INTEL® HPC DEVELOPER CONFERENCE

FUEL YOUR INSIGHT

Containers for Science, Reproducibility and Mobility

SINGULARITY P2

Presented By:

Gregory M. Kurtzer
HPC Systems Architect
Lawrence Berkeley National Lab
gmkurtzer@lbl.gov
http://singularity.lbl.gov/

CONTAINERS
(YESTERDAY IN A NUTSHELL)

SO… WHAT IS ALL THE COMMOTION ABOUT?

- Reproducibility and archival software and environment stacks
- Mobility of Compute (portable, sharable, distributable container images)
- User defined and controlled environments (BYOE)
- Integratable with existing shared infrastructures and scheduling subsystems
- Properly make use of the existing high performance physical hardware
- Must support running as the user to facilitate scheduling and MPI workflows
- Make use of all of the work that has been done in Docker so far
- We needed it yesterday (so it must be compatible with today's technology)!

CONTAINERS IN HPC: SINGULARITY


ubuntu
POWERED BY APACHE
NGINX
http server
PostgreSQL
MariaDB
CentOS

SINGULARITY: EXTREME MOBILITY AND PORTABILITY

SINGULARITY: WHO'S USING IT / NAME DROPS / BANDWAGON

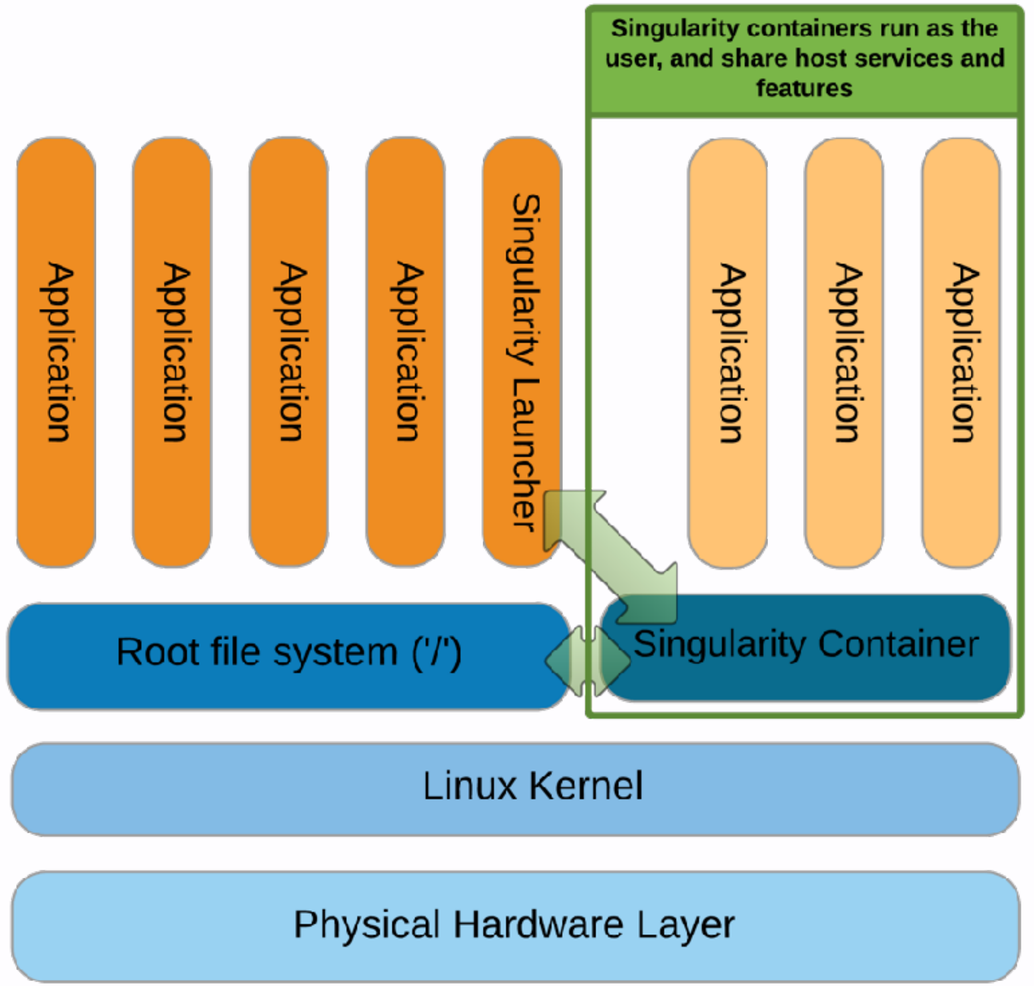
- Texas Advanced Computing Center: 462,462 cores / Stampede
- GSI Helmholtz Center for Ion Research: 300,000 cores / GreenCube
- National Institute of Health: 54,000 cores / Biowulf
- UFIT Research Computing at University of Florida: 51,000 cores / HiPerGator
- San Diego Supercomputing Center: 50,000 cores / Comet and Gordon
- Lawrence Berkeley National Laboratory: 30,000 cores / Lawrencium
- Holland Computing Center at UNL/LHC: 14,000 cores / Crane and Tusker

SINGULARITY ARCHITECTURE

SINGULARITY: ARCHITECTURE OVERVIEW

Applications which run in a container run with the same “distance” to the host kernel and hardware as natively running applications.

Singularity launches the container as the calling user in the appropriate process context. There is no root daemon process and no escalation of privileges within the container.



SINGULARITY: THE CONTAINER PROCESS OVERVIEW

- Singularity application is invoked and shell code evaluates the commands, options, and variables
- The Singularity execution binary (sexec/sexec-suid) is executed via execv()
- Namespaces are created depending on configuration and process requirements
- The Singularity image is checked, parsed, and mounted in the 'CLONE_NEWNS' namespace
- Bind mount points, additional file systems, and hooks into host operating system are setup
- The 'CLONE_FS' namespace is used to virtualize the new root file system
- Singularity calls execv() and Singularity process itself is replaced by the process inside the container
- When the process inside the container exists, all namespaces collapse with that process, leaving a clean system

SINGULARITY: CONTAINERS SUPPORTED

- **Singularity Image**: The standard Singularity image format (built for HPC efficiency)
  - Standard POSIX file system inside image with an offset
  - Header contains an interpretative loader for launching images directly
- **SquashFS**: A kernel standard compressed loopback file system
- **Directory**: Standard Unix Directories containing a root container image
- **Archive Formats**: tar.gz, tar.bz2, tar, cpio, cpio.gz (inside directories)
- **URI**: http://, https://, docker://

SECURITY

SINGULARITY: CONTAINERS INSECURE?

Is Docker, runC, or RKT inherently insure?

NO!

Do their usage models translate securely into HPC?

NO!

Is this one of the reasons HPC has not adopted these container solutions?

YES!

SINGULARITY: PRIVILEGE ESCALATION MODELS

Containers all rely on the ability to use privileged system calls which can pose a problem when allowing users to run containers.

Root Owned Process

- Risk of vulnerability in any root owned daemon
- No ACLs or user limits
- Generally incompatible with HPC resource managers
- Good for service virtualization

SUID

- Typical target for attack
- Code must be easily audit-able
- Allows users to run code with escalated permission
- Easy to leverage with a continuous workflow

User Namespace

- This is the elusive pink unicorn
- Allows users to access some privileged system calls
- As of today, it is unstable

SINGULARITY: PRIVILEGE ACCESS MODELS

- Default run mode for Singularity is SUID
  - Works on all systems, provides an auditing trace, supports all features
  - Requires root to install and only obey's config when owned by root
- Singularity also supports the User Namespace
  - Singularity can be built and used completely unprivileged
  - Some features are limited
  - CAUTION: Not all kernels support this equally… Maybe a while.

SINGULARITY: CONFIGURATION FILE

- Knowing that Singularity has some superuser abilities, final control is given to the system administrator alone via the configuration file
- The configuration file defines what is allowed and what is not allowed
- Singularity only trusts the configuration file when owned by root

```
# USER BIND CONTROL: [BOOL]
# DEFAULT: yes
# Allow users to influence and/or define bind points at runtime? This will allow
# users to specify bind points, scratch and tmp locations. (note: User bind
# control is only allowed if the host also supports PR_SET_NO_NEW_PRIVS)
user bind control = yes
```

SINGULARITY: DEBUG/AUDIT OUTPUT

```
$ singularity --debug shell --bind /opt /tmp/Centos-7.img
 … snip …
VERBOSE [U=1000,P=126582]  util/util.c:87:envar()                      : Obtained input from environment 'SINGULARITY_BINDPATH' = '/opt,'
DEBUG   [U=1000,P=126582]  userbinds.c:48:singularity_mount_userbinds(): Checking for 'user bind control' in config
DEBUG   [U=1000,P=126582]  config_parser.c:146:singularity_config_get_bool(): Called singularity_config_get_bool(user bind control, 1)
DEBUG   [U=1000,P=126582]  config_parser.c:111:singularity_config_get_value(): Called singularity_config_get_value(user bind control)
VERBOSE [U=1000,P=126582]  config_parser.c:122:singularity_config_get_value(): Got config key user bind control (= 'yes')
DEBUG   [U=1000,P=126582]  config_parser.c:152:singularity_config_get_bool(): Return singularity_config_get_bool(user bind control, 1) = 1
DEBUG   [U=1000,P=126582]  userbinds.c:59:singularity_mount_userbinds(): Parsing SINGULARITY_BINDPATH for user-specified bind mounts.
DEBUG   [U=1000,P=126582]  userbinds.c:76:singularity_mount_userbinds(): Found bind: /opt -> container:/opt
DEBUG   [U=1000,P=126582]  userbinds.c:78:singularity_mount_userbinds(): Checking if bind point is already mounted: /opt
DEBUG   [U=1000,P=126582]  rootfs.c:64:singularity_rootfs_dir()        : Returning singularity_rootfs_dir: /var/singularity/mnt/final
DEBUG   [U=1000,P=126582]  mount-util.c:42:check_mounted()             : Opening /proc/mounts
DEBUG   [U=1000,P=126582]  mount-util.c:48:check_mounted()             : Iterating through /proc/mounts
DEBUG   [U=1000,P=126582]  privilege.c:152:singularity_priv_escalate(): Temporarily escalating privileges (U=1000)
VERBOSE [U=0,P=126582]     userbinds.c:136:singularity_mount_userbinds(): Binding '/opt' to '/var/singularity/mnt/final//opt'
DEBUG   [U=0,P=126582]     privilege.c:179:singularity_priv_drop()     : Dropping privileges to UID=1000, GID=1000
DEBUG   [U=1000,P=126582]  privilege.c:191:singularity_priv_drop()     : Confirming we have correct UID/GID
DEBUG   [U=1000,P=126582]  userbinds.c:145:singularity_mount_userbinds(): Unsetting environment variable 'SINGULARITY_BINDPATH'
```

SINGULARITY: A FINAL NOTE ON SECURITY

- I am not deluded…
- Well, sometimes I am. But.. I am open to feedback and criticism
- If you find bugs, issues, questions, concerns…

LET ME KNOW!

SYSTEM INTEGRATION

SINGULARITY: FILE SYSTEMS

- File system agnostic
- Permissions are easy: user inside == user outside
- IO is passed directly through the container via bind's to the real host mount
- Performance impact is unobserved
- System administrators can control what gets shared
- Limitations: bind points must exist, overlayFS unstable, no bind point checks

SINGULARITY: INFINIBAND

- To support InfiniBand, the container must support it!
- Device nodes are passed through into the container
- Kernel/Userspace API alignment required for container/host compatibility
- This is not as bad of a problem as it used to be, but it does exist
- We are interested in collaborating with others that are also interested in investigating and possibly resolving some of these issues

SINGULARITY: GPUS

- Device nodes are passed through into container
- Cuda libraries must be aligned with kernel drivers (similar to OFED)
- Workarounds exists!
  - The host installs Cuda/Nvidia libraries to a directory
  - That directory is configured as a `bind point` within the global Singularity config
  - The library path is added to all container's environments (`/etc/singularity/init`) using the environment variable `LD_LIBRARY_PATH`

SINGULARITY: MIC/KNL

- Singularity is installed into the KNL operating system
- Yeah… That's it.

SINGULARITY: RESOURCE MANAGEMENT

- Scheduler/Resource manager agnostic
- No scheduler changes necessary
- Singularity has no daemon process and always runs as the calling user
- Runs contained applications directly in the user's shell and properly handles IO
- Users run Singularity containers from their own batch scripts
- MPI support is trivial…

SINGULARITY AND OPEN MPI

- Utilizes a hybrid MPI container approach (MPI exists both inside and outside)
- This solves many complexities with remote node addressing and RM coordination
- High performance hardware and architecture can be easily utilized
- No additional issues for scheduling and resource management
- Logical and intuitive execution pathway
- Very little (if any) performance penalty has been observed

QUESTIONS?

SINGULARITY: ADVANCED SINGULARITY WORKSHOP (2)

HTTPS://GITHUB.COM/SINGULARITYWARE/INTEL-HPC-DEVCON

SINGULARITY: LOG INTO AWS COMPUTE INSTANCES

HTTPS://LAB.PORTABLE-HPC.NET/

SINGULARITY: INSTALLATION

```
# Required to build Singularity
$ sudo yum groupinstall "Development Tools"

# Download and build Singularity from the GitHub master branch
$ mkdir ~/git
$ cd ~/git
$ git clone https://github.com/singularityware/singularity.git
$ cd singularity
$ ./autogen.sh
$ ./configure
$ make dist
$ rpmbuild -ta singularity-2.2.tar.gz

# Install the newly build Singularity RPM package
$ sudo yum install $HOME/rpmbuild/RPMS/x86_64/singularity-2.2-0.1.el7.centos.x86_64.rpm

# Install dependencies for bootstrapping a Debian container
$ sudo yum install epel-release
$ sudo yum install debootstrap
```

SINGULARITY: BOOTSTRAP DEFINITION/RECIPE

```
$ cat examples/debian.def
# Copyright (c) 2015-2016, Gregory M. Kurtzer. All rights reserved.
#
# "Singularity" Copyright (c) 2016, The Regents of the University of California,
# through Lawrence Berkeley National Laboratory (subject to receipt of any
# required approvals from the U.S. Dept. of Energy).  All rights reserved.

BootStrap: debootstrap
OSVersion: stable
MirrorURL: http://ftp.us.debian.org/debian/


%post
    echo "Hello from inside the container"
    apt-get update
```

SINGULARITY: BOOTSTRAP

```
$ sudo singularity create -F /tmp/Debian.img
Creating a new image with a maximum size of 768MiB...
Executing image create helper
Formatting image with ext3 file system
Done.
$ sudo singularity bootstrap /tmp/Debian.img examples/debian.def
Bootstrap initialization
Checking bootstrap definition
Executing Prebootstrap module
Executing Bootstrap 'debootstrap' module
… snip …
I: Base system installed successfully.
Executing Postbootstrap module
+ echo Hello from inside the container
Hello from inside the container
+ apt-get update
Ign http://ftp.us.debian.org stable InRelease
Get:1 http://ftp.us.debian.org stable Release.gpg [2373 B]
Hit http://ftp.us.debian.org stable Release
Get:2 http://ftp.us.debian.org stable/main amd64 Packages [6787 kB]
Get:3 http://ftp.us.debian.org stable/main Translation-en [4583 kB]
Fetched 11.4 MB in 3s (3432 kB/s)
Reading package lists... Done
Done.
```

SINGULARITY: USAGE AND OPTION EXAMPLES

```
# Some command example tests
$ singularity exec /tmp/Debian.img pwd
$ singularity exec /tmp/Debian.img whoami
$ singularity --debug exec /tmp/Debian.img true

# Notice the PS output is now in a new process namespace
$ singularity exec -p /tmp/Debian.img ps auxf

# What happens when Contained? Create some files in your home, are they persistent?
$ singularity shell --contain /tmp/Debian.img

# Contain but define a new directory to use for your home
$ singularity shell --contain --home ~/git /tmp/Debian.img

# User defined bind points. What happens if you specify a bind point that doesn't exist? What
# about if you bind ontop of a system location (e.g. /bin/)?
$ singularity shell --bind /tmp:/opt /tmp/Debian.img

# How is the shell environment transposed into the container?
$ singularity exec /tmp/Debian.img env
$ singularity exec /tmp/Debian.img env | wc -l
$ env -i singularity exec /tmp/Debian.img env | wc -l
$ env -i FOO=BAR singularity exec /tmp/Debian.img env
```

OPEN MPI

SINGULARITY: INSTALL OPEN MPI ONTO THE HOST

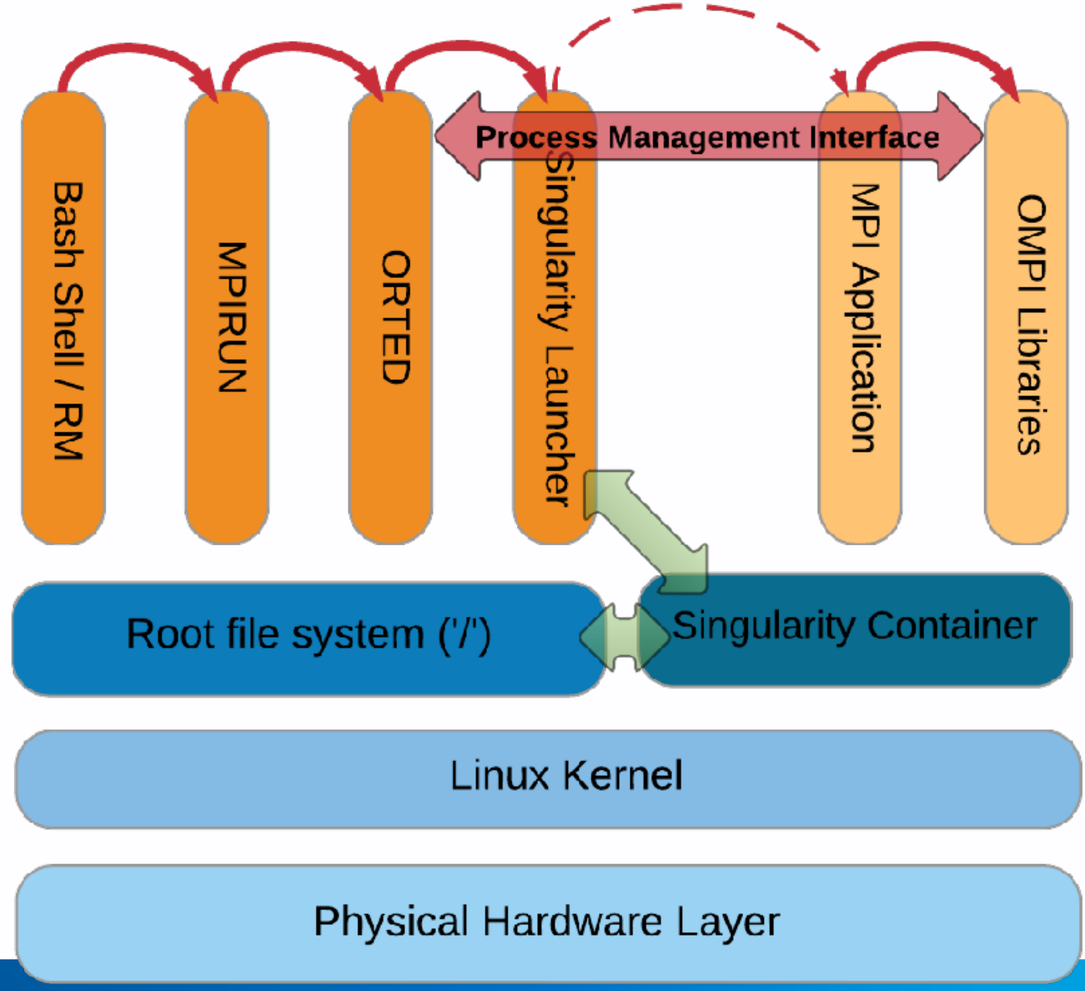
- Open MPI must be newer or equal to the version inside the container
- We build Open MPI from the GitHub master branch on the host first
- Because we are pulling from the master branch,… Let's hope everything works!

```
$ mkdir ~/git
$ git clone https://github.com/open-mpi/ompi.git
$ cd ompi
$ ./autogen.pl
$ ./configure --prefix=/usr/local
$ make -j 10
$ sudo make install
```

SINGULARITY: BUILDING THE NEW CONTAINER

```
$ mpirun -np 4 singularity exec /tmp/Centos-7.img /usr/bin/mpi_ring
```

- Starting with your Bash Shell or resource manager…
- MPI run gets executed which forks an Orted process
- Orted launches Singularity which starts the container process
- The MPI application within the container is linked to the Open MPI runtime libraries within the container
- The Open MPI runtime libraries then connect and communicate back to the Orted process via a universal PMI



SINGULARITY: EXECUTING THE RUNSCRIPT

```
$ cat examples/contrib/centos7-ompi_master.def
BootStrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/os/$basearch/
Include: yum

%post
    echo "Installing Development Tools YUM group"
    yum -y groupinstall "Development Tools"
    echo "Installing OpenMPI into container..."
    mkdir /tmp/git
    cd /tmp/git
    git clone https://github.com/open-mpi/ompi.git
    cd ompi
    ./autogen.pl
    ./configure --prefix=/usr/local
    make -j 10
    make install
    /usr/local/bin/mpicc examples/ring_c.c -o /usr/bin/mpi_ring
    cd /
    rm -rf /tmp/git
    exit 0

%test
    /usr/local/bin/mpirun --allow-run-as-root /usr/bin/mpi_ring
```

SINGULARITY: BUILDING THE NEW CONTAINER

```
$ sudo singularity create -F —size 2048 /tmp/Centos7-ompi.img
Creating a new image with a maximum size of 768MiB...
Executing image create helper
Formatting image with ext3 file system
Done.
$ sudo singularity bootstrap /tmp/Centos7-ompi.img examples/contrib/centos7-ompi_master.def
Bootstrap initialization
Checking bootstrap definition
Executing Prebootstrap module
Executing Bootstrap 'yum' module
Found YUM at: /bin/yum
base                                                                    | 3.6 kB  00:00:00
(1/2): base/x86_64/group_gz                                             | 155 kB  00:00:00
(2/2): base/x86_64/primary_db                                           | 5.3 MB  00:00:01
Resolving Dependencies
--> Running transaction check
---> Package centos-release.x86_64 0:7-2.1511.el7.centos.2.10 will be installed
---> Package coreutils.x86_64 0:8.22-15.el7 will be installed
--> Processing Dependency: rtld(GNU_HASH) for package: coreutils-8.22-15.el7.x86_64
--> Processing Dependency: ncurses for package: coreutils-8.22-15.el7.x86_64
 … snip …
```

THE SINGULARITY CONFIGURATION FILE

SINGULARITY: CONFIGURATION

- The Singularity configuration must be owned by root if running in SUID mode
- System admins have control over the execution options when running privileged (SUID)
- When running with user namespace, root ownership of config file isn't necessary
- The Location of the configuration file is hard coded into the binary

SINGULARITY: CONFIGURATION

Allow SUID?

```
# ALLOW SETUID: [BOOL]
# DEFAULT: yes
# Should we allow users to utilize the setuid binary for launching singularity?
# The majority of features require this to be set to yes, but newer Fedora and
# Ubuntu kernels can provide limited functionality in unprivileged mode.
allow setuid = yes
```

SINGULARITY: CONFIGURATION

Allow usage of the PID Namespace?

You may wish to disable the PID namespace as on some systems it confuses the resource manager

```
# ALLOW PID NS: [BOOL]
# DEFAULT: yes
# Should we allow users to request the PID namespace?
allow pid ns = yes
```

SINGULARITY: CONFIGURATION

Enable the OverlayFS

```
# ENABLE OVERLAY: [BOOL]
# DEFAULT: yes
# Enabling this option will make it possible to specify bind paths to locations
# that do not currently exist within the container. Some limitations still exist
# when running in completely non-privileged mode. (note: this option is only
# supported on hosts that support overlay file systems).
# note: currently disabled because RHEL7 kernel crashes with it... :(
enable overlay = no
```

SINGULARITY: CONFIGURATION

Automatically adjust container files at runtime

- Automatically entries in `/etc/passwd` and `/etc/group` within the container
- Add the master's `/etc/resolv.conf` into the container

```
# CONFIG PASSWD: [BOOL]
# DEFAULT: yes
# If /etc/passwd exists within the container, this will automatically append
# an entry for the calling user.
config passwd = yes
```

SINGULARITY: CONFIGURATION

Automatically Bind mount system file systems

```
# MOUNT HOME: [BOOL]
# DEFAULT: yes
# Should we automatically determine the calling user's home directory and
# attempt to mount it's base path into the container? If the --contain option
# is used, the home directory will be created within the session directory or
# can be overridden with the SINGULARITY_HOME or SINGULARITY_WORKDIR
# environment variables (or their corresponding command line options).
mount home = yes
```

SINGULARITY: CONFIGURATION

Bind paths to always try to include into container

```
# BIND PATH: [STRING]
# DEFAULT: Undefined
# Define a list of files/directories that should be made available from within
# the container. The file or directory must exist within the container on
# which to attach to. you can specify a different source and destination
# path (respectively) with a colon; otherwise source and dest are the same.
#bind path = /etc/singularity/default-nsswitch.conf:/etc/nsswitch.conf
#bind path = /opt
bind path = /global
bind path = /etc/hosts
```

SINGULARITY: CONFIGURATION

Allow users to request arbitrary bind points

```
# USER BIND CONTROL: [BOOL]
# DEFAULT: yes
# Allow users to influence and/or define bind points at runtime? This will allow
# users to specify bind points, scratch and tmp locations. (note: User bind
# control is only allowed if the host also supports PR_SET_NO_NEW_PRIVS)
user bind control = yes
```

SINGULARITY: WAITING FOR THE OMPI CONTAINER TO FINISH BUILDING…

```
… snip …
+ /usr/local/bin/mpicc examples/ring_c.c -o /usr/bin/mpi_ring
+ cd /
+ rm -rf /tmp/git
+ exit 0
+ /usr/local/bin/mpirun --allow-run-as-root /usr/bin/mpi_ring
Process 0 sending 10 to 1, tag 201 (4 processes in ring)
Process 0 sent to 1
Process 0 decremented value: 9
Process 0 decremented value: 8
Process 0 decremented value: 7
Process 0 decremented value: 6
Process 0 decremented value: 5
Process 0 decremented value: 4
Process 0 decremented value: 3
Process 0 decremented value: 2
Process 0 decremented value: 1
Process 0 decremented value: 0
Process 0 exiting
Process 1 exiting
Process 2 exiting
Process 3 exiting
Done.
```

SINGULARITY: OMPI TEST

```
$ mpirun -np 4 singularity exec /tmp/Centos7-ompi.img /usr/bin/mpi_ring
Process 0 sending 10 to 1, tag 201 (4 processes in ring)
Process 0 sent to 1
Process 0 decremented value: 9
Process 0 decremented value: 8
Process 0 decremented value: 7
Process 0 decremented value: 6
Process 0 decremented value: 5
Process 0 decremented value: 4
Process 0 decremented value: 3
Process 0 decremented value: 2
Process 0 decremented value: 1
Process 0 decremented value: 0
Process 0 exiting
Process 1 exiting
Process 2 exiting
Process 3 exiting
```

QUESTIONS?

INTEL® HPC DEVELOPER CONFERENCE

FUEL YOUR INSIGHT

Containers for Science, Reproducibility and Mobility

SINGULARITY P2

Presented By:

Gregory M. Kurtzer
HPC Systems Architect
Lawrence Berkeley National Lab
gmkurtzer@lbl.gov
http://singularity.lbl.gov/