HPC Containers: Tips, Challenges and Solutions

Smahane Douyeb

April 10th, 2019

Notices & Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration.

No product or component can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. For more complete information about performance and benchmark results, visit http://www.intel.com/benchmarks .

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit http://www.intel.com/benchmarks .

Intel® Advanced Vector Extensions (Intel® AVX)* provides higher throughput to certain processor operations. Due to varying processor power characteristics, utilizing AVX instructions may cause a) some parts to operate at less than the rated frequency and b) some parts with Intel® Turbo Boost Technology 2.0 to not achieve any or maximum turbo frequencies. Performance varies depending on hardware, software, and system configuration and you can learn more at http://www.intel.com/go/turbo.

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

*Other names and brands may be claimed as property of others.

Agenda TBD

- Container choices and why Singularity?
- How to build, run, and share a container?
- Complexities and ways around
- Other useful container features

Selecting between container technologies: Why Singularity?

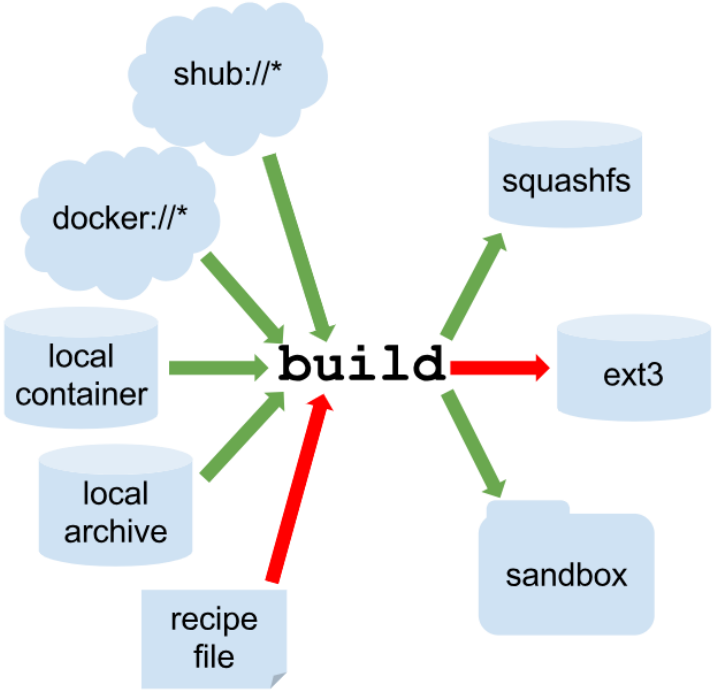
- User permissions are the same inside and outside the container (very important for HPC clusters)
- HPC optimized and friendly to use allowing a full utilization of the host software and hardware resources
- Containers are a self-contained small, lightweight bundle of one or more applications, dependencies and results in less performance loss
- Singularity is a container technology that is built for HPC
- Allows integration with HPC cluster apps (InfiniBand, *MPI*...), compatible with other container technologies (Docker), and resource managers (SLURM, TORQUE...)
- Singularity community and support are HPC folks too

How to Build, Run and Share a Container

Build a Container

Several ways to build a container:

- URI beginning with **shub://** to build from Singularity Hub
- URI beginning with **docker://** to build from Docker Hub
- Path to a **existing container** on your local machine
- Path to a **directory** to build from a sandbox
- Path to an **archive** in .tar or compressed .tar.gz format
- Path to a **Singularity recipe file –** **The best way to build a customized image**



$ sudo singularity build <container-name.simg> Singularity.recipeFile

https://www.sylabs.io/docs/

Example of a LAMMPS Recipe File – “Header” Section

```
BootStrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/os/$basearch/

Include: yum
```

Example of a LAMMPS Recipe File – “Runscript” section

```
#############################################
$ runscript
############
cd $WORKDIR

NUMCORES="$1"
OMP_NUM_THREADS="$2"

nproc=`nproc`
files=`echo in.intel.*`
for file in $files
do
  name=`echo $file | sed 's/in\.intel\.//g'`
  log="${HOME}/${LOG}_${name}"
  echo -n "Running: $name " |tee -a $HOME/$RESULTS
  mpiexec.hydra -np $NUMCORES ./lmp_intel_cpu_intelmpi -in $file -log none -pk intel 0 omp 2 -sf intel -v m 0.2 -screen $log
  grep 'Perform' $log | awk 'BEGIN{n=1}n%2==0{c=NF-1; print "Performance:",$c,"timesteps/sec"}{n++}' |tee -a $HOME/$RESULTS
done
```

Example of a LAMMPS Recipe File – “Setup” Section

```
##############################################
%setup
######
#Commands in the %setup section are executed on the host system outside of the container after the base OS has been installed

base=`pwd`
# Get the codes and any dependencies
        rm -rf lammps
        git clone https://github.com/lammps/lammps.git $base/lammps
#Build your code

        echo "Build LAMMPS binaries"

        cd $base/lammps/src/
        make yes-asphere yes-class2 yes-kspace yes-manybody yes-misc yes-molecule
        make yes-mpiio yes-opt yes-replica yes-rigid
        make yes-user-omp yes-user-intel
        export LMP_ROOT="../../../"
        .......

#Create a work directory inside the container
        WORKDIR="$SINGULARITY_ROOTFS/lammps"
        mkdir -p $WORKDIR

# Copy all the binaries and anything else needed to run your binaries
        cp -rf $BENCH_DIR/* $WORKDIR
        exit 0
```

Example of a LAMMPS Recipe File – “Post” Section

```
##############################################
%post
######

#Commands in the %post section are executed within the container after the base OS has been installed at build time.
#This is where the meat of your setup will live, including making directories, and installing software and libraries
#You cannot copy files from the host to your container in this section, but you can of course download with commands like git clone and

yum install -y sudo wget vi which numactl
yum install -y hostname lscpu uptime redhat-lsb

#installing runtime libs
rpm --import https://yum.repos.intel.com/2018/setup/RPM-GPG-KEY-intel-psxe-runtime-2018
rpm -Uhv https://yum.repos.intel.com/2018/setup/intel-psxe-runtime-2018-reposetup-1-0.noarch.rpm
yum install intel-psxe-runtime -y
yum install libhfil libpsm2 -y
yum install libnl -y
```

Run a Container

- **Run your image as an executable**

  $ singularity **run** <container-name.simg>

- **Run from inside the container :**

  $ singularity **shell** <container-name.simg>

  $ cd $WORKDIR #inside the container

  $ run commands #inside the container

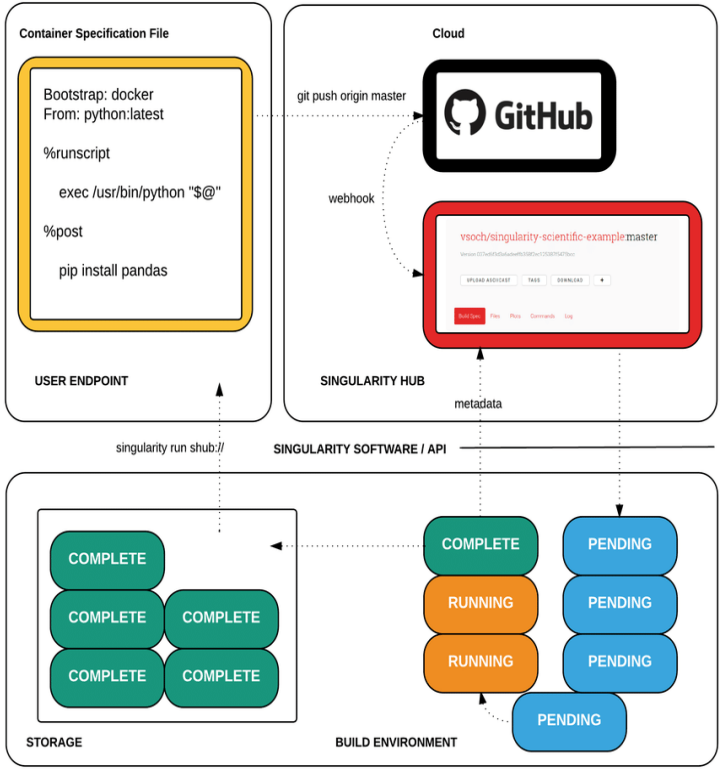
- **Run with the exec command**

  $ singularity **exec** <container-name.simg> /$WORKDIR/$BINARY <localhost>/$workload arg1 arg2 ...

Storing , Distributing and Sharing the Container

- Store the container's definition file in a public source control (github or gitlab)
- Create an account on Singularity Hub
- Connect the Github repository to a new container collection
- Push the Singularity build recipe (The container is built on the cloud)
- Share or pull the container

  $ singularity pull shub://$reponame/$container-name



https://tin6150.github.io/psg/blogger_container_hpc.html

Complexities and Ways Around

Observations!

- Documentations are pretty good but elementary
- A learning curve and mindset are needed
- Container building and deployment is not straightforward and it is not a drop-in solution – human interaction is still needed
- MPI for multi-node runs can still be confusing and version mismatch can be an issue
- File systems and bind methods are tricky and need experimentation
- Space of the container and the OverlayFS available as a user is limited
- Profile an application inside a container is still very new

Container adoption in HPC is not easy!

- Problems:
  - Mentalities of HPC folks and system admins are a bit hard to change
  - Security is still a concern for the majority of traditional HPC clusters
  - Super user access is required to build a container (usually not an option)
  - Containers for real HPC applications are complex. It takes months for the one to get their head around what's going on.
  - Support is minimal due to the shortage of skillset in this area
  - No standard way of development, deployment, or managing the containers. Everyone is doing their own thing…

Common issue : difficulty building codes inside the container

- Problem:
  - Compiler and dependencies are too large
  - Most requirement have licensing obligations (compilers, 3rd party libraries …)
- One workaround :
  - Keep all in one definition file and in a single run:
    - Build the binaries and workloads on the host system
    - Copy the binaries/workloads to the container

```
##############################################
%setup
######
#Commands in the %setup section are executed on the host system
# outside of the container after the base OS has been installed
base=`pwd`

# Get the codes and any dependencies
        rm -rf lammps
        git clone https://github.com/lammps/lammps.git $base/lammps
        cd $base/lammps
        git checkout 0c287a55cd0be3103bc67be0d8f0688e1a6db345

#Build your code

        echo "Build LAMMPS binaries"

        cd $base/lammps/src/
        make yes-asphere yes-class2 yes-kspace yes-manybody yes-misc yes-molecule
        make yes-mpiio yes-opt yes-replica yes-rigid
        make yes-user-omp yes-user-intel
        export LMP_ROOT="../../../"
        source /opt/intel/compilers_and_libraries_2018.3.222/linux/bin/compilervars.sh intel64
        make intel_cpu_intelmpi -j
        LMP_BIN="$base/lammps/src/lmp_intel_cpu_intelmpi"

        echo "Create data files"
        BENCH_DIR="$(pwd -P)/workloads"
        mkdir -p $BENCH_DIR
        cp -rf $LMP_BIN $BENCH_DIR
        cp -rf ./USER-INTEL/TEST/in.* ./USER-INTEL/TEST/mW* $BENCH_DIR
        cp -rf ../bench/Cu_u3.eam ../bench/data.rhodo $BENCH_DIR
        cp -rf ../bench/POTENTIALS/Si.* $BENCH_DIR
        cp -rf ../examples/airebo/data.airebo ../potentials/CH.airebo $BENCH_DIR

        cd $BENCH_DIR
        files=`echo in.*`;
        for file in $files
        do
               sed -i 's/\${root}.*\///g' $file
        done

#Create a work directory inside the container
        WORKDIR="$SINGULARITY_ROOTFS/lammps"
        mkdir -p $WORKDIR

# Copy all the binaries and anything else needed to run your binaries
        cp -rf $BENCH_DIR/* $WORKDIR
```

Single node run using shared memory between MPI ranks

- Problem:
  - MPI is not always available on the targeted run environment
- One workaround :
  - In the %post section of the container definition file, install Intel MPI through the RPM packages:
    - $ rpm --import https://yum.repos.intel.com/2018/setup/RPM-GPG-KEY-intel-psxe-runtime-2018
    - $ rpm -Uhv https://yum.repos.intel.com/2018/setup/intel-psxe-runtime-2018-reposetup-1-0.noarch.rpm
    - $ yum install intel-psxe-runtime -y

Single-node and multi-node in the same container

- Problem:
  - Most examples use **exec** command to execute the binary– becomes too complicated in some cases
  - How to encapsulate single and multi-node run commands in the same container?
- Current workaround :
  - Treat single run and multi-run as 2 different applications that use the same binary but each have a different run script

Example

$apprun singlenode

```
mpiexec.hydra -np $NUMCORES ./lmp_intel_cpu_intelmpi -in $file -log none -pk intel 0 omp $OMP_NUM_THREADS -sf intel -v m 0.2
```

$apprun multinode

```
./lmp_intel_cpu_intelmpi -in $file -log none -pk intel 0 omp 2 -sf intel -v m 0.2 -screen $log
```

Run the container as:

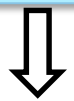
$ singularity run –app singlenode <container-name>

$ mpirun –n $NODES –ppn $PPN singularity run –app multimode <container-name>

Multi-node runs using high-performance interconnect

- Problem:
  - No well defined common supported API for interconnect portability across multiple implementations (InfiniBand, OmniPath, Ethernet, etc)
  - No clear instructions of how to make the network device accessible to the container
- One workaround depending on the interconnect of choice:
  - **Install the necessary libraries** to communicate with the hardware and **set the environment variables** at runtime

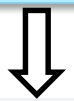
Include these in the %post section of the container definition file



```
$ yum install -y libhfil libpsm2
$ yum install –y libnl
```

For Intel® Omni-Path fabrics

Include these in your MPI run command (mpirun –genv … singularity run …)



```
I_MPI_FABRICS=shm:ofi
I_MPI_TMI_PROVIDER=psm2
```

```
$ cd /tmp/MLNX_OFED_LINUX-$VER-$OS
$ ./mlnxofedinstall
```

For Mellanox EDR

```
I_MPI_FABRICS=shm:ofi
I_MPI_DAPL_PROVIDER=verbs
```

Space of the container and the OverlayFS available as a user is limited

- The size of the container image is determined at container build time
- The OverlayFS is created at runtime as a place that small changes, when running the container, can be written to in RAM and will be lost when the container exits

```
Singularity lammps.simg:~> df -h
Filesystem      Size  Used Avail Use% Mounted on
singularity     1.0M     0  1.0M   0% /
```

- Problem:
  - OverlayFS space problem when running as a user: only 1 MB of space is available
  - The container size is fixed and can't increase unless we expend it

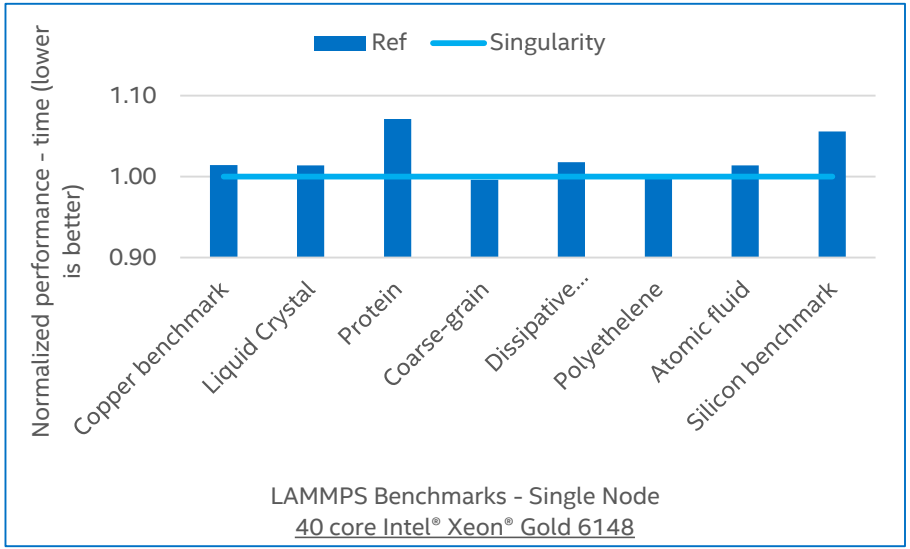
- One workaround:
  - Save the input files to the host and mount its location to the container at run time:

    ```
    $ singularity run –B <localdir/InputFiles>:/workload –app singlenode <container-name>
    ```

  - Modify your application to write to a local directory (like /tmp) not a directory inside the container or the OverleyFS

Performance between container vs bare metal?

- Problem:
  - Perception that container overhead impact the performance
- Response:
  - No significant impact using singularity container



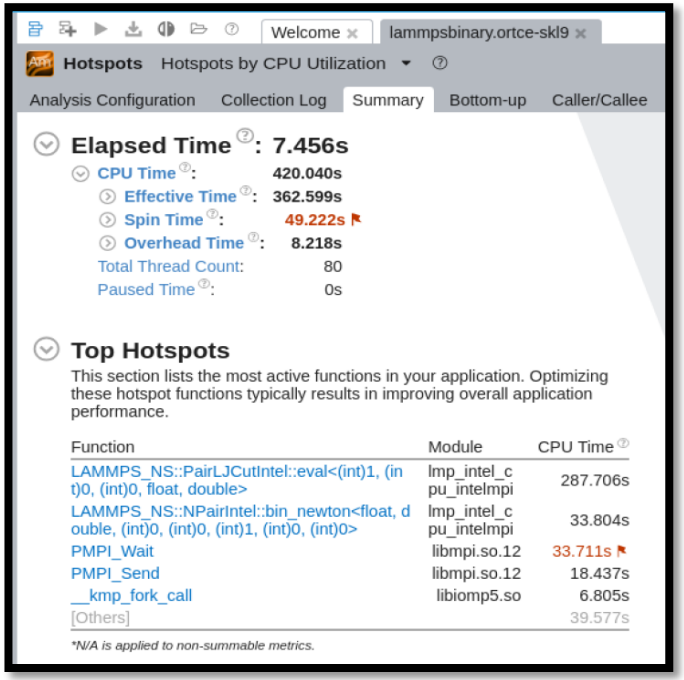
Other Useful Container Features

Intel® VTune™ Amplifier Analysis with and without Container

Intro: Intel® VTune™ Amplifier collects key profiling data and presents it with a powerful interface that simplifies its analysis and interpretation.

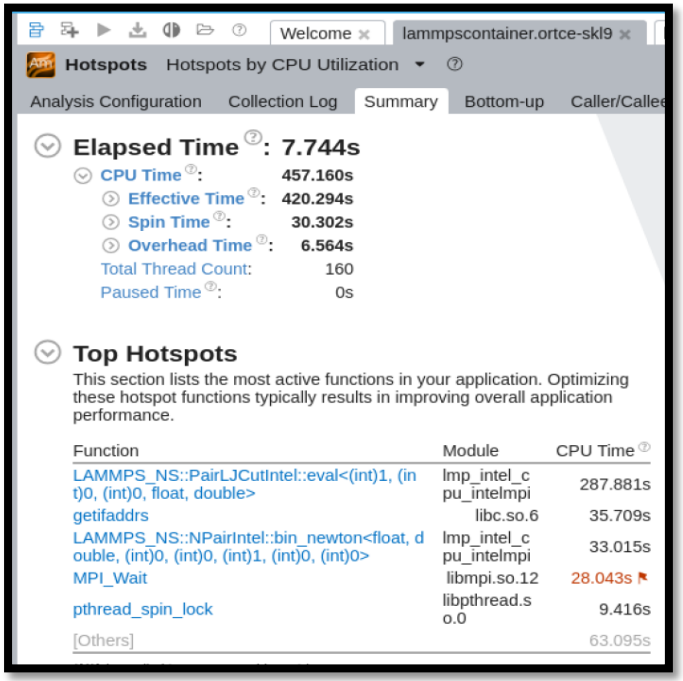
- The configuration of a Singularity container for the Intel® VTune™ Amplifier analysis to identify hotspots is supported
- Profiling a target application running in the Singularity container is only supported from the same container. **Running the VTune™ Amplifier outside the container for Singularity profiling is not supported.**
- The container must have access to the VTune™ binary and source files from the host system

LAMMPS Example



VTune™ analysis inside the container

VTune™ analysis without the container



After you set up your environment :

$ mpiexec.hydra -np 40 /opt/intel/vtune_amplifier<PATH>/amplxe-cl collect hotspots -r <resultsDir> ../lmp_intel_cpu_intelmpi -in in.intel.lj -log none -pk intel 0 omp 2 -sf intel -v m 0.2

$ singularity shell -B /opt/intel/:/mnt lammps.simg

$ mpiexec.hydra -np 40 /mnt/vtune_amplifier<PATH>/amplxe-cl -collect hotspots -r <resultsDir> ./lmp_intel_cpu_intelmpi -in in.intel.lj -pk intel 0 omp 2 -sf intel -v m 0.2

The “Help” section is everything!

- Assume that **the user** of your container is a newbie
- Provide clear instructions of **how to use the container**
- Clarify how to run on **different core count/platforms** if applicable
- Give an example of how to run over **specific interconnects**
- Show how to **cross mount a directory** to your container
- Explain **where are the input files** and how to get them
- If the container writes **output files**, provide the **location**

Example

```
%help
######
EXAMPLES:
  - Available apps:
        $ singularity apps <container-name.simg>
            gromacs
            multinode
            sysinfo
            appinfo
            clean

  - Available workloads inside the container:

        Workload Name           : Argument to pass to the container
        ------------------------------------------------------------
        ion_channel_pme         : ion_channel.tpr
        water_pme               : topol_pme.tpr
        water_rf                : topol_rf.tpr
        lignocellulose_rf       : lignocellulose-rf.tpr

  - Running recommendations for SKL:
        For topol_pme (both tpr and rf) – 30000 steps (over 50 (for pme) and 30 (for rf) sec on 16 nodes)
        For ion_channel.tpr – 55000 steps (over 30 sec on 16 nodes).
        For lignocellulose-rf.tpr – 8000 steps (over 30 sec on 16 nodes).

  - Single node, to run one workload:
        $ singularity run --app gromacs <container-name.simg> $NTHREADS $WORKLOAD $NSTEPS

    for example to run the water_pme workload:
        $ singularity run --app gromacs gromacs.simg 40 topol_pme.tpr 30000

  - Single node to run all workloads:
        $ singularity run --app gromacs <container-name.simg> $NTHREADS

  - Cluster:
        $ mpirun -n $NP -hostfile nodelist singularity run  --app multinode <container-name.simg>  $WORKLOAD $NSTEPS

   for example to run lignocellulose_rf workload on a 4/40 cores per node:
        $ mpirun -n 160 -hostfile nodelist singularity run  --app multinode gromacs.simg lignocellulose-rf.tpr 8000

  - Run multiple apps:
        $ for app in sysinfo appinfo gromacs ; do singularity run --app $app <container-name.simg> ; done
```

Results management

How useful is a container if I can't get results at the end of a run?

Always remember to:

- Parse the performance results right after the run of the application
- Isolate a figure of merit into a file
- Collect system info at runtime
- Provide the application information that's available inside the container (App version, build command, compiler flags…)

Example

Add platform data collection and application information as “applications” inside your container

```
##############################################
%apprun sysinfo
###############
echo "Getting system configuration"
cd $WORKDIR
./sysinfo.sh > $RESULTS_DIR/$SYSCONFIG

echo "The $SYSCONFIG is located at $RESULTS_DIR"
```

```
##############################################
%apprun appinfo
#############
./appinfo.sh |tee $RESULTS_DIR/$APPINFO

echo "The application data information is located at $RESULTS_DIR
```

Run as:
$ for app in sysinfo appinfo ; do singularity run --app $app container.simg; done

Know the container

- Use the “Help” message if available
  $ singularity **help** container.simg
- Extract the content of the container
  $ singularity **image.export** container.simg > container.tar
- Get the definition file from a container
  $ singularity **inspect –d** container.simg
- Shell into the Container and navigate the file system
  $ singularity **shell** container.simg
- Write into the Container
  $ sudo singularity **exec –writable** container.simg yum install wget

Thank you!

LAMMPS example used in this demo is available here:

https://github.com/intel/HPC-containers-from-Intel/

Questions?

smahane.douyeb@intel.com