Mejorar las infraestructuras HPC para el análisis interactivo de datos climáticos.

Enhance HPC infrastructures for interactive climate data analysis.

Trabajo de Fin de Máster
para acceder al
Máster Interuniversitario en Ciencia de Datos

Curso Académico 2022-2023

Autor: Juan Vélez Velasco
Director: Ezequiel Cimadevilla Álvarez
Codirector: Antonio S. Cofiño
Julio - 2023

Agradecimientos

Tras cuatro años de carrera y un año de máster, creo que doy por finalizada la vida universitaria, que puede ser de las mejores que pueden existir. Con este trabajo acabo el máster de Data Science y empiezo la vida laboral, que no me llama tanto como la vida universitaria. En esta etapa he conocido gente que me llevo para toda la vida, gente con la que he compartido días y días durante semanas y semanas sin esfuerzo ninguno. Gente con la que he viajado y voy a viajar, que para mi es unas de las cosas que más feliz me hace. Creo que no habría disfrutado tanto este proceso de no ser por ellos, por eso quiero agradecerles también todo este tiempo a mis amigos de la uni, en especial a "Los Chemas", Chus, Borja.. y muchos más que ya saben quienes son, y a Ángela. También a todos los que no tienen nada que ver con la uni pero que siempre están cuando se les necesita.

A mi familia, que me han hecho ser quien soy y me han apoyado siempre en cualquier situación.

A mis profesores por ayudarme siempre que han podido.

A mis dos tutores de este proyecto, Ezequiel y Antonio, que fueron tutores míos también en el Trabajo de Fin de Grado y lo vuelven a ser esta vez, gracias a ellos elegí hacer este máster y estoy muy contento de seguir sus consejos. Han sido la clave en este proyecto, ya que estando mucho mas experimentados en estos temas, me han podido sacar de cualquier problema o duda que tuviese. Una vez más, me han empujado en mas de una ocasión. Sus enormes conocimientos en este ámbito y sus ganas de ayudar y mejorar, me han hecho aprender mucho a mi también. Les tengo mucho cariño y quien sabe si en algún futuro nuestros caminos se vuelven a cruzar. La verdad es que la experiencia trabajando junto a ellos dos me ha sido muy gratificante. Tampoco me quiero olvidar de Nuria, que me ha ayudado y salvado en mas de una ocasión en la que me cargaba la máquina y siempre fue capaz de solucionar.

Resumen

Este trabajo Fin de Máster (TFM) se centra en la implementación de JupyterHub en la infraestructura de computación de alto rendimiento (HPC) del grupo de meteorología de la Universidad de Cantabria y el Consejo Superior de Investigaciones Científicas (CSIC).
El contexto de este trabajo está centrado en el aumento de aplicaciones que hacen uso de Machine Learning en el campo del clima y meteorología, impulsado por el aumento de datos masivos y de las capacidades computacionales de los HPC.
La motivación del proyecto radica en que la actual infraestructura del grupo de meteorología UC/CSIC, a pesar de contar con un petabyte de almacenamiento y nodos de cálculo accesibles mediante un sistema de colas, carece de una interfaz web que facilite el análisis de datos de forma gráfica e interactiva para los investigadores.
El objetivo de este TFM consiste en la implementación de un data hub para el análisis interactivo de datos climáticos en la infraestructura del HPC del grupo de meteorología (UC/CSIC). El data hub dispondrá de las herramientas necesarias para que los usuarios puedan realizar análisis de datos a través de una interfaz web interactiva, aprovechando los nodos del *clúster* del HPC.

Palabras clave

JupyterHub, HPC, análisis de datos, Grupo de Meteorología de Santander, sistema de colas, nodos de cómputo.

Abstract

This Master Thesis (TFM) focuses on the implementation of JupyterHub in the High Performance Computing (HPC) infrastructure of the meteorology group of the University of Cantabria. High Performance Computing infrastructure of the meteorology group of the University of Cantabria and the Consejo Superior de Investigaciones Cientificas (CSIC).
The context of this work is focused on the increase of applications that make use of Machine Learning in the field of weather and climate. Learning in the field of climate and meteorology, driven by the increase of massive data and of the computational capabilities of HPCs. The motivation of the project lies in the fact that the current infrastructure of the UC/CSIC, despite having one petabyte of storage and computational nodes accessible through a queueing system, lacks a web interface to facilitate data analysis in a graphical and interactive way for researchers.
The objective of this TFM is the implementation of a climate data hub for the interactive analysis of climate data in the HPC infrastructure of the meteorology group (UC/CSIC). The data-space will have the necessary tools for users to perform data analysis through an interactive web interface, taking advantage of the HPC cluster nodes.

Keywords

JupyterHub, HPC, data analysis, Grupo de Meteorologia de Santander, queueing system, compute nodes.

Índice

Índice de figuras

1. Introducción

1.1. Contexto

Las aplicaciones de Machine Learning (ML) en el campo del clima y la meteorología están ganando impulso debido a dos factores principales: El auge de los datos masivos y el aumento de la capacidad de computación de alto rendimiento (HPC).

- Auge de datos masivos: Las tecnologías modernas, incluyendo satélites, estaciones meteorológicas y los avances en el desarrollo de modelos del clima están dando lugar a una gran cantidad de datos climáticos y meteorológicos. El análisis de estos datos climáticos es cada vez más complejo. Además, el Machine Learning es especialmente útil para analizar estas ingentes cantidades de datos y descubrir patrones y relaciones que no son evidentes mediante el uso de métodos tradicionales. Los algoritmos de ML ayudan a identificar patrones que son indicativos de fenómenos climáticos específicos, como los ciclones tropicales, y pueden ser utilizados también para realizar predicciones sobre futuros eventos climáticos basándose en observaciones de datos históricos.

  Los datos de clima y tiempo son un claro ejemplo de datos masivos gracias a varias características, entre ellas:

  - Volumen: Los datos de clima y tiempo son extremamente grandes en términos de tamaño.
  - Velocidad: Estos datos se generan continuamente por lo que se actualizan cada pocos minutos.
  - Variedad: Asimismo, estos datos adoptan una gran variedad de formas, incluyendo imágenes satelitales, mediciones de estaciones meteorológicas y datos de modelos climáticos.

- Aumento de la capacidad de HPC: La computación de alto rendimiento (HPC) se refiere a la utilización de supercomputadores y clústers para realizar cálculos avanzados a una velocidad que no es posible con los computadores normales. Las técnicas de Machine Learning a menudo requieren una gran cantidad de cálculos, especialmente cuando se trata de conjuntos de datos masivos. Por lo tanto, el aumento de la capacidad de HPC ha propiciado que sea práctico aplicar el Machine Learning a problemas de clima y tiempo. Con HPC, los investigadores pueden entrenar modelos más complejos y precisos, y pueden ejecutarlo a mayor velocidad.

  La computación de alto rendimiento (HPC, por sus siglas en inglés) es una tecnología esencial en muchos campos de la ciencia y la ingeniería que requieren una gran cantidad de cálculos, como sucede en la meteorología y la climatología, que a menudo requieren simular procesos físicos complejos en la atmósfera y los océanos.

  El aumento de la capacidad de HPC ha surgido gracias a:

  - Aumento de la potencia de cálculo: La transformación eficiente de los algoritmos en el clima y la meteorología ha permitido la realización de más cálculos con la misma cantidad de recursos de computación.

- Aumento de la eficiencia de los algoritmos: Los algoritmos utilizados en la meteorología y la climatología, incluyendo los algoritmos de ML, se han vuelto más eficientes. Esto significa que pueden realizar más cálculos con la misma cantidad de recursos de computación.

El aumento de la capacidad de HPC ha permitido a los investigadores aplicar técnicas de ML a problemas de clima y tiempo de formas que antes no eran posibles. Tal es así, que pueden entrenar modelos con grandes conjuntos de datos climáticos y meteorológicos, lo que contribuye a realizar modelos más precisos. También pueden usar estos algoritmos para analizar los resultados de las simulaciones, lo que puede ayudar a mejorar nuestra comprensión de los procesos climáticos. Además, el HPC puede permitir a los investigadores ejecutar simulaciones climáticas a una resolución más alta, lo que puede llevar a predicciones más precisas.

1.2. Motivación y Objetivo

Actualmente, la infraestructura HPC del grupo de meteorología (UC/CSIC) [5] consta de un petabyte de almacenamiento y nodos de cómputo accesibles mediante un sistema de colas. Sin embargo, esta infraestructura carece de una interfaz web que permita el análisis de datos de forma gráfica e interactiva para los investigadores.

El objetivo de este proyecto es implementar JupyterHub en la infraestructura HPC del grupo de meteorología (UC/CSIC) para poner fin a estas limitaciones. JupyterHub es una plataforma que permite a los usuarios interactuar con Jupyter Notebooks a través de una interfaz web, proporcionando así un entorno fácil e usar para el análisis interactivo de datos. Al desplegar JupyterHub, se les proporciona a los investigadores una forma más intuitiva y visual de trabajar con sus datos.

Además, JupyterHub, permitirá aprovechar la capacidad de cómputo del HPC, haciendo uso de los nodos mediante un sistema de colas. Los investigadores podrán ejecutar sus *notebooks* en el servidor y acceder a ellos desde cualquier lugar, lo que favorece el intercambio de análisis y resultados.

Por último, JupyterHub también apoya la reproducibilidad de la investigación. Los notebooks de Jupyter permiten combinar código, resultados, visualizaciones y texto explicativo en un solo documento, lo que facilita el seguimiento de la obtención de resultados, y así, permitir a otros investigadores reutilizar y construir sobre el trabajo previo. De esta forma, se apoya la reproducción de proyectos.

2. Estado del arte

2.1. Introducción

Las ciencias climáticas y meteorológicas llevan mucho tiempo unidas a las infraestructuras HPC. Esto se debe a la gran complejidad de los algoritmos usados y de la necesidad de usar computadores con grandes capacidades. Por ello, son necesarias aplicaciones software que permitan a los investigadores la posibilidades de manejar los *notebooks* de una manera sencilla y de acceder a *clústeres* de computación de formas mas eficaces. Existen varias aplicaciones que cumplen con estos requisitos, entre otras están Amazon SageMaker, Google Vertex AI Workbench o JupyterHub. También existe la posibilidad de usar Google Colab, pero solo permite usar conjuntos de datos limitados, además, tienen capacidades de cálculo y de memoria limitadas, no se integran con otras herramientas de desarrollo de software, como el control de versiones, no tienen control de acceso basado en roles y tienen capacidades limitadas a la hora de trabajar con grandes conjuntos de datos. [2]

Actualmente, en el HPC del SMG, el sistema de archivos usados es Lustre, un sistema de archivos distribuido de código abierto diseñado para aplicaciones de computación de alto rendimiento a gran escala. Lustre se destaca por su capacidad para soportar la escritura y lectura de datos a alta velocidad en paralelo desde y hacia muchos discos a la vez. Este sistema puede escalar para manejar decenas de miles de nodos clientes y petabytes de datos.
Las características principales de Lustre son:

- **Arquitectura escalable:** Lustre es un sistema de archivos que escala horizontalmente, esto aumenta la capacidad de almacenamiento y rendimiento. Se puede añadir más servidores a medida que la necesidad de almacenamiento y rendimiento crece.
- **Diseño orientado a la computación de alto rendimiento (HPC):** Lustre está diseñado para entornos de computación de alto rendimiento, donde se necesita un acceso rápido y eficiente a grandes volúmenes de datos. Soporta la transferencia de grandes cantidades de datos entre los nodos de almacenamiento y los nodos de cálculo a alta velocidad.
- **Almacenamiento distribuido:** Lustre distribuye los datos entre varios servidores de almacenamiento, lo que permite un acceso paralelo a los datos. Esto significa que muchos clientes pueden leer y escribir datos simultáneamente sin cuellos de botella de rendimiento.
- **Compatibilidades con POSIX:** Lustre es compatible con la interfaz estándar de sistema de archivos POSIX, lo que significa que las aplicaciones no necesitan ser modificadas para usarlo.
- **Control de metadatos separado:** En Lustre, los metadatos (como los nombres de los archivos, los permisos, etc.) se almacenan por separado de los datos reales. Esto permite un acceso más rápido a los metadatos y una mayor escalabilidad.

El sistema de colas utilizado en el HPC es el **Portable Bacth System**, más conocido como PBS, es un sistema software de código abierto que se utiliza para gestionar trabajos de computación en clústeres, supercomputadores y sistemas de computación de alto rendimiento. PBS proporciona las capacidades necesarias para programar y monitorizar trabajos, así como para administrar y

optimizar el uso de los recursos disponibles.
Las características principales del sistema de colas PBS son:

- **Colas de trabajos**: Los usuarios pueden enviar trabajos a una cola y PBS se encargará de ejecutarlos en el orden adecuado, teniendo en cuenta las prioridades y las políticas establecidas.
- **Planificación de recursos**: PBS puede asignar recursos específicos a los trabajos en función de sus necesidades. Los recursos pueden incluir cosas como la cantidad de memoria necesaria, el número de núcleos de CPU, el tiempo de ejecución, etc.
- **Gestión de la carga de trabajo**: PBS puede equilibrar la carga de trabajo en el sistema, asegurándose de que los recursos se utilizan de manera eficiente y que no se producen cuellos de botella.
- **Soporte para diferentes políticas de planificación**: PBS admite diferentes políticas de planificación, incluyendo la planificación basada en prioridades, la planificación basada en cuotas, la planificación basada en reservas, etc. Esto permite a los administradores del sistema adaptar el comportamiento del sistema a sus necesidades específicas.
- **Interfaz de línea de comandos**: PBS proporciona una interfaz de línea de comandos para que los usuarios puedan interactuar con el sistema, enviar trabajos, consultar el estado de los trabajos, etc.

En resumen, el sistema PBS es una herramienta poderosa y flexible para la gestión de trabajos en entornos de computación de alto rendimiento. Su capacidad para gestionar y programar eficientemente los recursos del sistema permite a los usuarios ejecutar trabajos complejos y demandantes de recursos de forma eficaz. Sin embargo, es importante destacar que PBS se maneja principalmente a través de una interfaz de línea de comandos. Aunque esta interfaz es altamente funcional y versátil, puede presentar una curva de aprendizaje para los usuarios menos familiarizados con este tipo de interacción. Además, no proporciona una interfaz gráfica directa para acceder y administrar los nodos de la cola. En cambio, los usuarios deben interactuar con el sistema principalmente a través de comandos textuales. A pesar de este desafío, el uso eficiente de PBS puede ofrecer beneficios significativos en el manejo de trabajos en un entorno de HPC.

Los datos climáticos son almacenados de forma habitual en el formato **NetCDF**. NetCDF es un formato de archivos que está diseñado para ser eficiente y efectivo cuando se trabaja con grandes volúmenes de datos. En un sistema HPC, donde se pueden manejar y procesas grandes cantidad de datos, el uso de un formato como NetCDF puede hacer que el almacenamiento y el acceso a datos sea mucho más eficiente. Además como NetCDF es un formato auto-descriptivo, puede facilitar el intercambio y la interpretación de los datos entre diferentes usuarios en un entorno HPC. Así, en un sistema HPC, el análisis de datos climáticos se puede llevar a cabo de manera eficiente y efectiva, utilizando herramientas como PBS para la gestión de trabajos y Lustre para el almacenamiento de datos. Y con el formato de archivo NetCDF, estos datos pueden ser almacenados y accedidos de manera eficiente, facilitando el intercambio de datos y el análisis colaborativa.

El SMG cuenta con un amplio repositorio de datos climáticos de distintas iniciativas entre las que se encuentran CMIP [6] y CORDEX. Los datos de clima proceden de distintos repositorios de

datos, siendo la federación ESGF [7] una de las fuentes principales de datos. Además, el SMG cuenta con versiones propias de datasets del interés como es el caso de ERA5 [8]. El SMG ofrece servicios de análisis de datos de clima mediante acceso remoto, además del sistema de colas utilizado de forma interna por los miembros del grupo. La implementación de un servicio JupyterHub permite una forma más rápida, cómoda y eficiente de realizar tareas de análisis de datos para los miembros del grupo de investigación, siguiendo la tendencia en la migración hacia este tipo de entornos en las ciencias del clima [9].

2.2. Pangeo

Pangeo Forge [10] es una plataforma impulsada por la comunidad que acelera la ciencia al proporcionar marcos de trabajo y una infraestructura informática en la nube para producir y acceder a datos listos para el análisis y optimizados para la nube (**ARCO**). Su objetivo es abrir la puerta a una comunidad más amplia de científicos para participar en la producción de datos ARCO y fomentar la colaboración y la participación abierta. Pangeo Forge es una plataforma de código abierto compuesta por diferentes componentes modulares.

En el campo del análisis de datos ambientales en la nube, se han desarrollado diversas plataformas, y se reconoce la necesidad de contar con datos listos para el análisis y optimizados para la nube (**ARCO**) en estas plataformas. Sin embargo, la producción de datos ARCO ha sido un proceso complejo y laborioso, que requiere conocimientos técnicos especializados. Pangeo se ha diseñado para reducir esta carga y facilitar la contribución de científicos con conocimientos específicos en la curación de datos ARCO.

Pangeo se basa en los conceptos de datos listos para el análisis y optimizados para la nube, destacando la importancia de la eficiencia y el acceso directo a subconjuntos de datos. Los datos ARCO deben pasar por un proceso de preprocesamiento para cumplir con los estándares de calidad requeridos y deben almacenarse en formatos que permitan un acceso eficiente.

Un aspecto destacado de Pangeo es su naturaleza de código abierto, lo que garantiza la transparencia y la posibilidad de revisar y mejorar el proceso de producción de datos ARCO. La replicabilidad es un valor fundamental, y se asegura que, incluso cuando se utilizan tecnologías comerciales, el código de aplicación siempre es de código abierto.

La inspiración para Pangeo proviene del exitoso modelo de Conda Forge, que ha demostrado cómo la colaboración y la contribución abierta pueden resolver problemas complejos. Pangeo busca seguir ese mismo camino, permitiendo la colaboración de la comunidad científica para producir y compartir datos ARCO de alta calidad.

En resumen, Pangeo es una plataforma de código abierto que facilita la producción y el acceso a datos ARCO en la nube. Su objetivo es reducir la carga de trabajo y fomentar la participación abierta y colaborativa de la comunidad científica en la producción de datos ARCO de alta calidad.

2.3. Jupyterhub en sistemas HPC

En el artículo [1], se habla sobre el análisis interactivo y reproducible a gran escala en sistemas HPC utilizando la plataforma Jupyter. Los autores destacan que, si bien Jupyter ha demostrado ser una herramienta popular y efectiva para la interactividad y colaboración científica, no fue diseñada originalmente para su uso en entornos HPC, lo que plantea desafíos específicos.

El artículo describe los esfuerzos de los investigadores para abordar estos desafíos y desarrollar capacidades de análisis interactivo y reproducible en sistemas HPC. Se presenta un caso de estudio en el campo de las simulaciones geofísicas y las inversiones para la imagen del subsuelo, que tiene aplicaciones en la exploración de recursos naturales y estudios medioambientales.

Los autores se centran en tres aspectos clave del flujo de trabajo científico: la reproducibilidad, la escalabilidad y la interactividad. Para lograr la reproducibilidad, se ha desarrollado un entorno que captura todo el conjunto de software necesario, incluyendo el código utilizado. En términos de escalabilidad, se ha diseñado una solución que permite la ejecución de flujos de trabajo interactivos utilizando múltiples nodos de procesamiento de forma distribuida en sistemas HPC. Además, se han desarrollado herramientas que facilitan la interactividad y visualización de grandes volúmenes de datos en tiempo real.

En el artículo se discuten los detalles de la implementación de esta solución, que se basa en la integración de diferentes tecnologías en la plataforma Jupyter, como Binder, Science Capsule y Dask. Los autores presentan ejemplos y resultados de su enfoque, demostrando la eficacia de la plataforma en el análisis interactivo y reproducible a gran escala en sistemas HPC.

En conclusión, el artículo destaca la importancia de desarrollar capacidades de análisis interactivo y reproducible en entornos HPC utilizando la plataforma Jupyter. Los autores proporcionan una propuesta concreta y detallada que aborda los desafíos específicos de este tipo de análisis, y presentan resultados prometedores en el caso de estudio de simulaciones geofísicas. Este enfoque tiene un potencial significativo para mejorar la eficiencia y efectividad del análisis científico a gran escala.

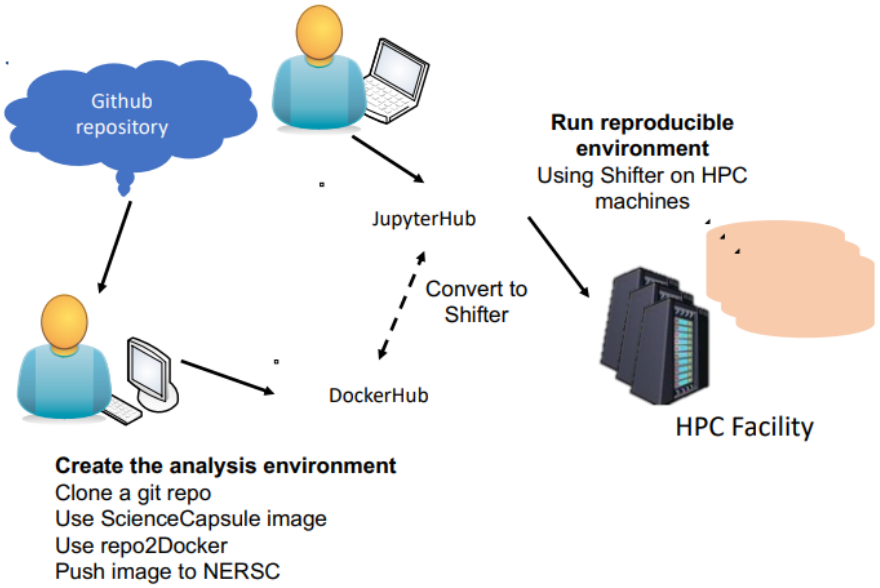

Figura 1: Proceso de JupyterHub en HPC. Fuente: [1]

2.4. Amazon SageMaker

SageMaker es el producto de aprendizaje automático de AWS, que cuenta con un conjunto de funciones similares a Vertex AI de Google. Puede ser accedido a través de la consola de AWS y permite iniciar sesión a través de SSO. Ofrece computación escalable y los datos se guardan en almacenamiento de archivos elásticos, donde pueden ser usados por múltiples cuadernos. Como desventajas. solo se pueden compartir copias de los cuadernos y el coste es bastante elevado, además, personalizar el entorno JupyterLab es bastante complicado.
En la figura 2 se puede ver un ejemplo del manejo de cuadernos en la plataforma.

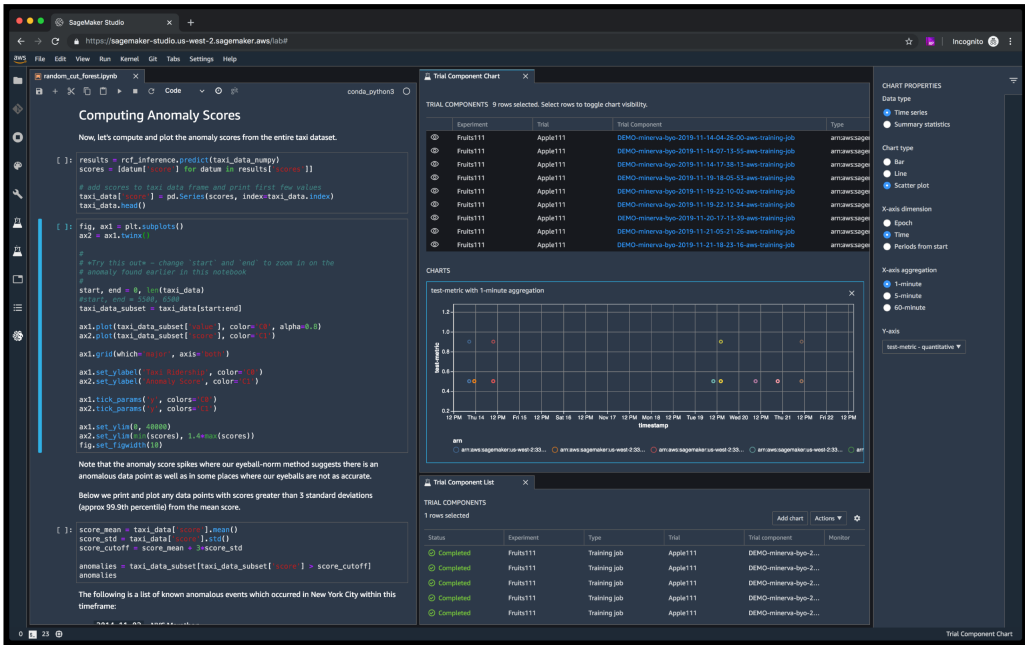

Figura 2: Amazon SageMaker. Fuente: [2]

2.5. Google Vertex AI Workbench

La única opción que Google presenta es el Google Vertex AI Workbench, que no es más una instancia de JupyterLab que permite seleccionar entre varios kernels. Es facilmente integrable con GitHub y permite automatizar ejecuciones de cuadernos y guardar los resultados en Google Cloud Storage. Además, permite gestionar las capacidades de la maquina, es decir, modificar el *hardware*.
En la figura 3 se puede ver la aplicación en funcionamiento.
El uso de esta plataforma supone costes equivalentes al gasto de recursos computacionales, el gasto de almacenamiento y los gastos de gestión.

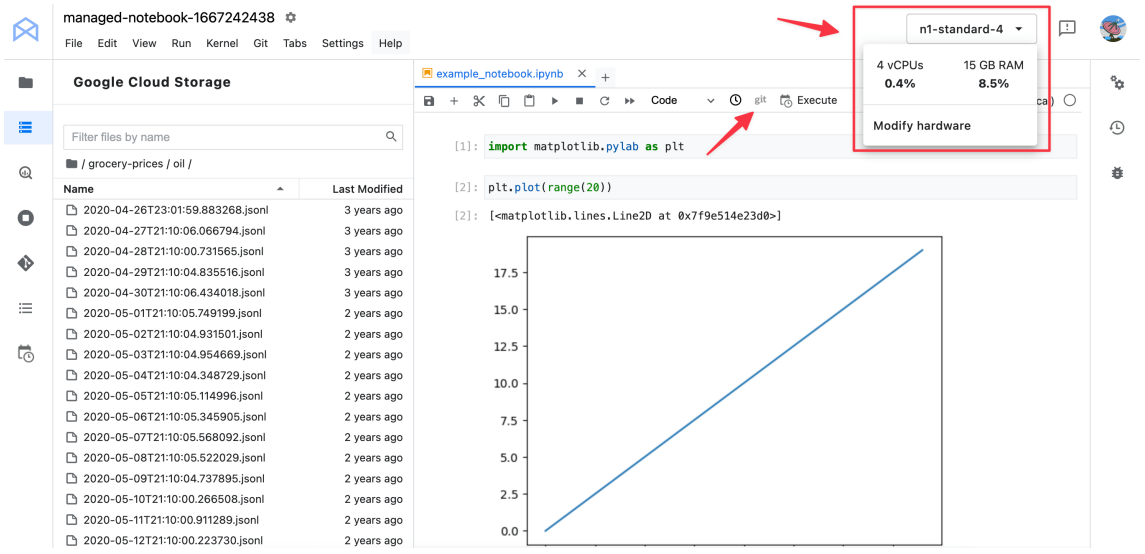

Figura 3: Google Vertex AI Workbench . Fuente: [2]

2.6. JupyterHub

Por último, tenemos la opción de instalar JupyterHub por nuestra cuenta en nuestra propia infraestructura y tener así un control total de la configuración. Esta opción es la más adecuada en nuestra caso por las ventajas que trae consigo.

2.6.1. Componentes de JupyterHub

JupyterHub está compuesto por tres principales subsistemas:

- **Multi-user Hub (Proceso Tornado/Python)**: Es el corazón de JupyterHub, se encarga de las cuentas de usuario, de su autenticación y de la gestión de los servidores individuales usando un Spawner.
- **Configurable-http-proxy (node-http-proxy)**: Este componente maneja todas las solicitudes entrantes y las enruta a los servidores de notebooks correctos, una vez que los usuarios se han autenticado y sus servidores de notebooks se han iniciado.
- **Spawner**: Este componente es responsable de iniciar los servidores de notebooks para los usuarios. Existen diferentes tipos de Spawners que se pueden usar dependiendo de las necesidades. Por ejemplo, se puede usar el LocalProcessSpawner para iniciar servidores de notebooks en el mismo servidor que el Hub, o se puede usar el DockerSpawner para iniciar servidores de notebooks en contenedores Docker.
- **Authenticator**: Este componente maneja la autenticación de los usuarios. JupyterHub viene con un autenticador básico, pero puede ser personalizado para integrar JupyterHub con otros sistemas de autenticación, como el OAuth2 de GitHub o de Google.

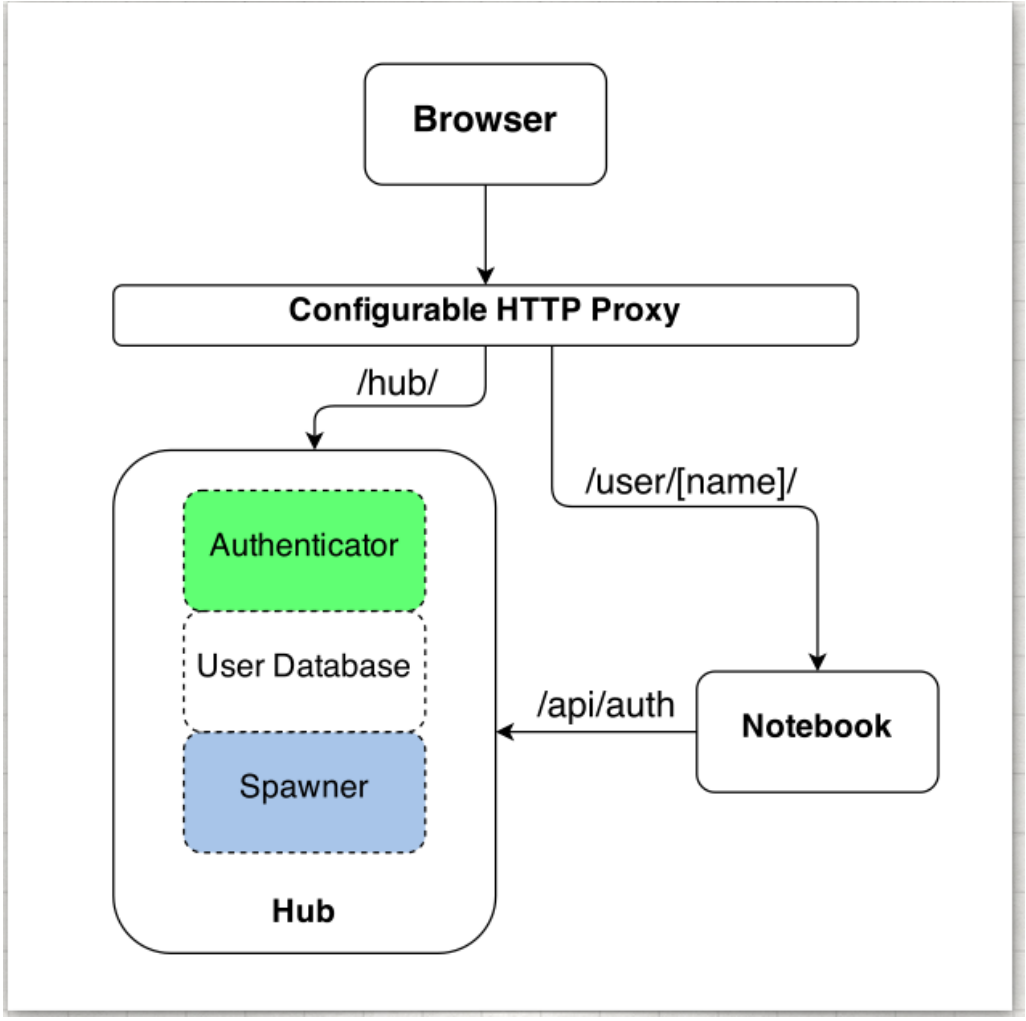

Figura 4: Componentes de JupyterHub. Fuente: [3]

2.6.2. Proceso de JupyterHub

- **Lanzamiento del Proxy:** Cuando se inicia JupyterHub, el primer paso es lanzar el Proxy. Este es un servidor HTTP que se coloca entre los usuarios y tanto el Hub como cualquier Servidor de Usuario que se inicie. El Proxy puede manejar miles de conexiones simultáneas y enrutarlas adecuadamente.
- **Redirección de Solicitudes al Hub:** Una vez que el Proxy está en marcha, todas las solicitudes HTTP que llegan al JupyterHub son inicialmente recibidas por el Proxy. Por defecto, todas estas solicitudes son redirigidas al Hub.
- **Manejo de Autenticación:** El Hub recibe la solicitud y determina si el usuario está autenticado. Para ello, se utiliza el componente Autenticador que se encarga de la autenticación de los usuarios. Si el usuario no está autenticado, se le presenta una pantalla de inicio de sesión y se le pide que proporcione sus credenciales.
- **Lanzamiento del Servidor del Usuario:** Una vez que el usuario está autenticado, el Hub

necesita asegurarse de que el servidor de notebooks del usuario está en marcha. Esto se hace con el componente Spawner. El Spawner tiene la tarea de iniciar el servidor de notebooks del usuario y asegurarse de que esté listo para recibir solicitudes.

- **Redirección de Solicitudes al Servidor del Usuario:** Cuando el servidor de notebooks del usuario está listo, el Hub añade una nueva ruta al Proxy que enlaza la URL única del servidor de notebooks del usuario con este. A partir de ese momento, todas las solicitudes a esa URL son redirigidas por el Proxy directamente al servidor de notebooks del usuario.
- **Interacción con el Servidor de Notebooks:** Con la redirección en su lugar, el usuario puede interactuar con su servidor de notebooks como lo haría normalmente. Puede crear y editar notebooks, ejecutar celdas, etc. Todas estas acciones se realizan en el servidor de notebooks del usuario y no afectan a otros usuarios.
- **Cierre y Limpieza:** Cuando el usuario termina su sesión, el Hub es notificado y el Spawner se encarga de apagar el servidor de notebooks del usuario y liberar los recursos que estaba utilizando. Una vez que el servidor de notebooks se ha apagado correctamente, la ruta en el Proxy se elimina.

En la figura 5 se puede ver la principal arquitectura de un sistema de JupyterHub general, en el que el proceso del Hub es el corazón de la aplicación, y a sus lados tiene interconectados diferentes procesos, como son el `http-configurable-proxy`, el archivo de configuración `jupyter_config.py`, el autenticador, que decide si el cliente tiene permitido el acceso o no, la base de datos, que es la encargada de almacenar los datos de los usuarios y los spawners, que son lanzados para cada usuario por separado.

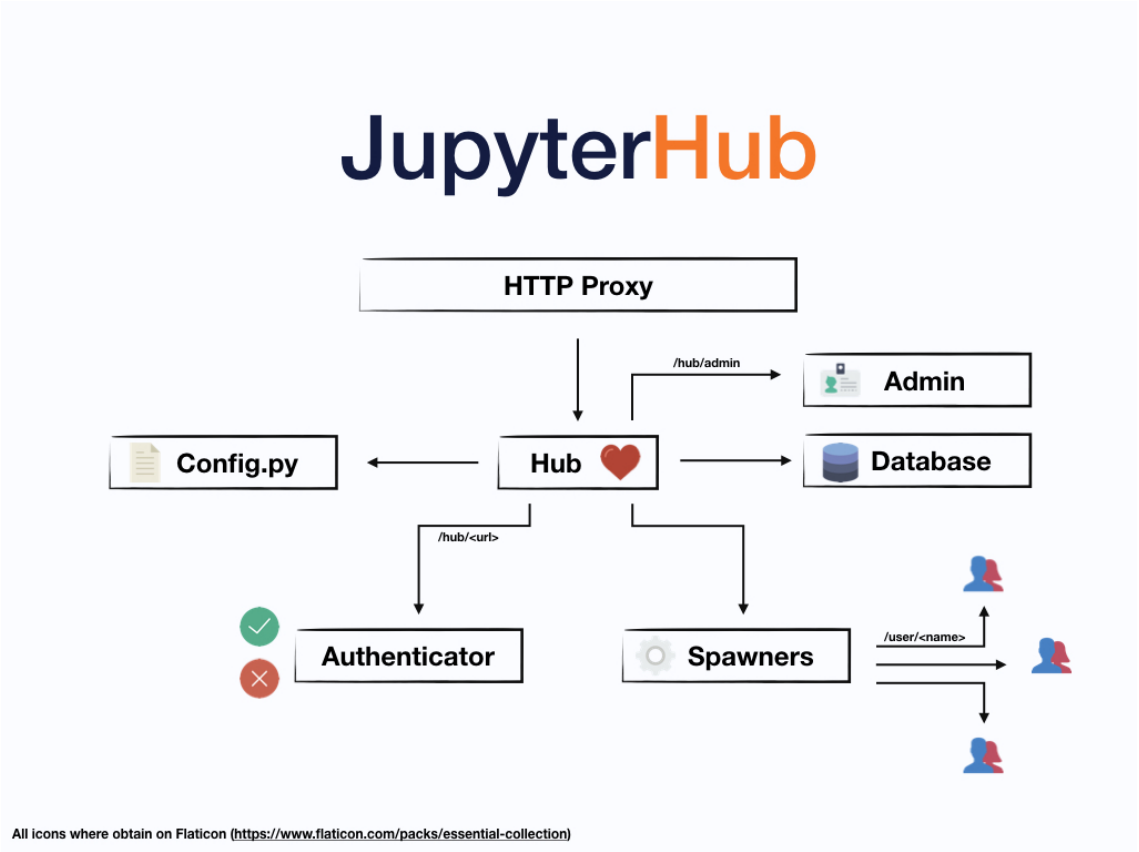

Figura 5: Arquitectura de JupyterHub. Fuente: [3]

3. Metodología

3.1. Introducción

El desarrollo del trabajo de fin de máster, se basa en gran medida en la utilización de JupyterHub, motivo por el cuál hacer un estudio de las opciones de instalación que existen es de vital importancia. Existen varias formas de configurar e instalar JupyterHub, cada una de ellas con sus propias ventajas y desventajas, por lo que la elección del método de instalación depende en gran medida de las necesidad y circunstancias especificas del proyecto, así como de la infraestructura en la que trabajamos.

3.2. Control de versiones

Durante el desarrollo de este proyecto, se utilizó Git como sistema de control de versiones para gestionar el código desarrollado y los archivos relacionados. Git proporciona un entorno colaborativo y permite mantener un historial de cambios, lo cual nos facilita el seguimiento de las versiones y el posible arreglo de errores en el futuro. Además, se usó GitLab como plataforma para alojar el repositorio Git. Gitlab proporciona herramientas para colaborar de forma eficiente y controlar el flujo de trabajo. Utilizar Git y Gitlab permitió un desarrollo estructurado y la posibilidad de tener un manejo de versiones perfecto.

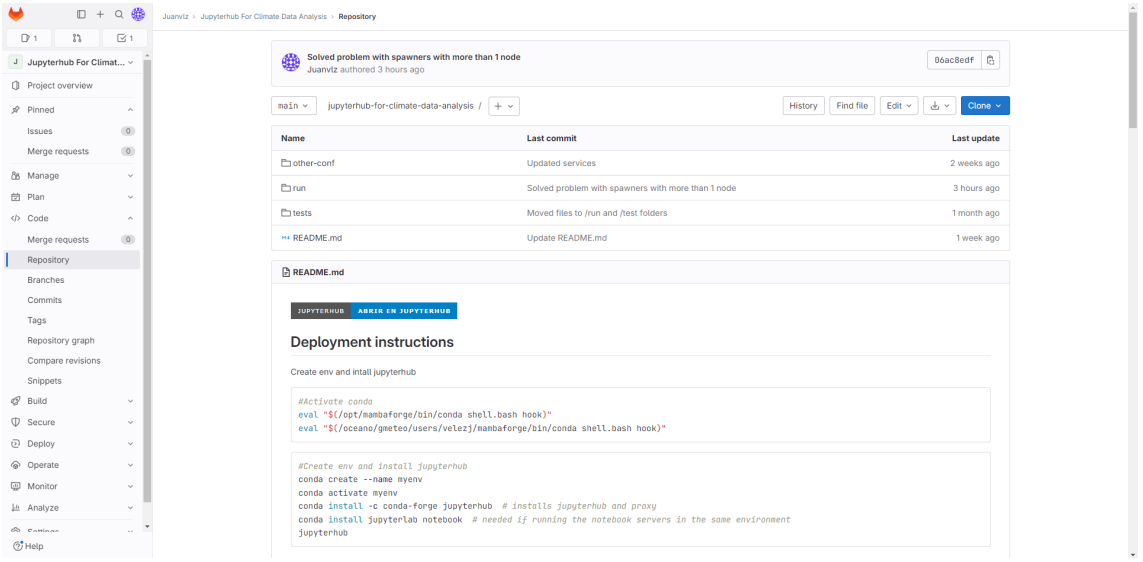

Figura 6: Repositorio en GitLab.

A continuación, en la figura 7 se muestran los diferentes archivos que contiene la carpeta `run` que corresponde a la carpeta `/opt/jupyterhub/run` en la máquina Ganymede:

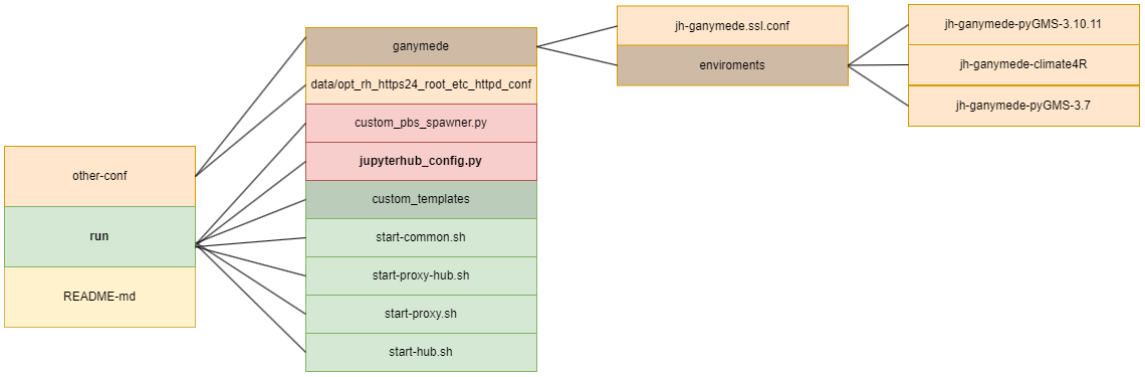

Figura 7: Repositorio en GitLab.

3.3. Opciones de instalación de JupyterHub

Las principales opciones de instalación de JupyterHub son:

- **Servicios de cuadernos alojados para ti**: Proveedores de la nube como Google y Microsoft proporcionan servicios gratuitos de cuadernos que son alojados por soluciones externas para que cualquier persona pueda desarrollar y ejecutar código en un navegador de forma rápida. Estos servicios suelen venir con paquetes preinstalados y soporte para múltiples lenguajes, pero no tienen la capacidad de personalizar la configuración del hardware y la escalabilidad.
- **JupyterHub para Kubernetes - Autoalojado**: Integrar JupyterHub con un motor Kubernetes permite una versión robusta y escalable de JupyterHub, que da la capacidad de desplegar entornos en contenedores sin estar atado a ningún proveedor de la nube en particular.
- **Visual Studio Online - Alojado para ti**: Visual Studio Online proporciona entornos de desarrollo impulsados por la nube e instancias para cualquier actividad. Este servicio soporta trabajar con Jupyter Notebooks de forma nativa y puede ser configurado para usar JupyterHub como un hub central para todos los usuarios o individualmente para cada usuario.
- **Cuadernos Jupyter locales - Autoalojado**: Quizás la opción más sencilla, ejecutar cuadernos Jupyter localmente permite personalizar completamente su configuración y ejecutar JupyterHubs en su propio servidor. Este es el entorno de desarrollo local original para los cuadernos Jupyter que aprovecha el hardware de su propia computadora para el cálculo. En nuestro caso, hemos optado por esta configuración, instalando JupyterHub en local con conda en un sistema de HPC (Computación de Alto Rendimiento). Esta opción permite utilizar la potente infraestructura del HPC para ejecutar análisis de datos y cálculos intensivos.

3.4. Elección de la instalación de JupyterHub

En el contexto del proyecto, la opción elegida es la instalación de JupyterHub en local utilizando conda. Esta decisión se tomó por varias razones. En primer lugar, al ejecutar JupyterHub localmente, tenemos un control total sobre el entorno y podemos personalizarlo según nuestras necesidades especificas. En segundo lugar, conda es una herramienta de gestión de paquetes que facilita la instalación de JupyterHub y sus dependencias, y es aompliamente utiliza en la comunidad científica y de análisis de datos. Por último, al ejecutar JupyterHub en nuestros popios servidores, podemos asegurarnos de que todos los datos permanezcan locales y sean accesibles, lo que es importante para el control de la privacidad y la seguridad de los datos.

3.5. Spawners en JupyterHub

Los Spawners en JupyterHub son los responsables de iniciar cada servidor de notebooks para un solo usuario. Los Spawners representan una interfaz abstracta para un proceso, y un Spawner personalizado necesita poder realizar tres acciones:

1. Iniciar un proceso.
2. Verificar si un proceso sigue en ejecución.
3. Detener un proceso.

Existen varios tipos de Spawners, algunos de ellos son:

- **DockerSpawner**: es utilizado para generar servidores de usuarios en contenedores Docker. Incluye dos variantes, el `dockerspawner.DockerSpawner` que genera contenedores Docker idénticos para cada usuario y `dockerspawner.SystemUserSpawner` que genera contenedores Docker con un entorno y directorio de inicio para cada usuario. Ambas variantes también trabajan con Docker Swarm para lanzar contenedores en máquinas remotas.
- **SudoSpawner**: permite a JupyterHub ejecutarse sin ser root, generando un proceso intermedio a través de `sudo`.
- **BatchSpawner**: se utiliza para generar servidores remotos utilizando sistemas de lotes.
- **YarnSpawner**: se utiliza para generar servidores de notebooks en contenedores YARN en un clúster de Hadoop.
- **SSHSpawner**: se utiliza para generar notebooks en un servidor remoto utilizando SSH.
- **KubeSpawner**: se utiliza para generar servidores de notebooks en un clúster de Kubernetes.

Cada Spawner tiene métodos de control como `start`, `poll` y `stop`, y también puede manejar estados para persistir información que puede restaurarse más tarde. Además, los Spawners también pueden ofrecer opciones a los usuarios para influir en cómo se inician sus servidores. Esto puede incluir implementaciones basadas en clústeres, donde los usuarios especifican qué recursos deben estar disponibles, o implementaciones basadas en Docker donde los usuarios pueden seleccionar de una lista de imágenes base.

3.6. Elección de Spawners

En este trabajo, se han utilizado tres Spawners: SudoSpawner, BatchSpawner y ProfilesSpawner. El SudoSpawner es útil porque permite a JupyterHub ejecutarse sin ser root, lo cual es esencial para garantizar la seguridad del servidor y limitar los permisos de los usuarios. Por otro lado, BatchSpawner es especialmente útil en entornos de HPC como Altamira, ya que permite generar servidores remotos utilizando sistemas de lotes, lo que hace posible que los trabajos de los usuarios se ejecuten en la cola de trabajos del HPC en lugar de en el servidor JupyterHub en sí.

Además, se ha utilizado el ProfilesSpawner, que es un Spawner de envoltorio que permite a los usuarios seleccionar de una lista de perfiles de configuración al lanzar su servidor. Este Spawner es especialmente útil en entornos donde es necesario dar a los usuarios opciones para el tipo de servidor que quieren lanzar, como diferentes configuraciones de hardware o diferentes entornos de software.

Optamos por estos Spawners debido a la naturaleza de nuestra implementación de JupyterHub. Dado que estamos trabajando en un entorno de HPC, el uso de BatchSpawner nos permite aprovechar al máximo los recursos del HPC. Asimismo, el uso de SudoSpawner nos permite mantener un nivel adecuado de seguridad y control sobre los permisos de los usuarios. Finalmente, el uso del ProfilesSpawner nos proporciona flexibilidad y opciones personalizadas para los usuarios a la hora de lanzar sus servidores.

3.7. Separación del Proxy en JupyterHub

El elemento al que los usuarios se conectan directamente en JupyterHub es el proxy, que por defecto es el *configurable-http-proxy*. Este proxy redirige a los usuarios al hub (para el inicio de sesión y la gestión de servidores), o a sus propios servidores de un solo usuario. Por lo tanto, siempre que el proxy siga funcionando, el acceso a los servidores existentes continuará, incluso si el hub se reinicia o se cae.

Cuando se configura el hub por primera vez, es posible que no se perciba esto porque el proxy es administrado automáticamente por el hub. Aunque esto es ideal para empezar y para la mayoría de los casos de uso, cada vez que se reinicia el hub, todas las conexiones de los usuarios también se reinician. Sin embargo, también es sencillo ejecutar el proxy como un servicio separado del hub, lo que puede permitir reconfigurar el hub mientras sólo se interrumpe a los usuarios que están esperando que se inicie su servidor de cuadernos.

Para configurar el proxy por separado, deben establecerse algunas opciones de configuración:

- `c.JupyterHub.cleanup_servers = False`: Esto indica al hub que no detenga los servidores cuando el hub se reinicia.
- `c.ConfigurableHTTPProxy.should_start = False`: Esto indica al hub que el proxy no debe iniciarse (porque tú lo inicias por ti mismo).
- `c.ConfigurableHTTPProxy.auth_token = ÇONFIGPROXY_AUTH_TOKEN"`: Debe establecerse un token para autenticar la comunicación con el proxy.
- `c.ConfigurableHTTPProxy.api_url = 'http://localhost:8001'`: Debe establecerse la URL que el hub utiliza para conectarse a la API del proxy.

Además, es necesaria la creación de un servicio de *systemd* que mantenga el proxy en ejecución. En nuestra implementación, hemos seguido estas pautas para separar el proxy del hub. Esta configuración nos permite reiniciar o reconfigurar el hub según sea necesario, sin interrumpir el acceso a los servidores de cuadernos de los usuarios. Además, también facilita la gestión del proxy, ya que podemos iniciar, detener y reiniciar el proxy de forma independiente del hub.

3.8. Arquitectura del despliegue de JupyterHub en Ganymede

En la figura 8 se puede observar la arquitectura usada en el despliegue total de la aplicación. La arquitectura consta de varios componentes interconectados entre sí que permiten acceder y analizar Jupyter Notebooks de manera colaborativa. A continuación, se describe cada uno de estos componentes de manera individual:

1. **Cliente del investigador**: Es el usuario que accede a JupyterHub desde su navegador web a través de Internet. El cliente del investigador se conecta inicialmente al nodo de inicio de sesión de Ganymede.

2. **Nodo de inicio de sesión (Login Node)**: Es un nodo de Ganymede responsable de manejar la autenticación y el acceso de los usuarios. Aquí es donde los usuarios ingresan sus credenciales y obtienen acceso al JupyterHub.

3. **JupyterHub Server**: Es el componente principal que gestiona y coordina los recursos de JupyterHub en Ganymede. El JupyterHub Server consta de dos componentes principales:

   - **HTTP Configurable Proxy**: Este componente se encarga de enrutar las solicitudes de los usuarios a los servidores de Jupyter Notebook correspondientes. Actúa como una capa de proxy inverso y asegura que las solicitudes se dirijan al servidor correcto en función de la configuración y las preferencias del usuario.
   - **PBS Spawner**: Este componente se utiliza para la creación y gestión de los servidores de usuario de Jupyter Notebook. Utiliza el sistema de colas PBS (Portable Batch System) para programar y asignar recursos en el clúster PBS. Cada usuario tiene su propio servidor de Jupyter Notebook, que se crea y administra dinámicamente según la demanda. El proceso del script PBS se lleva a cabo de la siguiente manera:

     a) **Cola de requisitos (`req_queue`)**: Se especifica la cola de PBS en la que se deben ejecutar los servidores de usuario. En este caso, se establece la cola de requisitos como 'tintel'.

     b) **Tiempo de ejecución (`req_runtime`)**: Se establece el tiempo de ejecución máximo permitido para cada servidor de usuario. Aquí, se configura un tiempo de ejecución máximo de 1 hora (`1:00:00`).

     c) **Memoria (`req_memory`)**: Se establece la cantidad de memoria requerida para cada servidor de usuario. En este caso, se configura una memoria de 10 GB.

     d) **Prefijo de ejecución (`exec_prefix`)**: Se especifica el prefijo de ejecución que se utilizará para ejecutar el comando del servidor de usuario. Aquí, se utiliza el prefijo `'sudo -E -u {username}'` para ejecutar el comando con privilegios de usuario.

e) **Comando del servidor de usuario (`cmd`)**: Se define el comando que se ejecutará para iniciar el servidor de usuario de Jupyter Notebook. En este caso, se utiliza `["jupyter-labhub"]` como el comando para iniciar JupyterLab.

f) **Script de lote (`batch_script`)**: Se proporciona el script de lote completo que se enviará al sistema de colas PBS para iniciar el servidor de usuario de Jupyter Notebook. El script contiene directivas PBS, como la cola, el tiempo de ejecución, la cantidad de nodos, la memoria y el nombre del trabajo. También se especifica la ruta de salida y error para los registros del servidor de usuario. Además, se incluye la configuración del entorno con variables relevantes de JupyterHub y se activa el entorno de conda necesario. Por último, se ejecuta el comando del servidor de usuario proporcionado. [11] A continuación se muestra el script PBS usado por defecto:

```
c.BatchSpawnerBase.req_queue = 'tintel'
c.BatchSpawnerBase.req_runtime = '1:00:00'
c.BatchSpawnerBase.req_memory = '10gb'
c.PBSSpawner.exec_prefix = 'sudo -E -u {username}'
c.Spawner.cmd = ["jupyter-labhub"]
c.BatchSpawnerBase.batch_script= '''#!/bin/sh
#PBS -q {queue}
#PBS -l walltime={runtime}
#PBS -l nodes=1:ppn={nprocs}
#PBS -l mem={memory}
#PBS -N jupyterhub-singleuser
#PBS -o /oceano/gmeteo/users/{username}/out
#PBS -e /oceano/gmeteo/users/{username}/err
#PBS -v PATH,CONDA_DEFAULT_ENV,LANG,JUPYTERHUB_API_TOKEN,JPY_API_TOKEN
    ,JUPYTERHUB_CLIENT_ID,JUPYTERHUB_HOST,
    JUPYTERHUB_OAUTH_CALLBACK_URL,JUPYTERHUB_OAUTH_SCOPES,
    JUPYTERHUB_OAUTH_ACCESS_SCOPES,
    JUPYTERHUB_OAUTH_CLIENT_ALLOWED_SCOPES,JUPYTERHUB_USER,
    JUPYTERHUB_SERVER_NAME,JUPYTERHUB_API_URL,JUPYTERHUB_ACTIVITY_URL,
    JUPYTERHUB_BASE_URL,JUPYTERHUB_SERVICE_PREFIX,
    JUPYTERHUB_SERVICE_URL

env
# env > $PBS_WORKDIR/$PBS_JOBID.env
eval "$(/software/jupyterhub/mambaforge/bin/conda shell.bash hook)"
conda activate base
which python
which batchspawner-singleuser
which jupyterhub-singleuser
{cmd}
#batchspawner-singleuser jupyter-labhub
'''
```

4. **Cluster PBS**: Es el clúster PBS que usa Ganymede, que consta de varios nodos de computación interconectados. Estos nodos ejecutan los servidores de usuario de Jupyter Notebook

en respuesta a las solicitudes de los usuarios. El sistema de colas PBS administra la programación y asignación de recursos en el clúster.

El clúster PBS es una infraestructura de computación distribuida compuesta por varios nodos interconectados que trabajan en conjunto para realizar tareas computacionales de manera eficiente. Estos nodos pueden ser servidores físicos o máquinas virtuales y están conectados mediante una red de alta velocidad.

El sistema de colas PBS se encarga de administrar y asignar los recursos de computación disponibles en el clúster. Cuando los usuarios envían tareas al clúster, estas tareas se enfilan en una cola de espera y se programan para su ejecución en función de diversos criterios, como la prioridad, los recursos requeridos y la disponibilidad.

La administración y asignación de recursos se realiza de manera equitativa y eficiente gracias al sistema de colas PBS. Este sistema gestiona la programación de las tareas en función de las políticas establecidas, como el tiempo máximo de ejecución, la cantidad de memoria requerida y la disponibilidad de nodos. Los administradores del clúster pueden establecer políticas de prioridad y uso de recursos para diferentes usuarios o grupos de usuarios.

Cuando se solicita la ejecución de un servidor de usuario de Jupyter Notebook, el sistema de colas PBS asigna los recursos necesarios en el clúster. Esto implica seleccionar un nodo adecuado y reservar los recursos necesarios, como memoria y tiempo de CPU. Una vez completada la asignación de recursos, el servidor de usuario se inicia en el nodo asignado y los usuarios pueden comenzar a trabajar en sus Jupyter Notebooks.

En resumen, el clúster PBS que usa Ganymede está compuesto por varios nodos de computación interconectados que ejecutan los servidores de usuario de Jupyter Notebook. El sistema de colas PBS administra la programación y asignación de recursos en el clúster, garantizando una ejecución eficiente y equitativa de las tareas solicitadas por los usuarios [12].
Al hacer un `qstat` en la máquina *Ganymede*, los *jobs* del JupyterHub se ven de esta manera:

```
1181178.encina JupyterHub-singleuser velezj 00:00:02 C tintel
```

En la salida del comando podemos ver las propiedades principales del job:

- **id:** 1181178.encina
- **nombre:** JupyterHub-singleuser
- **usuario:** velezj
- **tiempo:** 00:00:02
- **cola:** tintel

5. **Sistema Lustre**: Es un sistema de archivos de alto rendimiento que se utiliza para almacenar datos. El sistema Lustre proporciona un almacenamiento compartido y distribuido que permite a los usuarios acceder y colaborar en datos de manera eficiente.

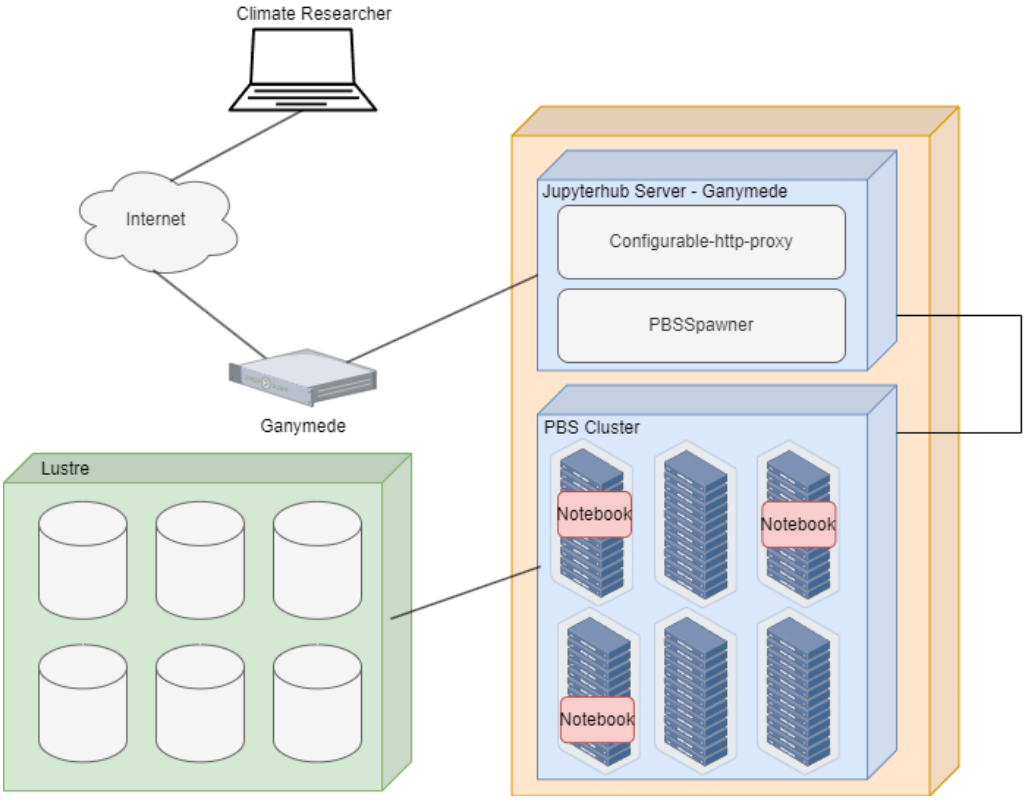

Figura 8: Arquitectura del JupyterHub.

4. Implementación

4.1. Introducción

El trabajo ha consistido en la instalación de un JupyterHub en el HPC del SantanderMetGroup.

El proceso de implementación se va dividió en varias fases.

Al principio del proceso, se barajaron distintas opciones para instalar JupyterHub, siendo la primera la utilización The Littlest JupyterHub [13], que consiste en una pequeña distribución de JuyterHub que sirve para una cantidad máxima de 100 usuarios, está pre-configurado y tiene un nivel de abstracción mayor a la instalación de JupyterHub desde cero. Justo por esto, tras probar la instalación y despliegue de The Littlest JupyterHub, nos dimos cuenta de que no cumplía con los requisitos que necesitábamos, y tomamos la decisión de usar la instalación desde cero, ya que permite una configuración y control total del mismo.

4.2. Preparación de servidor

El primer paso para poder cumplir con nuestro objetivo fue la creación de una máquina dentro del contexto del HPC del SantanderMetGroup. Se decidió que la maquina iba a llamarse **Ganymede** y la conexión se haría mediante *SSH* [14].

4.2.1. Características de la maquina Ganymede

- Propiedades de la CPU.

```
velezj@ganymede:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   40 bits physical, 48 bits virtual
CPU(s):                          1
On-line CPU(s) list:             0
Thread(s) per core:              1
Core(s) per socket:              1
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           44
Model name:                      Intel(R) Xeon(R) CPU           E5645  @ 2.40
    GHz
Stepping:                        2
CPU MHz:                         2399.973
BogoMIPS:                        4800.17
Hypervisor vendor:               Xen
Virtualization type:             full
L1d cache:                       32 KiB
L1i cache:                       32 KiB
L2 cache:                        256 KiB
```

```
L3 cache:                        12 MiB
NUMA node0 CPU(s):               0
```

- Propiedades de los discos.

```
velezj@ganymede:~$ lsblk
NAME                      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop1                       7:1    0 63.5M  1 loop /snap/core20/1891
loop2                       7:2    0 67.8M  1 loop /snap/lxd/22753
loop3                       7:3    0 91.9M  1 loop /snap/lxd/24061
loop4                       7:4    0 63.5M  1 loop /snap/core20/1950
loop5                       7:5    0 53.3M  1 loop /snap/snapd/19361
loop6                       7:6    0 53.3M  1 loop /snap/snapd/19457
xvda                      202:0    0   20G  0 disk
    xvda1                   202:1    0     1M  0 part
    xvda2                   202:2    0   1.8G  0 part /boot
    xvda3                   202:3    0 18.2G  0 part
      ubuntu    --vg-ubuntu--lv 253:0    0    10G  0 lvm  /
xvdb                      202:16   0   20G  0 disk
    xvdb1                   202:17   0   20G  0 part /opt
```

4.2.2. Creación de usuario sin privilegios

Una de las primeros decisiones que tuvimos que tomar, fue la de lanzar JupyterHub desde el usuario *velezj*, es decir, mi usuario, o de crear un usuario auxiliar. Por seguridad, se decidió crear un usuario que fuera capaz de ejecutar todas las instrucciones que se le manda pero que no tenga permisos de administrador. Para lograr esto, se sigue la guía de JupyterHub nos proporciona para ejecutar su aplicación con sudo pero sin permisos de administrador [15].
En resumen, se crea el usuario **rhea** y se le da los permisos necesarios en el archivo */etc/sudoers*. De aquí en adelante, es el usuario auxiliar **rhea** el que se encarga de ejecuta la aplicación.

4.3. Instalación de JupyterHub desde Cero

Este proceso fue largo y tedioso, ya que mientras se iba a avanzando, aparecían nuevos errores que retrasaban mucho el proceso. Voy a intentar explicar el proceso en diferentes pasos.

- Instalación de un *conda root* en /opt/jupyterhub. De esta forma de descargue del repositorio oficial la ultima versión de mambaforge y se descomprime el fichero.

```
    curl -Ls https://micro.mamba.pm/api/micromamba/linux-64/latest | tar -xvj
        bin/micromamba
```

- Creación del entorno que contendrá los paquetes necesarios para desplegar JupyterHub.

```
    conda create --name myenv
```

- Instalar jupyterhub:

```
mamba install -c conda-forge jupyterhub
```

- Crear archivo de configuración:

```
sudo /opt/jupyterhub/bin/jupyterhub --generate-config
```

- Ahora mismo, la aplicación se puede desplegar, pero necesitamos editar el archivo de configuración *jupyterhub_config.py* para desplegar la aplicación en la dirección y puertos correctos.

```
c.JupyterHub.bind_url = 'http://192.168.202.189:8000/ganymede'
```

- Para ejecutar la aplicación con el usuario rhea basta con activar el entorno y ejecutar como rhea:

```
eval "$(/opt/mambaforge/bin/conda shell.bash hook)"
conda activate myenv
sudo -E -u rhea jupyterhub -f jupyterhub_config.py
```

- Para conseguir que la aplicación es ejecutándose constantemente, es necesario crear un servicio de *systemd* que ejecute la aplicación. El servicio creado es de esta forma:

```
[Unit]
Description=Jupyterhub
After=jupyterhub-proxy
Requires=jupyterhub-proxy

[Service]
User=rhea
WorkingDirectory=/opt/jupyterhub/run
Environment="PATH=/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
    opt/mambaforge/envs/myenv/bin"
ExecStart=/opt/jupyterhub/run/start-hub.sh

[Install]
WantedBy=multi-user.target
```

El *script* al que hace referencia el servicio `jupyterhub-hub.service` es el encargado de lanzar la aplicación, el *script* `start-hub.sh` tiene la forma:

```
#!/usr/bin/env bash
source start-common.sh
# Ejecuta JupyterHub
jupyterhub -f ${JH_RUN_DIRECTORY}/jupyterhub_config.py
```

- Además, se quiso separar el *proxy* que usa JupyterHub de la propia aplicación por las mejoras que esto supone, en un principio, JupyterHub se encarga de manejar el *configurable-http-proxy* por su cuenta, y no nos damos cuenta de su funcionamiento, pero a la hora de realizar buenas prácticas, lo correcto es separarlo en otro proceso, ya que si por ejemplo queremos reconfigurar el *Hub*, las conexiones que están activas pueden continuar funcionando. Se decidió seguir la guía oficial de JupyterHub. [16]. Este *proxy* se encarga de redirigir a los usuarios al *hub* (para iniciar sesión y gestionar servidores), o a sus propios servidores de usuario. Para poder separarlo basta con editar la configuración de *jupyter_config.py*:

```
c.JupyterHub.cleanup_servers = False
c.ConfigurableHTTPProxy.should_start = False
c.ConfigurableHTTPProxy.auth_token = "CONFIGPROXY_AUTH_TOKEN"
c.ConfigurableHTTPProxy.api_url = 'http://localhost:8001'
```

Seguido a esto, es necesario iniciar el *configurable-http-proxy* en un servicio, en este caso, lanzado por **rhea**.

```
[Unit]
Description=JupyterHub Proxy
```

```
After=syslog.target network.target

[Service]
User=rhea
WorkingDirectory=/opt/jupyterhub/run
Environment="PATH=/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
    opt/mambaforge/envs/myenv/bin"
ExecStart=/opt/jupyterhub/run/start-proxy.sh

[Install]
WantedBy=multi-user.target
```

El *script* al que hace referencia el servicio `jupyterhub-proxy.service` es el encargado de lanzar el proxy, el *script* `start-proxy.sh` tiene la forma:

```
#!/usr/bin/env bash
source start-common.sh
#Start dynamic proxy
configurable-http-proxy --ip 0.0.0.0 --port 8000 --api-ip 127.0.0.1 --api-port
    8001 --default-target http://127.0.0.1:8081 --error-target http
    ://127.0.0.1:8081
```

4.3.1. Spawners usados en el JupyterHub de Ganymede

Como se menciona en la Metodología, se han usado tres Spawners para que JupyterHub funcione como debe, estos van a ser explicados más a fondo, tanto su uso, como su instalación y su configuración.

- **BatchSpawner:** Este spawner está diseñado para lanzar servidores Jupyter en sistemas de programación por lotes. En nuestro caso hacemos uso de este Spawner para mandar los servidores Jupyter de cada usuario a una cola Torque, que se encarga de ejecutar los *jobs* mediante el script que se la asigna. Aquí es donde entra en juego el siguiente Spawner, el `wrapspawner.ProfilesSpawner` A continuación, se muestra una parte del archivo `jupyter_config.py`, en la que se configura el BatchSpawner.

```
from batchspawner import PBSSpawner

JH_RUN_DIRECTORY = os.getenv('JH_RUN_DIRECTORY', os.getcwd())
#Add PATH to sys.path for the custom_pbs_spawner.py
custom_spawner_path = os.path.abspath(JH_RUN_DIRECTORY)
if custom_spawner_path not in sys.path:
    sys.path.insert(0, custom_spawner_path)
import custom_pbs_spawner

c.JupyterHub.spawner_class = custom_pbs_spawner.CustomPBSSpawner

c.Spawner.env = {'PATH': '/usr/local/torque-2.5.12/bin/qsub:/usr/local/bin:/
    usr/bin:/bin'}
```

```
13
14 c.Spawner.cmd = ["jupyter-labhub"]
15 c.BatchSpawnerBase.batch_script= '''#!/bin/sh
16 #PBS -q {queue}
17 #PBS -l walltime={runtime}
18 #PBS -l nodes=1:ppn={nprocs}
19 #PBS -l mem={memory}
20 #PBS -N jupyterhub-singleuser
21 #PBS -o /oceano/gmeteo/users/{username}/out
22 #PBS -e /oceano/gmeteo/users/{username}/err
23 #PBS -v PATH,CONDA_DEFAULT_ENV,LANG,JUPYTERHUB_API_TOKEN,JPY_API_TOKEN,
      JUPYTERHUB_CLIENT_ID,JUPYTERHUB_HOST,JUPYTERHUB_OAUTH_CALLBACK_URL,
      JUPYTERHUB_OAUTH_SCOPES,JUPYTERHUB_OAUTH_ACCESS_SCOPES,
      JUPYTERHUB_OAUTH_CLIENT_ALLOWED_SCOPES,JUPYTERHUB_USER,
      JUPYTERHUB_SERVER_NAME,JUPYTERHUB_API_URL,JUPYTERHUB_ACTIVITY_URL,
      JUPYTERHUB_BASE_URL,JUPYTERHUB_SERVICE_PREFIX,JUPYTERHUB_SERVICE_URL
24
25 env > $PBS_WORKDIR/$PBS_JOBID.env
26 eval "$(/software/jupyterhub/mambaforge/bin/conda shell.bash hook)"
27 conda activate base
28 which python
29 which batchspawner-singleuser
30 which jupyterhub-singleuser
31 {cmd}
32 '''
```

- **wrapspawner.ProfilesSpawner:** Este Spawner permite al administrador de JupyterHub la posibilidades de definir una serie de diferentes configuraciones de Spawners. Al usuario se le presenta un menú desplegable para escoger entre los diferentes Spawners. Este método permite una forma segura y sencilla de elegir diferentes configuraciones del `BatchSpawner`. En el siguiente fragmento de código se puede ver la configuración de perfiles en el JupyterHub de Ganymede.

```
1 c.JupyterHub.spawner_class = 'wrapspawner.ProfilesSpawner'
2 c.ProfilesSpawner.profiles = [
3     ('tintel - 1 core, 800 MB, 24 hours', 'tintel', 'custom_pbs_spawner.
        CustomPBSSpawner',
4       dict(req_nprocs='1', req_queue='tintel', req_runtime='24:00:00',
            req_memory='800mb')),
5     ('long - 1 core, 7 GB, 48 hours', 'long', 'custom_pbs_spawner.
        CustomPBSSpawner',
6       dict(req_nprocs='1', req_queue='long', req_runtime='48:00:00',
            req_memory='7gb')),
7     ('himem - 1 core, 32 GB, 48 hours', 'himem', 'custom_pbs_spawner.
        CustomPBSSpawner',
8       dict(req_nprocs='1', req_queue='himem', req_runtime='48:00:00',
            req_memory='32gb')),
9     ('himem4 - 4 core, 32 GB, 48 hours', 'himem4', 'custom_pbs_spawner.
        CustomPBSSpawner',
10      dict(req_nprocs='4', req_queue='himem', req_runtime='48:00:00',
            req_memory='32gb')),
```

```
]
```

Como se puede observar, en este caso existen cuatro opciones de configuración del BatchSpawner. En cada una de ellas se diferencia la cola, el número de nodos de cómputo, la memoria requerida y el tiempo de ejecución requerido. Además, cabe destacar, para un mejor entendimiento de la configuración, que se ha usado el spawner `PBSSpawner`, como extensión de la clase `BatchSpawner`, ya que es preferible para nuestro sistema. Además, se ha extendido la clase `PBSSpawner` en un nuevo Spawner personalizado llamado `CustomPBSSpawner`, en el que configuramos el Spawner a nuestro gusto.

A continuación se muestra el archivo `custom_pbs_spawner.py`, que contiene el `CustomPBSSpawner`:

```python
# custom_pbs_spawner.py

from traitlets import Unicode
from batchspawner import PBSSpawner

class CustomPBSSpawner(PBSSpawner):
    # outputs job id string
    batch_submit_cmd = Unicode("qsub").tag(config=True)
    # outputs job data XML string
    batch_query_cmd = Unicode("qstat -x {job_id}").tag(config=True)
    batch_cancel_cmd = Unicode("qdel {job_id}").tag(config=True)
    # search XML string for job_state - [QH] = pending, R = running, [CE] = done
    state_pending_re = Unicode(r"<job_state>[QH]</job_state>").tag(config=True)
    state_running_re = Unicode(r"<job_state>R</job_state>").tag(config=True)
    state_exechost_re = Unicode(r"<exec_host>((?:[\w-]+\.?)+)/\d+").tag(config=True)
    http_timeout = 3600
    start_timeout = 600
```

A continuación, en la figura 9 se pueden ver los cuatro perfiles que pueden ser elegidos para *spawnear* en la cola. Cada uno de ellos lanzará *jobs* con configuraciones diferentes.

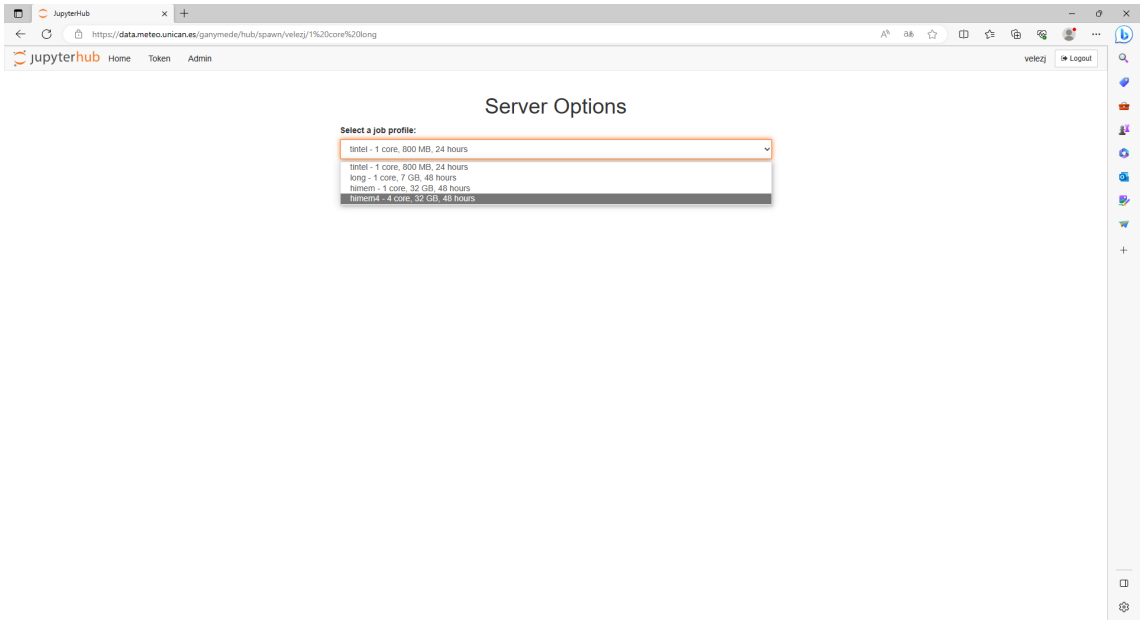

Figura 9: Perfiles del JupyterHub.

Además, cabe destacar que el JupyterHub ha sido configurado de forma que cada usuario puede tener un máximo de cinco servidores de usuario. Para configurar esta opción es necesario habilitar dos opciones en el archivo de configuración `jupyterhub_config.py`:

```
c.JupyterHub.allow_named_servers = True #Allow naming servers.
c.JupyterHub.named_server_limit_per_user = 5 #Maximun number of servers per user.
```

En la figura 10 se pueden observar los distintos servidores que los usuarios pueden crear.

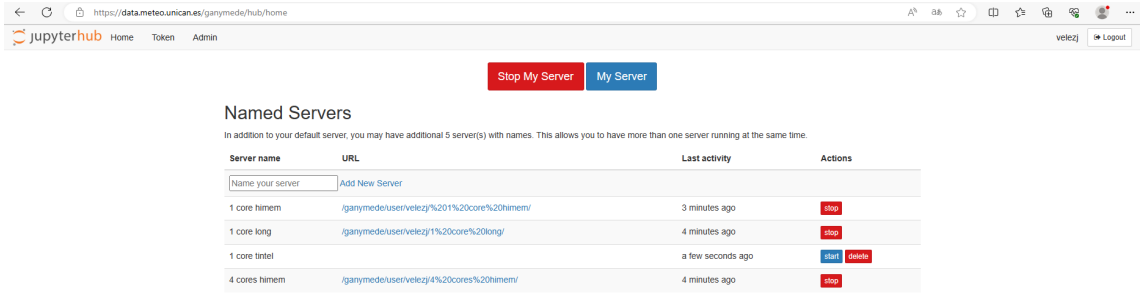

Figura 10: Servidores de usuario.

4.3.2. Kernels habilitados en el JupyterHub de Ganymede

En JupyterHub, un *kernel* es un programa que ejecuta y gestiona los cálculos para los *notebooks*. Cada notebook se asocia normalmente a un *kernel* y este es el encargado de ejecutar el código en un lenguaje de programación específico. los *kernels* pueden tener bibliotecas adicionales y dependencias instaladas para proporcionar funcionales más avanzadas.
En el caso del JupyterHub de Ganymede, se han desplegado tres *kernels*:

1. **jh-ganymede-pyGMS-3.7**: Este kernel se basa en Python 3.7 e incorpora una serie de bibliotecas poderosas para el análisis de datos y visualización, incluyendo:

   `xarray`: Es una biblioteca de Python diseñada específicamente para el manejo y análisis de datos multidimensionales etiquetados, como los datos climáticos. Está construida sobre las estructuras de datos de `pandas` y ofrece una abstracción poderosa y eficiente para trabajar con conjuntos de datos climáticos de gran tamaño.

   Una de las principales ventajas de `xarray` es su capacidad para etiquetar y organizar los datos a lo largo de múltiples dimensiones. Esto permite una representación más intuitiva y un acceso más fácil a los datos climáticos. Además, `xarray` proporciona una amplia gama de operaciones avanzadas de indexación, filtrado y agregación, lo que facilita el análisis y la manipulación de los datos. También ofrece una integración fluida con otras bibliotecas populares, como `numpy` y `matplotlib`, lo que amplía aún más su funcionalidad.

   `dask`: Es una biblioteca de Python diseñada para el cálculo paralelo y distribuido, especialmente útil cuando se trabaja con conjuntos de datos grandes que no caben en la memoria. `Dask` se integra de manera transparente con `xarray`, lo que permite aprovechar la potencia de `dask` para realizar operaciones eficientes en conjuntos de datos climáticos distribuidos.

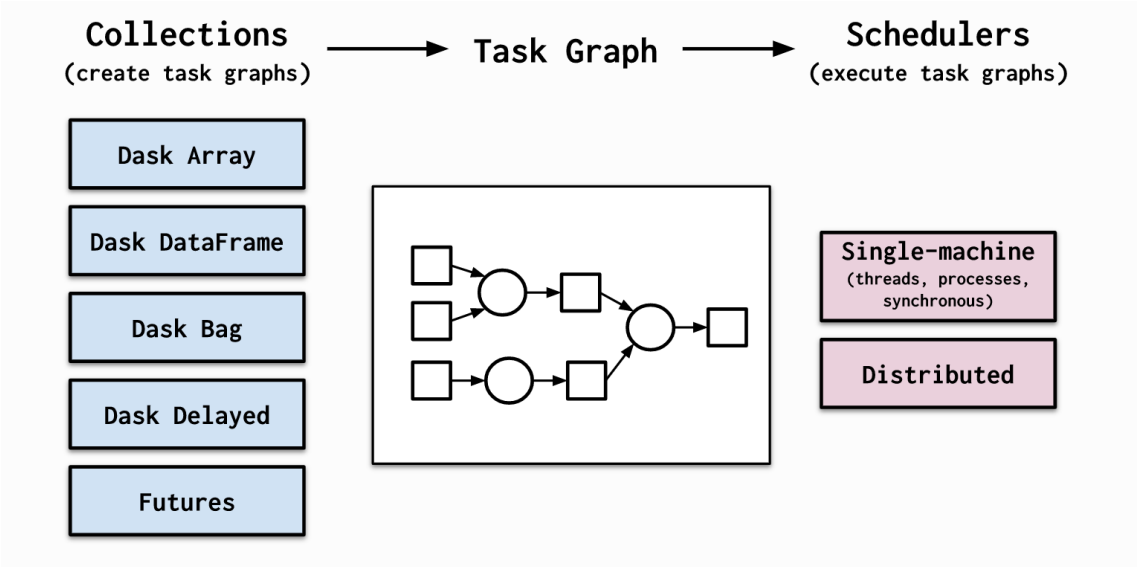

Figura 11: Dask. Fuente: [4]

La principal característica de `dask` es su capacidad para dividir el trabajo en tareas más

pequeñas y ejecutarlas en paralelo. Esto se logra mediante el concepto de "gráficos de tareas", donde las operaciones se representan como una serie de pasos intermedios que se pueden ejecutar en diferentes procesadores o incluso en diferentes máquinas. `Dask` optimiza automáticamente la ejecución de estas tareas para maximizar la eficiencia y minimizar el tiempo de procesamiento.

Al utilizar `dask` en conjunto con `xarray`, se puede aprovechar al máximo la capacidad de procesamiento paralelo y distribuido para manipular y analizar conjuntos de datos climáticos de gran escala. Esto permite realizar cálculos complejos y realizar operaciones en paralelo, acelerando significativamente los tiempos de procesamiento y permitiendo el análisis de datos más grandes y detallados.

En resumen, `xarray` y `dask` son bibliotecas complementarias y altamente compatibles que ofrecen una solución completa y eficiente para el manejo, análisis y visualización de datos climáticos. `Xarray` proporciona una interfaz intuitiva y flexible para trabajar con datos multidimensionales etiquetados, mientras que `dask` habilita el cálculo paralelo y distribuido para el procesamiento eficiente de conjuntos de datos climáticos de gran tamaño. La combinación de estas bibliotecas facilita el análisis de datos climáticos a gran escala y promueve un flujo de trabajo eficiente en entornos de análisis de datos climáticos.

jh-ganymede-pyGMS-3.11: Este kernel es una versión actualizada del entorno de Python **jh-ganymede-pyGMS-3.7** que se basa en Python 3.11. Al igual que su versión anterior, este kernel está diseñado específicamente para el análisis de datos climáticos y proporciona un conjunto de bibliotecas poderosas y útiles.

Además de las bibliotecas mencionadas anteriormente, el kernel **jh-ganymede-pyGMS-3.11** incluye dos bibliotecas adicionales de aprendizaje profundo:

`tensorflow`: Es una biblioteca de aprendizaje automático y aprendizaje profundo de código abierto desarrollada por Google. TensorFlow proporciona un entorno flexible para construir y entrenar modelos de aprendizaje automático en una variedad de plataformas. Ofrece una amplia gama de herramientas y funciones para construir redes neuronales, realizar cálculos numéricos eficientes y optimizar modelos de aprendizaje profundo.

`keras`: Es una biblioteca de alto nivel para el desarrollo de modelos de aprendizaje profundo que se ejecuta sobre TensorFlow. Keras ofrece una interfaz sencilla y modular para construir redes neuronales, lo que la hace muy accesible para principiantes y facilita el prototipado rápido de modelos. Permite construir modelos de aprendizaje profundo utilizando capas predefinidas que se pueden combinar y personalizar para crear arquitecturas de redes neuronales complejas.

Al incluir las bibliotecas `tensorflow` y `keras`, el kernel **jh-ganymede-pyGMS-3.11** proporciona capacidades adicionales para el análisis de datos climáticos utilizando técnicas de aprendizaje profundo. Estas bibliotecas permiten construir, entrenar y desplegar modelos de aprendizaje automático y aprendizaje profundo en el contexto de los datos climáticos, lo que abre nuevas posibilidades para el análisis y la predicción basada en datos climáticos.

2. **jh-ganymede-climate4R**: Este kernel está construido sobre el lenguaje de programación R y está especialmente diseñado para trabajar con datos climáticos utilizando el framework de

climate4R. Este entorno proporciona una serie de paquetes y herramientas específicamente desarrollados para el análisis y procesamiento de datos climáticos.

El kernel **jh-ganymede-climate4R** incluye los siguientes paquetes y componentes:

Conjunto de paquetes climate4R: Estos paquetes, como `r-easyverification`, `r-loader`, `r-transformer`, `r-downscaler`, `r-visualizer`, `r-climate4r.value`, `r-climate4r.udg`, `r-value`, `r-loader.java`, `r-convertr`, y `r-drought4r`, forman el núcleo del framework climate4R. Cada uno de estos paquetes proporciona funcionalidades específicas para realizar tareas como carga de datos, transformaciones, verificación de modelos, downscaling, visualización y más. Estos paquetes ofrecen una amplia gama de funciones y métodos para el análisis y manipulación de datos climáticos.

`r-irkernel`: Es un paquete que permite la integración de R con JupyterHub, lo que te permite ejecutar código R en un entorno de cuaderno interactivo. Con este paquete, puedes aprovechar todas las capacidades de R dentro de JupyterHub y utilizar las funcionalidades del kernel **jh-ganymede-climate4R** de manera eficiente.

`nbgitpuller`: Es una extensión de Jupyter que facilita la sincronización de notebooks con repositorios Git remotos. Con esta extensión, puedes mantener tus notebooks actualizados y sincronizados con los cambios realizados en un repositorio Git, lo que facilita el trabajo colaborativo y el seguimiento de versiones de tus análisis climáticos.

`cdo`: Es una herramienta de línea de comandos utilizada para manipular y analizar datos de archivos netCDF. CDO (Climate Data Operators) proporciona un conjunto completo de operaciones y funciones para el procesamiento y análisis de datos climáticos en formato netCDF. Con esta herramienta, puedes realizar operaciones como promedios espaciales y temporales, re-muestreo, interpolación, agregación y muchas otras transformaciones en tus datos climáticos.

En resumen, el kernel **jh-ganymede-climate4R** está diseñado específicamente para trabajar con datos climáticos utilizando R y el framework climate4R. Proporciona un conjunto de paquetes y herramientas especializadas que permiten la carga, manipulación, análisis y visualización de datos climáticos de manera eficiente. Además, ofrece integración con JupyterHub, sincronización con repositorios Git y acceso a la potente herramienta `cdo` para el procesamiento avanzado de archivos netCDF.

Estos kernels personalizados proporcionan a los usuarios un entorno de ejecución potente y flexible para el análisis de datos. En la siguiente figura 12 se pueden ver los kernels instalados, en la máquina, los *kernels* mencionados anteriormente son los principales.

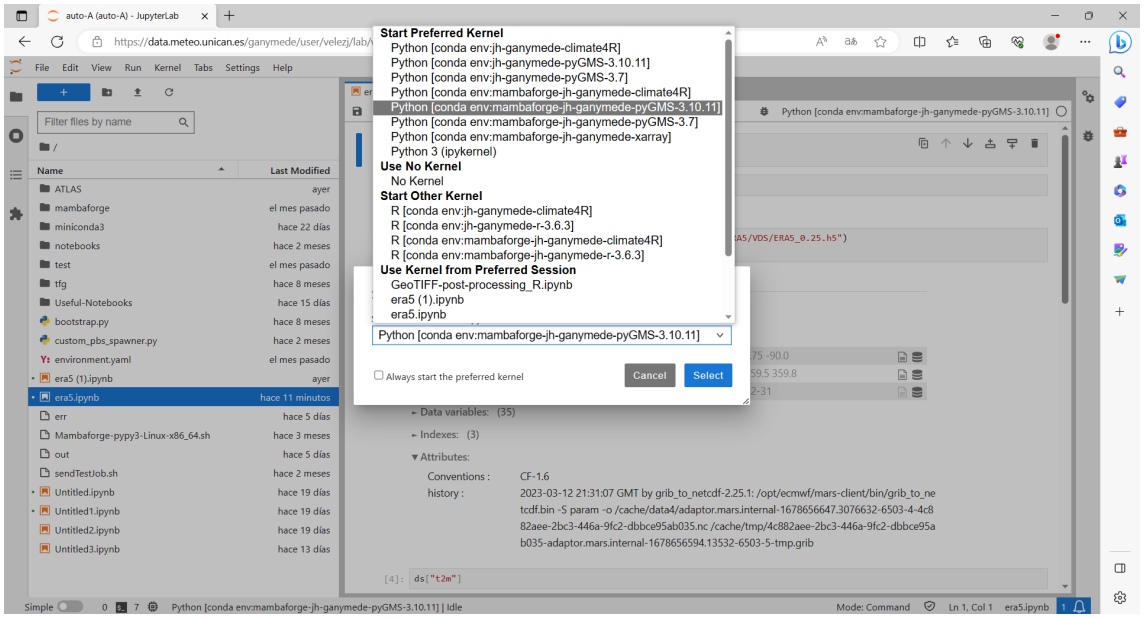

Figura 12: Kernels.

4.3.3. Autenticación en el JupyterHub de Ganymede

La autenticación es un componente crucial en cualquier aplicación que requiera un acceso de usuario, por lo tanto es de vital importancia tener un control en JupyterHub. Existen varios métodos de autenticación en JupyterHub, lo que permite al administrador elegir el que mejor se adapte a las necesidades del proyecto.
En el caso del JupyterHub de Ganymede, inicialmente se consideraron dos métodos de autenticación: GitHub y PAM (Pluggable Authentication Module).

1. **Autenticación de GitHub**: Este método permite a los usuarios iniciar sesión en JupyterHub utilizando sus credenciales de GitHub. Aunque este método puede ser útil en entornos donde los usuarios ya tienen cuentas de GitHub y estas cuentas se utilizan para gestionar el acceso a otros recursos, requiere acceso a Internet y no es adecuado si los usuarios no tienen una cuenta de GitHub o si se prefiere mantener el control de la autenticación dentro de la propia organización.

   El proceso de autenticación de GitHub con OAuth2 implica los siguientes pasos:

   a) Crear una aplicación en GitHub.

   b) Proporcionar detalles de la aplicación, como nombre, descripción y URL de autorización de redireccionamiento.

   c) Obtener el Client ID y Client Secret de la aplicación creada en GitHub.

   d) Implementar el flujo de autenticación en la aplicación JupyterHub utilizando el Client ID y Client Secret.

   e) Redirigir al usuario a la página de autorización de GitHub para que inicie sesión y otorgue permisos a la aplicación.

f) Intercambiar el código de autorización por un token de acceso a través de la API de GitHub.

g) Utilizar el token de acceso para realizar solicitudes a la API de GitHub en nombre del usuario autenticado.

Como se puede ver en la figura 13, el inició se sesión por Github predeterminado es:

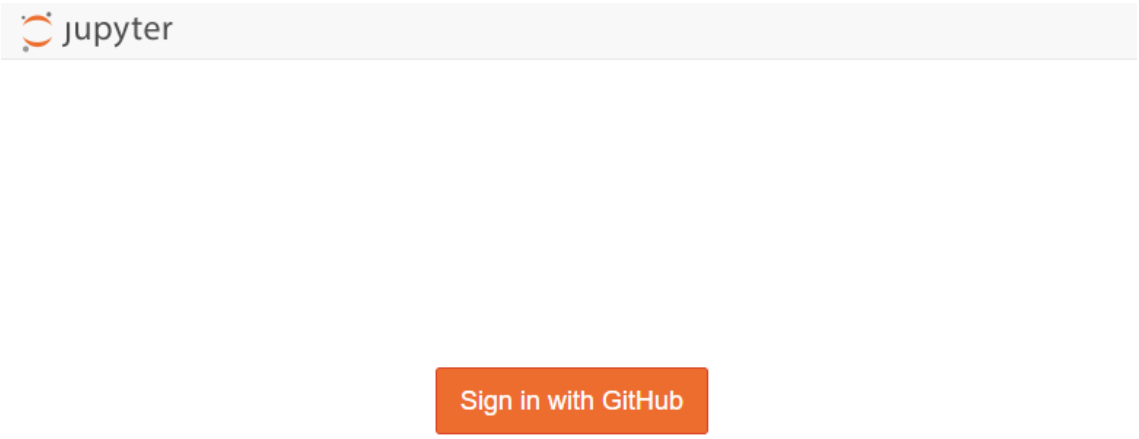

Figura 13: Login por GitHub.

2. **Autenticación PAM**: Este método permite a los usuarios iniciar sesión en JupyterHub utilizando las mismas credenciales que utilizan para iniciar sesión en el sistema que aloja JupyterHub. PAM es un sistema de autenticación flexible que se utiliza en muchos sistemas Unix y Linux, y permite a los administradores del sistema controlar cómo se autentican los usuarios.

Aunque las dos opciones han sido implementadas y pueden llegar a usarse conjuntamente, finalmente, se decidió utilizar únicamente la autenticación PAM para el JupyterHub de Ganymede.

Esta decisión se tomó debido a varias razones. Se prefirió mantener la autenticación dentro de la organización en lugar de depender de un proveedor externo como GitHub. Además se prefiere que todos los usuarios de JupyterHub tengan cuentas en el sistema anfitrión, por lo que tiene sentido utilizar las mismas credenciales para JupyterHub. Además, la autenticación PAM no requiere acceso a Internet, lo cual es una ventaja si el acceso a Internet es limitado o intermitente.

La autenticación es un aspecto importante de la configuración de JupyterHub, ya que controla quién puede acceder al sistema. Al elegir un método de autenticación, los administradores del sistema deben tener en cuenta factores como la facilidad de uso para los usuarios, la seguridad y la integración con otros sistemas. En el caso del JupyterHub de Ganymede, la autenticación PAM era la mejor opción dados los requisitos y las circunstancias en la actualidad.

5. Resultados y análisis

Para llevar a cabo esta evaluación, se creará un caso de uso sencillo que acceda de manera completa a una variable del dataset ERA5 disponible en el sistema HPC del SMG. El dataset tiene un tamaño en disco de 3.2TB teniendo en cuenta todo el conjunto de variables que conforman el dataset. Para llevar a cabo el test de rendimiento, se ha realizado un notebook que se ejecuta usando el despliegue de JupyterHub implementado en este trabajo. Para evaluar la escalabilidad del sistema, se ejecutará el test de rendimiento usando distinta cantidad de procesos.
A continuación, se muestran los resultados de la ejecución del *notebook*.

- **Importación de bibliotecas:**

```
import xarray
import dask
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Esta sección consiste en la importación de las bibliotecas necesarias para el análisis de los datos y la visualización de los resultados.

- **Configuración de Dask:**

```
dask.config.set(scheduler="processes")
```

Aquí se configura Dask para utilizar el esquema de programación de "processes", lo que permite realizar cálculos en paralelo utilizando múltiples procesos.

- **Carga de datos:**

```
ds = xarray.open_dataset("/lustre/gmeteo/WORK/DATA/C3S-CDS/ERA5/VDS/ERA5_0.25.h5")
```

En esta sección se carga el conjunto de datos.

- **Selección y fragmentación de datos:**

```
t2m = ds["t2m"].chunk({"time": 10})
```

Aquí se selecciona la variable "t2m"del conjunto de datos y se fragmenta en bloques más pequeños utilizando el método 'chunk' de Dask. Aquí se selecciona la variable "t2m"del conjunto de datos y se fragmenta en bloques más pequeños utilizando el método 'chunk' de Dask. A continuación, en la figura 14, se muestra la imagen resultado:

[13]: xarray.DataArray 't2m' (**time**: 30316, **latitude**: 721, **longitude**: 1440)

	Array	Chunk
Bytes	117.25 GiB	39.61 MiB
Shape	(30316, 721, 1440)	(10, 721, 1440)
Dask graph	3032 chunks in 2 graph layers	
Data type	float32 numpy.ndarray	

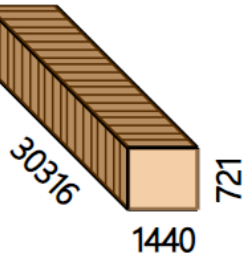

Figura 14: Tamaño de la variable en memoria.

- **Cálculo de la media de temperatura:**

```
t2m_mean = t2m.mean(["latitude", "longitude"]).compute(num_workers=4)
```

Se calcula la media de la temperatura utilizando la variable 't2m' a lo largo de las dimensiones "latitudez "longitude". El cálculo se realiza utilizando Dask y se especifica el número de trabajadores como 4 para realizar el cálculo en paralelo.

- **Remuestreo y gráfico:**

```
t2m_mean.resample({"time": "Y"}).mean().plot()
```

En esta sección se realiza un remuestreo de los datos de temperatura media ('t2m_mean') utilizando el método 'resample'. Luego, se calcula la media de los datos remuestreados utilizando el método 'mean'. Finalmente, se genera un gráfico 15 que muestra la temperatura media a lo largo del tiempo.

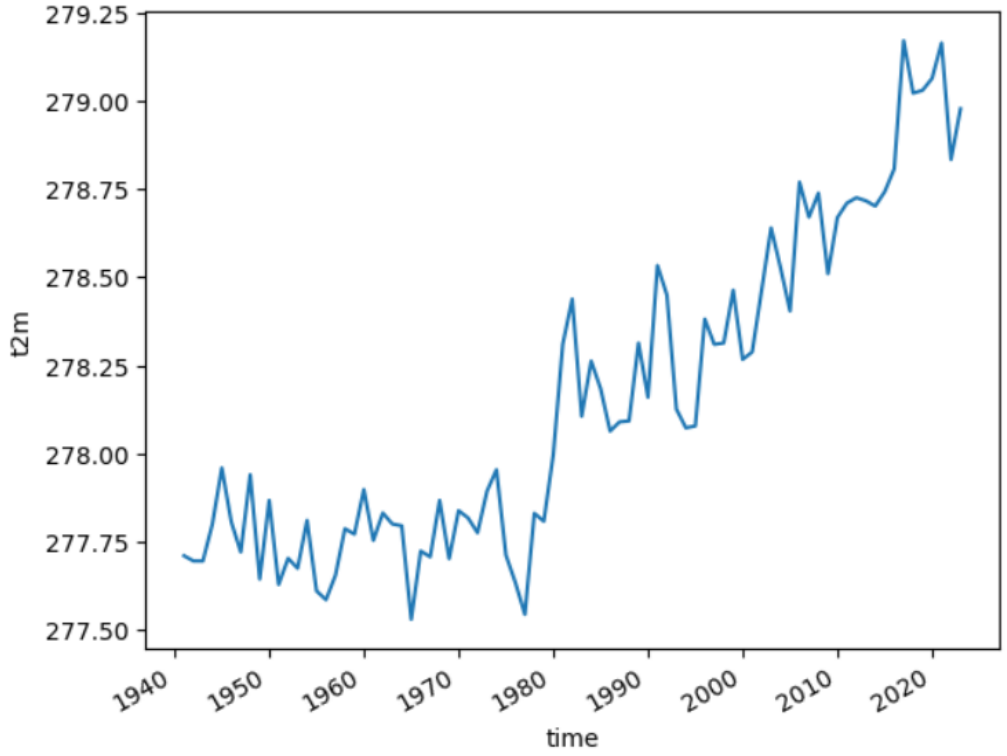

Figura 15: Gráfico que muestra el *resample*.

- **Cálculos paralelos y rendimiento:**

```
results = []
nworkers = [1,2,4]

for n in nworkers:
    start = time.time()
    t2m.mean(["latitude", "longitude"]).compute(num_workers=n)
    end = time.time()
    results.append((n,end-start))

df = pd.DataFrame(results, columns=["workers", "seconds"])
df["throughput"] = t2m.size * t2m.dtype.itemsize / 2**20 / df["seconds"]
```

En esta sección se realizan cálculos paralelos utilizando diferentes cantidades de trabajadores (‘nworkers‘). Se mide el tiempo de ejecución utilizando la función ‘time.time()‘ y se registra en la lista de resultados. Luego, se crea un DataFrame de pandas (‘df‘) con los resultados de los tiempos y se calcula el rendimiento en términos de rendimiento de datos (‘throughput‘).

- **Gráficos de rendimiento:**

```
fig, axes = plt.subplots(1,2,figsize=(8,4))
df.plot.bar(x="workers", y="seconds", title="Elapsed time (s)", legend=False,
    xlabel="Workers", ax=axes[0], rot=0)
df.plot.bar(x="workers", y="throughput", title="Throughput (MiB/s)", legend=
    False, xlabel="Workers", ax=axes[1], rot=0)
```

En esta sección se generan dos gráficos de barras utilizando los datos del DataFrame 'df'. El primer gráfico 16 muestra el tiempo transcurrido en función del número de trabajadores, mientras que el segundo gráfico 16 muestra el rendimiento en términos de rendimiento de datos en función del número de trabajadores.

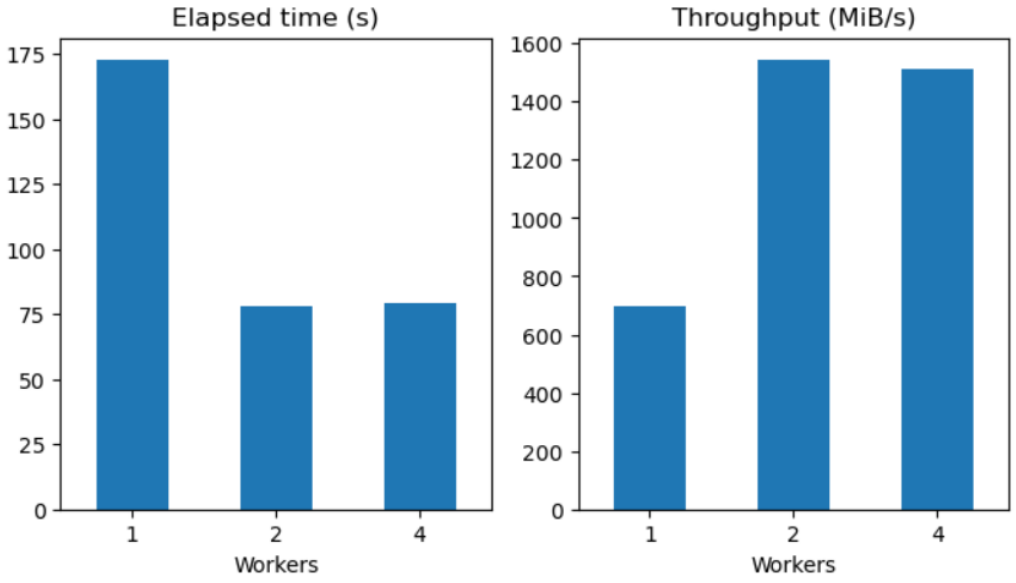

Figura 16: Gráfico que muestra el tiempo y el *throughput*.

6. Conclusiones y trabajos futuros

En esta sección se tratara de explicar el cumplimiento o no de los objetivos iniciales, de los problemas que ha habido, y de lo que queda por hacer en un futuro.
Como objetivo principal, en este proyecto se debía implementar un JupyterHub en el sistema HPC del Grupo de Meteorología de Santander que ofreciece el acceso a los investigadores del mismo grupo, así como el uso de las nodos del HPC. También se debía hacer un estudio de viabilidad de varias formas de instalación de JupyterHub. Estos objetivos se han cumplido, así como el resto de objetivos menores.

La realización de este proyecto ha supuesto un desafío inicial debido a la necesidad de trabajar en un entorno basado en Linux. Dado que mi sistema operativo habitual es Windows, las tareas que requerían el uso de Linux avanzado fueron complicadas, pero con el paso el tiempo conseguí un dominio correcto para la realización del resto de tareas.

En cuanto a los problemas surgidos, hubo bastantes, debido a que dependemos de aplicaciones que están siendo actualizadas actualmente y de manera continua. Por ejemplo, a la hora de descargar e instalar el BatchSpawner, la versión actual tenia un *bug*, días antes a verlo nosotros, un usuario de la comunidad consiguió corregirlo y abrir un *commit*, entonces tuvimos que descargar el *BatchSapwner* desde un *commit* especifico, debido a esto, estuvimos bastantes días buscando un problema que en realidad no dependía de nosotros directamente. En esta ocasión, aprendí la manera de instalar desde el propio repositorio y desde un *commit* en concreto.
La manera de instalarlo fue esta:

```
pip install 'git+https://github.com/jupyterhub/batchspawner@COMMIT\_ID#egg=
    batchspawner'}
```

Para mencionar algunos problemas menores, un objetivo que fue bastante dificil de conseguir fue la ejecución de los servidores de usuario en los nodos del HPC, debido a que queríamos que fuera el propio usuario el que lanzara el *job* a la cola, y no el usuario auxiliar `rhea`. Esto se consiguió modificando la instrucción que manda el BatchSpawner.

Se ha intentado seguir la planificación establecida al comienzo del proyecto, pero al surgir percances inesperados que han tomado más tiempo de lo esperado, se ha hecho difícil conseguir ajustarse a los plazos establecidos. A pesar de esto, el proyecto ha resultado exitoso y funciona correctamente, aunque tiene posibles mejoras.

Como trabajos futuros, se quiere volver a activar el inicio de sesión por Github, y hacer un mapeo de alguna manera con los usuarios que ya tienen usuario en el HPC. Además, esn un futuro se quiere incorporar un *DataCube*, ya que en nuestro proyecto nos hemos quedado en poder facilitar el acceso al *Data-Lake* que hay en el clúster.

Por mi parte, la realización de este proyecto me ha permitido entender algunas de las cosas que he estudiado en el máster y en la carrera, que no había llegado a ver en el mundo real. Asimismo, me

he dado cuenta que no todo sale como en un principio planificas debido a ciertas dificultades que te encuentras por el camino, las cuáles son mas difíciles de corregir que lo que uno en una primera instancia piensa. También me he dado cuenta de que a veces no todo está en Internet, y que hay que desarrollar por primera vez, ya que nadie lo ha hecho nunca.

En conclusión, me gustaría dar las gracias a mis dos tutores, Ezequiel y Antonio, por ayudarme con todo y juntos conseguir que este proyecto salga adelante. También me gustaría dar las gracias a Nuria, por estar siempre atenta a cualquier percance que surgiera con la máquina Ganymede. Desarrollar este proyecto me ha parecido una buena manera de dar por finalizado el máster y de dar un uso real a muchos de los conocimientos aprendidos durante este años.

Referencias

[1] S. Cholia, L. Heagy, M. Henderson, D. Paine, J. Hays, L. Bianchi, D. Ghoshal, F. Perez, and L. Ramakrishnan, "Towards interactive, reproducible analytics at scale on HPC systems," in *2020 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*. IEEE, pp. 47–54. [Online]. Available: https://ieeexplore.ieee.org/document/9307626/

[2] "Choosing Multiuser Jupyter Platform." [Online]. Available: https://blog.reviewnb.com/choosing-multiuser-jupyter-platform/#amazon-sagemaker

[3] "Jupyterhub architecture." [Online]. Available: https://jupyterhub.readthedocs.io/en/latest/reference/technical-overview.html

[4] "Dask Documentation." [Online]. Available: https://docs.dask.org/en/stable/

[5] "SantanderMetGroup." [Online]. Available: https://github.com/SantanderMetGroup

[6] V. Eyring, S. Bony, G. A. Meehl, C. A. Senior, B. Stevens, R. J. Stouffer, and K. E. Taylor, "Overview of the coupled model intercomparison project phase 6 (CMIP6) experimental design and organization," vol. 9, no. 5, pp. 1937–1958. [Online]. Available: https://www.geosci-model-dev.net/9/1937/2016/

[7] D. N. Williams, V. Balaji, L. Cinquini, S. Denvil, D. Duffy, B. Evans, R. Ferraro, R. Hansen, M. Lautenschlager, and C. Trenham, "A Global Repository for Planet-Sized Experiments and Observations," *Bulletin of the American Meteorological Society*, vol. 97, no. 5, pp. 803–816, May 2016. [Online]. Available: https://journals.ametsoc.org/doi/10.1175/BAMS-D-15-00132.1

[8] H. Hersbach, B. Bell, P. Berrisford, S. Hirahara, A. Horányi, J. Muñoz-Sabater, J. Nicolas, C. Peubey, R. Radu, D. Schepers, A. Simmons, C. Soci, S. Abdalla, X. Abellan, G. Balsamo, P. Bechtold, G. Biavati, J. Bidlot, M. Bonavita, G. Chiara, P. Dahlgren, D. Dee, M. Diamantakis, R. Dragani, J. Flemming, R. Forbes, M. Fuentes, A. Geer, L. Haimberger, S. Healy, R. J. Hogan, E. Hólm, M. Janisková, S. Keeley, P. Laloyaux, P. Lopez, C. Lupu, G. Radnoti, P. Rosnay, I. Rozum, F. Vamborg, S. Villaume, and J. Thépaut, "The ERA5 global reanalysis," vol. 146, no. 730, pp. 1999–2049. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/qj.3803

[9] G. Eynard-Bontemps, R. Abernathey, J. Hamman, A. Ponte, and W. Rath, "The pangeo big data ecosystem and its use at CNES." [Online]. Available: https://archimer.ifremer.fr/doc/00503/61441/

[10] C. Stern, R. Abernathey, J. Hamman, R. Wegener, C. Lepore, S. Harkins, and A. Merose, "Pangeo forge: Crowdsourcing analysis-ready, cloud optimized data production," vol. 3, p. 782909. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fclim.2021.782909/full

[11] "batchspawner." [Online]. Available: https://github.com/jupyterhub/batchspawner

[12] "PBS Queue." [Online]. Available: https://docs.pace.gatech.edu/software/PBS_script_guide/

[13] “The Littlest JupyterHub.” [Online]. Available: https://tljh.jupyter.org/en/latest/

[14] “SSH.” [Online]. Available: https://en.wikipedia.org/wiki/Web-based_SSH

[15] “Running JupyterHub without sudo.” [Online]. Available: https://jupyterhub.readthedocs.io/en/stable/howto/configuration/config-sudo.html

[16] “Running proxy separately from the hub.” [Online]. Available: https://jupyterhub.readthedocs.io/en/stable/howto/separate-proxy.html