Electronic Communications of the EASST
Volume 85 Year 2025

deRSE25 - Selected Contributions of the 5th Conference for Research Software Engineering in Germany

Edited by: René Caspart, Florian Goth, Oliver Karras, Jan Linxweiler, Florian Thiery, Joachim Wuttke

Deploying Infrastructure-as-a-Service at GSI with JupyterHub

Jeremy Wilkinson

DOI: 10.14279/eceasst.v85.2691

Electronic Communications of the EASST (https://eceasst.org).
Published by **Berlin Universities Publishing**
(https://www.berlin-universities-publishing.de/)

Deploying Infrastructure-as-a-Service at GSI with JupyterHub

Jeremy Wilkinson[1]

GSI Helmholtzzentrum für Schwerionenforschung GmbH, Darmstadt, Germany[1]

Abstract: This contribution outlines the proposed deployment of the JupyterHub platform for use with the HPC cluster at the Gesellschaft für Schwerionenforschung (GSI), the Helmholtz centre for heavy-ion research in Darmstadt. Particular focus is given to the security considerations in setting up such a service, and the stack of plugins and external software required to integrate it into the HPC cluster at GSI.

Keywords: RSE, Python, JupyterHub, HPC

1 Introduction

Increasingly large datasets and complex machine learning (ML) tasks necessitate the use of high-performance computing (HPC) clusters for data analysis and theoretical simulations in modern physics research. As many of the most prominent ML libraries and analysis tools are written in Python, Python-based program documentation and data visualisation tools such as Jupyter Notebooks and JupyterLab [JLb] are particularly valuable for both the ease of use and reproducibility of analysis code. However, the specific network setup of institutional HPC clusters, such as firewall rules and internal proxy configurations, can make it difficult for non-expert users to access a Jupyter session running on one of the execution nodes on the cluster, especially from off-site. This can be solved using a service called JupyterHub [JHba], which serves as a single Web-based access point for users to spawn, manage, and interact with Jupyter sessions on dedicated hardware.

This contribution outlines the planned deployment of the JupyterHub service for use with the Virgo HPC cluster at the GSI Helmholtz centre for heavy-ion research in Darmstadt, with a focus on the security considerations in the planned deployment and the stack of existing and custom open-source software needed to bring the service into production.

2 The Virgo HPC cluster at GSI

The Virgo HPC cluster at GSI [Vir] comprises ∼500 compute nodes with differing hardware configurations, with a total of ∼100k CPU cores. A dedicated partition of GPU-equipped nodes is available for the processing of ML and detector reconstruction tasks. This is backed by a Lustre-based network storage system with a capacity of 60 PB.

The submission of computation jobs on the cluster is handled by the Slurm batch job scheduler [Slu]. This system allows the resources to be shared seamlessly between multiple purposes, including as a primary computing facility for the future experiments at the FAIR accelerator, a dedicated analysis facility for the ALICE experiment at the CERN Large Hadron Collider as part of the Worldwide LHC Grid, and for individual tasks submitted by users. Typically, users do not have direct access to connect to the cluster worker nodes, instead logging in to a dedicated Slurm submission node and submitting their jobs from there. Further to this, the software environment is supported by containers using the Apptainer runtime [App], allowing both for the deployment of centrally managed virtual application environments (VAEs) as well as user-created custom containers that allow the use of software without requiring central installation by the cluster administrators.

3 JupyterHub

JupyterHub [JHba] is a web-based hub for serving Jupyter applications to large numbers of users. It consists of a central web server process that manages user authentication and coordinates the deployment of user sessions through a spawner plugin; a web proxy that routes HTTPS connections to the hub interface and to individual notebook sessions; and a series of single-user notebook servers that are started for the user when they log into the service. The key advantage of this service over users running standalone Jupyter Notebook sessions is that it streamlines the process of setting up the session and connecting to the server for the end user, while also ensuring that the connection to a running session is properly secured. It also makes it easier to standardise the resource requirements for Jupyter sessions, as default upper bounds on the available running time, cores, and memory can be set as administrator-defined parameters in the service configuration.

4 Security considerations

When making cluster resources and user data available through a web service such as JupyterHub, it is essential to consider security to prevent unauthorised access by third parties. The key security concepts behind the deployment of JupyterHub at GSI are as follows.

- The HPC cluster nodes, including the submitter nodes that are able to authorise Slurm job submissions, are behind a firewall and only accessible from within the internal GSI network, usually via login with an encrypted SSH key. These nodes should not be exposed directly to the internet by opening a listening port to direct connections from outside of GSI. Instead, the web service should be used as a proxy to access the cluster resources, and should only be authorised to do so when requested by an authenticated user.

- Use of the GSI HPC cluster is subject to having a valid Linux account on the cluster, as well as being subscribed to a Slurm account coordinated by one of the departments at GSI for job accounting. As such, the session on the HPC cluster should be run as the user, with the user's regular file permissions on the cluster storage and with the resource usage correctly assigned to their Slurm account.

- Users at GSI are provided with a Web login account, which is used for several web-based services provided by GSI. This Web login is separate from the user's Linux account, with a different username and password. According to GSI IT security policy, any mapping between a user's Web login and their Linux account to allow access to cluster resources and storage via web services strictly requires 2-factor authentication, in order to prevent unauthorised access by third parties.

- Users should not be able to access each other's Jupyter sessions, to prevent the risk of third parties having access to a user's files.
- The web service should avoid directly handling user credentials, and instead rely on a dedicated Single Sign-On (SSO) provider to authorise access. This limits the potential attack surface of the JupyterHub service itself, as well as limiting the number of components that require direct access to the user database.
- The JupyterHub service should run with minimal privileges on the server and should not be able to directly impersonate other user accounts or perform privileged actions on their behalf, instead using a limited-duration token signed by the SSO provider to log into the submission node on their behalf to spawn the session.
- Any customisable job parameters should be validated to ensure that they conform to the resource limits configured by system administrators, and sanitised to protect against invalid or potentially malicious inputs that may interfere with the job submission or otherwise affect the normal functioning of the service.
- The environment used to run the JupyterLab session should be loaded from a preconfigured image and not import any user-defined environment variables or base software. This is to ensure that the configuration of the connection between the worker node running the JupyterLab session and the JupyterHub server cannot be tampered with [JHbb]. This does not exclude the use of a customised software environment from within an already running JupyterLab session.
- All communication between JupyterHub and the JupyterLab sessions should be encrypted using SSL. This is configurable in JupyterHub using the "internal_ssl" option, allowing JupyterHub to serve as an ephemeral certification authority and sign the SSL certificate to be used by each JupyterLab instance.

5 JupyterHub plugins and supporting software

JupyterHub runs as a Python-based web service that deploys and manages JupyterLab sessions on behalf of users. It is supported by a set of "authenticator" and "spawner" plugins that handle the user login into the service and the actual creation and management of the user sessions, respectively. Default generic implementations of the authenticator and spawner plugins are included with the JupyterHub package, and mostly support the creation of JupyterLab sessions on a local host, using local Linux accounts authenticated by a PAM module. This functionality can be extended by installing additional plugins from the community or implementing one according to JupyterHub's API. The proposed scheme for logging in and spawning a session in the GSI JupyterHub deployment is outlined in Fig. 1. In the following, the additional plugins and external software used to support the JupyterHub deployment are outlined.

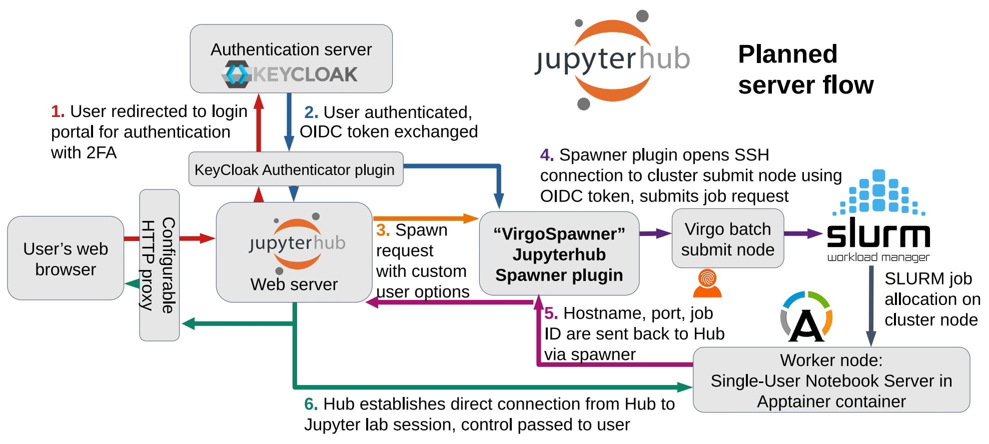
Authentication server
KEYCLOAK
jupyterhub
Planned server flow
1. User redirected to login portal for authentication with 2FA
2. User authenticated, OIDC token exchanged
KeyCloak Authenticator plugin
4. Spawner plugin opens SSH connection to cluster submit node using OIDC token, submits job request
User's web browser
Configurable HTTP proxy
jupyterhub Web server
3. Spawn request with custom user options
"VirgoSpawner" Jupyterhub Spawner plugin
Virgo batch submit node
slurm workload manager
SLURM job allocation on cluster node
5. Hostname, port, job ID are sent back to Hub via spawner
Worker node: Single-User Notebook Server in Apptainer container
6. Hub establishes direct connection from Hub to Jupyter lab session, control passed to user

Figure 1: Schematic of login and spawner workflow for the JupyterHub service.

5.1 KeyCloakAuthenticator plugin

To ensure that the GSI HPC facilities and data are not exposed to unauthorised access through web-based services, the mapping between the GSI Web login account and the Linux cluster account strictly requires two-factor authentication. This will be handled by the Keycloak [Key] service at GSI, which is an open-source SSO portal that provides an interface with the LDAP user database and serves as an OpenID Connect (OIDC) token provider.

The KeyCloakAuthenticator [KCA] plugin for JupyterHub, developed at CERN for its Service for Web-based ANalysis (SWAN) deployment, allows communication between the JupyterHub service instance and the Keycloak server. It is configured with the URI for the Keycloak realm it should query, along with a client ID and secret for client verification. The Keycloak server's API endpoints for authorisation and token generation are automatically determined via the server's Well-Known OpenID Configuration endpoint.

On connecting to the JupyterHub service (step 1 in Fig. 1), if there is no existing session cookie in the user's browser, the user is redirected to the SSO portal for authentication. On the side of the Keycloak SSO service, the client is configured to require two-factor authentication from the user: the user's GSI Web login username and password, followed by a one-time passcode from a preconfigured OTP generator.

After authentication (step 2 in Fig. 1), the KeyCloakAuthenticator plugin receives a login token from Keycloak containing the user's defined roles as mapped in Keycloak, and authorises the login if the correct role for JupyterHub access is found. At the same time, a token exchange is performed to retrieve a signed, limited-duration OIDC token for the Keycloak client registered to the SSH service, containing the Linux username retrieved from LDAP. This token is encrypted and stored in JupyterHub's session database for use by the spawner plugin to deploy the JupyterLab session.

5.2 ssh-oidc

The ssh-oidc toolset [SSH] is used to allow for the SSH connection between the JupyterHub server and the Slurm submission node on the cluster, using the OIDC token obtained from Keycloak as a login credential. In this way, we avoid the need for the user to provide any Linux login credentials such as confidential SSH private keys to JupyterHub directly, instead relying on the Keycloak service to handle the login authorisation and handling access tokens internally within the service.

On the Slurm submission node where the SSH connection is targeted, the ssh-oidc service that runs to handle connection via OIDC tokens consists of two main components: Motley Cue [MCu] and pam-ssh-oidc [SSH]. Motley Cue (the Mapper Oidc To Local idEntitY with loCal User managEment) handles the mapping between the username and Keycloak unique user ID (UUID) claims in the OIDC token and the Linux account on the target system, which can be backed either by a local UNIX account database or (as in our case) an LDAP server. The signature of the token is validated against the Keycloak server's public key to prevent token manipulation attacks. Motley Cue performs a query of the LDAP database for the user's UUID in the Keycloak service, which is stored as an additional field in the user's LDAP entry, and uses this to authorise the login for that Linux username.

The pam-ssh-oidc PAM module provides the means to use the token as a login credential when the SSH connection is established. This is provided as a PAM 'KeyboardInteractive' prompt, accepting a base-64-encoded JSON Web Token (JWT) as a response. Notably, the token itself must be fewer than 1023 characters in order to be usable for the login, so the token scope for the SSH audience is limited to only the necessary fields for verification.

5.3 Apptainer

Apptainer [App] is a file-based container runtime allowing simple deployment of containerised environments. As mentioned in Section 4 and advised in the JupyterHub security documentation [JHbb], the environment and version of JupyterLab used by JupyterHub's spawner should be controlled by the service administrators, in order to ensure that the establishment of the connection between JupyterLab and JupyterHub cannot be tampered with. This is done by providing a fixed container image containing the JupyterLab installation in a dedicated software repository on the cluster. In addition, we ensure that when the container is loaded, it does not source any environment variables from user-defined scripts such as .profile or .bashrc. When the Apptainer session is started, the Lustre network storage element is bound to the session, so that the user is able to access and load any stored Python kernels from their own directories there. This means that the central JupyterLab container can be kept minimal, and not require the installation of extra Python packages on the side of the cluster administrators. Once the session has been spawned, it is then possible for the user to run their own nested Apptainer container within it, allowing them to run customised software within the job without being able to modify the actual JupyterLab process.

5.4 JupyterHub VirgoSpawner plugin

The spawner plugin of JupyterHub is the component that allows the service to create, destroy, and manage instances of JupyterLab on behalf of the user. Several generic implementations of the spawner plugin exist, allowing for session spawning on the local host, via SSH, or via a batch submission system. Other existing implementations such as the OutpostSpawner [Out] in use for the Jupyter4NFDI service at the Jülich Super Computing Centre [Jü] allow for the JupyterHub web service to remotely spawn Jupyter sessions. However, the OutpostSpawner solution requires a Kubernetes cluster, which is not presently available at GSI and so cannot be directly deployed in this case.

To perform the spawning of jobs over SSH and the remote Slurm controller, a custom ”VirgoSpawner” Python plugin has been developed at GSI. This is able to operate independently from the Slurm controller, without any persistent JupyterHub components running on the Slurm submission node or modifications to the GSI network setup. The spawning of the job proceeds as follows (steps 3–6 in Fig. 1):

(1) The authenticated user is presented with a web form with several customisable job parameters, for example the required CPU/memory resources and whether their task requires a GPU.

(2) After submission, the user inputs are validated against administrator-defined ranges and sanitised to ensure that no malicious or nonsensical job submission parameters are passed to the job submission. If there are any validation errors, the user is redirected back to (1) with an error message.

(3) An SSH connection is established from JupyterHub to the Slurm submission node, using the OIDC token exchanged during the authentication step (see Section 5.1) as the credential. This is done using an extension to the asynchronous SSH Python package asyncssh, extending its SSH client class to support the use of the token in the KeyboardInteractive authentication challenge from pam-ssh-oidc (see Section 5.2). The following steps are then run as a Bash script through the established SSH session.

(4) A Slurm allocation is made for the job on the user’s Slurm account, using the resource parameters given in (1).

(5) A script is run within the allocated job to determine a random free TCP port on the assigned cluster node. This port will be used as the listening port for the JupyterLab session, set via an environment variable. This allows the presence of multiple separate JupyterLab sessions on the same cluster node without clashes in connection.

(6) The JupyterLab session is started inside a dedicated Apptainer container on the assigned node. A secret key for the session and a trust bundle for SSL encryption are automatically generated by JupyterHub and provided to the spawned JupyterLab process. The secret key is passed as an environment variable and used in the HTTPS header for all requests between JupyterHub and the JupyterLab instance to secure the session and prevent access from outside of JupyterHub or by other users within the network.

(7) The SSH session returns a string containing the hostname, port, and Slurm job ID assigned to the notebook session. These parameters are parsed from the response and stored in the JupyterHub session database for session management.

(8) The JupyterLab session attempts to establish a connection to the JupyterHub server. Once JupyterHub receives this request, it communicates back to the job host through its listening port with the secret key established in (6), and the session is established. The user is presented with their JupyterLab notebook running on the HPC cluster through the JupyterHub webpage. If the connection is not established within a pre-defined timeout, then the spawning is considered as failed and the user is redirected back to (1).

At all stages of this process, if an error is determined in the job deployment or in establishing an HTTPS connection between the JupyterLab instance and JupyterHub, then JupyterHub sends a signal to Slurm to terminate the job allocation and deletes the session from its local store. An error message is displayed through the JupyterHub webpage to indicate to the user if the issue was e.g. due to invalid job parameters, a missing Slurm account on the remote host, or a timeout in job creation, and prompting them to retry the job creation.

5.5 Deployment of service configuration

The installation and configuration of the JupyterHub service on the web server and of the ssh-oidc tools on the Slurm submission node will be handled by an Ansible playbook [Ans], providing a repeatable and idempotent method for deploying the services and bringing them into commission. The configuration files and web templates will be kept in an internal GSI Git repository for version management and ease of maintenance. The Apptainer container to be used as the environment for the individual JupyterLab sessions will be built and stored on a shared read-only filesystem, and can be rebuilt and re-deployed as needed for updates to the container's operating system.

6 Summary

The deployment of the JupyterHub service at GSI will streamline the interactive use of Python software for research activities on the local HPC cluster. The setup for this service has been conceived with security and ease of use as two main priorities, and this can be achieved through a combination of existing open-source software packages and custom plugins in order to integrate it with the specific configuration of the HPC cluster at GSI. The service is planned to go into testing and eventual deployment during 2025, and the accompanying VirgoSpawner plugin will be published as an open-source package once it is ready to be used in production.

Bibliography

[Ans] Ansible documentation. https://docs.ansible.com.

[App] Apptainer runtime. https://apptainer.org.

[JHba] JupyterHub documentation. https://jupyterhub.readthedocs.io.

[JHbb] JupyterHub documentation – Security overview. https://jupyterhub.readthedocs.io/en/stable/explanation/websecurity.html.

[JLb] JupyterLab documentation. https://jupyterlab.readthedocs.io.

[Jü] Jupyter4NFDI, Forschungszentrum Jülich. https://nfdi-jupyter.de/.

[KCA] KeyCloakAuthenticator Python package. https://pypi.org/project/keycloakauthenticator.

[Key] Keycloak homepage. https://www.keycloak.org.

[MCu] motley-cue homepage. https://dianagudu.github.io/motley_cue.

[Out] JupyterHub OutpostSpawner documentation. https://jupyterhub-outpostspawner.readthedocs.io/en/latest/.

[Slu] Slurm workload manager. https://slurm.schedmd.com.

[SSH] ssh-oidc GitHub repository. https://github.com/EOSC-synergy/ssh-oidc.

[Vir] GSI Virgo documentation. https://virgo-docs.hpc.gsi.de.

Acronyms

API Application Programming Interface.

CPU Central Processing Unit.

FAIR Facility for Antiproton and Ion Research.

GPU Graphics Processing Unit.

GSI Gesellschaft für Schwerionenforschung (Helmholtz Centre for Nuclear Research, Darmstadt).

HPC High-Performance Computing.

HTTPS Hypertext Transfer Protocol Secure.

JSON JavaScript Object Notation.

JWT JSON Web Token.

LDAP Lightweight Directory Access Protocol.

ML	Machine Learning.
OIDC	OpenID Connect.
OTP	One-Time Password.
PAM	Pluggable Authentication Modules.
SSH	Secure Shell Protocol.
SSL	Secure Sockets Layer.
SSO	Single Sign-On.
TCP	Transmission Control Protocol.
URI	Uniform Resource Identifier.
UUID	Universally Unique Identifier.
VAE	Virtual Application Environment.