1 Finding and Building Images

In order to use containers on TACC systems, you must first understand our requirements. For instance:

- What MPI would you like to use?
- What CPU architecture are you running on?
- Does your code need accelerators (GPUs)?

For this example on Frontera, you use the module defaults. The MPI is Intel MPI, version 19.0.9. The architecture is x86, and no GPUs are used.

1.1 Finding MPI Versions

In order to find out the MPI version on Frontera and many other HPC Machines, these commands can be used.

```
$ module list

Currently Loaded Modules:
  1) intel/19.1.1 2) impi/19.0.9 3) git/2.24.1 4) autotools/1.2 5) python3/3.7.0 6) cmake/3.16.1 7) pmix
       /3.1.4 8) hwloc/1.11.12 9) xalt/2.10.2 10) TACC
```

You can see the Intel MPI version here. It is important to note there are multiple MPI versions available on Frontera, so please match the one you are using to the container.

1.2 Finding CPU Architecture

In order to find the architecture, submit a job for an interactive job in the partition (queue) that you will be running. Some HPC machines may have multiple architectures so it is important to use the correct partition.

```
$ idev -p small -N 1 -n 56 -A Frontera-Training
 -> After your idev job begins to run, a command prompt will appear,
 -> and you can begin your interactive development session.
 -> We will report the job status every 4 seconds: (PD=pending, R=running).

 -> job status: PD
  -> job status: R

 -> Job is now running on masternode= c211-020...OK
 -> Sleeping for 7 seconds...OK
 -> Checking to make sure your job has initialized an env for you....OK
 -> Creating interactive terminal session (login) on master node c211-020.

ssh -Y -A -o StrictHostKeyChecking no c211-020
TACC Frontera System
Provisioned on 10-Feb-2021 at 08:13

$ uname -a
Linux c211-020.frontera.tacc.utexas.edu 3.10.0-1127.19.1.el7.x86_64 #1 SMP Tue Aug 25 17:23:54 UTC 2020
     x86_64 x86_64 x86_64 GNU/Linux
```

Looking at the output of uname, you can see that this kernel is x86_64, which is the default for most container images. As a counter example on the TACC Longhorn system, which is a Power9 architecture, the command looks like this:

```
$ uname -a
Linux c002-001.longhorn.tacc.utexas.edu 4.14.0-115.10.1.el7a.ppc64le #1 SMP Wed Jun 26 09:32:17 UTC 2019
      ppc64le ppc64le ppc64le GNU/Linux
```

You can see that the architecture is ppc64le. More information on this platform is below.

1.3 Selecting the Correct TACC Container

TACC curates a number of containers for TACC systems. They are available in DockerHub (https://hub.docker.com/u/tacc) for easy building, however, they are also available on GitHub (https://github.com/TACC/tacc-containers) along with documentation about the containers and the dockerfile that is used to create them.

On the GitHub page there is a table of all base containers, including containers that have known-working MPI installations. In the table you see that there is a CentOS 7 container with Intel MPI 19 that works with Frontera called: tacc-centos7-impi19.0.7-common. This container will be used.

1.4 Building a TACC Container

To start the build process you can pull the dockerfile directly from DockerHub with the singularity command. An image will be built from this dockerfile.

```
$ module load tacc-singularity
$ singularity pull docker://tacc/tacc-centos7-impi19.0.7-common
INFO: Converting OCI blobs to SIF format
INFO: Starting build...
...

...
INFO: Creating SIF file...
$ ls
tacc-centos7-impi19.0.7-common_latest.sif
```

1.5 Testing the Container

Now that you have a singularity image you want to make sure this works. First find an exemplar MPI code you are familiar with. For this example VPIC is chosen. First make sure the application runs correctly without the container.

```
$ ibrun ./VPIC_Example
TACC: Starting up job 3069405
TACC: Starting parallel tasks...
...

...
Task completed successfully
```

Now try running our exemplar application.

```
$ ibrun singularity exec tacc-centos7-impi19.0.7-common_latest.sif ./VPIC_Example
TACC: Starting up job 3069405
TACC: Starting parallel tasks...
singularity_tutorial/VPIC_Example: error while loading shared libraries: libmpicxx.so.12: cannot open
     shared object file: No such file or directory
```

You can see that the application cannot find the the MPI library it was linked against. This is because my exemplar application was compiled using modules, and the modules aren't available. The reason they are not available is because you changed out the entire software stack with the container. The only thing common with the host is the kernel. Let's look at the binary and see what file it's looking for exactly.

```
$ ldd VPIC_Example
    linux-vdso.so.1 => (0x00007ffdd234c000)
    /opt/apps/xalt/xalt/lib64/libxalt_init.so (0x00002b44434ab000)
    libmpicxx.so.12 => /opt/intel/compilers_and_libraries_2020.4.304/linux/mpi/intel64/lib/libmpicxx.so
         .12 (0x00002b44436ed000)
...

...
    /lib64/ld-linux-x86-64.so.2 (0x00002b4443287000)
```

you see that it is looking for it in a directory in /opt. Now, can you see that file within the container?

```
$ singularity exec tacc-centos7-impi19.0.7-common_latest.sif ls /opt/intel/compilers_and_libraries_
    2020.4.304/linux/mpi/intel64/lib/libmpicxx.so.12
/bin/ls: cannot access /opt/intel/compilers_and_libraries_2020.4.304/linux/mpi/intel64/lib/libmpicxx.so
    .12: No such file or directory
$ singularity exec tacc-centos7-impi19.0.7-common_latest.sif ls /opt/intel/
bin compilers_and_libraries_2020 conda_channel imb parallel_studio_xe_2020 samples_2020
compilers_and_libraries compilers_and_libraries_2020.1.217 documentation_2020 impi parallel_studio_xe_
    2020.1.102 uninstall
```

So it looks like that file doesn't exist and the container uses an older version of the Intel compiler. you have two choices now, either recompile the application using the older version (ideally within the container) OR bring in the modules from the /opt file system on Frontera and thus get the newer version. For this exercise you choose the second.

In order to do this you will need to mount the directory on Frontera inside the container and setup our environment. For our environment you create new environment variables prefixed by SINGULARITYENV. This will work for any environment variables but for this exemplar application only PATH and LD_LIBRARY_PATH are needed. To mount the /opt filesystem you use the -B (bind) flag.

```
$ export SINGULARITYENV_PATH=$PATH
$ export SINGULARITYENV_LD_LIBRARY_PATH=$LD_LIBRARY_PATH
$ ibrun singularity exec -B /opt tacc-centos7-impi19.0.7-common_latest.sif ./VPIC_Example
...

...
Task completed successfully
```

Important to Consider: Containers are often used to provide software that is unavailable on the host system. It is not always possible to test your target application outside of the container, however, having an exemplar MPI application to test with can help to troubleshoot if problems are encountered.

2 Using Non X86 Architectures

For this example you will run our exemplar application on the Longhorn cluster at TACC. This cluster uses a Power 9 architecture (ppc64le) and v100 GPUs. First lets build a container that matches the architecture. From the table on the TACC-Containers GitHub you see that Longhorn has a few options, but since you are using GPUs you will use the GPU-optimized mvapich, mvapich-gdr. So you choose the tacc-centos7-ppc64le-mvapich2.3-ib image.

```
$ singularity pull docker://tacc/tacc-centos7-ppc64le-mvapich2.3-ib
INFO: Converting OCI blobs to SIF format
INFO: Starting build...
...

...
INFO: Creating SIF file...
INFO: Build complete: tacc-centos7-ppc64le-mvapich2.3-ib_latest.sif
```

2.1 With GPUs

Now, take what you know from the last section and try to run. CUDA will be needed in the container so you use the –nv flag with singularity.

```
export SINGULARITYENV_PATH=$PATH
export SINGULARITYENV_LD_LIBRARY_PATH=$LD_LIBRARY_PATH

c004-010.longhorn(235)$ ibrun singularity exec -B /opt --nv tacc-centos7-ppc64le-mvapich2.3-ib_latest.
    sif ./VPIC_Longhorn_Example
TACC: Starting up job 78875
TACC: Setting up parallel environment for MVAPICH2+mpispawn.
TACC: Starting parallel tasks...
./VPIC_Longhorn_Example: error while loading shared libraries: libcudart.so.10.2: cannot open shared
    object file: No such file or directory
```

You can see another library missing, lets use the method from the last section to see where that library lives.

```
c004-010.longhorn(237)$ ldd VPIC_Longhorn_Example
    linux-vdso64.so.1 => (0x0000200000050000)
    /opt/apps/xalt/xalt/lib64/libxalt_init.so (0x0000200000070000)
    libmpicxx.so.12 => /opt/apps/gcc7_3/mvapich2-gdr/2.3.4/lib64/libmpicxx.so.12 (0x00002000000e0000)
    libmpi.so.12 => /opt/apps/gcc7_3/mvapich2-gdr/2.3.4/lib64/libmpi.so.12 (0x0000200000130000)
    libcuda.so.1 => /lib64/libcuda.so.1 (0x0000200001060000)
    libcudart.so.10.2 => /usr/local/cuda-10.2/lib64/libcudart.so.10.2 (0x00002000020d0000)
...

$ ibrun singularity exec -B /opt -B /usr/local/cuda-10.2/ --nv tacc-centos7-ppc64le-mvapich2.3-ib_latest
    .sif ./VPIC_Longhorn_Example
TACC: Starting up job 78875
TACC: Setting up parallel environment for MVAPICH2+mpispawn.
TACC: Starting parallel tasks...
Unable to load GDRCOPY Library "/usr/lib64/libgdrapi.so" (/usr/lib64/libgdrapi.so: cannot open shared
    object file: No such file or directory)
```

You found a library not listed in ldd. That is ok because it tells us exactly where it is. However, the problem is that there is already a /usr/lib64 directory within the container, and you don't have write access. Let's mount that bind mount that file, instead of the directory, with -B.

```
$ ibrun -n 8 singularity exec -B /opt -B /usr/local/cuda-10.2/ -B /usr/lib64/libgdrapi.so --nv tacc-
    centos7-ppc64le-mvapich2.3-ib_latest.sif ./VPIC_Longhorn_Example
TACC: Starting up job 78875
TACC: Setting up parallel environment for MVAPICH2+mpispawn.
TACC: Starting parallel tasks...
...

...
Task completed successfully
```

Now you have a container running an application that is fully GPU accelerated with mvapich-gdr on a Power 9 platform.

3 File Systems at TACC

As you have seen it is incredibly easy to mount existing host directories in containers. You may have also noticed that your scratch, work and home directory are also automatically mounted within the container. This is called an underlay and it is configured in the tacc-singularity module. The underlay mounts all shared file systems for any given cluster. For many uses of containers these directories are sufficient to run, however, lets say you have a particularly picky piece of software who's dependencies are not easily installed in your scratch or work directory.

3.1 Writing Containers

At this point you may want to be able to write to the container, but the container images are read-only. The method we suggest at TACC is to use the -w flag and that will allow you to write anywhere in the container that your user has permissions using an overlay. An overlay is a file in which changes to the container are recorded, and when used with read-only image creates a new and modifiable file system.
For instance if there is a directory in the container owned by your user, you can modify it. Most likely there is not a directory you can write to unless you modified a dockerfile or are using an overlay. So in order to make this useful you will need superuser (root) permissions. Once you run singularity as root with the -w flag and the overlay you will be able to install software or make any changes to the container as you see fit. Unfortunately TACC does not provide root access on any machine so you will need to bring the singularity image somewhere where you have root access, modify it and copy it back to a TACC machine. This is currently the best method for modifying containers.

For this example, let's install vim. First, at TACC, you confirm vim is not found.

```
$ singularity exec tacc-centos7-ppc64le-mvapich2.3-ib_latest.sif which vim
which: no vim in (...)
```

Then on a system you have root on (not TACC) copy the image over, create a 500M overlay and embed it into the singularity image.

```
# Create a file for the image
# dd if=/dev/zero of=overlay.img bs=1M count=500
500+0 records in
```

```
500+0 records out
524288000 bytes (524 MB) copied, 0.233896 s, 2.2 GB/s

# # Format the image
# mkfs.ext3 overlay.img
mke2fs 1.42.9 (28-Dec-2013)
overlay.img is not a block special device.
Proceed anyway? (y,n) y
...

...
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

# #Embed the overlay
# singularity sif add --datatype 4 --partfs 2 --parttype 4 --partarch 2 --groupid 1 tacc-centos7-ppc64le
    -mvapich2.3-ib_latest.sif overlay.img
```

Now, still on the remote machine, lets install vim

```
# singularity exec -w tacc-centos7-ppc64le-mvapich2.3-ib_latest.sif yum install vim
Loaded plugins: fastestmirror, ovl
Determining fastest mirrors
 * base: mirror.dal.nexril.net
...

...
Installed:
  vim-enhanced.ppc64le 2:7.4.629-8.el7_9

Dependency Installed:
  gpm-libs.ppc64le 0:1.20.7-6.el7 vim-common.ppc64le 2:7.4.629-8.el7_9 vim-filesystem.ppc64le
      2:7.4.629-8.el7_9

Complete!
```

Now lets copy it back and try it on TACC's system as your normal user.

```
$ singularity exec tacc-centos7-ppc64le-mvapich2.3-ib_latest.sif which vim
/bin/vim
```

The overlay file you created is now embedded in the image, you can see the image has increased in size by 500M. The software is installed and persistent. We hope to improve this process in the future.