Containers For Scientific Computing
Greater IPAC Technology Symposium IX

Containers For Scientific Computing
Greater IPAC Technology Symposium IX

Karan Vahi
vahi@isi.edu

USC Viterbi
School of Engineering
Information Sciences Institute

Pegasus

Outline

Motivation	*Reproducibility* *Virtual Machines*
Containers Overview	*Technologies* *Orchestration* *Repositories*
Containers Deployments	*Infrastructure* *Middleware*
Conclusion	*Questions*

Reproducibility in Scientific Analysis

- Why?
  - As a scientist you want to be able write code that can be independently verified
  - Reliability
    - Analysis should be able to run on newer hardware
    - As a user very hard to keep up with changing computing landscape
  - Can scale up dependent on target execution environments
    - Parameter Sweep
    - Pipeline code together as scientific workflows

Reproducibility in Scientific Analysis

- Why?
  - Ease of Use and Portability
    - Don’t limit the execution environments
    - Ideally, users can reliably recreate your analysis on varied execution environments
      - Local Desktop (Windows, Linux, MACOS)
      - Local HPC Cluster (Mainly Linux oriented)
      - Computing Grids (Collection of University HPC clusters, such as OSG)
      - Leadership Class HPC Systems (Linux variants like Cray)
      - Cloud Environments (Choice of OS and architectures available)

Challenges to Reproducibility?

Custom Execution Environments

- When you start using shared resources you loose control over the hardware and OS
- Hard to ensure homogeneity: Users will run your code on same platform/OS it was developed on.
- Some dependent libraries required for your code may conflict with system installed versions
  - TensorFlow requires specific python libraries and versions.
  - Some libraries maybe easy to install on latest Ubuntu, but not on EL7
- If running on shared computing resources such as computational grids
  - you run on a site with heterogeneous nodes and your job lands on a node where OS is incompatible with your executable

Solutions

- Enforce similar computing environment
  - Limit users to where they can run your code
  - Not practical if you want your analysis to scale up

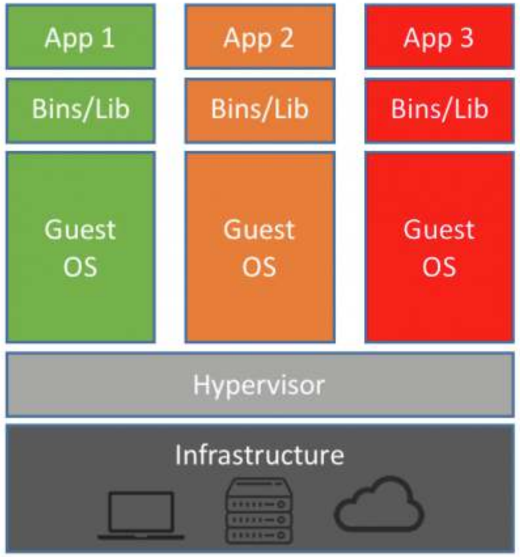
- Engage with computing facilities
  - Install libraries in your shared space and make sure environment refers to those libraries
  - Need cooperation from administrators

- Lmod: Identify and setup modules required for your application.
  - Can vary from one cluster to another

Solutions: Virtual Machines

- Originally developed to efficiently exploit increase in server processing capabilities
  - Applications could not natively make use of increased memory and processing power
  - Allow multiple applications running on different OS to run on the same hardware

- Virtual Machines enabled better use by running multiple self contained applications on the same machine
  - Uses software to emulate a particular hardware on physical servers
  - Each Virtual Machine runs it's own Guest Operating System
    - Bundles it own binaries and libraries
  - Typical size of a VM is on order of GB's



Machine Virtualization

- Achieves portability at cost of speed
  - VM's take a long time to boot up and start
  - Each VM runs a fully copy of the OS + virtual copy of hardware to run OS

- Not particularly appealing running large pipelines and deploying VM's on demand to execute individual steps

Image Source: https://blog.netapp.com/wp-content/uploads/2016/03/Screen-Shot-2018-03-20-at-9.24.09-AM-935x500.png

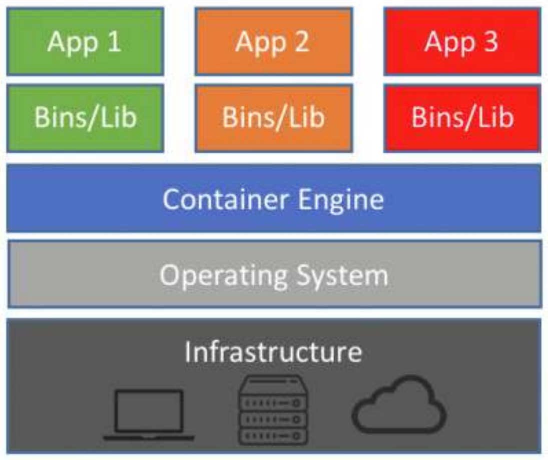
Solutions: Containers

- Virtualizes the OS instead of the Hardware
  - Sits on top of the physical server and the host OS
  - Each container shares the Host kernel and binaries and libraries

- Separates the application from the node OS.

- Lightweight
  - Instead of GB's size is on order of MB's
  - Take seconds to start instead of minutes
  - Can pack more applications on the same node compared to VM's

- Perfect for deploying on demand.



Containers

Image Source: https://blog.netapp.com/wp-content/uploads/2016/03/Screen-Shot-2018-03-20-at-9.24.09-AM-935x500.png

Solutions: Containers

- Reproducible
  - Usually described as a recipe file that captures the steps to configure and setup the container

```
FROM centos:7

LABEL maintainer "Mats Rynge <rynge@isi.edu>"

RUN yum -y upgrade
RUN yum -y install epel-release yum-plugin-priorities

# osg repo
RUN yum -y install http://repo.opensciencegrid.org/osg/3.4/osg-3.4-el7-release-latest.rpm
..

# install relevant packages
RUN yum -y install \
    astropy-tools \
    ...
    python-astropy \
    python-devel \
    python-future \
    python-pip \
    wget
```

```
# Build File Continued
# Cleaning caches to reduce size of image
RUN yum clean all


# wget -nv http://montage.ipac.caltech.edu/download/Montage_v5.0.tar.gz
RUN cd /opt && \
    wget -nv https://github.com/Caltech-IPAC/Montage/archive/master.zip && \
    unzip master.zip && \
    rm -f master.zip && \
    mv Montage-master Montage && \
    cd Montage && \
    make


RUN mkdir /opt/montage-workflow-v2


ADD * /opt/montage-workflow-v2/
```

Complete Build File: https://github.com/pegasus-isi/montage-workflow-v2/blob/master/Dockerfile

However: Containers are no magic bullet

- **Unlike VM's they do not provide complete security isolation**
  - By default, usually you run as root in the container (Security Risk)
  - Possibility of privilege escalation on the Host OS

- **Need to take precautions**
  - Drop privileges , and run as non root wherever possible
  - Be careful of what you install in the container.
  - Install from trusted repositories even within the container
  - Mount host file systems as read only where possible
  - Write into the container filesystem

Outline

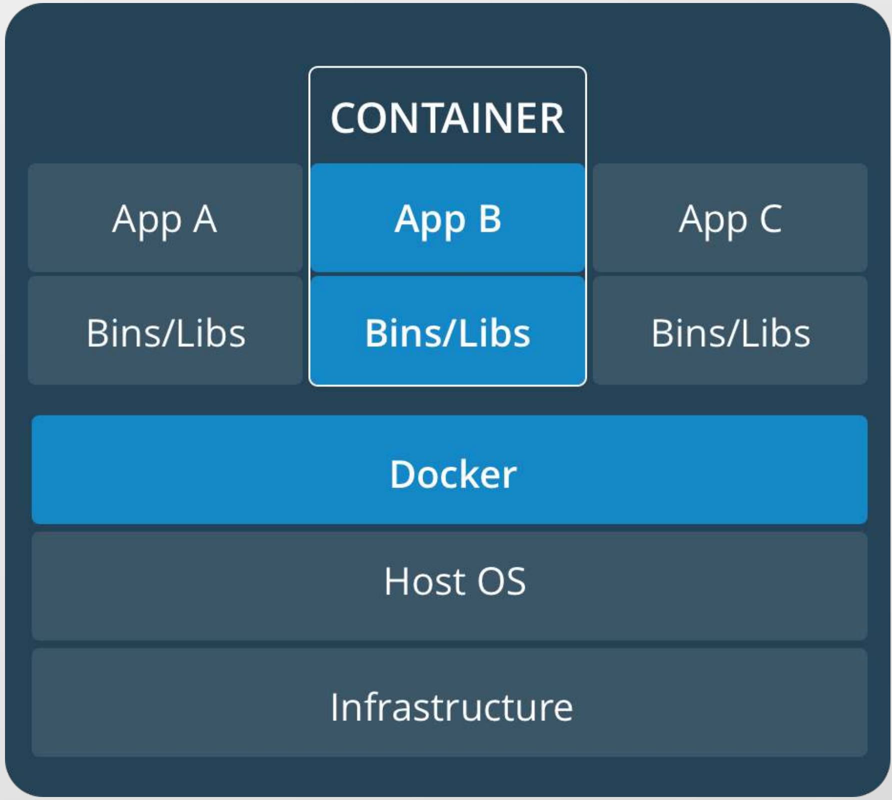
Motivation	*Reproducibility* *Virtual Machines*
Containers Overview	*Technologies* *Orchestration* *Repositories*
Containers Deployments	*Infrastructure* *Middleware*
Conclusion	*Questions*

Container Technologies

- Popular Container Technologies
  - Docker
    - Popular in the enterprise world.
    - By default, application launched in container run as root
      - A concern when running on shared infrastructure
  - Singularity
    - Popular in HPC environments.
    - Is run in user space.
  - Shifter
    - Developed by NERSC and optimized for running on HPC machines
    - Allows users to securely run Docker images at scale
    - Images cannot be exported

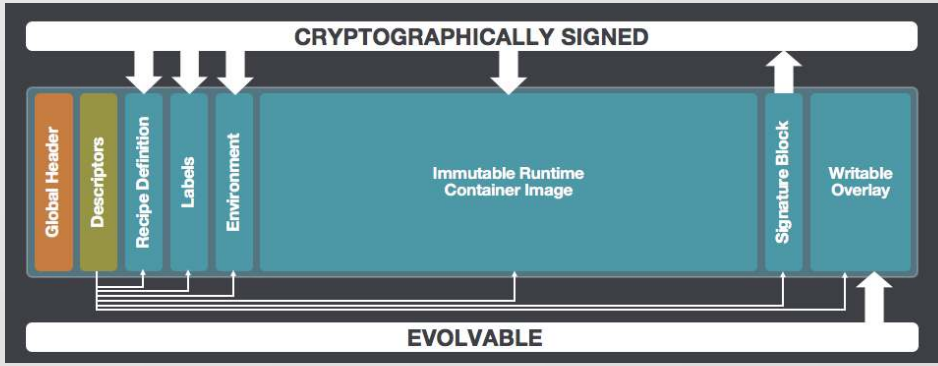
Container Technologies - Docker

- Available for both Linux and Windows based applications
- The containers run on a Docker agent called Docker Engine running on a node
  - Has to be run as root
  - Images can be loaded and unloaded on each node
- Powerful command line tool (*docker*) to build, deploy and run images
- Images are stored as a series of layers
- Images can be built from a recipe file called Dockerfile.
- Docker Hub enables sharing and discovery of Images
  - Also provides private Image Registries



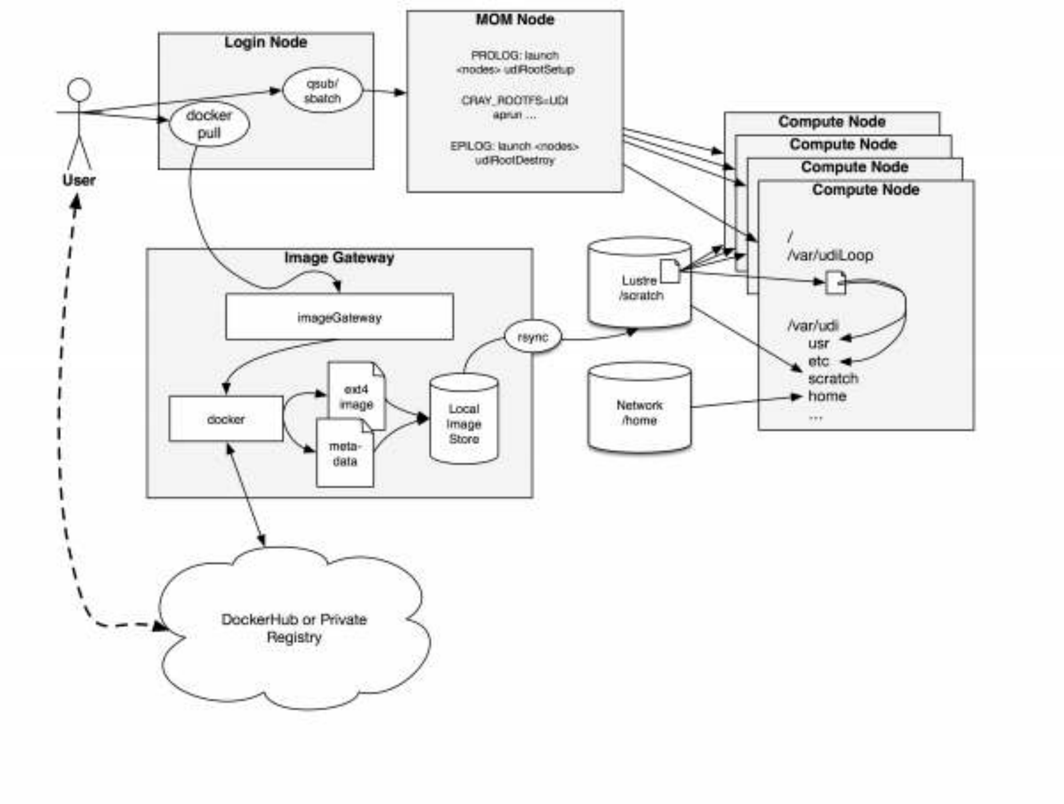
Container Technologies - Singularity

- Developed for HPC and geared towards research computing
- Available mainly for Linux
  - Limited support for Windows and MACOS using Vagrant
- Deployed on most of the large supercomputers
- Runs in user space
  - No root owned daemon processes
- Same user is the same inside and outside of the container
  - Prevents a user from escalating privileges.
- Container is stored as a single runtime file (.sif)
  - Makes it easier to ship around and deploy
- Can pull images from Docker Hub
- Like Docker, supports a Singularity Library for image sharing



Container Technologies - Shifter

- Shifter is NERSC's approach to implement containers in HPC
- It provides a simple (but restricted) command line interface
- It allows users to pull Docker images to a local image store from DockerHub and private registries
- Has good integration with NERSC's batch scheduling system (SLURM)
- Container images are made available to the compute nodes transparently at job submission
- Supports volume mapping within the container
- Supports internode communication (MPI jobs)



Container Orchestration

- Fits well with Microservices architecture
  - structures an application as a collection of loosely coupled services
  - services are fine-grained and the protocols are lightweight
  - Improves modularity and parallelizes development

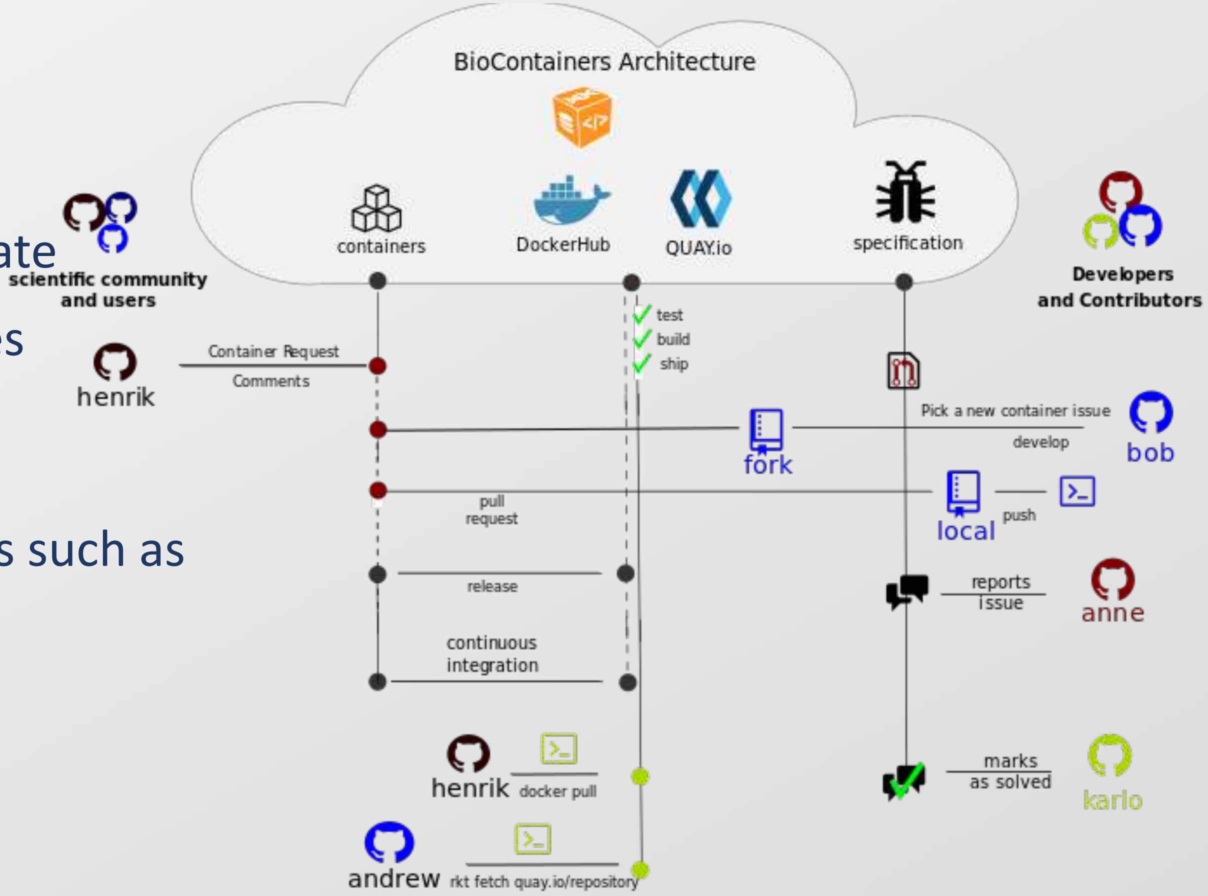
- However, as you scale up in terms of applications and number of containers, container orchestration becomes important
  - Provisioning and deployment of containers
  - Redundancy and availability of containers
  - Scaling up or removing containers to spread application load evenly across host infrastructure

Container Orchestration – How?

- Use one of the available orchestration tools
  - Kubernetes
  - Docker Swarm
- Describe configuration of application in YAML or JSON file
  - Tell source of containers e.g. Docker Hub
  - Establishing networking between containers
  - how to mount storage volumes, and
  - where to store logs for that containers
- However, as you scale up in terms of applications and number of containers, container orchestration becomes important
  - Provisioning and deployment of containers
  - Redundancy and availability of containers
  - Scaling up or removing containers to spread application load evenly across host infrastructure

Container Repositories- BioContainers

- Community Driven Project
  - https://biocontainers.pro/#/
- Goals
  - Provides infrastructure and basic guidelines to create and manage and distribute bioinformatics packages
- Focus
  - Ready to use containers in bio-informatics domains such as proteomics, genomics etc.
- Supports automated builds
- Provides Registry
  - BioContainers Registry is a hosted registry of all BioContainers images that are ready to be used
  - You can create your own free repositories on DockerHub or Quay.io



Outline

Motivation — *Reproducibility*, *Virtual Machines*

Containers Overview — *Technologies*, *Orchestration*, *Repositories*

Containers Deployments — *Infrastructure*, *Middleware*

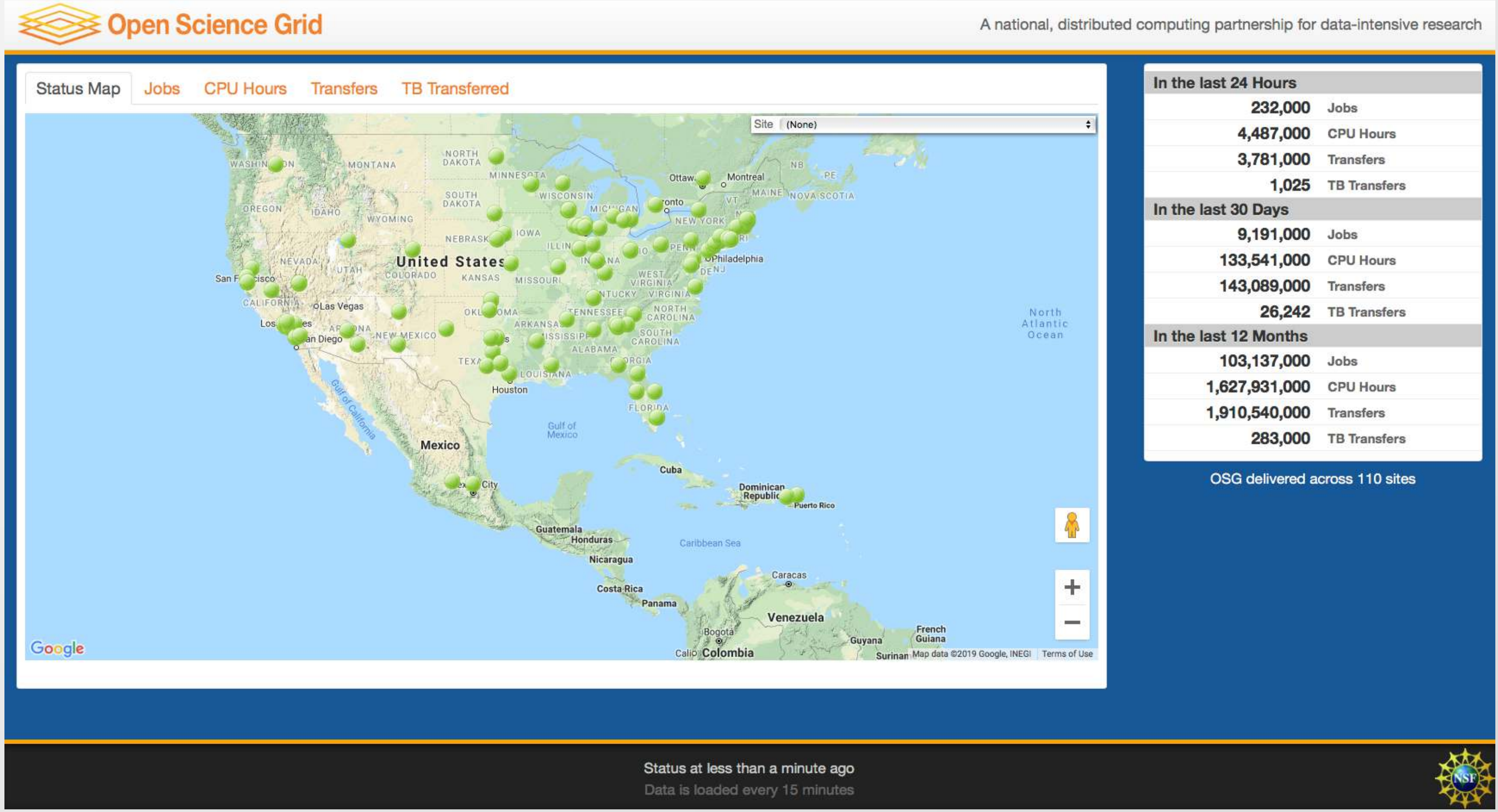
Conclusion — *Questions*

Containers in Computing Infrastructure – Open Science Grid

- A framework for large scale distributed resource sharing addressing the technology, policy, and social requirements of sharing computing resources.

OSG is a consortium of software, service and resource providers and researchers, from universities, national laboratories and computing centers across the U.S., who together build and operate the OSG project. The project is funded by the NSF and DOE, and provides staff for managing various aspects of the OSG.

Integrates computing and storage resources from over 100 sites in the U.S.



From: https://display.opensciencegrid.org

Slides Courtesy: Mats Rynge (OSG)

Open Science Grid – Container Motivations

- **Consistent environment (default images)** - If a user does not specify a specific image, a default one is used by the job. The image contains a decent base line of software, and because the same image is used across all the sites, the user sees a more consistent environment than if the job landed in the environments provided by the individual sites.

- **Custom software environment (user defined images)** - Users can create and use their custom images, which is useful when having very specific software requirements or software stacks which can be tricky to bring with a job. For example: Python or R modules with dependencies, TensorFlow

- **Enables special environment such as GPUs** - Special software environments to go hand in hand with the special hardware.

- **Process isolation** - Sandboxes the job environment so that a job can not peek at other jobs.

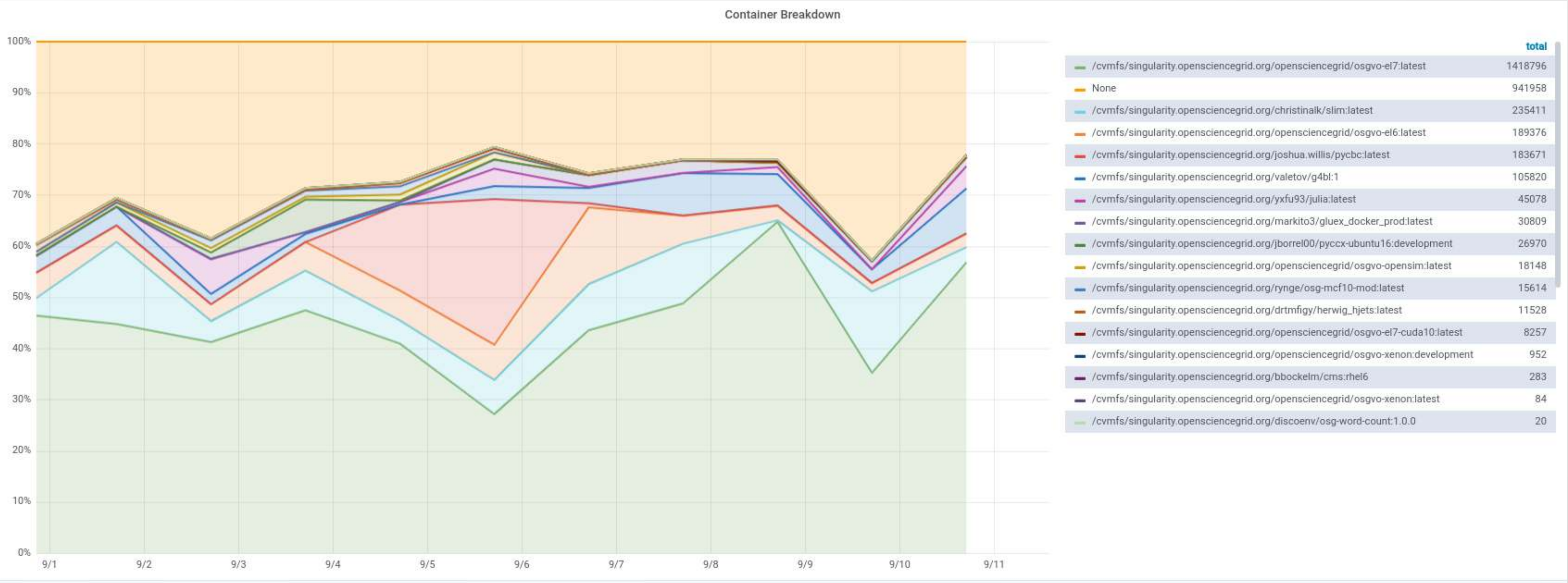
- **File isolation** - Sandboxes the job file system, so that a job can not peek at other jobs' data.

Slides Courtesy: Mats Rynge (OSG)

Open Science Grid – Container Instances Per Day

Slides Courtesy: Mats Rynge (OSG)

Open Science Grid – Job Breakdown



Slides Courtesy: Mats Rynge (OSG)

Open Science Grid – Extracted Images

- OSG stores container images on CVMFS in extracted form. That is, we take the Docker image layers or the Singularity img/simg files and export them onto CVMFS. For example, ls on one of the containers looks similar to ls / on any Linux machine:

```
$ ls /cvmfs/singularity.opensciencegrid.org/opensciencegrid/osgvo-el7:latest/
cvmfs     host-libs   proc     sys     anaconda-post.log        lib64
dev        media               root    tmp       bin            sbin
etc       mnt         run      usr     image-build-info.txt     singularity
home      opt         srv      var     lib
```

- Result: Most container instances **only use a small part** of the container image **(50-150 MB)** and that part is heavily cached in CVMFS!

Slides Courtesy: Mats Rynge (OSG)

Pegasus Workflow Management System

Automates complex, multi-stage processing pipelines

Enables parallel, **distributed computations**

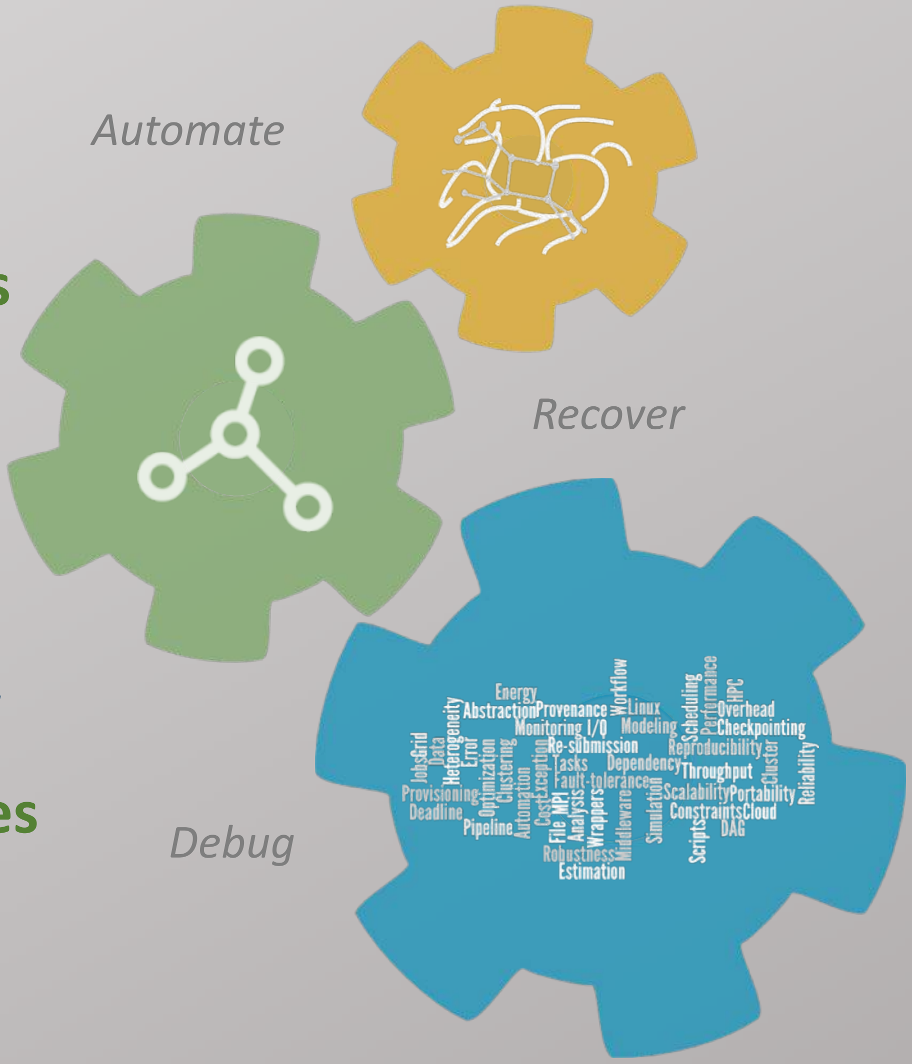
Automatically executes data transfers

Reusable, aids **reproducibility**

Records how data was produced (**provenance**)

Handles **failures** with to provide reliability

Keeps track of data and **files**

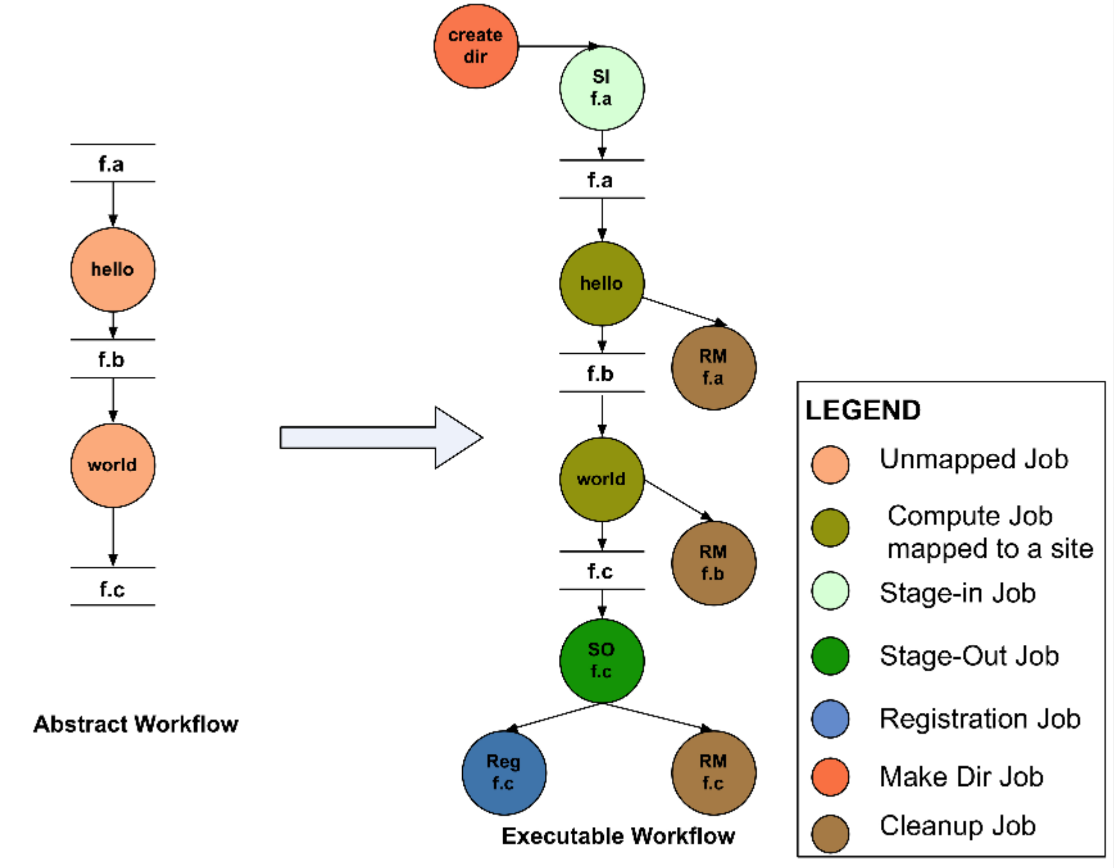


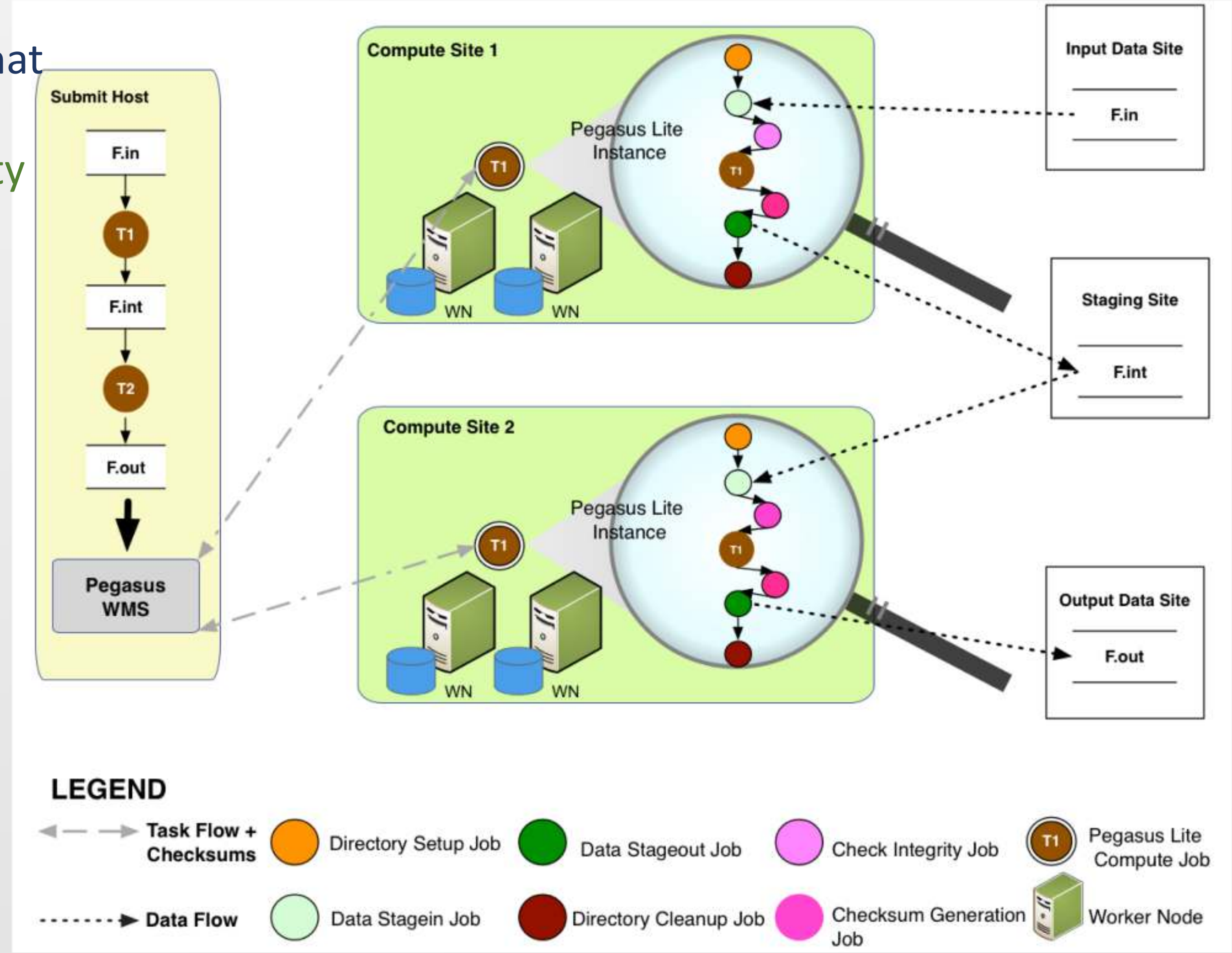
HTCondor
High Throughput Computing

NSF funded project since 2001, with close collaboration with HTCondor team

Pegasus

- Users describe their pipelines in a portable format called Abstract Workflow, without worrying about low level execution details.
- Pegasus takes this and generates an executable workflow that
  - has data management tasks added
  - transforms the workflow for performance and reliability





Pegasus: Container Support

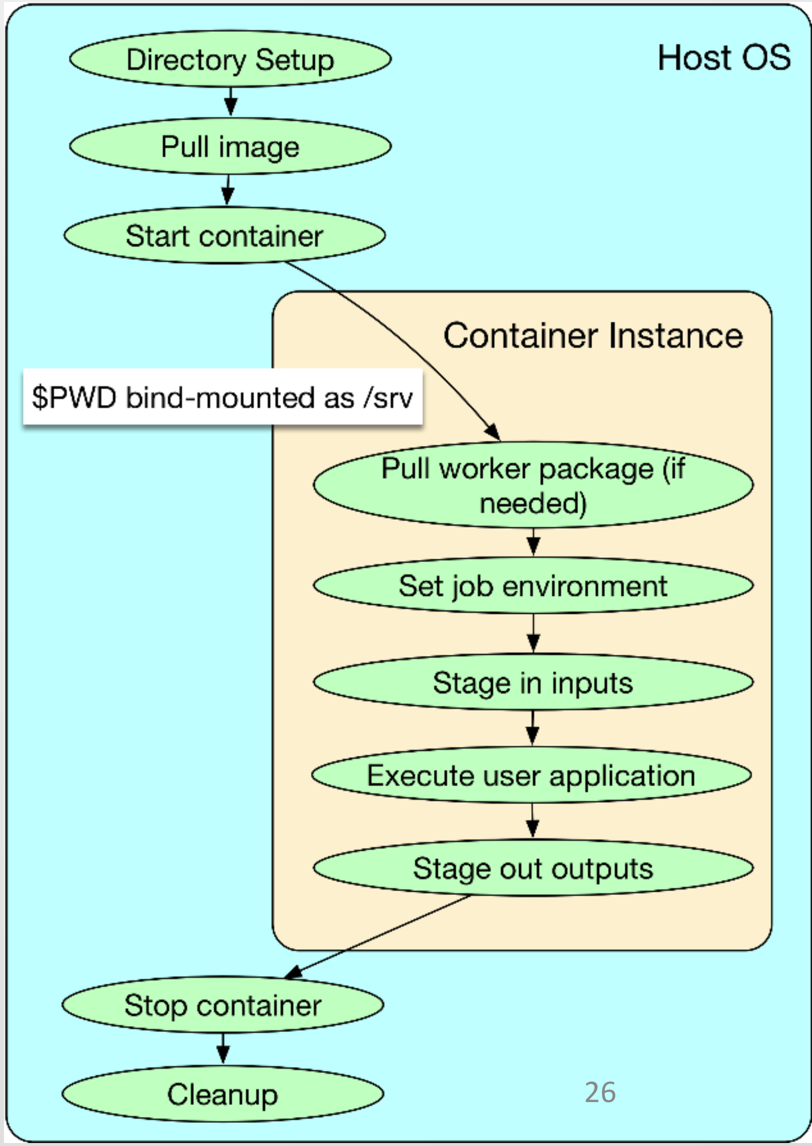
Data Management

- Users can refer to container images as
  - Docker Hub or Singularity Library URL's
  - Docker Image exported as a TAR file and available at a server , just like any other input dataset.

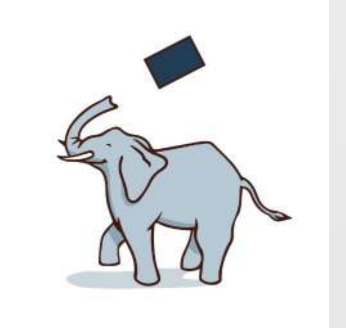
- If an image is specified to be residing in a hub
  - The image is pulled down as a tar file as part of data stage-in jobs in the workflow
  - The exported tar file is then shipped with the workflow and made available to the jobs
  - Motivation: Avoid hitting Docker Hub/Singularity Libraryrepeatedly for large workflows

- Pegasus worker package is not required to be pre-installed in the container
  - If a matching worker package is not installed, the required worker package is installed at runtime when container starts

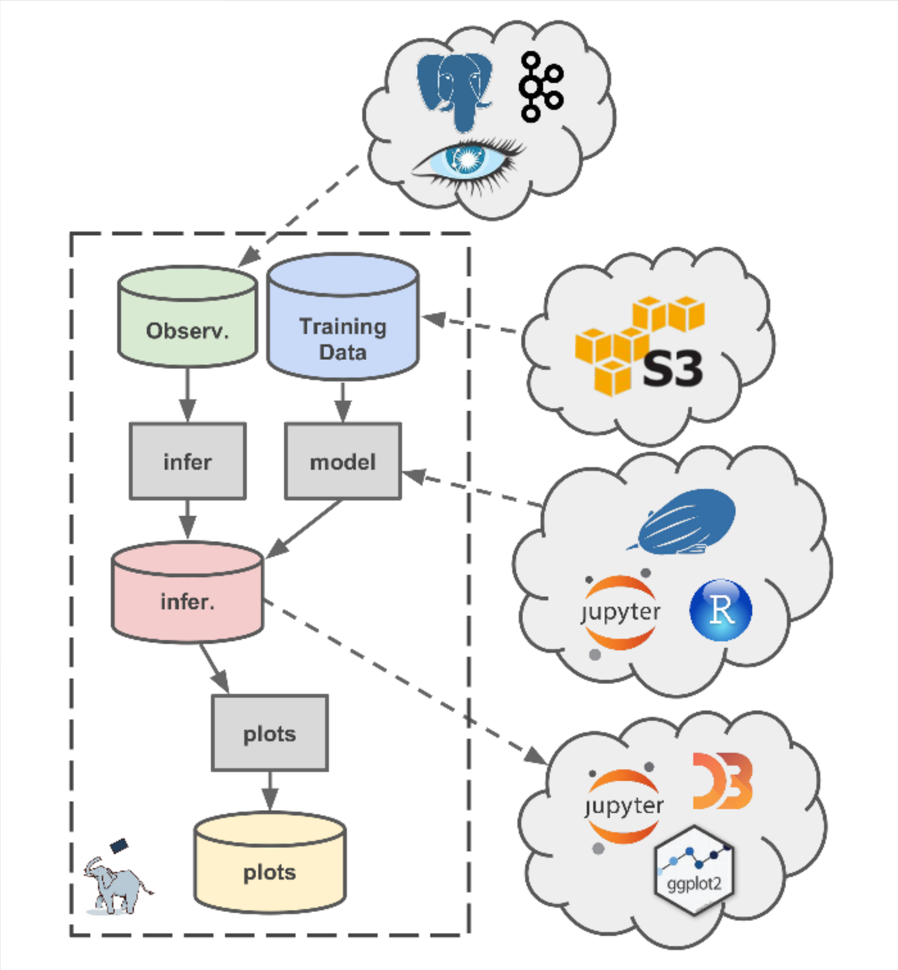
Container Execution Model



Pachyderm:



- Pachyderm allows users to build reproducible pipelines
- Fully-containerized solution and can only run in containers
- Data Oriented instead of being Task Oriented
  - Everything is versioned (files, pipelines) following the git model (commit, branch, repository etc)
- Everything runs into containers (Docker)
- Pachyderm relies on the containers orchestration solution: Kubernetes
- Pachyderm requires a cloud solution + an object-based storage to run (AWS, Azure, Google Cloud)



Outline

Motivation	*Reproducibility* *Virtual Machines*
Containers Overview	*Technologies* *Orchestration* *Repositories*
Containers Deployments	*Infrastructure* *Middleware*
Conclusion	*Questions*

Containers: Closing Thoughts

- Containers are attractive, lightweight proposition for service isolation.
- Help in addressing reproducibility concerns that have arisen in the past few years , especially in scientific computing
- Lowers barriers for sharing.
- Wide adoption in commercial enterprise and scientific computing.
- Container based middleware for data processing is being developed grounds up
    - Pachyderm is a fully-containerized solution and can only run in containers
- Traditional task oriented data processing systems have also implemented support
    - Pegasus
    - MakeFlow
    - Nextflow
    - Airflow

Questions?