The government seeks individual input; attendees/participants may provide individual advice only.

Middleware and Grid Interagency Coordination (MAGIC) Meeting Minutes
February 7, 2018, 12-2 pm
NCO, 490 L'Enfant Plaza, Ste. 811
Washington, D.C. 20024

Participants (*In-Person Participants)

Gonzalo Alvarez	LBL	Miron Livny	UW-Madison
Ben Brown	DOE/SC	Brian Lyles	ORNL
Rich Carlson	DOE/SC	David Martin	ANL
Shane Canon	LBL	Sergey Panitkin	BNL
Luca Cinquini	NASA	Gilberto Pastorello	LBL
Cam Clemence	McAllister & Quinn	Rajiv Ramnath	NSF
Kaushik De	UTA	Lavanya Ramakrishnan	LBL
Dave Dystra	FNAL	Matyas Selmici	UW-Madison
Devarshi Ghoshal	LBL	Alan Sill	TTU
Dave Godlove	Sylabs	Derek Simmel	PSC
Dan Gunter	LBNL	Cory Snavely	LBL
Shantenu Jha	Rutgers	Adam Soroka	Smithsonian
Padma Krishnaswamy	FCC	Douglas Thain	UND
Greg Kurtzer	Sylabs	Todd Tannenbaum	UW-Madison
Joyce Lee*	NCO	Rick Wagner	Globus

Action Items: Follow up with speaker suggestions and continue fleshing out tasking.

Proceedings
This meeting was chaired by Rich Carlson (DOE/SC) and Rajiv Ramnath (NSF).

Speaker Series: Containerization & Virtualization Technologies: current containerization and virtualization technologies

- Shane Canon (Shifter) - Project Engineer, Data & Analytics Services Group, National Energy Research Scientific Computing Center (NERSC)
- Cory Snavely (SPIN) - Lead, Infrastructure Services Group, NERSC
- Douglas Thain (Docker) - Associate Professor, Computer Science and Engineering, University of Notre Dame
- Rick Wagner (Kubernetes) - Professional Services Manager, Globus
- Dave Godlove (Singularity) - Software Engineer, Singularity
- Greg Kurtzer (Sylabs) - CEO, Sylabs, Inc.

Shifter, Containers for HPC - Shane Canon
Background

- NERSC: Mission HPC computing facility that has historically supported only modeling and simulation. Currently, it is spending more time supporting data intensive use cases. (Slides 2-4)

    - Converging systems geared for HPC and data intensive computing (e.g., Cori)
        - Benefits: Lowers bar to entry (e.g., lower barrier to porting process)
        - Facilitates running communities' applications on HPC systems
        - Problems: HPC systems are not necessarily amenable to some workloads
- Docker: Outside of HPC community, it enables building, shipping and running applications in a portable way; images can be pulled down and executed on different platforms. (Slides 5-9)
    - Helps address challenges and provides flexibility: Facilitates starting point for users and give them more control over environments in which they are executing
    - Problem: All or nothing security model- once a user is able to run a docker container, it is given full privileges on system. Problematic in shared environments

Shifter

- Developed to leverage Docker image ecosystem (e.g., Docker Hub). (Slide 9,19)
- Allows users to securely run containers in an HPC environment with native access to shared file system and highspeed network on the system. (https://github.com/nersc/shifter).

Components (Slide 10)

- Shifter Image Gateway: imports and converts images from DockerHub and Private Registries to a format more friendly to the HPC system
- Shifter Runtime: Must be available on compute nodes (translates user's request for desired image, creates the environment and starts user's application)
- Work Load Manager (WLM) Integration (optional): integrates Shifter with WLM; valuable for scaling.

Main motivation (Slide 11)

- Productivity
- Facilitates bringing codes in to centers like NERSC
- Performance benefits: generally for applications with many dynamic libraries that want to load (e.g. Python); can accelerate performance of some applications due to packaging

Somewhat unique features (Slide 12)

- Centralized Image Management model: system site (not user) controls images
- Native support for MPI (message passing interface) and GPU (graphics processing unit), next release (Slide 14)
    - Worked to integrate and can easily run MPI apps and have somewhat portable containers. Can get native performance with MPI application and sometimes better
- Optional integration with resources (important for startup times and scaling)
- Per-node write cache: useful for sites (e.g., NERSC) that want to emulate having a local disk
    - Looks and behaves like a local disk on a compute node, but private to that node. IOPS (input/output operations per second) scales to number of nodes (Slide 13)
- Focused on scaling

Use Case: Shifter pivotal for achieving a science mission. Re-packaged Shifter as Docker image and ran at full scale at Cori across 600,000 cores (Slide 15)

Future Releases and Future Work (Slide 17)

- Beta version supporting private images with improvements (GPU and MPI support, etc.)

- Continue advocating for container adoption for science
- Tracking evolving Docker community; may not need Shifter in the future.
- Shifter may work with Docker community to address gaps

NERSC Spin Project Overview - Cory Snavely

Spin - part of a 2-prong container strategy at NERSC: 1) Shifter – Container strategy for computational work and 2) Spin provides system to build and deploy science gateways/"edge services" quickly and easily using Docker Containers. (Slides 2-6)

- Key: everything is built out of containers (software defined-spun up quickly, easier to manage. Potential to deploy entire stacks in batch job prologue via API calls
- User access: HPC networks and other resources (file systems) at NERSC
- Scaling: simpler in a container environment due to quick, simple instantiation
- Conventional approach: more "pain" points and delays for users. Can automate around it but can't speed up like a container-based deployment. (Slide 5)
- Docker: Example of simplicity of declaring and instantiating a service; self-contained

"Orchestrating" with Rancher – Deployment of different services across a pool of servers. (Slide 7)

- Rancher: manages many user-defined services over the whole system. Orchestration decides location of individual containers
- Rancher environment was helpful to NERSC at the time in ease of use and robustness

Technology Gaps (Slide 8)

- Multi-tenancy is immature: current technical tools are not geared towards multi-tenancy
- Access Control: Hope community and orchestration tool providers will adopt capabilities (e.g., implement **access controls** in real time way at API level to enforce security policies)

Benefits (Slide 9-10)

- *Users*: 1) benefits of VMs without the overheads, and 2) cloud with tight integration to NERSC.
  - Can focus on developing an application instead of managing their infrastructure.
  - Free scalability and fault tolerance (re-deployment capability), flexibility
- *NERSC*:
  - Empowers user
  - Resources are assigned more flexibility; NERSC can maintain without being as impactful
  - Enhanced security

Strategic Overview of Docker – Douglas Thain

Docker popularized the idea of containers as a means for distributing software, performing resource management and providing applications for private, customized environments. Dockerfile produces an image to create a container, which can be moved around and provisioned at new sites.

Docker Implementation

- On every node for Docker use, run a root-owned server (dockerd). Dockerd responds to user commands run from Docker that operate on containers or images
- When making a change to containers, can commit the change into a new image at a key point
- Access control: As Docker commands can be run by any ordinary user, set up access controls on dockerd for the users who are allowed to perform operations

Image Sub-structure-each image has internal structure

- Dockerfile: Every change expressed in dockerfile is recorded as a new layer in that image not as a change to the existing layer
- To run a container, Docker creates new layer on top that is writeable, and accepts any changes made by the Container. Need to set up Docker to exploit the right capability in the underlying kernel, which enables the Container to see all layers in image simultaneously

De-Duplication - A single node can have a large degree of de-duplication (Slide 4)

- Allowed across different kinds of containers
- When constructing different kinds of containers sharing the same base operating system, no need to copy the same base operating system multiple times to different places

Docker Orchestration

In most situations using Docker, the goal is to deploy a fleet of containers that work together
Common case: deploy web application that consists of database, webserver and memcached

- Simplest orchestration method in Docker: *Docker Compose* writes a composed description describing each container. External supervision is required for stateless services
- More robust performance orchestration: *Docker Swarm*- Used widely in industry
  - Bring stateless services from Docker Compose into the cluster
  - Identify several nodes as manager nodes for tracking the state of your service
  - Ask swarms to create a service (e.g., web servers, memcast, etc.) to deploy on worker nodes cluster. Swarm managers will take responsibility if problems occur

Docker Ecosystem

- The services offering support for running Docker Container do not necessarily run Docker. But Docker Container has become the de facto standard. Recommends working with it
- Difficult to run your own Docker Cluster because of the complicated interaction between Docker and kernels; difficult to find right combination of operating system, kernel, docker and storage driver. Docker technology is very sensitive to the underlying kernel technology

Problems with Docker

- Garbage collection: Ongoing and biggest architectural problemNo limits on quantity or location of storage consumed by users and no storage removal process. All storage will be consumed and wedge docker
- Adding Docker to HPC is difficult: Docker ecosystem is designed to deploy micro services. Conflict when sites already have batch system for managing nodes and file system for managing storage. Other systems are addressing this conflict (e.g., Shifter, Singularity)

Current activities

- In context of large scale data intensive workflows, how can the user provide the container for executing the workflow? Can deploy the container regardless of the underlying site technology?
- "Wharf" IBM Project – how to make better use of Docker on shared file systems like GPFS

Kubernetes Overview – Rick Wagner

Open-source platform designed to automate deploying, scaling, and operating application containers
Sea change: currently in production/preview as a managed service on Amazon, Google, and Azure

Definition

- Portable - VM, physical hardware, deployed as managed services on commercial cloud providers

- Designed to be extensible (enables customization)
- Built-in self-healing through multiple pods running as replicas of each other

How it works (Slides 6-8)

- Pods (i.e., application) run on 1 or more nodes (VM or physical machines)
- Pluggable scheduler in Master (runs API server) picks set of nodes based on pods' needs
- Replication Controller - monitors pods to ensure enough pods are running
- Each service has a virtual IP address to avoid port collisions
- Secrets- can embed and store secrets to be handled appropriately
- ConfigMap: Containers are being defined to be reused/Kubernetes agnostic - Running without having to be focused solely on Kubernetes environment (can run in Kubernetes and in Rancher)

Data Storage (Slide 9)

- Container: ephemeral, data storage is tied to Container lifecycle
- Volumes: less ephemeral, tied to pod lifecycle
- PersistentVolume objects/PersistentVolumeClaim objects: not ephemeral; independent lifecycle

Research CI Perspective

Where Kubernetes fits (Slide 11)

- Common interface: can do on-premises ("on-prem") deployment of Kubernetes and can reach out to talk to Google compute engines or future Amazon services
  - In this single technology, familiarity is gained with lower startup costs
  - Researchers may be able to move between resources more freely; lower barrier because working with familiar technology and just adapting to site configurations

Who is it for? (Slide 12)

- Projects: need to control their applications and own data (less need for multi-user environment)
- Recommendation: Explore Kubernetes' understanding of a job and fault tolerance. If projects adopt Kubernetes, infrastructure providers may not have to heavily adapt to a specific project

Edge Models: Boundary of an institution where services running in DMZ (Slide 13); e.g., Slate Computation Institute and Pacific Research Project have servers running Kubernetes

- Decoupling between application and infrastructure; thus all around on-site expertise not required

Concerns

- Unfamiliar model: Running arbitrary services without any interaction between building application and infrastructure is challenging if the application and infra are housed in separate organizations
- Dockerhub is great, ubiquitous use for researchers, but single point of failure
- Federation: Can federate if have set of on-prem deployments. But AWS and Azure credentials don't translate and not motivated to do so. Processor instructions are a common issue
- Better for certain workloads: Rise of long-tailed based HPC, acknowledgement of small workloads, high throughput could justify native Kubernetes clusters running in a different environment to take on workloads in a different manner and pushing the jobs to HPC systems

More information*:* Zero JupyterHub tutorial; How we got started (Slides 16-17)

Singularity Containers – Dave Godlove

Containers for HPC, analytics, machine learning, reproducible and trusted computing

Unique Features as a Container Platform

Built for HPC/Science: response to HPC users' requests

Single file based container format (different from many other container solutions)

- Allows many things that cannot be done with currently existing, traditional container platforms
- Enables extreme mobility: can use same standard tools (used to transfer data from 1 location to another) to transfer singularity containers from 1 to another
- Controls-compliant: If have controls to protect data that is under security clearance, singularity container will also be protected
- Compatible with complicated architecture (HPC specific architectures)
- Security model: designed to support untrusted users from untrusted containers

Singularity Image Format (SIF)

- Global header and object descriptors reference raw data objects within file (e.g. file system housing container; on top are labels, recipe, etc.)
- File format enables cryptographic signatures for data objects and add signature block as data object in file format itself; so can check container received from colleague to ensure data objects not changed from original intention
- Expanding SIF file format to use for other things
  - create overlays that evolve over time- would enable a SIF container with signed data objects and also writeable overlay that allows container to evolve over time
- Potential use cases:
  - Checkpoint and restart container (pause and move to another system to restart). A writeable overlay could contain state information, so Singularity would know automatically how to restart container after it's checkpointed
  - Container used for Deep learning application could have framework in container image and data, model, etc. in writeable overlay. Copy container up to HPC resource, train model and copy it back down to local computer/lab

Reproducibility & Mobility

- In Singularity, the container is single file, which is run at runtime. If copy container, can ensure bit for bit software reproducibility (identical to original container)
- Current existing container solutions and infrastructure: If want to copy container, means rebuilding from scratch. If anything changes during that process, will not end up with the same reproducible container
- Can copy the container from 1 place to another to use whatever tools using to copy data from 1 place to another. Can easily move containers to cloud, etc.

HPC Compatible Security Model- based on untrusted users running untrusted containers

- Limit user's potential security contexts: If lack root privileges on system, won't have root privileges in the container
- User outside container is the same user inside container
  - Allows users to access their data that on system transparently
  - Also limits access to data that don't own: If create new file on host system while in singularity container, it will have correct ownership and UID from the start

HPC Compatible Container Architecture

- Native MPI support: Run hybrid model – leverages MPI from host system and runs in conjunction with MPI within containers
- GPU: Containers can leverage GPU on host system transparently
- Resource managers (RM): Works with RM because no other process is running, orchestrating container on the user's behalf. Singularity sets up processes are decedents of RM; controlled by the RM
- Single file container images – Single file container format is highly optimized for parallel file systems. Thus, Python containerized version of python installed in Conda runs quicker stored in container, e.g., Wolfgang Resch ran up to 5120 python interpreters simultaneously during the down time on up to 320 nodes, using Gpfs file system

Current Working on:

- Virtually booting containers to interact with containers more like virtual machines
- Evolving signed containers
- OCI compliance: can run Singularity in OCI compliance mode, which allows native Kubernetes support, etc.
- Performance profiling: if installed application within a Singularity container able to profile IO
- Checkpoint/restarting
- Future support of OS X and Windows so users can run on any platform

Greg Kurtzer- Sylabs Presentation available (note: presentation not completed due to time constraints)
Business Ecosystem - enterprise company runtimes are generally designed for microservices. But not every problem is a microservices answer

Missing niche: Many use cases don't fall under the purview of microservices or VM.

- Singularity has solved this for HPC and for niche area
- Singularity fits alongside existing paradigms
- Increasing demand for non-microservice containers

Singularity – Differentiated from a microservices-based architecture. (Slide 9)

- Image format enables simplistic way of moving workflows
  - Singularity and Sylabs become the data/substrate bridging resources together
- Microservices
  - Allows you not to have duplication of Spin
  - Not work for mobility as another registry is needed. Sometimes difficult to recreate a container to re-run it
- Azure batch uses Singularity to distribute and run jobs and containers
- Committed to the success of OS project and community
  - Open GitHub is the primary point for everyone to contribute to Singularity's development
  - All of Singularity's primary development on features, functionality will go into public OSS. Working on what OS traditionally lacks (long term support, backwards compatibility, etc.)
  - Singularity Pro: "snapshots" of public GitHub that will support, maintain/give long term life; distribute via host subscriptions. Precedence to be given to Singularity Pro subscribers

Next MAGIC meeting

March 7, 2018 at the National Coordination Office, 490 L'Enfant Plaza, Suite 8001