Docker - based Implementation for an Astronomical Data Analysis Cloud Service

M. Diaz[1], M. Araya[1], C. Jauregui, C. Valenzuela[1], L. Pizarro [1], M. Osorio[1], and M. Solar[1]
[1]*Universidad Técnica Federico Santa María, Valparaíso, Chile;*
madiaz@alumnos.inf.utfsm.cl

Abstract. The data deluge problem that astronomy research confronts not only require developing new algorithms and computing infrastructure, but also fostering modern software and services. For that matter, the cloud computing paradigm allows to perform the processing of the data where it is, minimizing the transfer to the end-user and taking advantage of the high-performance computing infrastructure of the data-centers. However, new challenges arise when these services are deployed under the high-availability principle in order to compete with the convenience of local processing. Between them we found multi-user support, load balancing, resource usage optimization, securing data and network security. In this paper we report the architecture and caveats of the deployment of JOVIAL: a notebook-based cloud service specifically designed for astronomy using the JupyterHub platform. This platform allows users to run arbitrary code, which is is at the same time its main advantage and menace. Therefore, we used Docker containers to spawn a Jupyter server on demand for each user, which increased both the protection of the user data and the protection of the hosts from users. Following the high-availability principle, the containers have to be orchestrated making good use of the high-performance infrastructure. For achieving this, we combined our user-based docker containers with the Kubernetes system, providing on-demand growth and load balancing between the hosts. We are currently exploring supporting high-availability of the storage system through LustreFS, and managing the whole infrastructure with Rancher, a software that provides user-friendly orchestration of docker within a Kubernetes deployment.

1. Introduction

The Virtual Observatory (VO) is not only about data access, search and transport. The management of large volumes of data poses new challenges for data analysis, which are forcing these computations to be made at data centers rather that at personal computers. (Djorgovski et al. 2003). Even though specialized libraries and computational tools designed to cope with this data deluge are now available, the platforms and services to run them at high-performance environments are less developed. In the Chilean Virtual Observatory (Solar et al. 2015) we are building a service that provides a web platform for astronomers to run code, visualize data and export/import their files to an account at our data center. The implementation of this platform, that we named "Jovial", requires the coordination of many software and hardware components. It's based on a JupyterHub server (Jupyter 2016) whose Docker containers are orchestrated by Kubernetes (Bernstein 2014). Besides this, users need an account where to save their code

and data. these files are managed by the LustreFS file system that allow us to manage huge amount of data with low latency (Braam & Zahir 2002). The sections 2 explains the gradual implementation of Jovial and the section 3 explains the user experience on it.

2. Jovial

2.1. Jupyter Notebook and JupyterHub

We selected Jupyter Notebook, which brings tools for programing, visualization, storage of files and other features, as our data analysis application. Each notebook in the application uses a kernel where the codes are processed, for this reason we use the Ipython kernel that allows the users to run code wrote in python.

The Jupyter Notebook server only allows one user per instance, so we implemented JupyterHub which allows multiple users, providing each user a Jupyter notebook through an account. A user can download a dataset inside its notebook and process it using the processing power, bandwidth and disk space of the data center instead of on its own computer, additionally all the work is stored in the account. We refer at the process of bringing a notebook to a user as "spawn", and where it spawns is determined by the "spawner".

By default the user of the account is a generic system user and the notebook spawns in the host server. This combined with the feature of running random code in the notebooks represents a risk for the server and the others users, for this reason we made two decisions:

- The notebooks have to spawn on an isolated instance with a limit of resources in order to keep the server and the other users safe.
- Each user has a private account with a file system where they can store their work.

These two decisions complement each other since the notebook use the file system of the user to exist, for this, the file system must be accessible by the spawner, and the spawner has to have writing and reading permission on the files.

2.2. Isolation and file system

The isolation goal can be accomplished by using Docker containers, these containers recreate a system environment (like a virtual machine but lighter) inside the server, so if we spawn the notebook inside a docker container the user will only see the system of the container. To use this approach, we have to provide the container with the files of the user through a docker volume, mounting the user's file system on the host that holds the container.

We use LustreFS to manage the users' files. LustreFS is a distributed file system optimized to operate with low latency over a large amount of data (Terabytes and Petabytes). This feature is fundamental since we work with astronomical data. The file system of the users is mounted through InfiniBand (a high performance low-level data transfer protocol) on the same server where docker containers are hosted. Then these files are passed to the containers as docker volumes.

2.3. Cloud control

Docker needs a host where to put its containers. This host is a node of a cluster. A node is a physical server that is a Lustre client, so it can mount the users' files. But if the node fails all the infrastructure fails. In order to make the infrastructure robust against failure of the node, we distributed the infrastructure through several nodes, so if one of these nodes fails the others can still support Jovial.

When we talk about distributing the infrastructure we mean to spawn the notebooks in several docker containers through different nodes. These containers have to connect with the Hub. This is not a trivial task so we leave it up to Kubernetes, a Docker container orchestrator. Kubernetes decides then where to create the container based on the current load of the nodes. If one node fails, Kubernetes will migrate all the containers to other available nodes. So Kubernetes is not only a load balancer but it ensures high availability of the infrastructure. For this reason, not only the notebooks are put in container managed by Kubernetes but also the JupyterHub server itself is put in a container (in this paper Kubernetes pods and Docker containers are treated as the same). The whole database and all the roles and daemons of Kubernetes are distributed and replicated through the nodes so that the infrastructure becomes resistant to the failure of multiple nodes.

For networking, Kubernetes uses a service called Flanneld that gives each container an IP address in a logic private network (different from the one of the nodes) that connects the containers all across the nodes (Zismer 2016).

The implementation of Kubernetes was made by Kubeadm since the management of all the Kubernetes deployment is a complex task for a person.

2.4. Kubernetes and the spawner

Users are managed globally using LDAP, a software protocol that makes entries in which important data of the user is stored (the user's id, name, etc). The spawner that allows us to use Kubernetes for JupyterHub is KubeSpawner. KubeSpawner takes the id and the name of the user from LDAP, recreates the same user in the container and mounts the user's file system in the container as a docker volume. This recreated user owns the file mounted in the container since it has the same id and name as the real one. Finally, KubeSpawner runs the notebook inside the file system of the user (that is, inside the container).

3. The Whole Picture

We keep all the servers in a private network and the public access to the JupyterHub server is gave by a proxy.

In general, the user enters the portal through the proxy and logs in with its account. Immediately, the JupyterHub server checks if the container with the Jupyter instance of the user already exists. If not, it asks Kubernetes to create it, spawns the files of the user into the container and contacts the server. An overview of the components that participate in this process is shown in Figure 1.

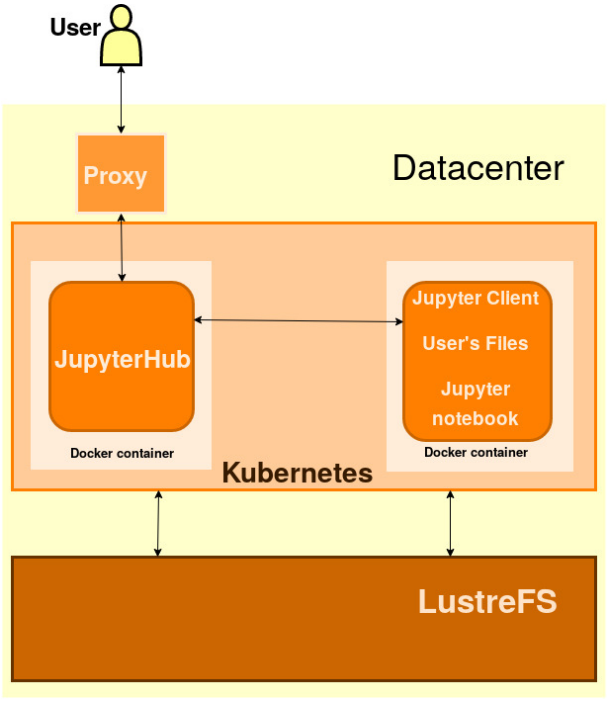

Figure 1. Abstraction of the infrastructure.

4. Conclusion

The objective of Jovial is to allow users to login to a personal account at a web application, download astronomical data from the virtual observatory to this account at the speed of the local network, and upload and download files selectively to their computers or another storage media.

With the notebooks distributed and confined to containers we can control the correct use of ours resources and prevent the users to compromise the server or the work of other users. The selection and arrangement of these particular tools was done in order to ensure high availability of the service and the protection of users' data.

Acknowledgments. This work has been partially funded by - FONDEF IT 15I10041. An special thanks to the JupyterHub team for their support on this project.

References

Bernstein, D. 2014, Containers and cloud: From lxc to docker to kubernetes
Braam, P. J., & Zahir, R. 2002, Lustre: A scalable, high performance file system
Djorgovski, S., Brunner, R., Mahabal, A., Williams, R., Granat, R., & Stolorz, P. 2003, in Statistical Challenges in Astronomy (Springer), 127–141
Jupyter, J. 2016, Github repository
Solar, M., Araya, M., Arévalo, L., Parada, V., Contreras, R., & Mardones, D. 2015, in Computing Conference (CLEI), 2015 Latin American (IEEE), 1
Zismer, A. 2016, Performance of docker overlay networks