Deploying an Educational JupyterHub for Exploratory Data Analysis, Visualization, and Running Idealized Weather Models on the Jetstream2 Cloud

Ana V. Espinoza
Unidata Program Center
UCAR
Boulder, USA
0000-0002-6292-073X

Julien Chastang
Unidata Program Center
UCAR
Boulder, USA
0000-0003-2482-3565

W. G. Blumberg
Department of Earth Sciences
Millersville University
Millersville, USA
0000-0001-7620-7962

***Abstract*—The Unidata Program Center dedicates two software engineers to the development and maintenance of a science gateway meant to serve the members of the Earth Systems Science community. Unidata collaborated with one such community member, Dr. Greg Blumberg of the Department of Earth Sciences at Millersville University, in order to provide three undergraduate courses in atmospheric science with access to a custom JupyterHub cluster on the Jetstream2 Cloud boasting preconfigured environments, a shared network drive, the capability to enable machine learning education, and the execution of the Weather Research and Forecasting (WRF) model. The implementation of these features through the Kubernetes orchestration engine is discussed in detail, including initial failures of the Unidata Science Gateway team and the resolution of the issues that arose as a result. The performance of WRF executed at scale using JupyterHub is discussed at a surface level, with more study necessary to make further conclusions. Finally, feedback from Dr. Blumberg, both positive and constructive, is discussed along with specific use cases for the cyberinfrastructure.**

***Index Terms*—science gateways, cyberinfrastructure, meteorology, atmospheric science, education, WRF, Jupyter, JupyterHub, Kubernetes, Docker**

I. Introduction

The Unidata Program Center (simply, Unidata), a part of the University Corporation for Atmospheric Research, develops and maintains a science gateway hosted on the Jetstream2 Cloud [2] serving the Earth Systems Science community. In addition to serving data via software technologies such as TDS, RAMADDA, and EDEX, all fed via the Unidata IDD, the Unidata Science Gateway [1] (USG) accepts requests from professionals and educators for custom JupyterHub clusters, co-located with the aforementioned data, for use in research and educational settings. The USG team has deployed short term JupyterHubs for workshops at conferences serving over 100 people and longer term JupyterHubs for semester long courses.

Unidata is funded by the National Science Foundation's Division of Atmospheric & Geospace Sciences (AGS), Award #1901712.

As the needs of the Unidata community have evolved, so has Unidata's ability to provide new features, such as deploying shared drives for quick access to large data sets or allowing for different compute profiles for synchronous and asynchronous teaching instruction. Many of these features were born from grass roots efforts along with colleagues at multiple academic institutions. One such collaboration was with Dr. Greg Blumberg of Millersville University of Pennsylvania, who approached the USG team with a request for a JupyterHub intended for use by over 40 undergraduate students across three different atmospheric science courses: Stats & Decision Making in Earth Science (ESCI 446), Atmospheric Radiative Transfer (ESCI 345), and Boundary Layers and Turbulence (ESCI 448).

We offer this case study to demonstrate how a JupyterHub deployed by a small team can facilitate education even when complex, computationally intensive software is part of the curriculum. Furthermore, we highlight the importance of soliciting feedback from users, as it can provide valuable insight into how your science gateway fulfills user needs, and where it may be lacking.

II. Motiviation & Requirements

The conversation between Unidata staff and Dr. Blumberg was initiated at the American Meteorological Society's 103rd annual meeting, which took place January 2023 in Denver, CO. Followup discussions took place via email correspondence and this line of communication was kept open for the duration of the spring semester.

A. Standard Requirements

The JupyterHub [3], [4] was to be configured to provide users with a JupyterLab single-user server that provided them with 10 GB of persistent storage, 4 GB of memory, and a guaranteed 1 virtual CPU (vCPU), with a limit of 2 vCPUs. In addition, a 300 GB drive was shared via the Network File System (NFS) protocol to provide students with large

meteorological data sets for exercises and assignments. Three GitHub repositories, one for each course, were desired to be pulled into each single user server. Finally, a conda environment was provided by Dr. Blumberg to the USG team which included packages and tools common to meteorological workflows in python, such as `siphon`, `netcdf4`, `cartopy`, and `wrf-python`.

B. Weather Research and Forecasting

In addition to the standard requirements, Dr. Blumberg made a request for the single user JupyterLab servers to be equipped with one of the idealized scenarios of the Weather Research and Forecasting model, commonly referred to as WRF [5]. Generally ran from a terminal shell, WRF consists of a suite of software executables and tools which can be used in a wide variety of numerical weather prediction (NWP) applications from daily operational weather forecasting, to in-depth research projects such as dissertations, and when guided by a skilled instructor, education. Dr. Blumberg required the idealized WRF "single column" model with a modified "registry" file which sets build time options. These particular modifications would instruct WRF to output specific fields relevant to his teaching objectives during runtime. For Dr. Blumberg's application, WRF would be ran via JupyterLab's built-in terminal shell in the web browser.

The complexity of the WRF system that makes it so multi-faceted also makes it difficult to compile, especially for nontechnical users or those inexperienced in building software from source. In addition, while system administration staff at one's institution may have the knowledge necessary to compile software, they may lack the NWP context in order to provide all the tools and features needed by the end user. Thus, there is a high barrier to entry to run the WRF model, especially for an undergraduate course in meteorology where students aren't typically expected to have all the technical know-how—after all, they are there to learn.

On the other hand, while there are many atmospheric science professionals who have likely compiled and ran WRF dozens, if not hundreds of times, doing so in a multi-user, user friendly environment presents its own challenges, such as acquiring SSL certificates or setting up user authentication. Furthermore even finding the time to acquire the compute resources in the first place, whether it be bare metal or cloud resources, may pose a challenge to university faculty—after all, they are there to do science, and to educate.

The Unidata Science Gateway team is uniquely positioned to offer the type of service that removes many of these aforementioned obstacles. Although the USG team is composed of only two full time software engineers and our time is often split between various projects, one of our main priorities is attending to the needs of our core community members such as Dr. Blumberg.

III. Deployment Strategy

The vast majority of the preliminary steps of our JupyterHub deployments is enabled by the work of Andrea Zonca of SDSC, who has modified the Kubespray project [6] for use on Jetstream2. Through Kubespray, Kubernetes (K8s) [10] and all associated virtual infrastructure—virtual machines, network resources, security groups, etc.—are installed and provisioned via a combination of Ansible and Terraform, respectively. SSL certificates are provided through cert-manager via Letsencrypt [8] and a K8s Ingress resource, rather than a load balancer, handles incoming traffic. The "main" K8s node, which provides the cluster's core functionality such as running the K8s API server, cert-manager pods, ingress controller, and cloud controller, and the "worker" nodes, which are intended to contain JupyterHub resources and pods, both run on the m3.medium VM flavor. This flavor comes equipped with 8 vCPUs, 32 GB of memory, and 60 GB of volume storage intended for use by the operating system, Ubuntu 20.04.

Cloud Configuration	
Cloud Provider	Jetstream2 running Openstack
VM Flavor	`m3.medium`
CPUs per node	8
Memory per node	32 GB
Storage Volume Size	60 GB
Operating System	Ubuntu 20.04
# of Main Nodes	1
# of JupyterHub "Core Nodes"	1
# of Worker Nodes	10
Total # of nodes	12
Certificate Provider	Letsencrypt via `cert-manager`
Cloud Controller	`cloud-provider-openstack`

TABLE I
THE PROVISIONING AND CONFIGURAITON OF THE KUBERNETES CLUSTER TOOK PLACE VIA THE KUBESPRAY PROJECT. CONFIGURATION FOR THE VM FLAVOR AND NUMBER OF NODES IS DONE VIA `CLUSTER.TFVARS` [7], THE TERRAFORM CONFIGURATION FILE. ANSIBLE AUTOMATES THE PROCESS OF INSTALLING KUBERNETES AND CONFIGURING IT TO COMMUNICATE WITH JETSTREAM2 THROUGH THE `CLOUD-PROVIDER-OPENSTACK` CLOUD CONTROLLER. THIS ALLOWS K8S TO, FOR EXAMPLE, CREATE AND MOUNT STORAGE VOLUMES.

The JupyterHub is deployed on top of this K8s cluster and is installed via the K8s package manager, `helm`, and is configured by a `.yaml` file [11]. It is here that many of the requirements described in Section II-A are satisfied. The Docker image [11] containing the JupyterLab instance that users will connect to when logging into the JupyterHub is also referenced in this `.yaml` file. This Docker image is based off of the Jupyter Docker Stacks scipy notebook and the python environment is built using the co-located conda environment file. In addition, the WRF single column model is compiled along with the `netcdf-c` and `netcdf-f` libraries which are necessary as dependencies. While not done in this instance, it is noted that any future Docker images built with WRF or similar software can and should be done in a multi-stage manner to reduce the number of layers and build smaller images. A Persistent Volume Claim was created and mounted to a dedicated Pod which exported this volume via a Service as an NFS mount to single user Pods so all users can access the shared data [9]. Lastly, GitHub OAuth was chosen to provide authentication as it is a mature and reliable technology. A summary of the cluster configuration is found in Table I.

IV. Lessons Learned

As the request for a JupyterHub came to the USG team's attention a short time before the start of the spring semester at Millersville University, the JupyterHub was deployed in two stages. First the Standard Requirements from Section II-A were implemented, with the exception of the NFS mount. Both the NFS mount and the WRF model, described in Section II-B, were not a priority for the beginning of the semester and were instead rolled out several weeks later. A small cluster with a single main node and a single worker node with otherwise identical configuration was deployed and used for development purposes. When a new milestone was reached, for example compiling WRF prior to any changes to the registry file, this was communicated to Dr. Blumberg so that he could verify that everything was functioning as desired.

This approach, however was flawed in one crucial aspect: the model was verified to run on a single user's JupyterLab server at a time, and no attempt was made to stress test the cluster with multiple users running the model at once. When Dr. Blumberg instructed his class to run the model in a synchronous classroom setting, the single-user pods in the cluster all became unreachable via the web browser interface.

When a JupyterHub is deployed in a standard manner as in Section III, K8s is configured to schedule all JupyterHub related pods on the aforementioned worker nodes, with no additional preferences or requirements. This means that the JupyterLab pods run the risk of being scheduled on the same node(s) as the JupyterHub "core pods," a set of pods that provide essential functionality such as running the JupyterHub API and proxy servers. When ran by multiple users, the WRF model used an entire node's resources, leaving nothing to the core pods, resulting in the crash. This was systematically confirmed to be the case by simultaneously running 1, 2, 3, then finally 4 WRF models on the development cluster. Each subsequent addition of a pod running WRF, using 2 vCPUs, resulted in a slower execution time, and finally a crash when the 4$^{\text{th}}$ pod ultimately used the last of the node's compute resources, preventing the JupyterHub core pods from performing their essential functions.

To remedy this, a dedicated node was added to the cluster. This new "core node" was given a K8s "Taint," which prevents pods from being scheduled on it unless they have the corresponding "Toleration," and a "Label," which acts as a "Selector" that must exist on a node for a pods with a "Node Affinity" for that label to be scheduled on that node. The Taint was added to the core node using the standard Kubernetes command: `kubectl taint nodes <node-name> hub.jupyter.org/dedicated=core:NoSchedule`.

The JupyterHub configuration was updated so that its core pods spawn with the appropriate Toleration and Node Affinity. As the single-user pods do not have these attributes by default, they will not spawn on the JupyterHub core node. Figure 1 shows the portion of the JupyterHub configuration file that allows for this behavior.

The new deployment strategy was applied to the development cluster and the same tests were undertaken. The separation of the WRF processes from those of the core pods ensured that the cluster remained operational. An overview of the final cluster arcitechture is shown in Figure 2

```
scheduling:
  corePods:
    tolerations:
      - key: hub.jupyter.org/dedicated
        operator: Equal
        value: core
        effect: NoSchedule
      - key: hub.jupyter.org_dedicated
        operator: Equal
        value: core
        effect: NoSchedule
    nodeAffinity:
      matchNodePurpose: require
```

Fig. 1. JupyterHub configuration that allows for a dedicated "core" node. Note that the core node must be tainted for this to have the desired effect.

# of Models	Min (s)	Max (s)	Mean (s)	Std Dev (s)
1	0.006	0.190	0.052	0.040
2	0.012	5.704	2.245	1.202
3	0.006	8.648	3.667	1.466
4	0.008	12.460	5.492	1.25

TABLE II

Basic statistics for execution time per model time step. While not a rigorous study, it is clear that the relationship is non-linear, with ≈ 1.64 orders of magnitude difference between the execution time per model time step between 1 and 2 simultaneous model execution as opposed to the ≈ 0.30 (i.e. $\log_{10} 2$) orders of magntitude difference that would be expected for a linear relationship.

The WRF single column model takes some initial conditions and integrates the governing equations of our atmosphere forward in time for a user-determined number of iterations. As before, the WRF model suffered a performance loss—using execution time per model time step as the metric—when multiple single-user servers were executing simultaneously. Some basic statistics on these measurements, meant as an exploratory exercise, are shown in Table II. A more robust study would be necessary to account for outliers, e.g. the reported integration time is higher when the current model state is written to disk. However it can be concluded that WRF does not scale well when ran in the manner described. Further studies can be carried out to determine whether removing some of these layers of abstraction and virtualization could restore some performance. Nevertheless, in an asynchronous work environment, i.e. homework assignments or take home projects, this implementation of WRF can still be beneficial to the educational experience.

V. Feedback & Use Cases

Dr. Blumberg's JupyterHub request was one of the largest ever made to the USG team for a semester long course, measured by both the number of courses/concurrent users and the range of capabilities requested. Although the persistently open

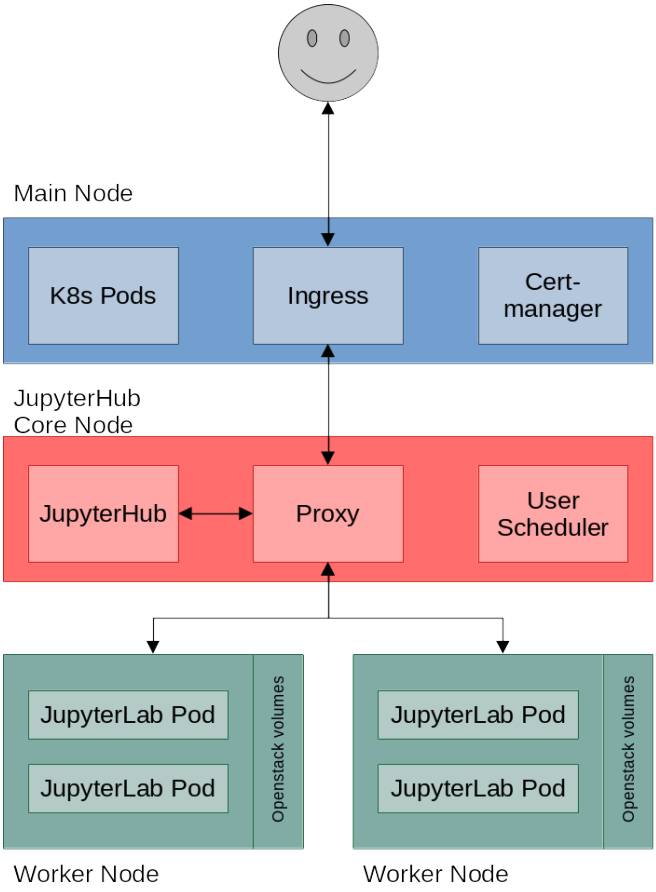

Fig. 2. When a user navigates to the JupyterHub in their web browser, their request makes it to the cluster via the ingress resource on the Main Node. Kubernetes then routes that request to the JupyterHub's proxy server found on the Core Node. If the user is not yet authenticated or makes a JupyterHub specific request—e.g. attempting to access the admin page—they will be routed to the JupyterHub pod. Once they have authenticated with GitHub, the JupyterHub will interface with the user scheduler and Kubernetes to spawn a single user JupyterLab Pod where the user is then routed to access the computing resources. User's access their data in the automatically mounted cloud storage volumes, which are originally created by Kubernetes' interface with Openstack.

line of communication allowed both parties to relay important information about progress updates and downtime, little was known by the USG team about what the cluster was actually being used for. A post-semester interview with Dr. Blumberg was organized to give him a chance to present the USG team with feedback and any information he felt was relevant. The discussions had in this conversation were invaluable, as they provided us with insight to the challenges faced by instructors in a classroom setting and how the USG helped ease that burden. Constructive feedback was also given and was much appreciated, as it made us aware of room for improvement. Furthermore, this experience has encouraged us to seek out feedback in such a manner in future collaborations as it builds trust and shows the USG team's dedication to investment in our community, two principles outlined as fundamental to ensuring adoption of science gateways and cyberinfrastructure [12].

A. *Positive Feedback*

Much of the positive feedback received concerned some of the objectives of all science gateways. In particular, the USG allowed easy and uniform access to cyberinfrastructure to a large number of concurrent users. Dr. Blumberg explained how, when he had led python tutorials in the past, much time and effort was spent simply setting up students' environments. Staging a JupyterHub server solves this issue. However, staff at his institution were unable to successfully configure their own JupyterHub, specifically user authentication, with the limited time that was given to them before the start of the spring semester. The USG made this problem trivial, as students only had to provide Dr. Blumberg with a GitHub username to be added to the allow list via the JupyterHub Admin Panel.

The availability of a shared drive reportedly made using data sets convenient. As the data was co-located with the compute resources, this not only reduced the data access time, but ensured that data would be available which may not always be the case if the data is fetched by a third party data server. As Dr. Blumberg developed course material in the form of notebooks or scripts, the `nbgitpuller` [13] workflow, in which updates to the remote GitHub repository are pulled into single-user servers at launch time, made shipping out these changes to multiple users much easier than would have been possible without this technology. Finally, the integration of the WRF single column model into the JupyterHub made running the model immediately accessible to an entire classroom of students without effort on the users' end.

B. *Constructive Feedback*

There were some areas that were identified as having room for improvement, with the crashing of the server when running WRF being an obvious case. However, it was communicated and mutually understood that this was a new capability that had not previously been thoroughly tested and that unforeseen issues could potentially arise. The USG team is grateful to Dr. Blumberg for his patience during the resolution of these issues, and the opportunity to develop new skills, features, and experience in more rigorous testing procedures.

While `nbgitpuller` made delivering material to students relatively simple, it was not without its criticisms. First, a "manual" restart of a single user server was sometimes necessary to pull in the latest course material. This was due to the implementation of `nbgitpuller` as part of the single user pod creation procedure. The JupyterHub was set to cull/destroy inactive single user pods, implying a recreation of the pod and the retriggering of `nbgitpuller` on a user's next login. If a user had their server persistently active on a browser tab/window, this culling would not occur and `nbgitpuller` would not activate when they returned to their still active JupyterLab session. Thus, a "manual" restart of the single user server was necessary in such cases. The addition of a notebook that calls `nbgitpuller` has since been added to the USG team's JupyterHub deployment workflow.

Secondly, a low maintenance method of delivering assignment answer keys was desired. Currently, the workflow

recommended by the USG team is to push out the answer key to the same repository as the assignment itself. However, as this is a public repository, unless one waits until after the assignment has been graded to push the keys, they would be visible to students from the start, defeating the purpose of a graded assignment. This also makes it difficult for instructors to reuse a repository when reteaching the course. One possible solution to this would be to push the answer keys to a private repository which is then manually cloned by the instructor into a shared drive. This, however, necessitates managing two separate repositories and leaves some simplicity to be desired. Further discussion on this issue is needed.

C. Use Cases

Much of the curriculum for the three courses taught by Dr. Blumberg was guided by the principle of learning by doing. For example, Atmospheric Radiative Transfer (ESCI 345) assignments, exercises, and notebooks were crafted such that students would not only learn the mathematical principles behind radiative transfer, but also be able to visualize their effects on incoming and outgoing radiation through the tweaking of parameters in a Jupyter notebook and reproducing plots.

In Stats & Decision Making in Earth Science (ESCI 446), exploratory data analysis and the application of statistical methods was facilitated by access to the JupyterHub. One course learning objective was to introduce students to the fundamentals of machine learning methods. One exercise had students use parameters derived from atmospheric soundings taken in storm environments which were known to have produced supercell thunderstorms [14], [15]. This data set also included whether or not the observed supercells were tornadic. Part of this data set was used to train two models using the `scikit-learn` libraries which predicted whether a tornado would form. The other part used to evaluate its accuracy. This lesson provided many students an introduction into simple machine learning workflows and procedures.

The WRF model was used in the Boundary Layers and Turbulence (ESCI 448) course for students to gain an intuition for the biases introduced by different parameterizaton schemes, which are packages used to predict unresolveable sub-grid scale atmospheric processes such as a cloud droplet formation or turbulence. One such example is shown plotted in Figure 3. Separately, students were tasked with performing real observations of the planetary boundary layer, the portion of the atmosphere most directly influenced by the earth's surface. These observations, which are collected by a kite-based platform, were provided by NASA's Science Activation Team - the AEROKATS and ROVER Education Network (AREN) [16]. The JupyterHub then was used by the students to create visualizations of their data to identify how key planetary boundary layer features could be influenced by surface features such as the amount of vegetative cover.

VI. Summary

The USG team is continually challenged by community members such as Dr. Greg Blumberg to improve the capabilities of our science gateway and offer new features. In the past, such collaborations would not go much further than the provisioning of the necessary cyberinfrastructure. Although anecdotal, this experience, in which we uncovered information that may prove useful to practitioners of multiple diverse disciplines such as science gateway developers and scientific educators, has convinced us that such conversations are not only useful, but often times necessary to push ones craft.

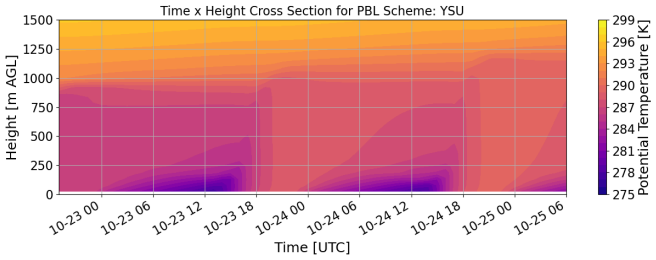

Fig. 3. Students were tasked with running the WRF Single Column Model, a simulation in time of the atmosphere for many vertical levels over one particular location above the Earth's surface. Demonstrated here is how the "potential temperature"—roughly described as the temperature of an air parcel translated "adiabatically" to the surface—varies in the vertical column throughout multiple "diurnal cycles". In this case, the the WRF model was ran with the "YSU" boundary layer parameterization scheme. Students executed the model with a different parameterization scheme by editing a runtime configuration file called `namelist.input` to compare results, all else being equal. Further discussion of this topic is beyond the scope of this paper.

Acknowledgment

We would like to thank Andrea Zonca of the San Diego Supercomputer Center for his continual involvement in pushing forward the Kubespray project applied to Jetstream2 and for his in depth tutorials showing how to best make use of the Zero to JupyterHub with Kubernetes project, both a large portion of our science gateway operations. Additionally, we would like to acknowledge the Jetstream2 team members based out of Indiana University, in particular Jeremy Fischer and Mike Lowe for all their work behind the scenes and fast response times when something does not go as planned.

References

[1] Unidata, (2023): Unidata Science Gateway [software]. Boulder, CO: UCAR/Unidata Program Center. (doi:10.5065/688s-2w73).

[2] David Y. Hancock, Jeremy Fischer, John Michael Lowe, Winona Snapp-Childs, Marlon Pierce, Suresh Marru, J. Eric Coulter, Matthew Vaughn, Brian Beck, Nirav Merchant, Edwin Skidmore, and Gwen Jacobs. 2021. "Jetstream2: Accelerating cloud computing via Jetstream." In Practice and Experience in Advanced Research Computing (PEARC '21). Association for Computing Machinery, New York, NY, USA, Article 11, 1–8. DOI: https://doi.org/10.1145/3437359.3465565

[3] B. E. Granger and F. Pérez, "Jupyter: Thinking and Storytelling With Code and Data," in Computing in Science & Engineering, vol. 23, no. 2, pp. 7-14, 1 March-April 2021, (doi: 10.1109/MCSE.2021.3059263).

[4] Project Jupyter. Zero-to-JupyterHub with Kubernetes. (https://github.com/jupyterhub/zero-to-jupyterhub-k8s).

[5] Skamarock, W. C., J. B. Klemp, J. Dudhia, D. O. Gill, Z. Liu, J. Berner, W. Wang, J. G. Powers, M. G. Duda, D. M. Barker, and X.-Y. Huang, 2019: A Description of the Advanced Research WRF Version 4. NCAR Tech. Note NCAR/TN-556+STR, 145 pp. (doi:10.5065/1dfh-6p97).

[6] Zonca, Andrea. San Diego, CA: San Diego Supercomputer Center. (https://github.com/zonca/jetstream-kubespray).

[7] Zonca, Andrea. San Diego, CA: San Diego Supercomputer Center. (https://github.com/zonca/jetstream_kubespray/blob/branch_v2.21.0/inventory/kubejetstream/cluster.tfvars).
[8] Zonca, Andrea. Setup HTTPS on Kubernetes with Letsencrypt. (https://www.zonca.dev/posts/2020-03-13-setup-https-kubernetes-letsencrypt.html)
[9] Zonca, Andrea. Deploy a NFS server to share data between JupyterHub users on Jetstream. (https://www.zonca.dev/posts/2023-02-06-nfs-server-kubernetes-jetstream)
[10] Kubernetes. (https://github.com/kubernetes/kubernetes).
[11] Unidata. JupyterHub configuration. (https://github.com/Unidata/science-gateway/tree/master/jupyter-images/spring-2023/mvu-spring-2023)
[12] Kee, K. F., Le, B., Jitkajornwanich, K. If you build it, promote it, and they trust you, then they will come: Diffusion strategies for science gateways and cyberinfrastructure adoption to harness big data in the science, technology, engineering, and mathematics (STEM) community. Concurrency Computat Pract Exper. 2021; 33:e6192. https://doi.org/10.1002/cpe.6192
[13] Project Jupyter. nbgitpuller. (https://github.com/jupyterhub/nbgitpuller).
[14] Jewell, R., 2010: The Sounding Analog Retrieval System (SARS). Preprints, 25th Conf. Severe Local Storms, Denver CO.
[15] Blumberg, W. G., K. T. Halbert, T. A. Supinie, P. T. Marsh, R. L. Thompson, and J. A. Hart, 2017: SHARPpy: An Open-Source Sounding Analysis Toolkit for the Atmospheric Sciences. Bull. Amer. Meteor. Soc., 98, 1625–1636, https://doi.org/10.1175/BAMS-D-15-00309.1.
[16] Bland, G., Bydlowski, D., Henry, A. AEROKATS and ROVER Education Network - a Multidisciplinary Approach. AGU Fall Meeting Abstracts. 2018. (https://ui.adsabs.harvard.edu/abs/2018AGUFMED31A..04B/abstract)