Integrating Interactive Jupyter Notebooks at the BNL SDCC

Ofer Rind[1,*], *William* Strecker-Kellogg[1,**], *Daniel* Allan[1], *Douglas* Benjamin[2], *Mizuki* Karasawa[1], and *Kristy* Li[1]

[1]Brookhaven National Laboratory, Physics Dept., P.O. Box 5000, Upton, NY 11973-5000, USA
[2]Argonne National Laboratory, 9700 S. Cass Avenue, Lemont, IL 60439, USA

Abstract. At the SDCC we are deploying a Jupyterhub infrastructure to enable scientists from multiple disciplines to access our diverse compute and storage resources. One major design goal was to avoid rolling out yet another compute backend, but rather to leverage our pre-existing resources via our batch systems (HTCondor and Slurm). Challenges faced include creating a frontend that allows users to choose what HPC resources they have access to as well as selecting containers or environments, delegating authentication to a MFA-enabled proxy, and automating deployment of multiple hub instances. This paper covers the design and features of our Jupyterhub service.

1 Introduction

The Scientific Data Center (SDCC) at Brookhaven National Laboratory serves an increasingly diverse community of more than two thousand users on over twenty projects ranging across the HEP, HENP, Photon Sciences, Astrophysics, LQCD and Condensed Matter disciplines. It provides a large High-Throughput Computing (HTC) farm accessed by the HTCondor batch system, along with experiment-specific job management layers, as well as a set of High-Performance Computing (HPC) clusters accessed via Slurm. Limited interactive access to these compute resources are provided via SSH gateways.

The diversity of resources and access modes at the SDCC allows for a multiplicity of computing modes and workflows. HTC modes rely on embarrassingly parallel processing models to run through large data sets, while modern HPC modes require GPU accelerators and high-speed, low-latency interconnects for parallelized approaches to a certain class of problems. The workflows, in both cases, can be interactive (during the development phase, for example) or can require a job workflow management interface to a batch system for full-scale data processing. Users often face a steep learning curve while buying into a complex workflow model that encompasses code development, compilation, data movement, small-scale interactive runs and large-scale batch processing. This paper describes an effort to flatten this learning curve by providing a flexible Jupyter notebook-based, interactive "Data Analysis As a Service" (DAAS) portal to the full range of SDCC resources.

[*]e-mail: rind@bnl.gov
[**]e-mail: willsk@bnl.gov

2 The Jupyterhub Analysis Portal

The SDCC analysis portal (https://jupyter.sdcc.bnl.gov) is built on a set of Jupyterhub servers configured to access the varied back-end computing resources through a unified interface. The system has been architected to satisfy local cybersecurity constraints for interactive access, while also providing a common interface to both HTC and HPC batch systems. The architectural components are shown in Fig. 1. The user accesses the portal through an authenticating proxy front-end via a web browser. The proxy decouples the Jupyterhub from the SDCC cybersecurity requirements, such as multi-factor authentication, by offloading the authentication component to the Keycloak-based facility infrastructure. The proxy also provides load-balancing across multiple Jupyterhub instances, all of which are deployed and managed by Puppet[1]. The Jupyterhub servers themselves are able to spawn notebooks locally or via the local batch systems. A custom Slurm spawner interface was created on the front-end to handle the account requirement specifications for job submissions on the HPC cluster. More details about the setup are provided in the following subsections.

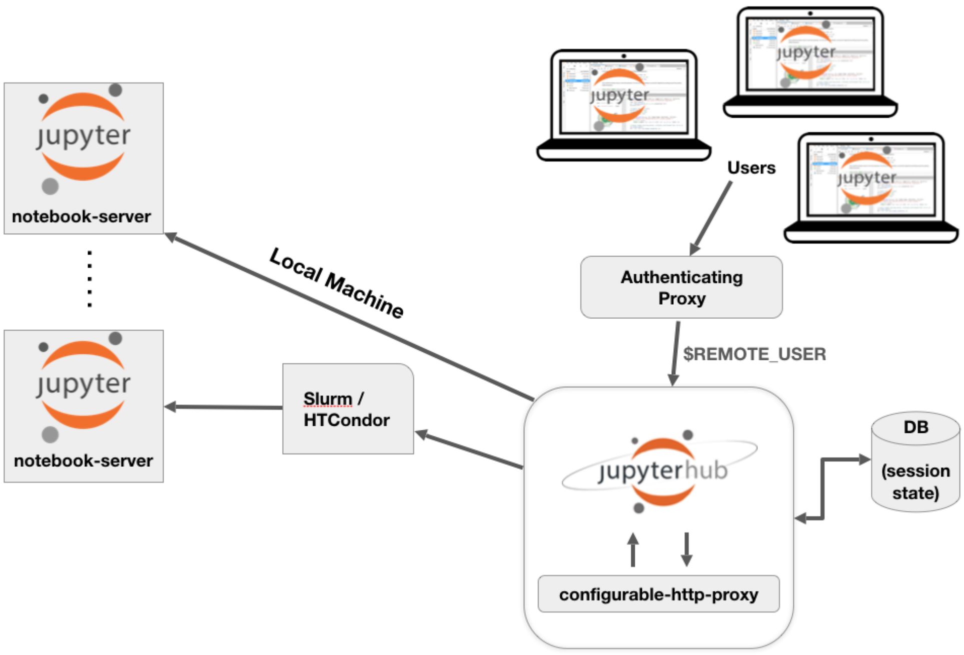
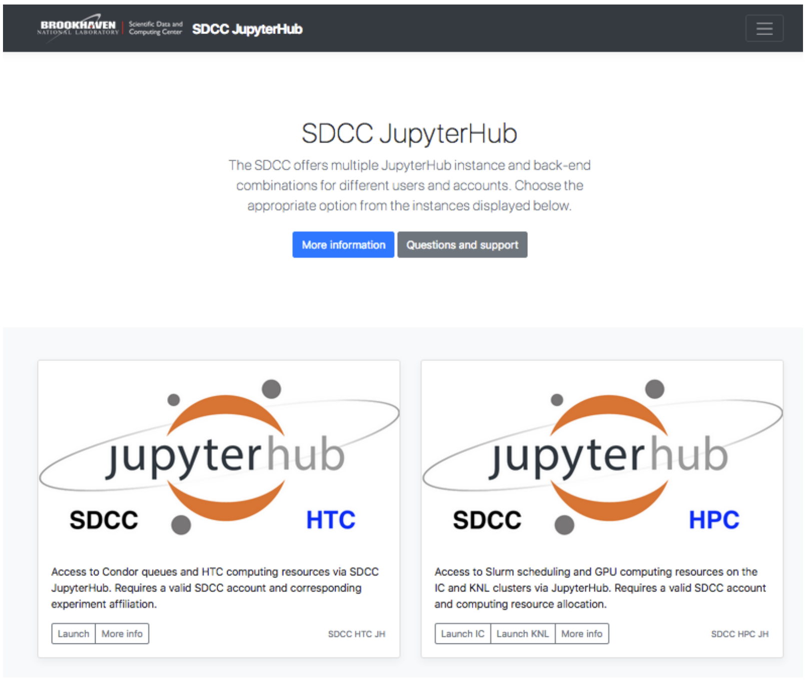


Figure 1. Schematic diagram of the Jupyterhub-based SDCC analysis portal architecture.

2.1 The Front-end Proxy Interface

Authentication to the analysis portal is handled by a proxy that sits in front of several different Jupyterhub instances. Delegating authentication to a front-end is important in our case because it allows the configuration and maintenance of a single MFA-enabled webserver and a large number of diverse backends supporting different use-cases and user-groups. The user is passed in a header to the back-ends, who then use it to determine the user. Most of the instances are one-to-one, but the HTC instance has several backends and a load-balancer configured in front. The Apache server chooses one of the back-ends and assigns a cookie that binds that client to the chosen back-end until the user selects "Log Out". The interface is shown in Fig. 2. The user is able to select either HTC or HPC resources and is given the option of spawning their session as a batch job. Spawning to the batch system provides a ready-made and ideal mechanism for ensuring exclusive, fair access to scarce resources. Containers can enter at the batch level to isolate external users or can be based on choice of environment.



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.

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

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.

Figure 2. SDCC Analysis Portal front-end interface to load-balancing proxy, showing HTC and HPC options.

2.2 Multi-factor Authentication

The SDCC requires two-factor authentication for interactive access to its compute resources. This requirement is fulfilled for the analysis portal by the front-end proxy interface to the Keycloak-based SDCC authentication infrastructure[2]. After the user authenticates with userid and password, the second factor is provided by a one-time challenge code via Google Authenticator or FreeOTP. Once authenticated, the user is provided a token by Keycloak. The initial setup is a simple process using a generated QR code.

2.3 Custom Slurm Spawner

A unique feature of the SDCC analysis portal is its Slurm spawner interface. This custom interface presents a form upon login, allowing the user to choose, based on the contents of the form, which spawner class to launch (Slurm, HTCondor, Local, etc...). The software is a

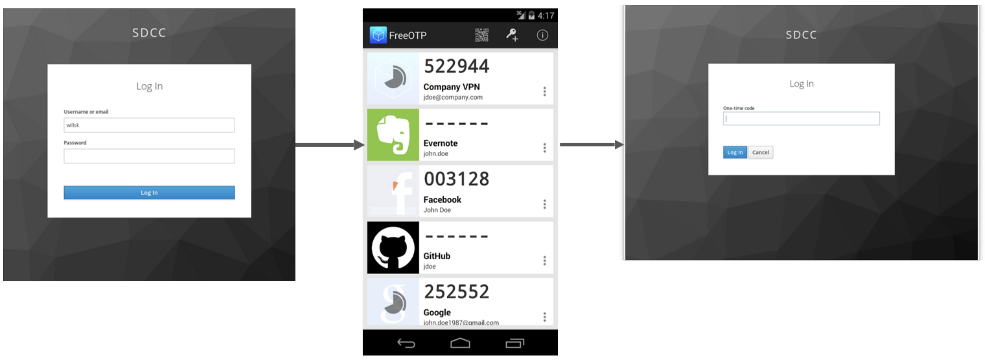
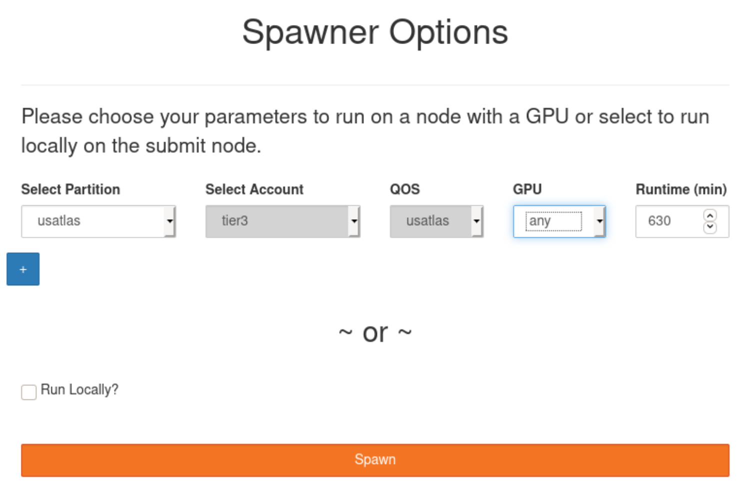


Figure 3. Illustration of the multi-factor authentication process for accessing the SDCC analysis portal.

class framework from which users must derive their own classes with a method that takes the form parameters as input and outputs a spawner class to use (for example in our setup if a user selects a "Run Locally" checkbox the logic will return a LocalProcessSpawner, otherwise it returns a SlurmSpawner).

Our implementation of a custom form (see Fig.4) sits in front of our HPC system and allows users to either submit the notebook session as a Slurm job or to run on the cluster's head node to allow, for example, a parallel scheduler like Dask to orchestrate jobs. The interface is dynamic, examining which accounts and partitions the user can access and providing dropdowns with only those options available. The code for this form is available at [3].


Spawner Options

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

Select Partition	Select Account	QOS	GPU	Runtime (min)
usatlas	tier3	usatlas	any	630

Figure 4. Example of the custom Slurm spawner interface. The user can request to run locally or on the GPU-based HPC cluster. The form only provides options that are available to the given user.

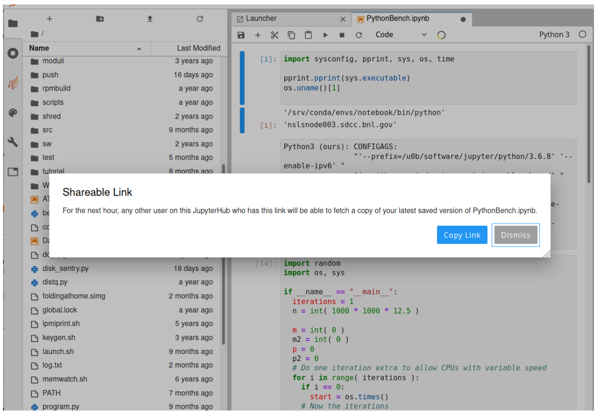
2.4 Orchestration

One of the most efficient use cases for Jupyter at the SDCC will be to act as a type of workflow management interface to the batch resources, replacing the typical array of shell scripts users employ to generate batch-job descriptions. To this end, we developed a proof-of-concept library that then inspired the HTMap Condor Library[4]. Using this library, the standard function map() → htmap(), which takes a list of inputs and breaks down each execution over that list into a separate Condor job, runs them on a cluster, and gathers the results into a list that is then available within the notebook (it serializes the entire state of the application and sends copies to each job).

Another popular interface is Dask, for which Slurm and HTCondor drivers exist[5], allowing an HPC or HTC cluster to be leveraged in this way from a notebook. The goal of these ongoing efforts, in essence, is to abstract away the fact that you are using a batch system at all, thus potentially lowering the bar for user adoption significantly. Dask integration is well-tested and fully functional with Slurm, and currently in beta stage with HTCondor, where it has been tested with simple workflows.

2.5 Notebook Sharing

Another highly-desired feature for Jupyterhub is the ability to share notebooks. A low-effort, short-term notebook sharing plugin, developed by D. Allan[6], enables users to generate a shareable link that allows other users on the same hub to import a copy of the last saved version of a notebook into their sessions. This link encodes the notebook options, such as kernel selection or container, in order to ensure a compatible software environment. Although the link does not bring active state with it, it can be refreshed to reflect the sender's saved updates during the link's period of validity. The link is set to expire after a predetermined time interval. More information on this plugin is available at [6]. See Fig. 5 for an example of its usage.



Name	Last Modified
moduli	3 years ago
push	16 days ago
rpmbuild	a year ago
scripts	a year ago
shred	2 years ago
src	9 months ago
sw	2 years ago
test	5 months ago
tutorial	8 months ago
disk_sentry.py	18 days ago
distq.py	a year ago
foldingathome.simg	2 months ago
global.lock	a year ago
ipmiprint.sh	5 years ago
keygen.sh	3 years ago
launch.sh	9 months ago
log.txt	2 months ago
memwatch.sh	6 years ago
PATH	7 months ago
program.py	9 months ago

```
[1]: import sysconfig, pprint, sys, os, time

     pprint.pprint(sys.executable)
     os.uname()[1]

     '/srv/conda/envs/notebook/bin/python'
[1]: 'nslsnode003.sdcc.bnl.gov'

     Python3 (ours): CONFIGAGS:
                    "'--prefix=/u0b/software/jupyter/python/3.6.8' '--
     enable-ipv6' "
```

Shareable Link

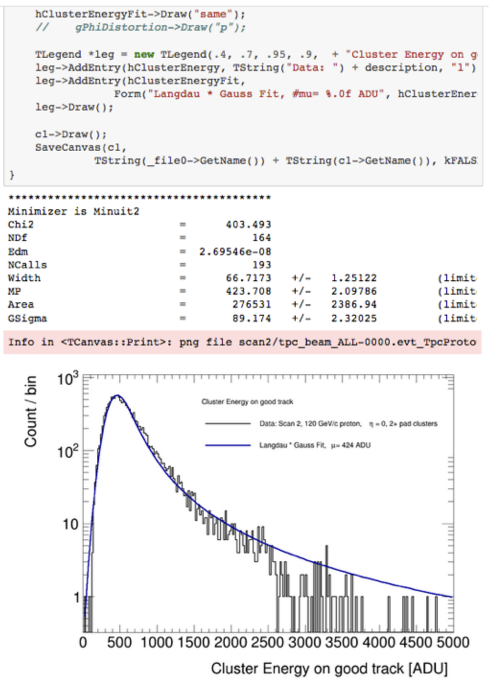
For the next hour, any other user on this JupyterHub who has this link will be able to fetch a copy of your latest saved version of PythonBench.ipynb.

```
[14]: import random
      import os, sys

      if __name__ == "__main__":
        iterations = 1
        n = int( 1000 * 1000 * 12.5 )

        m = int( 0 )
        m2 = int( 0 )
        p = 0
        p2 = 0
        # Do one iteration extra to allow CPUs with variable speed
        for i in range( iterations ):
          if i == 0:
            start = os.times()
          # Now the iterations
```

Figure 5. A temporary notebook share link is generated by right-clicking in the browser.

2.6 Usage

The SDCC analysis portal is being used by a growing number of facilities and experimental collaborations, including NSLS-II, US ATLAS, Belle II, and various supported Nuclear



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

    TLegend *leg = new TLegend(.4, .7, .95, .9,  + "Cluster Energy on g
    leg->AddEntry(hClusterEnergy, TString("Data: ") + description, "l")
    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
```

Count / bin
10^3
10^2
10
1
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Cluster Energy on good track [ADU]
Cluster Energy on good track
Data: Scan 2, 120 GeV/c proton, η = 0, 2+ pad clusters
Langdau * Gauss Fit, μ= 424 ADU

Figure 6. Example sPHENIX ROOT analysis on the SDCC analysis portal (Courtesy J. Huang)

Physics experiments. Fig. 6 illustrates just one example from a recent sPHENIX software tutorial.

3 Conclusions and Outlook

The BNL SDCC provides an easy-to-use Jupyterhub-based analysis portal enabling scientists from multiple disciplines to access its diverse HTC and HPC computing resources. The service is designed to meet facility requirements with minimal impact on the back-end infrastructure. It provides built-in support for experiment-based computing environments with a number of flexible access modes and workflows. The facility is continuing to develop and build upon this system in order to provide new capabilities for user collaboration.

References

[1] J.A. Smith, J.S.D.S. Jr, J. Fetzko, C. Hollowell, H. Ito, M. Karasawa, J. Pryor, T. Rao,W. Strecker-Kellogg, Journal of Physics: Conference Series **396**, 042056 (2012)
[2] S. Misawa, M. Karasawa, J. Hover, "Federated User Account Management," presented at 24th International Conference on Computing in High Energy and Nuclear Physics. Available from https://indico.cern.ch/event/773049/contributions/3473844
[3] W. Strecker-Kellogg, "sdcc_jupyter_spawners" [software]. Available from https://github.com/fubarwrangler/sdcc_jupyter [accessed 2020-03-14]
[4] J. Karpel, S. Sievert, "HTMap" [software], version 0.5.1. Available from https://github.com/htcondor/htmap [accessed 2020-03-14]

[5] Dask-Jobqueue project, "Deploy Dask on Job Queueing systems" [software], version 0.7.0. Available from https://github.com/dask/dask-jobqueue [accessed 2020-03-14]

[6] D. Allan, "JupyterHub Share Link" [software], version 0.1.1. Available from https://github.com/danielballan/jupyterhub-share-link [accessed 2020-03-14]