Pitfalls of Containers for HPC

SEA: CONTAINERS IN HPC SYMPOSIUM
4/4/17

Nathan Rini

Supercomputer Services Group

nate@ucar.edu

Pitfalls of containers for HPC

An incomplete list of observed and expected pitfalls of HPC containers:

- Leaky Abstraction
- Portability
- Performance
- Networking
- Security
- Maintainability
- Transparency and Usability
- Licensing
- Dockerfile vs Makefile vs cmake vs Shell script
- Caching

Computational & Information Systems Laboratory

Leaky Abstraction

"All non-trivial abstractions, to some degree, are leaky." --Joel Spolsky

- Containers are
  - just another abstraction layer composed of namespaces and cgroups.
  - not trivial by a long shot.
  - inheriting the nature of the machine host.
  - constantly changing in the linux kernel.

Illusion of Portability

- Applications in containers **should** be able to run anywhere
  - Applications can't cross machine architectures
    - Applications compiled for different steppings of Intel Processors may not even work.
  - Kernel ABI is mostly stable but
    - applications can hit a hard wall going downlevel in kernels
    - Kernel drivers level incompatibility
      - Ioctls lack any strong guarantees of compatibility
      - OFED changes all the time
    - Non-uniform kernel support for features across distros
      - RHEL disables cgroup namespaces
  - IPv4 vs IPv6 vs RDMA vs RoCE vs iWARP

- Please refer to JAVA 8 Compatibility Guide to see how this can go very wrong for systems **specifically designed to be portable**.
- The Kernel plumbers didn't design containers for portability.

Slow here, fast there?

Lowest common denominator between machines is usually the slowest.

- Compiling to specific CPUs stepping to get full performance removes portability:
  - gcc -march=native -O3
  - “Using -march=native will enable all instruction subsets supported by the local machine (hence the result might not run on different machines).” -GCC Docs
- Network abstraction will cost performance or portability (sometimes both):
  - Programs will need to be able to use the fastest available network. Users won’t usually have access to the fast interconnects outside of the supercomputers.
    - Example: Compile on your laptop and run on the Supercomputer will probably result in poor performance.
- Containers that use loopback devices of container images can make containers with small files much faster at risk of OOMing the nodes or using swap.

Not all packets are equal

- Software Defined Networks
  - Docker will generally try to put containers behind a bridge for compatibility and security.
  - Adding a Linux bridge will reduce bandwidth and increase latency and most HPC containers will use the host networking namespace.
- IPv4 vs IPv6
  - Containers that utilize IPv6 can fail on clusters with only IPv4 enabled.
- RDMA vs RoCE vs iWARP
  - Containers in virtual networks will suffer from different hardware limitations
- MPI Limitations
  - Several vendor proprietary MPI implementations expect certain networks topologies or configurations.

CISL Computational & Information Systems Laboratory

Security

- Docker gives users access to root directly and indirectly.
- Singularity allows administrators to lock users out from getting root (so far) but at cost of usability.
- Trusting contents of images which are blindly pulled from registries
- All images to need to be regularly updated to meet security requirements. Same issue many sites have with virtual machine sprawl.
- No central way to examine containers or track changes for forensics even on a single kernel instance.
- Giving users semi-privileged access in containers opens up the poorly tested parts of the kernel.

NCAR NSF

Maintainability

- Containers make it **very easy** to build a one off solution that is a nightmare to maintain.
- Users who believe containers are portable will very likely make very fragile containers that may not even run on other machines than their laptops and favorite site.
- User support will now have to deal with a flood of containers instead of 1 environment per cluster.
- Some sites may requires all applications to meet ongoing security compliance checks.
- Every library, every application in a container will need to receive regular security patches.
  - Use case for many users to avoid patching for security and locking into a single version of their preferred library.
- Many Dockerfiles pull from github directly:
  - Github doesn't guarantee files served now will be there later.

CISL Computational & Information Systems Laboratory

NCAR NSF

Transparency and Usability

- Users shouldn’t need to call their container system every time they try to call their application. Job launchers probably need to be aware of containers.
- Schedulers will need to be container aware to make containers “just work”.
- Pam is a good place to change user namespaces but most schedulers don’t use it.
- Integration with systemd is becoming more important but is non-existent.

Licensing

- Many vendors have not yet started to be explicit about licensing their software on containers.
    - Container systems appear to have no built in systems to detect or even warn about these issues.
- Users may easily run afoul of copyright by sharing containers.
    - A user need only keep a copy of the wrong file in their container when they upload it to shub or dockerhub.
    - Statically linked binaries may even have restrictions depending on the license.
- Many of the propriety MPI stacks are licensed by machine and can not be shared in any form via containers outside of the host cluster.

CISL Computational & Information Systems Laboratory
NCAR

Dockerfile vs Make vs CMake vs Scripts vs Recipes

- Dockerfiles have a 127 stacked image hard limit (aka 127 RUN commands max per container). Limit is inherited from Kernel for certain filesystems.
- Users will have to use the build system of the container system or transcode between container systems
  - Singularity can import Docker images directly but Docker won't import Singularity images.

- What is the benefit of using a container build script instead of a shell script or a simple Makefile?
- Users will need to make their container rebuild on every platform to maximise performance. This needs to be simple so users will actually do it.

Caching the Thundering Herd

- Building containers will hammer internet mirrors for Yum/Debian repositories, github, etc.
- Github uses TLS and makes caching very unfriendly. Github will throttle traffic (verified directly with them).
- Having all users build on login nodes or a smaller number of nodes can very easily get nodes throttled and result in nasty emails from vendors to your local abuse email.
- Setting up local instances of Apt-cacher can work around this issue if you can get your users to actually use it.
- Docker supports passthrough Dockerhub caching but must be configured in dockerd.
- Singularity Hub (shub) has no support for a passthrough cache.

Avoiding the Pitfalls?

- Use libfabric to abstract out the fabric. You’ve already started abstraction, might as well finish the job.
- Build application and all dependencies on the target host
  - If you are rebuilding each time the hardware changes, then you avoid the issues with binary compatibility.
  - Ensure your application can build with multiple compiler stacks (GCC, Intel, Clang, etc). This will not only force you to have more compatible code but your application can run on locations without licensed compilers.
- Separate out data from application.
  - Different sites will have different storage engines that may not be directly exposed in the container.
  - I/O intensive applications should consider using non-Posix filesystem access.
- Use site local caches

CISL Computational & Information Systems Laboratory

NCAR NSF

CISL Computational & Information Systems Laboratory

Thank You

NCAR