (blue, magenta). To be understood. For now, just analyzing 2+ pad clusters as the charge spread from zig-zag are expected to spread the charge to 3 pads.

PDF

```
In [7]: %cpp -d

Double_t langaufun(Double_t *x, Double_t *par)
{
//Fit parameters:
//par[0]=Width (scale) parameter of Landau density
//par[1]=Most Probable (MP, location) parameter of Landau density
//par[2]=Total area (integral -inf to inf, normalization constant)
//par[3]=Width (sigma) of convoluted Gaussian function
//
//In the Landau distribution (represented by the CERNLIB approximatio
//the maximum is located at x=-0.22278298 with the location parameter
//This shift is corrected within this function, so that the actual
//maximum is identical to the MP parameter.
```

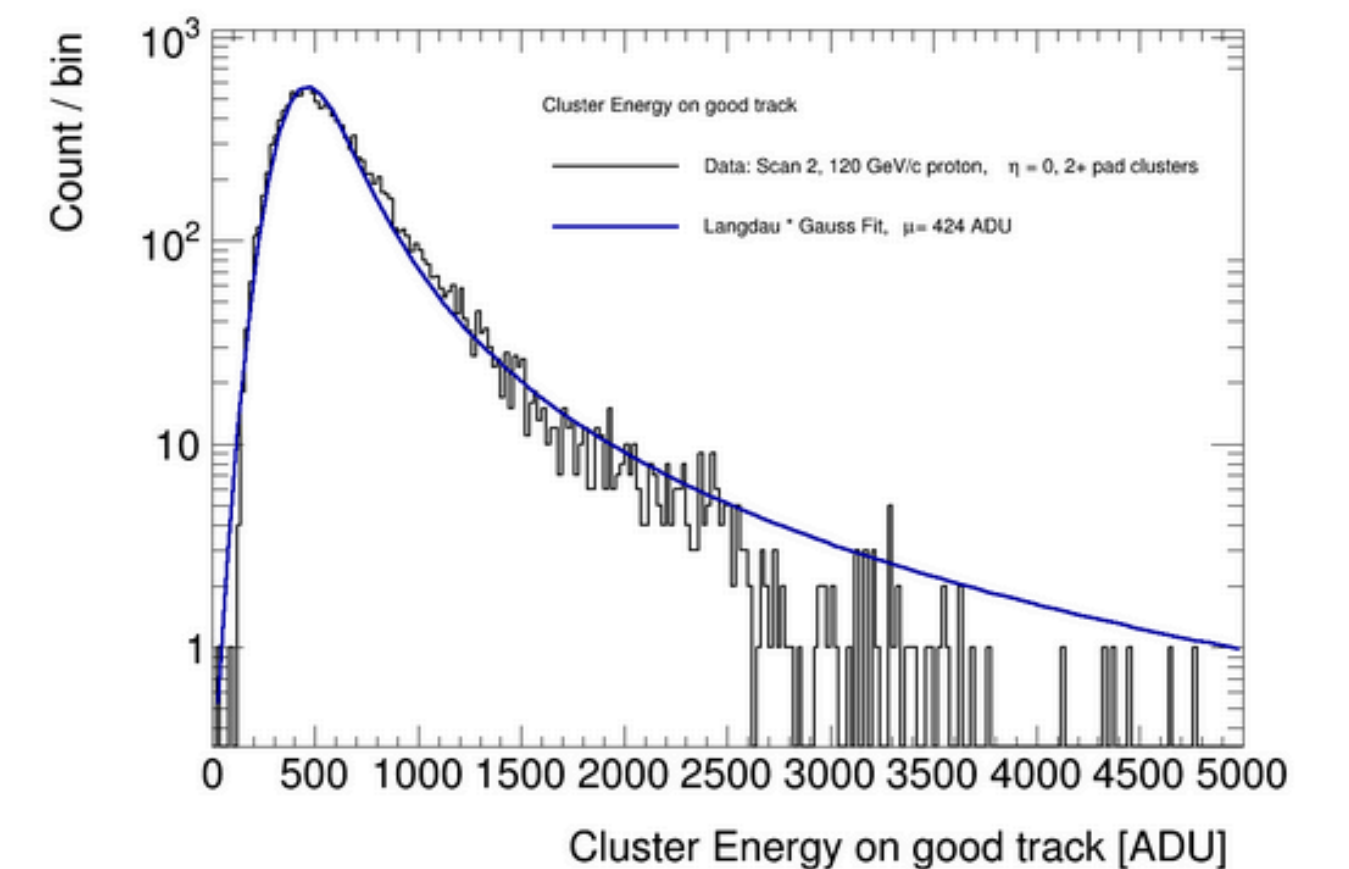
```
hClusterEnergyFit->Draw("same");
// gPhiDistortion->Draw("p");

TLegend *leg = new TLegend(.4, .7, .95, .9, + "Cluster Energy on g
leg->AddEntry(hClusterEnergy, TString("Data: ") + description, "1")
leg->AddEntry(hClusterEnergyFit,
Form("Langdau * Gauss Fit, #mu= %.0f ADU", hClusterEner
leg->Draw();

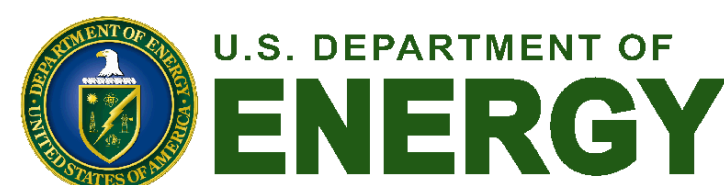
c1->Draw();
SaveCanvas(c1,
TString(_file0->GetName()) + TString(c1->GetName()), kFALS
}
```

```
*****
Minimizer is Minuit2
Chi2          =      403.493
NDF           =       164
Edm           =  2.69546e-08
NCalls       =       193
Width        =  66.7173  +/-  1.25122      (limit)
MP           =  423.708  +/-  2.09786      (limit)
Area         =  276531  +/-  2386.94      (limit)
GSigma       =   89.174  +/-  2.32025      (limit)
```

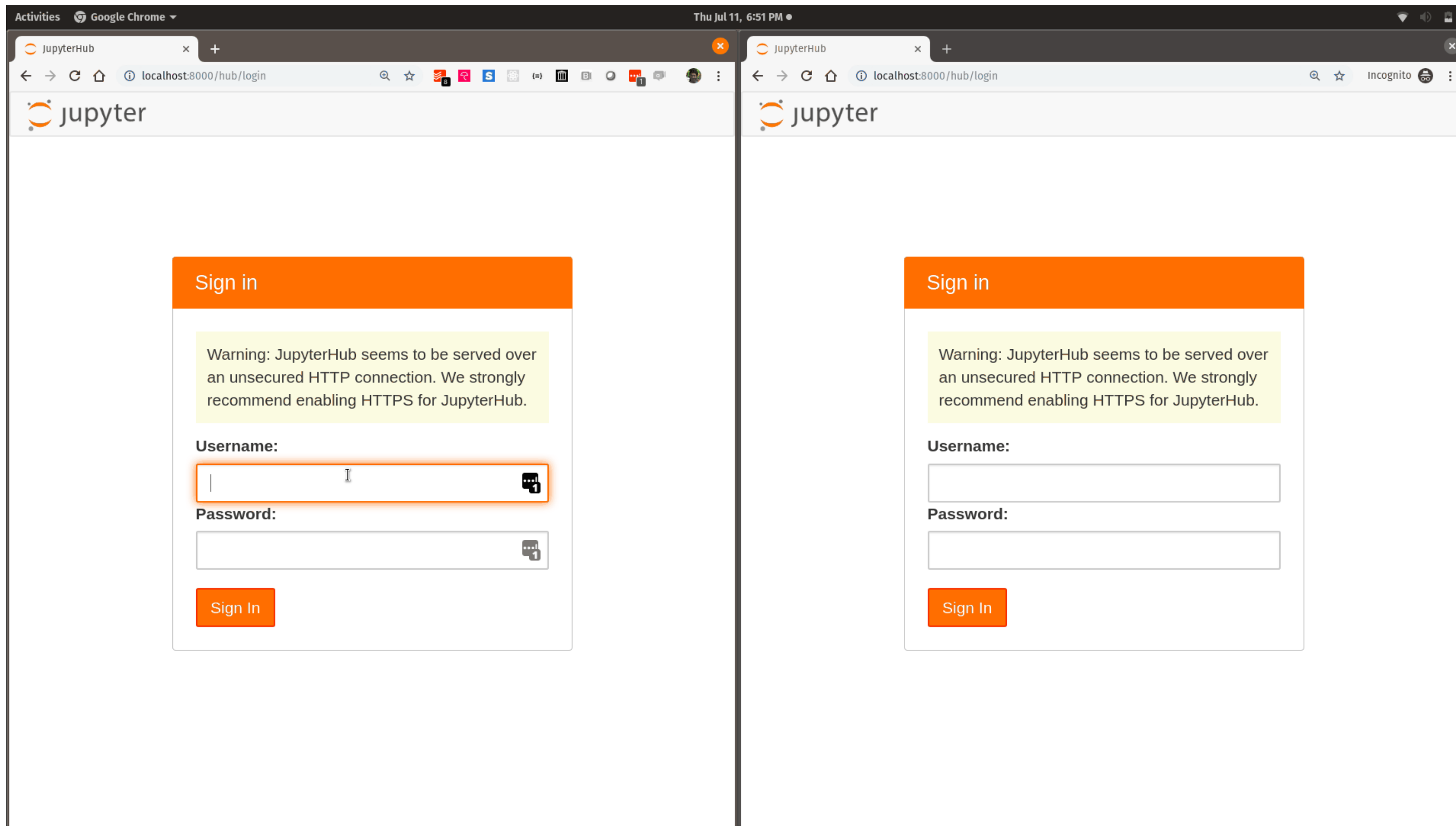
Info in <TCanvas::Print>: png file scan2/tpc_beam_ALL-0000.evt_TpcProto



** Notebook analysis courtesy of Jin Huang using custom sPHENIX Root Kernel



Notebook Sharing: Short Term



- Low-effort, short-term sharing between users on the same Hub
- Sender creates shareable link that provides last saved version of notebook to link recipient
 - Short-term link expires after certain time
 - Link encodes notebook options, such as container, to ensure compatible software environment
- See <https://github.com/danielballan/jupyterhub-share-link>

* Courtesy Daniel Allan, illustrative gif:
<https://github.com/danielballan/jupyterhub-share-link/blob/master/demo.gif?raw=true>

Notebook Archiving/Sharing

- Prepare a gallery of notebooks on Binder with a carefully defined software environment that anyone can recreate from a git repo with standard environment specs (e.g. requirements.txt)
 1. Enter URL of the repo
 2. Clicking "launch"
 3. Waiting and watching the build logs
 4. Copy a special link that will route directly to a Jupyter notebook running in a container that has repo contents and all software needed to run it successfully.
- Easy way for people to try your code and get running immediately
- Tightly coupled to Kubernetes and Docker, but developing similar workflows on HPC using Singularity

Build and launch a repository

GitHub repository name or URL
https://github.com/cho1dgraf/conda ← 1 2 GitHub ▾

Git branch, tag, or commit
Git branch, tag, or commit

Path to a notebook file (optional)
Path to a notebook file (optional) File ▾ launch

Copy the URL below and share your Binder with others:
https://mybinder.org/v2/gh/cho1dgraf/conda/master

Copy the text below, then paste into your README to show a binder badge: launch binder 4

3

Waiting building

Build logs

```
---> a5ca44eaa7ee
Step 25/38 : ARG REPO_DIR=${HOME}
---> Using cache
---> a25281372bef
Step 26/38 : ENV REPO_DIR ${REPO_DIR}
---> Using cache
---> 3d14afac5880
Step 27/38 : WORKDIR ${REPO_DIR}
---> Using cache
---> 5d5a1af05b90
Step 28/38 : ENV PATH ${HOME}/.local/bin:${REPO_DIR}/.local/bin:${PATH}
---> Using cache
---> 6adca6642720
Step 29/38 : USER root
---> Using cache
---> 3708d9fa7fc0
Step 30/38 : COPY src/ ${REPO_DIR}
---> 618e08487bd1
Step 31/38 : RUN chown -R ${NB_USER}:${NB_USER} ${REPO_DIR}
---> Running in 0ba0efbec2de
```

* Courtesy Daniel Allan

HTTP Frontend Configuration

- Authentication via Mellon plugin (for Keycloak)
- Subdivide URL space for different hub servers
 - /jupyterhub/\$cluster for HTC/HPC/others

- Load-balancing configuration
 - Need cookie for sticky-sessions
 - Newest apache on RHEL7
 - Requires websockets support

```
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=BALANCER_ROUTE_CHANGED
<Proxy "balancer://htccluster">
  BalancerMember "https://jupyter10.sdcc.bnl.gov:8000/jupyterhub/htc" route=1
  BalancerMember "https://jupyter11.sdcc.bnl.gov:8000/jupyterhub/htc" route=2
  BalancerMember "https://jupyter12.sdcc.bnl.gov:8000/jupyterhub/htc" route=3
  ProxySet stickysession=ROUTEID
</Proxy>
<Proxy "balancer://ws-htccluster">
  BalancerMember "wss://jupyter10.sdcc.bnl.gov:8000" route=1
  BalancerMember "wss://jupyter11.sdcc.bnl.gov:8000" route=2
  BalancerMember "wss://jupyter12.sdcc.bnl.gov:8000" route=3
  ProxySet stickysession=ROUTEID
</Proxy>
<Location /jupyterhub/htc>
  ProxyPass "balancer://htccluster"
  ProxyPassReverse "balancer://htccluster"
</Location>
RewriteCond %{HTTP:Connection} Upgrade [NC]
RewriteCond %{HTTP:Upgrade} websocket [NC]
RewriteRule /jupyterhub/htc/(.*) balancer://ws-htccluster/jupyterhub/htc/$1 [L,P]
```