An Overview of the Architecture of Juno: CHPC’s New JupyterHub Service

By Luan Truong, CHPC, University of Utah

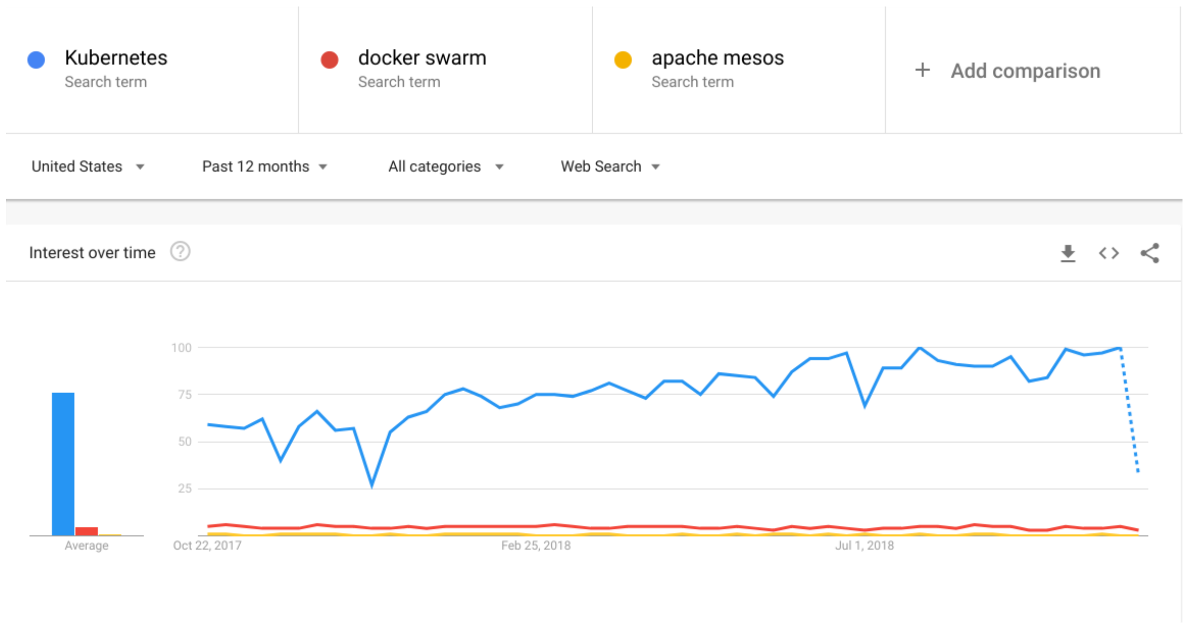
Introduction

Jupyter notebooks have emerged as a popular and open-source web application that allows users to create and share documents that contain code (in dozens of languages, most notably Python, R, and Julia), equations, visualizations and narrative texts. The notebooks have been adapted to provide environments for data analytics, statistical modeling, numerical simulation, and machine learning, among others. Users like Jupyter notebooks since they are simple to both learn and to use, and since they allow general use as a reliable customized workspace accessible from anywhere in the world, and on any device that can run a browser. The Jupyter notebook infrastructure is amenable to hands-on courses where each student can develop their own notebook, and the elegant design facilitates the development of additional workloads such as presentations, tests, visualizations, and group workspaces. Thus, support for Jupyter notebooks have been requested by a number of coursemasters, initially by faculty in chemical engineering, to teach a handful of programming and simulation classes.

The Center for High Performance Computing at the University has been working to help instructors with Jupyter notebook instances, however the popularity and demand has far exceeded resources available to the instructors. To overcome both limitations in scalability and difficulties in provisioning, the Jupyter notebook needed to be migrated from its default, single instance installation to a multi-user capable implementation that can scale flexibly and securely on a simple infrastructure. An ideal platform for this uses Kubernetes (a open-source container orchestration tool) and JupyterHub (which is a multi-user service that manages multiple instances of the single-user Jupyter notebook server) within a virtualized environment on dedicated compute resources.

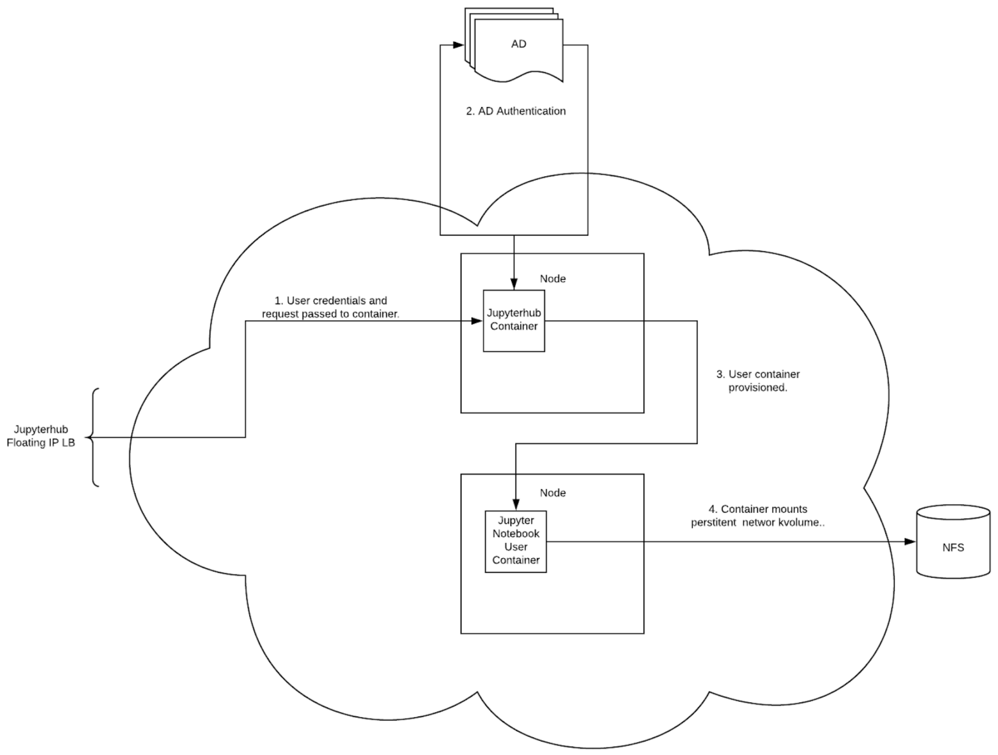
A Brief History

Borg, the predecessor to Kubernetes and Google’s best kept secret, is an ultra-resilient cluster management system that orchestrates at the container level. Container infrastructure is the way of the future. Many developers from Google jumped over to the Kubernetes project, incorporated some of its best ideas, and made it open source for all. Today, many common, daily utilized applications (Facebook, Github, Uber, etc.) are hosted on some form of container infrastructure whether it be Docker Swarm, Apache Mesos or Kubernetes. At the time of the paper,as shown by the above plot, Kubernetes owns the market, and many organizations running on Swarm or Mesos infrastructure are transitioning towards Kubernetes.



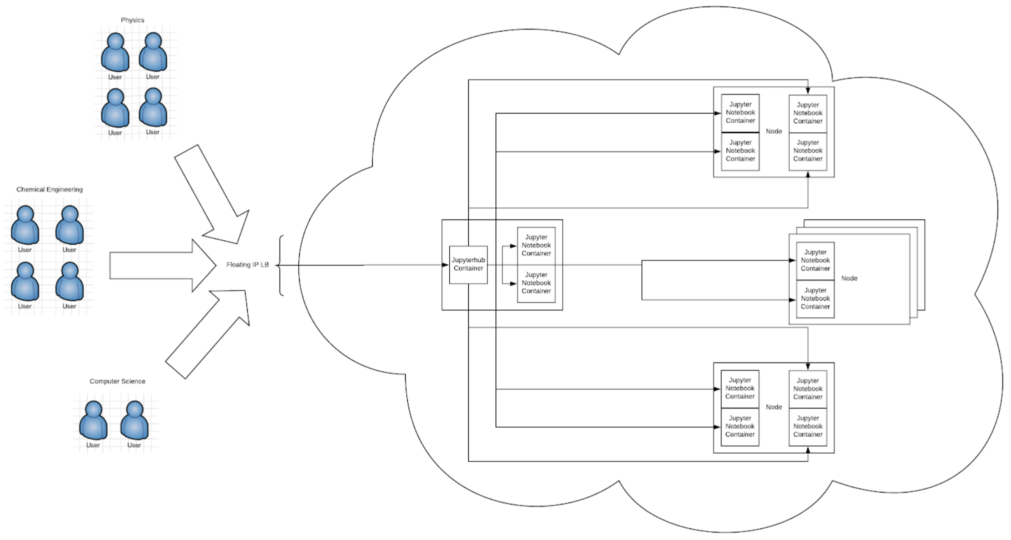
A schematic of the architecture of Jupyterhub on Kubernetes

Disclaimer: This is only a general architecture, and there are many Kubernetes specific objects and processes left out for simplicity.

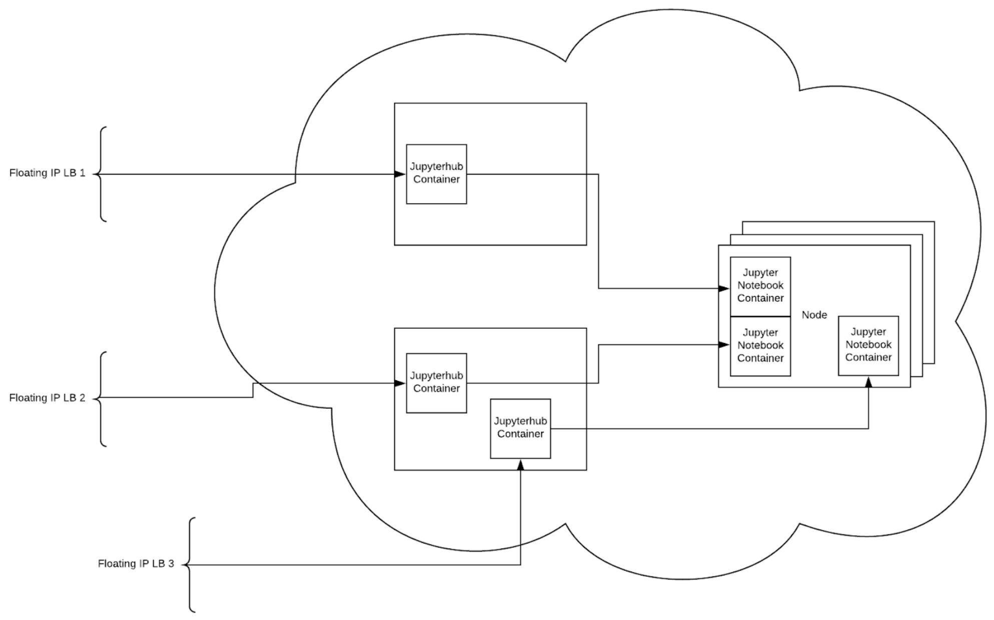


AD
2. AD Authentication
Node
1. User credentials and request passed to container.
Jupyterhub Container
3. User container provisioned.
Jupyterhub Floating IP LB
Node
Jupyter Notebook User Container
4. Container mounts perstitent networ kvolume..
NFS

All traffic is initiated through a web portal that exists as a container on the Kubernetes cluster, exposed publicly by an external load balancer (LB). Users interact exclusively with the load balancer. Credentials are first passed through to the internal container, with authentication deferred to Active Directory; upon successful login, a Jupyter Notebook container is provisioned for the user on a node in the cluster with adequate resources. An NFS volume is then mounted within the container for persistence. All traffic internally between containers within the cluster is dynamically configured via software defined networking. This happens automatically every time an object is added and removed from the cluster.


Physics
User
User
User
User
Chemical Engineering
User
User
User
User
Computer Science
User
User
Floating IP LB
Jupyterhub Container
Jupyter Notebook Container
Jupyter Notebook Container
Jupyter Notebook Container
Node
Jupyter Notebook Container
Jupyter Notebook Container
Jupyter Notebook Container
Jupyter Notebook Container
Jupyter Notebook Container
Node
Jupyter Notebook Container
Jupyter Notebook Container
Jupyter Notebook Container
Jupyter Notebook Container
Node

This process can happen for any number of users and can scale infinitely, given sufficient physical hardware resources.



Additional Jupyterhub instances with differing configurations can co-exist on one cluster, noting that they are logically segregated.

Statefulness, Resiliency, and Scalability

Kubernetes is different from a traditional cluster because it is stateful. For comparison, in a traditional cluster an admin pushes a change out and the change propagates, making other changes until a goal is achieved. In the Kubernetes world, you set the state of the system and the system tries to catch up to the state you designate.To use an analogy, pretend you're trying to kick a soccer ball into a goal post. In a traditional cluster you are the kicker and you have to estimate the force necessary, the wind resistance, etc; it might take you a few attempts to do this successfully and some of your attempts might land in the stands instead of in the goal.In Kubernetes you set the goal post and the ball will find its way into it.

This is important for two main reasons: ***Resiliency*** and ***Scalability***. Kubernetes is known for its self healing qualities. How it achieves this is by referencing its own ideal state. For example if you specify three nginx containers load balanced together across five nodes backed by NFS, Kubernetes will do its best to maintain that state by any means necessary. If a node goes down and it held a container, Kubernetes will spin it back up on a free node and attach that new container to NFS. This happens automatically for every single service you choose to host on Kubernetes. These qualities also make scaling trivial as all that has to be done is to add new nodes to the cluster and usage will be spread automatically for all applications on the infrastructure. The only true persistence involves storage, and as long as the storage remains intact and available, everything else can recover.

Package Management, Customizability and Ease of Use

Like yum and apt for Red Hat and Ubuntu, Kubernetes also has its own package manager, Helm. The aim of Helm is the same: to provide a fast and simple method of implementing a complex application with minimal configuration out of the box, with the option for ultimate customization. With one Helm command an admin can bring up an entire Jupyterhub instance. For customization, the configuration file can simply be copied and modified for uniqueness across additional instances.

Additional Potential

The ease of use for complex applications isn't limited to just Jupyterhub. Looking at the helm stable repository there are a number of other applications that could benefit the same from the power of Kubernetes, installed via Helm. With one command each, one can bring up complex applications like Spark, Hadoop, Prometheus, Elasticsearch and the like. The learning curve is greatly reduced because an administrator doesn't have to learn how it works subcomponent by subcomponent or install and configure each application subcomponent individually -- All that is taken care of for you by Helm and Kubernetes, the administrator only have to understand Kubernetes.

General Security Paradigm

Because all subcomponents of a complex application and its dependencies are containerized, packaged, and abstracted away, a deep understanding of every object in Kubernetes can be difficult. In theory this might it make it hard to secure these applications, however, much like how only a general understanding of Kubernetes is needed to build these complex applications, only a general understanding of Kubernetes is needed to secure them. This takes a paradigm shift however. Rather than understanding the knobs of every application, the paradigm is to treat every application and its subcomponent as a black box, understand how it behaves, and limit the black box via security policies provided by Kubernetes.

Kubernetes has ultimate, extremely granular security control at every layer. From bare metal to networks, to kernels, to cgroups (a Linux kernel feature that accounts for, limits, and isolates the resource usage of a collection of processes), to selinux (security enhanced Linux), to application configuration, Kubernetes controls it all. The stateful philosophy of Kubernetes is also advantageous for this purpose, as an administrator doesn’t need to spend a week learning selinux or another flavor of firewalls or how to get security in a specific application’s configuration. The administrator only has to specify the ideal security state via Kubernetes and the cluster with reach it without intervention.

In addition to existing security technologies, Kubernetes also introduces many new security concepts such as Role Based Access Control, encryption, and certificates and key management for objects within the cluster. Additionally, containers can also be upgraded to Kata container, which s are minimal virtualized containers that provide the speed of containers with the segregation and security of VMs.

Accountability and Monitoring

Metricbeats exists as a container on every node and forwards extremely granular cluster usage to Elasticsearch. For every object down to the specific processes it captures CPU/Disk/Network usage in the nanoscale. Also existing as a container per host is filebeats, forwarding docker logs. Container names and persistent volume naming schema contains usernames, and Elasticsearch can zone in on desired time frames.

Reliability

Two more valuable features are that every update of the desired state is recorded and that mistakes can be rolled back at a single command so there need not be any fear of experimentation. Additionally, all components refer to a single source of truth, the keystore. As long as the keystore is recoverable, no failure is fatal as far as cluster components are is concerned.