



JupyterHub Workshop 2016

Meeting Friday July 22nd, 2016

Quick Tips

- **Interact** for file sharing (URL based content sharing—makes it easier for users)
<http://www.inferentialthinking.com/>
- **Ansible** for deployment
<https://github.com/compmodels/jupyterhub-deploy>
- Jess' compmodels deployment as a base
<https://github.com/compmodels/jupyterhub-deploy>
- Reference Deployments
<https://github.com/jupyterhub/jupyterhub-deploy-teaching>
<https://github.com/jupyterhub/jupyterhub-deploy-docker>

Common Themes/Visions:

Possible Projects

- Find out how many users JupyterHub can support
Define "Modestly Sized" groups:
"JupyterHub is designed to be a simple multi-user server for modestly sized groups of semi-trusted users."
- Supporting many (500+) users on single JupyterHub server
- Flatten learning curve
currently too steep because of git
Interact (mentioned in "Quick Tips") as possible solution
- Saving history when you exit the shell
- Integrating .ipynb files with Google Drive's "open with"
- Make it easier to create hubs

Overall Notes

Brian Granger – Cal Poly SLO

“Deploying JupyterHub”

Small/medium groups of trusted users

Users are not developers

Ansible Based Deployment

- We have encoded this deployment scenario as a set of configurable ansible roles
<https://github.com/jupyterhub/jupyterhub-deployteaching>
- Used to teach data science at Cal Poly over multiple quarter, multiple instructors
- Can build a new server in <1 hour

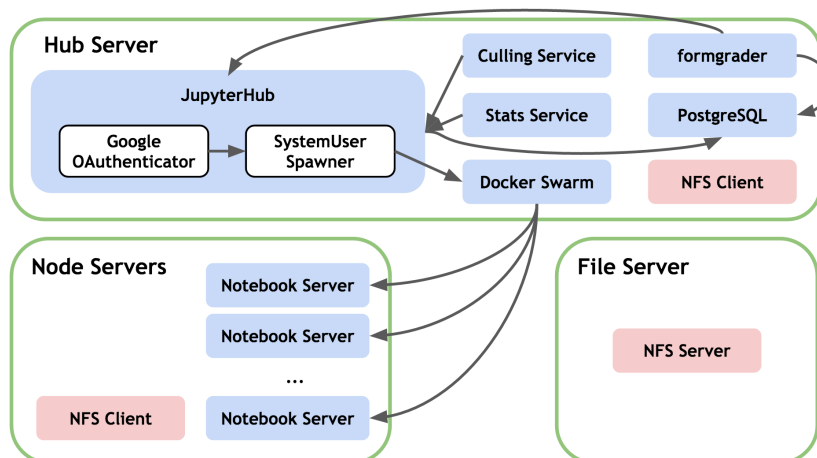
Install uses conda and some pip

Jess Hamrick – UC Berkeley

“Scaling Up JupyterHub for Education”

Scaling up to include more users (200+)

1. Isolating users with Docker
 - Isolated environment
2. Isolating 200+ users with Docker Swarm
3. Persistent files using NFS



4. Managing everything with Ansible and Docker Compose

[DockerSpawner and SystemUserSpawner:

<https://github.com/jupyterhub/dockerspawner>]

Talking to JupyterHub:

1. Get an API token
2. Access the JupyterHub API using `http://<url>:8081/hub/api/users`
 - Headers: {'authorization': 'token <hub_api_token>'}
 - <https://github.com/jupyterhub/jupyterhub/tree/master/jupyterhub/apihandlers>

Getting a JupyterHub URL:

1. Run your service at:
 - `http://<service_url>:<service_port>`
2. Access the proxy API
 - POST `http://<url>:8001/api/routes/myservice`
 - Headers: {'authorization': 'token <proxy_api_token>'}
 - Body: {'target': 'http://<service_url>:<service_port>'}
3. Access your service at:
 - `http://<url>:8000/myservice`

Authenticating with JupyterHub:

1. Get an API token:
 - JupyterHub token
2. Use the [HubAuth service](#)
[HubAuth service](#)
3. Example use case: nbgrader formgrade

Ryan Lovett – UC Berkeley

“Foundations of Data Science” Course

Computer Science & Statistics Dept.

- New Data Science course with “connector” courses

- Connector classes used skills from main class for specific disciplines
- Didn't want students to have to install too much software
- About 100 students (1st semester) grew to 480 students (2nd semester)
- There is a learning curve
 - Not all instructors or students familiar with git
- Needed a way to push notebooks and files to students
 - Interact** URL based content sharing
 - <http://www.inferentialthinking.com/>
 - hub side service that pulls notebooks from github repos when students click on a textbook link
- Azure spawning issues fixed by Min
- imagespawner: HTML form that allows the user to choose which container they want to run. Give instructors a choice between regular container and one where the user has elevated privileges
 - <https://github.com/ryanlovet/imagespawner>

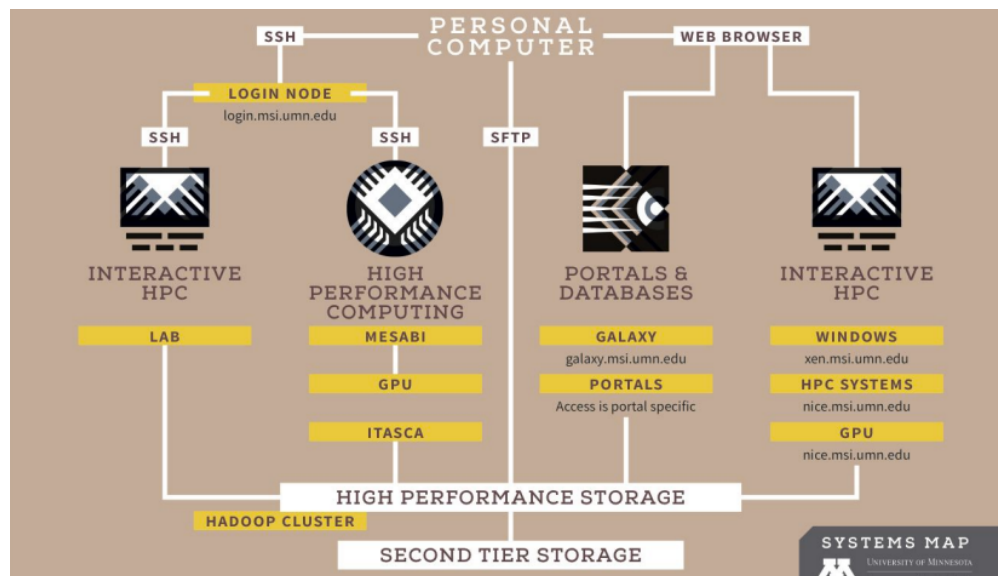
Shreyas Cholia– UC Berkeley

“Driving Supercomputers With Jupyterhub: NERSC Experiences On Cori”

<https://github.com/NERSC>

- Individual users to run their own notebook servers presents a potential security hole
- Custom Authenticator uses myproxy to login to NERSC CA server with username/password and retrieves credentials (X509 certificate)
- Jupyterhub runs as a standalone service and doesn't need root access
- We wrote an SSH Spawner that will will SSH into the Cori node with users credential
 - Uses GSISSH (but can also use regular SSH if you want to use SSH Keys instead of GSI Cer2ficates)
- SSH Spawner starts up notebook server process and goes away; Notebook server communicates directly with hub. No tunnels or persistent connections needed
- Keep track of the PID for poll and shutdown functions (also via SSH)

Michael Milligan – University of Minnesota
 “Scaling Up JupyterHub for Education”



Use Cases:

- Supercomputer Gateway for Interactive HPC
- Enabling technology for Data Science Gateways
 - Apache web server
 - [reverse proxy, SSL termination, Shibboleth auth]
 - ⇓
 - configurable-http-proxy
 - ⇓
 - JupyterHub server
 - [batchspawner, profiles, remoteuser]
 - ⇓
 - Job scheduling engine (Torque)
- 2016 SciPy talk: <https://github.com/mbmilligan/scipy2016-jupyterhub>

I. Allison, J. Colliander, M. Lamoureux – Pacific Institute for the Mathematical Sciences
 “JupyterHub using Docker and Flocker on Openstack via Ansible and Vagrant”

- We want to offer similar services to orders of magnitudes more researchers

- Federated identity - Shibboleth for authentication
- Also more research focused machines (HPC) for various groups.
At the moment we're experimenting with a larger single installation of JupyterHub
How far can we push a single hub and where are the bottlenecks?
We're trying out Swarm and Flocker for this
- Also looking at automated deployments for researchers -
Hubs on demand

They want to do this in a completely automated fashion

- **Vagrant** - "Reproducible development environments"
 - o Allows us to define and instantiate resources
 - o Providers: OpenStack, Rackspace, AWS, ...
 - o Provisioners: sh, ansible
- **Ansible** - "Simple IT Automation"
 - o Can take generic image and build up entire infrastructure for Docker
 - o Swarm in minutes
 - o Consul - distributed K/V store
- **Docker swarm** - API similarities make transition simple
 - o swarm nodes can be provisioned as needed (and automated)
- **Flocker** - "container data volume manage"
 - o maps common backends (EBS, GCE, Cinder) to containers

Alec Aivazis & Mike Polino – DataScience

"Lessons in transitioning a data science team to JupyterHub"

Difficulties

- VM drift, difficult debugging, reduced control for engineering team
- Discomfort in git (especially for diff'ing json docs)
- Limited by RAM in their laptops
- Sharing notebooks across Docker containers without mounting volumes or using git
 - o Restful content manager:
<https://github.com/datascienceinc/RestfulContentManager>
- Users want control over their notebook server resources (RAM / cores)
 - o Mesos and DockerSpawner, re-size accessible from the UI
- Analysts want to collaborate quickly, don't want to "check out" code

- locking mechanism and websockets
- Power users want to install notebook-specific Python packages
 - Python virtualenv per notebook

Jonathon Anderson – CU Boulder

“Jupyterhub at CU Boulder”

vSphere

PAM spawner for upstream

IP's limited

Should say how much memory is required

Status field is better with feeding que status

Spawn terminals into remote compute environment

Cory Gwin – Berkeley Applied Analytics

“Custom Jupyterhub OAuthenticator”

<https://github.com/jupyterhub/oauthenticator>

- Simplify sign on for user so they don't have to sign in twice
- Organize user's notebooks based on username folder structure
- Edit existing OAuthenticator to fit your needs
 - Subclasses must override:
 - login_service (string identifying the service provider), used for the button name
 - login_handler (likely a subclass of OAuthLoginHandler)
 - authenticate (method takes one arg - the request handler handling the oauth callback)
- Next Steps:
 - Store API credentials for the User so they can access them.
 - Setup an API using these stored credentials inside the Notebook.
 - Investigating sharing folders based on user groups.