SLURM: Seamless Integration With Unprivileged Containers

Felix Abecassis, Julie Bernauer, Jonathan Calmels,
Louis Capps, Michael Knox, Luke Yeager

Agenda

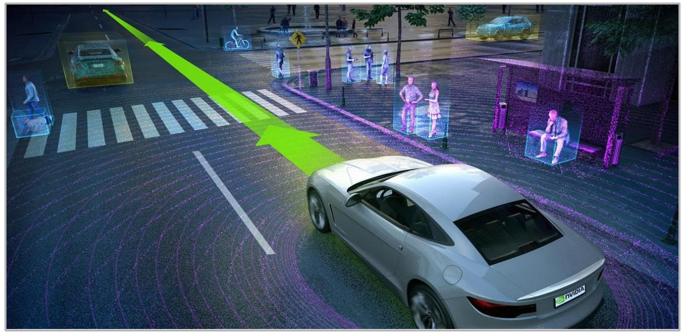
1. HPC, DL, and Containers at NVIDIA
2. We built a new “container runtime”
3. We wrote a SLURM plugin for it

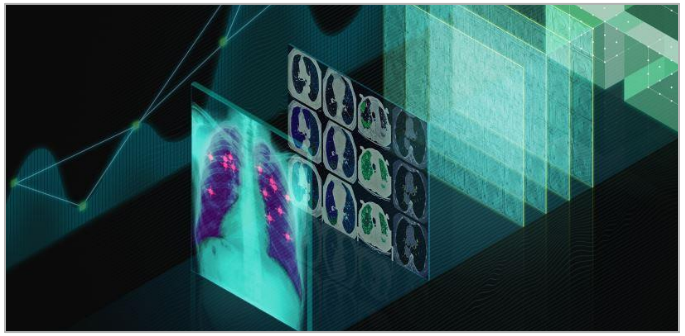
HPC and Deep Learning at NVIDIA

a.k.a. “Data Science”

Our users’ workloads aren’t typical HPC workloads.

- Many applications don’t use MPI at all. Even those that do generally only use it for initial bootstrapping.
- Peer-to-peer GPU access is critical.
- We run continuous integration (CI) on our HPC clusters.

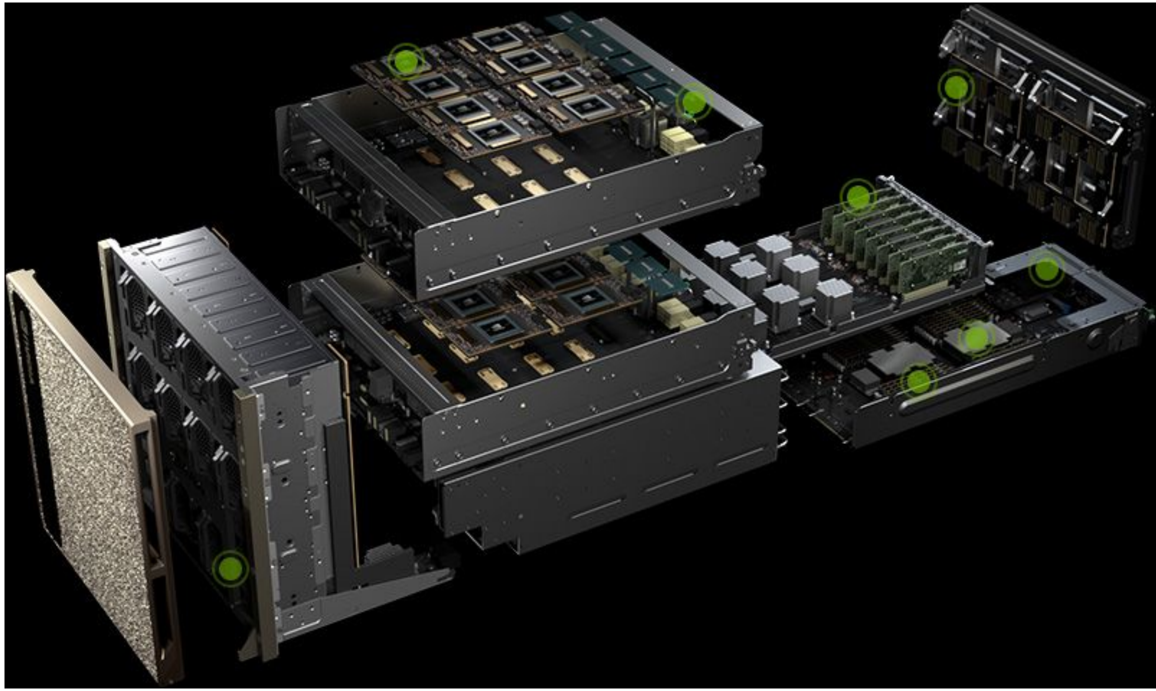




Infrastructure at NVIDIA

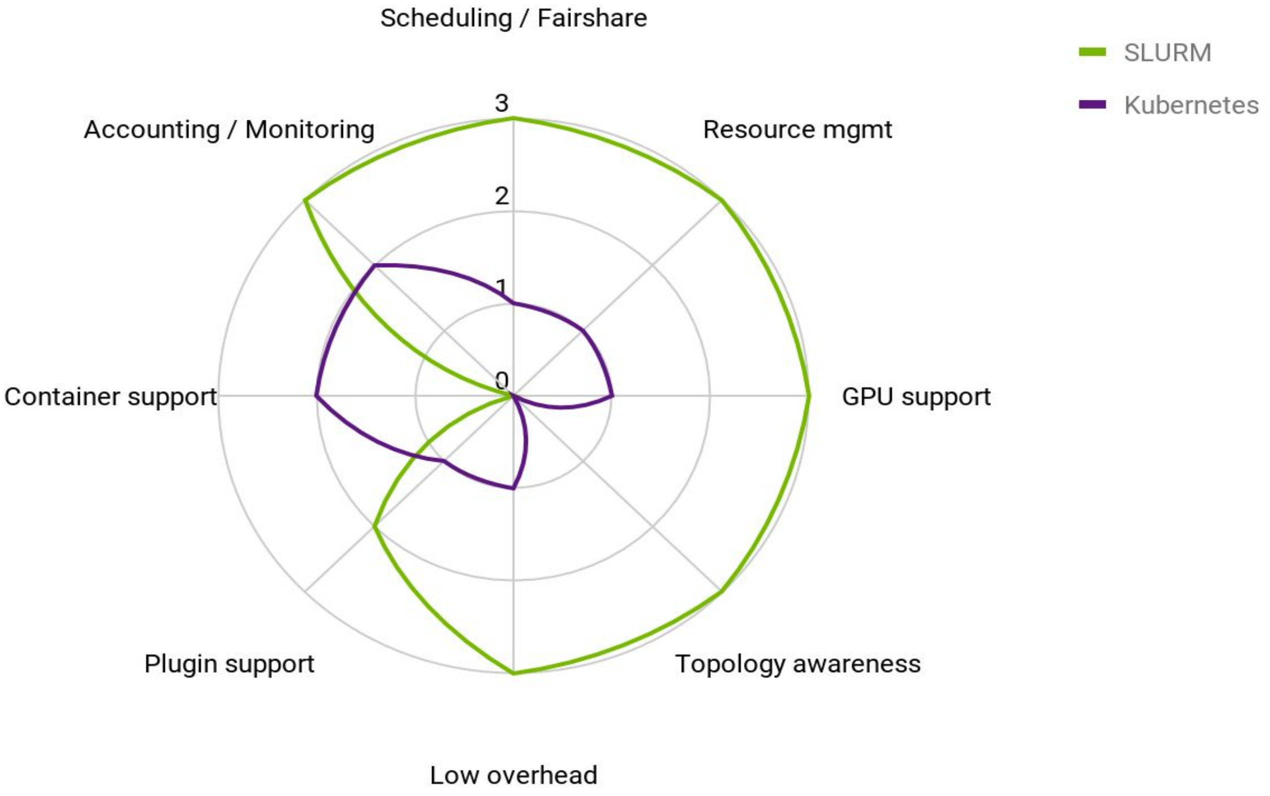
Circe, aka DGX SuperPOD (Top500 #22)

- 96 DGX-2H’s
- 1,536 Volta GPUs
- 144TB system memory
- 49TB GPU memory
- 10 Mellanox cx5’s in each machine
- Mellanox Infiniband EDR, non-blocking by rail, 2:1 blocking at top level



SLURM vs Kubernetes

or, “HPC” vs “Data Science”



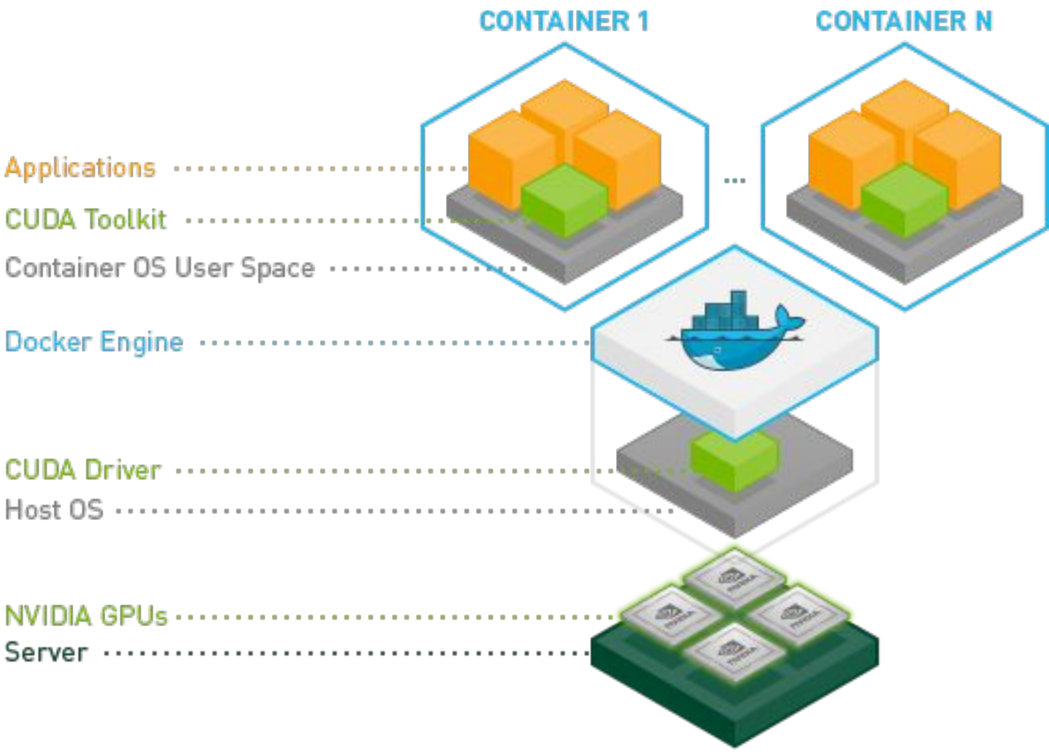
NGC Containers

We built libnvidia-container to make it easy to run CUDA applications inside containers.

We release optimized container images for each of the major DL frameworks every month, and provide them for anyone to use.

We use containers for everything on our HPC clusters - R&D, official benchmarks, etc.

Containers give us portable software stacks without sacrificing performance.



Example

SLURM+Docker+MPI

Excerpts from an actual script used to launch jobs for the MLPerf v0.5 benchmark (208 LOC total)

1. Setup docker flags
2. Setup mpirun flags
3. Setup SSH
4. Start sleep containers
5. Launch mpirun in rank0 container

```bash
#!/bin/bash

## Docker params
export VOLS="-v $DATADIR:/data -v $LOGDIR:/results"
export CONTNAME="mpi_${SLURM_JOB_ID}"
export DOCKEREXEC="nvidia-docker run --rm --net=host --uts=host --ipc=host --ulimit stack=67108864 --ulimit
memlock=-1 --security-opt seccomp=unconfined  $IBDEVICES"

MPICMD="mpirun --allow-run-as-root --tag-output --bind-to none -x SLURM_NTASKS_PER_NODE=$SLURM_NTASKS_PER_NODE -x
GPUS=$GPUS -x BATCHSIZE=$BATCHSIZE -x KVSTORE=$KVSTORE -x LR=$LR -x WARMUP_EPOCHS=$WARMUP_EPOCHS -x
EVAL_OFFSET=$EVAL_OFFSET -x DGXSYSTEM=$DGXSYSTEM ./run_and_time.sh"

MASTER_IP=`getent hosts \`hostname\` | cut -d ' ' -f1`
export hosts=( `scontrol show hostname |tr "\n" " "` )
unique_hosts=( $(echo "${hosts[@]}" | tr ' ' '\n' | sort -u | tr '\n' ' ' ) )
export MASTER_HOST=${hosts[0]}

VARS="-e OMPI_MCA_mca_base_param_files=/dev/shm/mpi/${SLURM_JOB_ID}/mca_params.conf -e GPUS -e BATCHSIZE -e KVSTORE
-e LR -e WARMUP_EPOCHS -e EVAL_OFFSET -e CONT -e DGXSYSTEM=$DGXSYSTEM -e MASTER_HOST -e MASTER_IP -e
SLURM_JOB_NUM_NODES -e SLURM_NNODES -e SLURM_NTASKS_PER_NODE "

docker pull $CONT

mkdir -p ${HOME}/.ssh/sbatch/${SLURM_JOB_ID}
ssh-keygen -t rsa -b 2048 -n "" -f "${HOME}/.ssh/sbatch/${SLURM_JOB_ID}/sshkey.rsa" -C "mxnet_${SLURM_JOB_ID}_"
&>/dev/null
echo command=\"/dev/shm/mpi/${SLURM_JOB_ID}/sshentry.sh\",no-port-forwarding,no-agent-forwarding,no-X11-forwarding
$(cat ${HOME}/.ssh/sbatch/${SLURM_JOB_ID}/sshkey.rsa.pub) >> ${HOME}/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
srun  -n $SLURM_JOB_NUM_NODES --ntasks-per-node=1 bash -c "mkdir -p /dev/shm/mpi/${SLURM_JOB_ID}; cp -R ${HOME}/.ssh
/sbatch/${SLURM_JOB_ID} /dev/shm/mpi; chmod 700 /dev/shm/mpi/${SLURM_JOB_ID}"
sleep 2

# Create mpi config file
srun  -n $SLURM_JOB_NUM_NODES --ntasks-per-node=1 tee /dev/shm/mpi/${SLURM_JOB_ID}/mca_params.conf <<-EOF
    plm_rsh_agent = /usr/bin/ssh
    plm_rsh_args = -i /dev/shm/mpi/${SLURM_JOB_ID}/sshkey.rsa -oStrictHostKeyChecking=no
-oUserKnownHostsFile=/dev/null -oLogLevel=ERROR -l ${USER}
    orte_default_hostfile = /dev/shm/mpi/${SLURM_JOB_ID}/mpi_hosts
    btl_openib_warn_default_gid_prefix = 0
    mpi_warn_on_fork = 0
    allow_run_as_root = 1
EOF

# Create ssh helper script that transfers an ssh into a compute node into the running container on that node
srun -n $SLURM_JOB_NUM_NODES --ntasks-per-node=1 tee /dev/shm/mpi/${SLURM_JOB_ID}/sshentry.sh <<-EOF
    #!/bin/bash
    echo "::sshentry: entered \$(hostname)"
    [[ -f $CONTNAME ]] && "::worker container not found error" && exit 1
    echo "::sshentry: running \$SSH_ORIGINAL_COMMAND"
    exec docker exec $CONTNAME /bin/bash -c "\$SSH_ORIGINAL_COMMAND"
EOF

# Create mpi hostlist
for h in ${hosts[@]}; do
   echo "$h slots=${SLURM_NTASKS_PER_NODE}" >> /dev/shm/mpi/${SLURM_JOB_ID}/mpi_hosts
done
srun -n $SLURM_JOB_NUM_NODES --ntasks-per-node=1 bash -c "cp $(which ssh) /dev/shm/mpi/${SLURM_JOB_ID}/.;  chmod 755
/dev/shm/mpi/${SLURM_JOB_ID}/mca_params.conf;  chmod 755 /dev/shm/mpi/${SLURM_JOB_ID}/sshentry.sh"

# Launch containers behind srun
srun  -n $SLURM_JOB_NUM_NODES --ntasks-per-node=1 $DOCKEREXEC --name $CONTNAME $VOLS $VARS $CONT bash -c 'sleep
infinity' & rv=$?
sleep 30

# Launching app
$(eval echo $SSH) docker exec $VARS $CONTNAME $MPICMD

# Clean up
docker rm -f $CONTNAME
```

Containers at NVIDIA

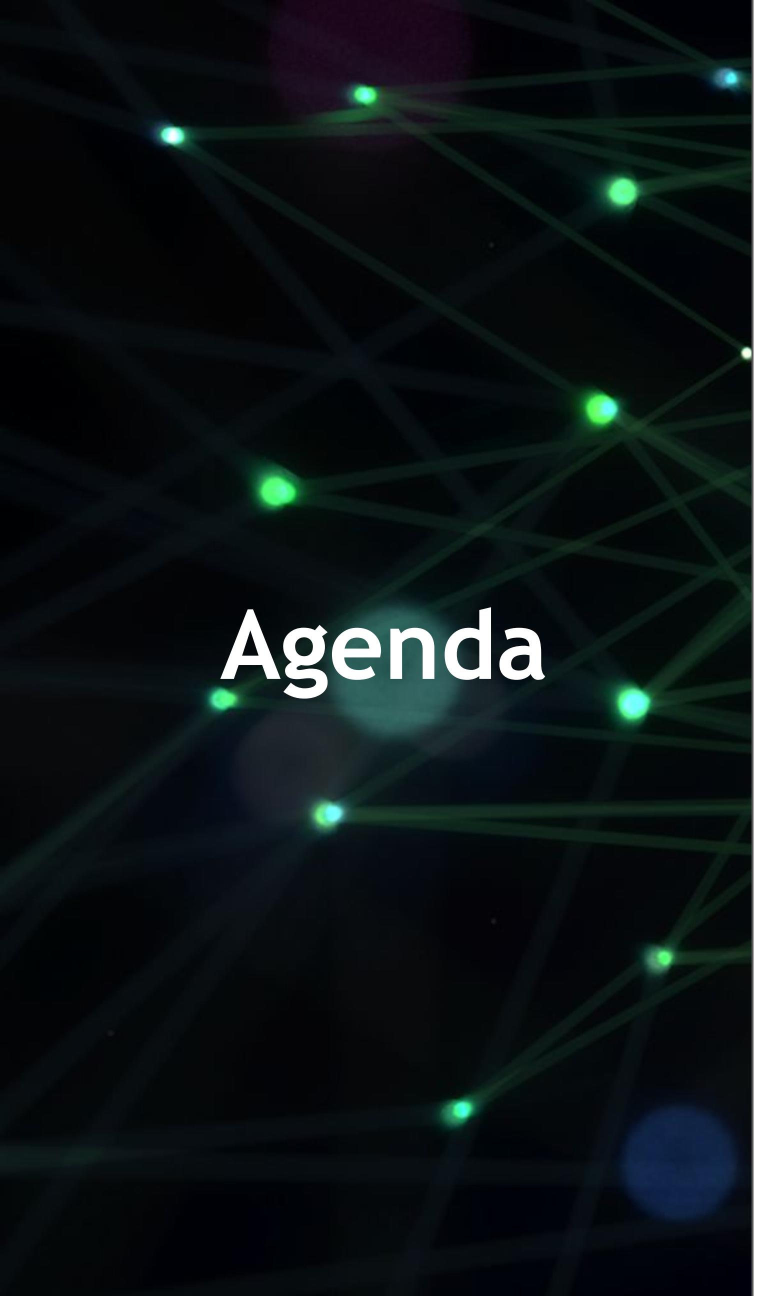
What do we need?

What we **need**

- High performance
- Unprivileged runtime
- Uses docker image format

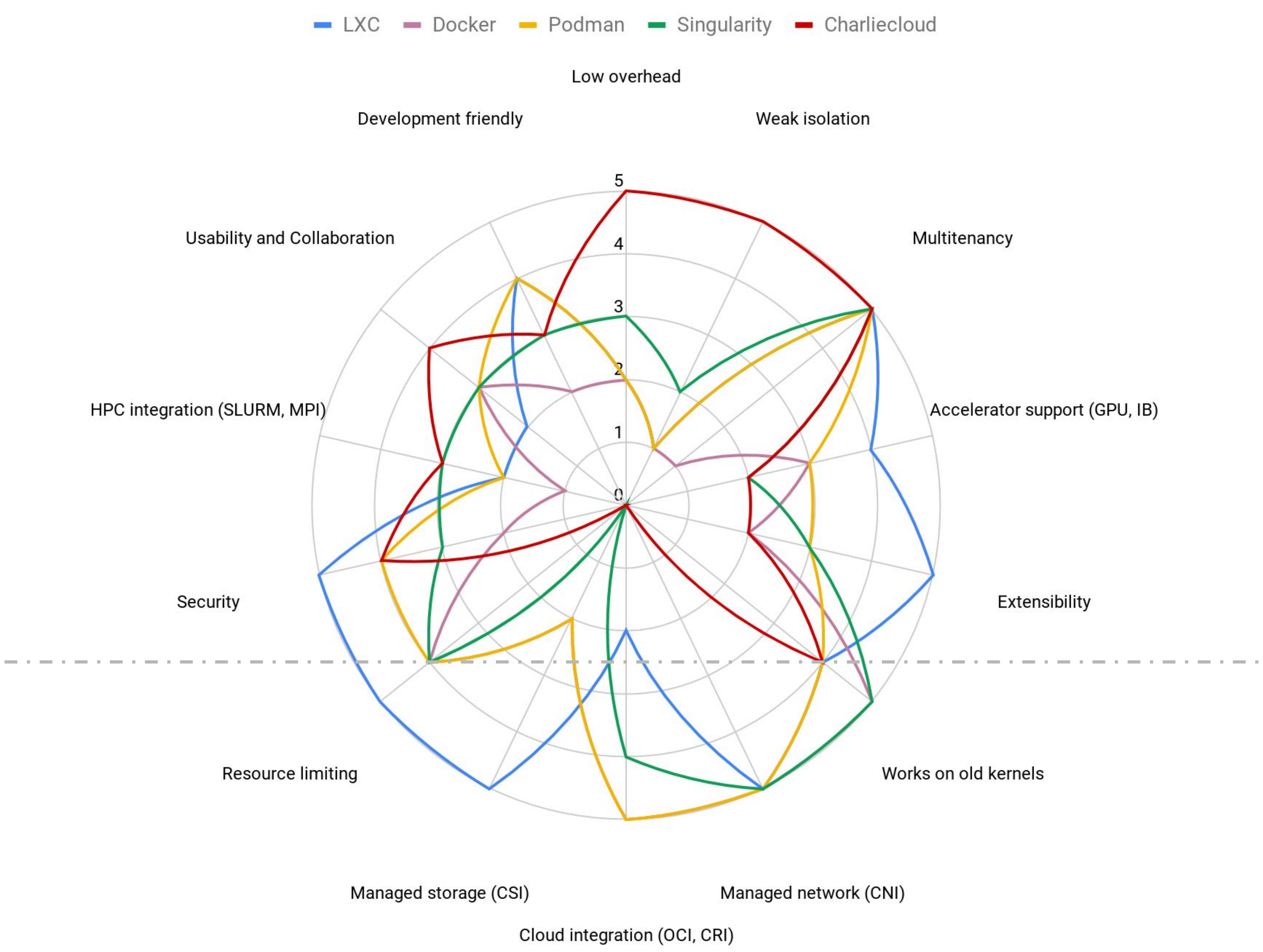
What we **want**

- Preserve SLURM cgroups
- NVIDIA+Mellanox devices are available by default
- MPI between containers is easy
- Can install packages inside containers



Agenda

1. HPC, DL, and Containers at NVIDIA
2. We built a new “container runtime”
3. We wrote a SLURM plugin for it



LXC
Docker
Podman
Singularity
Charliecloud
Low overhead
Weak isolation
Multitenancy
Accelerator support (GPU, IB)
Extensibility
Works on old kernels
Managed network (CNI)
Cloud integration (OCI, CRI)
Managed storage (CSI)
Resource limiting
Security
HPC integration (SLURM, MPI)
Usability and Collaboration
Development friendly
0
1
2
3
4
5


Enroot
Low overhead
Weak isolation
Multitenancy
Accelerator support (GPU, IB)
Extensibility
Works on old kernels
Managed network (CNI)
Cloud integration (OCI, CRI)
Managed storage (CSI)
Resource limiting
Security
HPC integration (SLURM, MPI)
Usability and Collaboration
Development friendly
5
4
3
2
1
0

ENROOT

Summary

Fully unprivileged “chroot” (with optional root-remapping)

Standalone (no daemon, no extra process)

Simple and easy to use (UNIX philosophy, KISS principle)

Little isolation, no overhead

Docker image support (5x pull speedup, shared cache)

Simple image format (single file + UNIX configs)

Composable and extensible (system/user configs, lifecycle hooks)

Advanced features (runfiles, scriptable configs, in-memory containers)

ENROOT

Basic usage

```
$ enroot import docker://nvcr.io#nvidia/tensorflow:19.08-py3
$ ls nvidia+tensorflow+19.08-py3.sqsh

$ enroot create --name tensorflow nvidia+tensorflow+19.08-py3.sqsh
$ ls -d ${XDG_DATA_PATH}/enroot/tensorflow

$ enroot start tensorflow nvidia-smi -L

$ enroot start --root --rw tensorflow apt update && apt install …

$ enroot bundle --output tensorflow.run nvidia+tensorflow+19.05-py3.sqsh
$ ./tensorflow.run python -c 'import tensorflow as tf; print(tf.__version__)'
```

ENROOT

Improved Linux utils

enroot-unshare	: like unshare(1), creates new namespaces
enroot-mount	: like mount(8), mounts filesystems
enroot-switchroot	: like switch_root(8), changes rootfs
enroot-aufs2ovlfs	: converts AUFS whiteouts to OverlayFS
enroot-mksquashovlfs	: like mksquashfs(1) on top of OverlayFS

ENROOT

“Container” from scratch

```
$ curl https://cdimage.ubuntu.com/[...]/ubuntu-base-16.04-core-amd64.tar.gz | tar -C ubuntu -xz

$ enroot-unshare bash

$ cat << EOF | enroot-mount --root ubuntu -
  ubuntu      /         none bind,rprivate
  /proc       /proc     none rbind
  /dev        /dev      none rbind
  /sys        /sys      none rbind
EOF

$ exec enroot-switchroot ubuntu bash
```

Agenda

1. HPC, DL, and Containers at NVIDIA
2. We built a new “container runtime”
3. We wrote a SLURM plugin for it

Pyxis

```
# run a command on a worker node
$ srun grep PRETTY /etc/os-release
PRETTY_NAME="Ubuntu 18.04.2 LTS"

# run the same command, but now inside of a container
$ srun --container-image=centos grep PRETTY /etc/os-release
PRETTY_NAME="CentOS Linux 7 (Core)"

# run inside the container, but mount the file from the host
$ srun --container-image=centos \
    --container-mounts=/etc/os-release:/etc/os-release \
    grep PRETTY /etc/os-release
PRETTY_NAME="Ubuntu 18.04.2 LTS"
```

Pyxis

Internals

1. slurm_spank_init()
   a. Add flags to srun

2. slurm_spank_user_init() - runs for each JOBSTEP
   a. Download a container image from a registry *(enroot import)*
   b. Unpack the image to a new container rootfs *(enroot create)*
   c. Start up a new "container" process *(enroot start)*
   d. Copy environment variables
   e. Save namespaces for later

3. slurm_spank_task_init() - runs for each TASK
   a. `setns(CLONE_NEWUSER) # join user namespace`
   b. `setns(CLONE_NEWNS)   # join mounts namespace`
   c. `chdir()`
   d. Setup PMIx, if active

Examples

Pyxis, MPI workload

```
srun -N4 --ntasks-per-node=1 --mpi=pmix \
   --container-image "${docker_image}" \
   --container-mounts "/raid/datasets/imagenet:/data,/scratch:/scratch" \
   caffe train --solver "/scratch/snikolaev/rn50/solver_idl_4k_mpi.prototxt" --gpu=all
```

Examples

Pyxis, MPI workload

```
srun -N4 --ntasks-per-node=1 --mpi=pmix \
   --container-image "${docker_image}" \
   --container-mounts "/raid/datasets/imagenet:/data,/scratch:/scratch" \
   caffe train --solver "/scratch/snikolaev/rn50/solver_idl_4k_mpi.prototxt" --gpu=all
```

1. No need to pass through environment variables (Pyxis inherits them all)
2. No need for any of these docker args: `--rm --net=host --uts=host --ipc=host --pid=host`
3. No need to configure mpirun (SLURM handles it)
4. No need to setup SSH (PMIx doesn't use it)

What Could Be Next

Allow pyxis to use a squashfile directly

Add pyxis flags to sbatch/salloc

Add backends for different “container runtimes”

Conclusion

1. We built a new container tool
    a. Unprivileged
    b. Lightweight, without excessive isolation
    c. Flexible plugins, including support for NVIDIA and Mellanox devices

2. We integrated it with SLURM
    a. Tasks seamlessly land inside containers
    b. MPI just works between containerized tasks

http://github.com/nvidia/enroot

http://github.com/nvidia/pyxis

Thanks to our coauthors: Felix Abecassis, Julie Bernauer, Louis Capps, Michael Knox

NVIDIA®

Supplementary Material

Pyxis

Enabling PMI2 and PMIx

PMI2 just works because we don't close any open file descriptors ($PMI_FD is still valid).

For PMIx:

1. Mount $PMIX_SERVER_TMPDIR inside the container
2. Make some MCA parameters available inside the container via envvars:

```
PMIX_MCA_ptl=PMIX_PTL_MODULE
PMIX_MCA_psec=PMIX_SECURITY_MODE
PMIX_MCA_gds=PMIX_GDS_MODULE
```