Containers in Slurm

Scott Hilton

Slurm User Group 2023

slurm | SCHEDMD

Contents

- Slurm in containers
- Slurm Container Support
  - sbatch/salloc/srun --container
- scrun
  - Container runtimes can submit jobs to to Slurm through scrun
    - Slurm is hidden to the user
  - Example with Docker
  - Example with Podman

Slurmd daemons can run in containers

- slurmctld and slurmdbd and mysql are already able to run in containers
- slurmd can run now run in a container (23.02)
  - Requires cgroup v2
  - Limited support in privileged mode with cgroup v1 (not ideal)

How Slurm can run containers: sbatch/salloc/srun --container

Slurm Container Support

- Added '--container' (21.08) support to the following:
  - srun
  - salloc
  - sbatch
- Added viewing job container [bundle path] (21.08) and container-id (23.02) to the following:
  - scontrol show jobs
  - scontrol show steps
  - sacct
    - If passed as part of the '--format' argument using "Container"
  - slurmd, slurmstepd, slurmdbd & slurmctld logs (too many places to list)

Slurm Container Support

srun example

```
$ srun --container=/tmp/centos grep ^NAME /etc/os-release
NAME="CentOS Linux"
```

salloc example

```
$ salloc --container=/tmp/centos grep ^NAME /etc/os-release
salloc: Granted job allocation 65
NAME="CentOS Linux"
salloc: Relinquishing job allocation 65
```

sbatch example

```
$ sbatch --container=/tmp/centos --wrap 'grep ^NAME /etc/os-release'
Submitted batch job 24419
$ cat slurm-24419.out
NAME="CentOS Linux"
```

Slurm Container Support

Example:

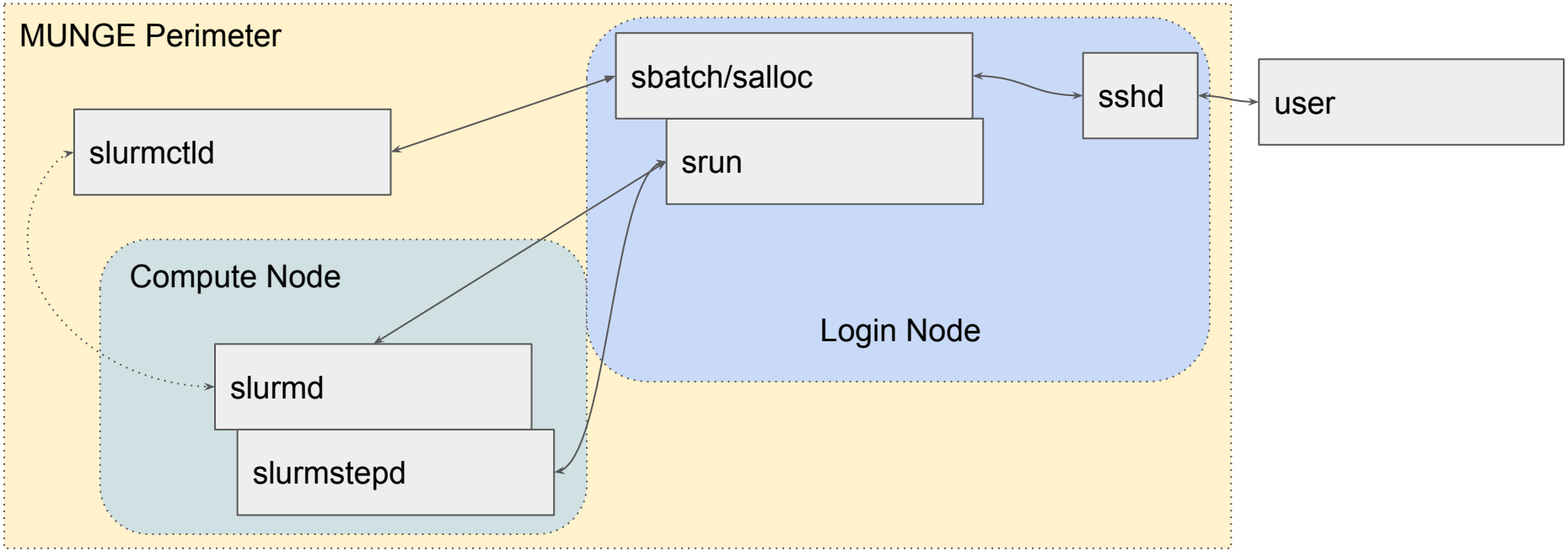
```
$srun --container ~/oci_images/alpine/ uptime
srun: job 13772 queued and waiting for resources
srun: job 13772 has been allocated resources
17:12:33 up 2 days, 20:27,  0 users,  load average: 1.78, 3.42, 3.44

$sacct --format=jobid,container%40
13772                /home/scott/oci_images/alpine/
13772.extern
13772.0              /home/scott/oci_images/alpine/

$scontrol show jobs | grep -i -E 'container|jobid'
JobId=13772 JobName=uptime
     Container=/home/scott/oci_images/alpine/ ContainerID=(null)
```

Slurm Container Support

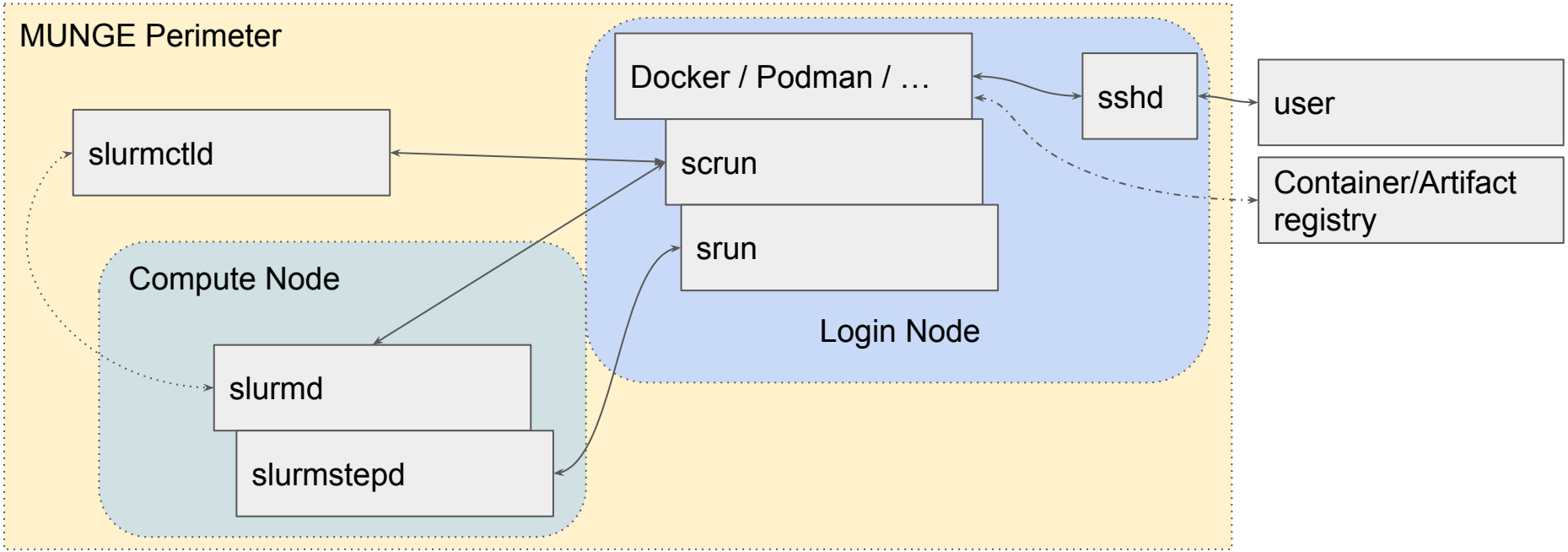
- If it works outside a container it should work inside a container
  - Slurm cgroups features apply to containers
  - Slurm only supports unprivileged containers currently
  - Containers must be able to function in an existing host network
- Per host configuration via ‘oci.conf’ in /etc/slurm/
  - slurm.schedmd.com/oci.conf.html
- Environment variables SLURM_CONTAINER (22.05) and SLURM_CONTAINER_ID (23.02) will always be set with a value (if present).

How container runtimes can submit jobs to Slurm: scrun

Using slurm user commands frontend



Using container runtime frontend through scrun



OCI runtime proxy - scrun

- scrun's goal is to make Slurm transparent to users running Docker/Podman
  - Users interface directly with OCI runtime clients (Docker, Podman, Podman-HPC ...)
  - Easier to onramp users already familiar with Docker/Podman
  - Site administrators will have to do setup and maintenance on the configuration
-

OCI runtime proxy - scrun

- Uses Slurm's existing infrastructure to run containers on compute nodes
- Allows users to work with the tools they want while running work on the Slurm cluster
- scrun is a new CLI command to join srun, sbatch and salloc, but no user should ever have to call it directly or even really need to be aware of it
  - scrun is still relatively new and we welcome tickets with requests for enhancements and especially bug reports
- Ends requirement that users manually prepare their images on compute nodes.
  - We recommend using a network file system
  - We also have the option to implement automatic staging out and in of containers
    - Lua
    - SPANK
- scrun is not meant to be called directly by users

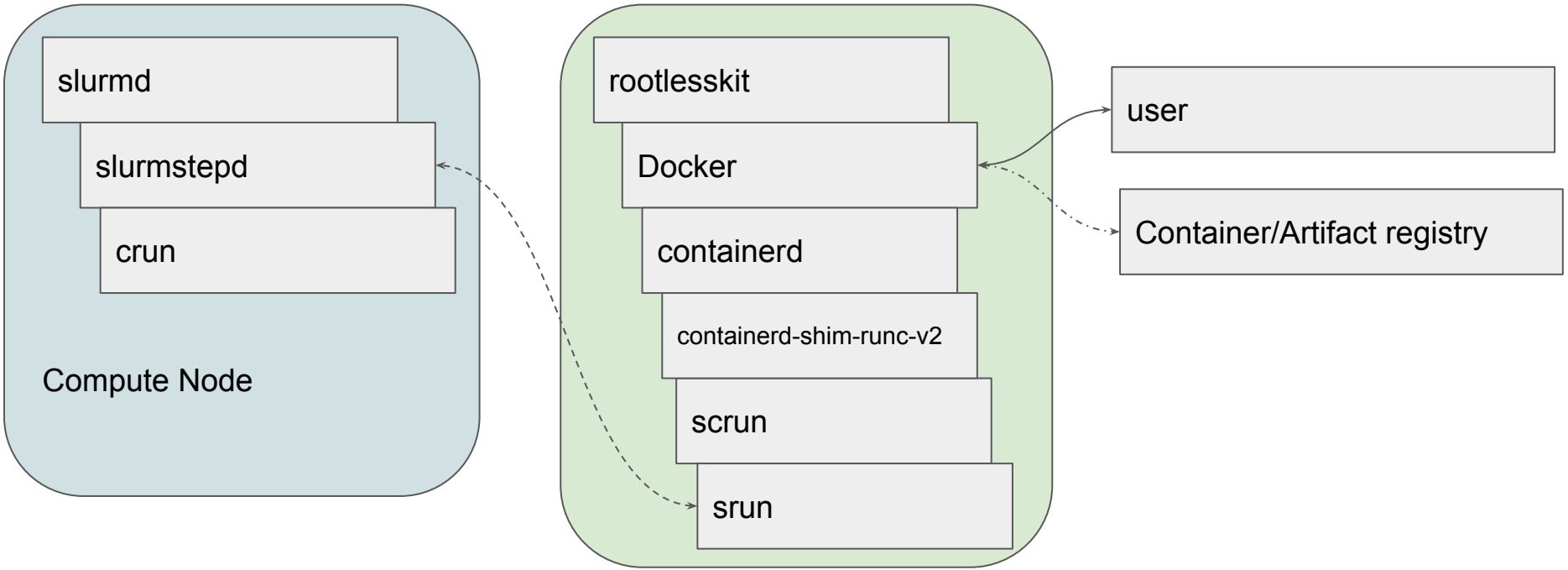
scrun via rootless Docker

scrun via rootless Docker (23.02)

Example:

```
$ export DOCKER_HOST=unix://$XDG_RUNTIME_DIR/docker.sock
$ export DOCKER_SECURITY=”--security-opt label:disable --security-opt
seccomp=unconfined  --security-opt apparmor=unconfined --net=none”
$ docker run $DOCKER_SECURITY -i ubuntu /bin/sh -c 'grep ^NAME /etc/os-release'
NAME="Ubuntu"
$ docker run $DOCKER_SECURITY -i centos /bin/sh -c 'grep ^NAME /etc/os-release'
NAME="CentOS Linux"
```

Rootless Docker Process Trees



Rootless Docker config (23.02)

~/.config/docker/daemon.json

```
{
  "default-runtime": "slurm",
  "runtimes": {
    "slurm": {
      "path": "/usr/local/slurm/sbin/scrun"
    }
  },
  "experimental": true,
  "iptables": false,
  "bridge": "none",
  "no-new-privileges": true,
  "rootless": true,
  "selinux-enabled": false
}
```

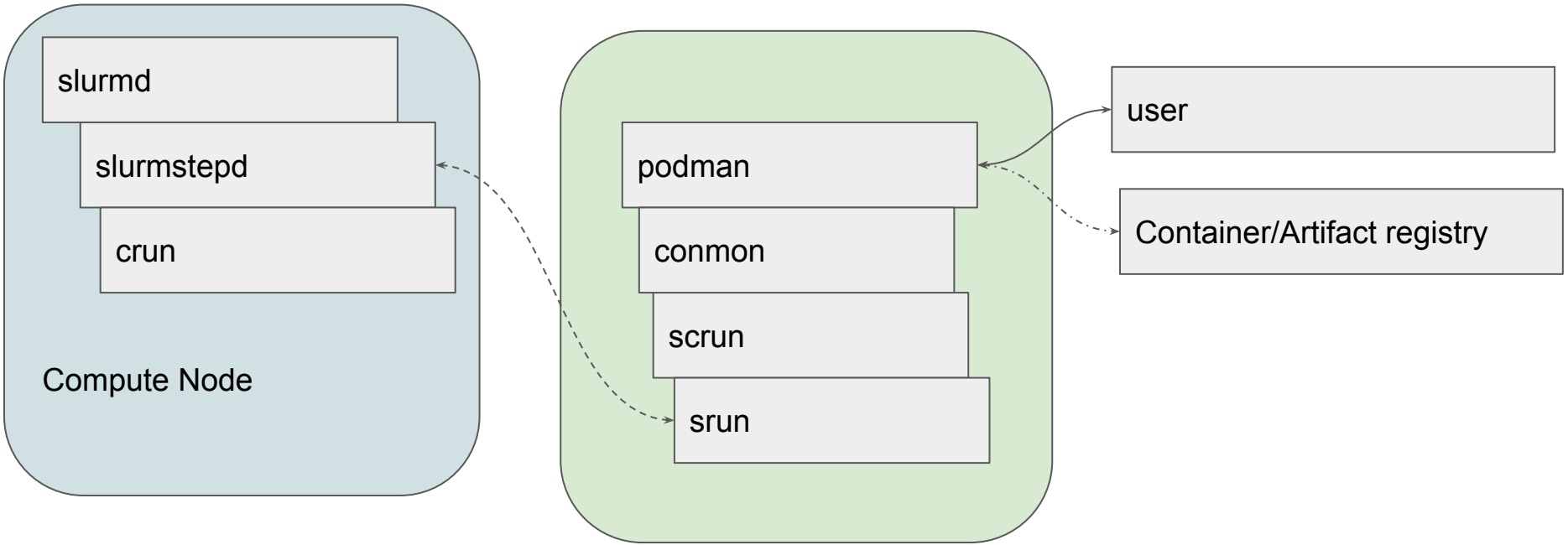
scrun via rootless Podman

scrun via rootless Podman (23.02)

example:

```
$ podman run ubuntu /bin/sh -c 'grep ^NAME /etc/os-release'
NAME="Ubuntu"
$ podman run centos /bin/sh -c 'grep ^NAME /etc/os-release'
NAME="CentOS Linux"
$ podman run centos /bin/sh -c 'printenv SLURM_JOB_ID'
77
$ podman run centos /bin/sh -c 'printenv SLURM_JOB_ID'
78
```

Podman Process Trees



Podman config for scrun (23.02)

~/.config/containers/containers.conf:

```
[containers]
apparmor_profile = "unconfined"
cgroupns = "host"
cgroups = "enabled"
default_sysctls = []
label = false
netns = "host"
no_hosts = true
pidns = "host"
utsns = "host"
userns = "host"
[engine]
runtime = "slurm"
runtime_supports_nocgroups = [ "slurm" ]
runtime_supports_json = [ "slurm" ]
remote = false

[engine.runtimes]
slurm = [ "/usr/local/slurm/sbin/scrun" ]
```

Questions?

- Documentation:
  - https://slurm.schedmd.com/containers.html


SCHEDMD
The Slurm Company

Appendix 1: Limitations of scrun

scrun - limitations (23.02)

- Annotations fully supported in scrun
  - Annotations not recorded in Slurm's accounting - will not show in sacct/sreport
  - Annotations not known to slurmctld - will not show with scontrol show jobs/steps
- No network namespaces support in slurm-23.02
  - All containers must run under host network
  - There are no design limitations in Slurm related to network namespaces. This is just functionality that has not been implemented.
    - Sites are welcome to submit RFEs to begin a discussion about adding it.
- Kubernetes operators and Kubernetes CNI support are also not implemented.
  - This is just functionality that has not been implemented.
    - Sites are welcome to submit RFEs to begin a discussion about adding it.

scrun - limitations (23.02)

- Rootless Docker/Podman on the login nodes should not have cgroup/apparmor/selinux support active
  - In most cases, these restrictions/systems will not work in rootless mode of Docker/Podman independent of Slurm. They just don't apply when using user namespaces.
  - All the containers are executed remotely making the local system's security systems irrelevant to the container. Slurm will enforce the existing limits as defined for the Slurm cluster to the containers being run under scrun.
  - If Docker/Podman are being run under any selinux tagging, then that tagging will be automatically inherited by scrun during the staging operations. The resultant jobs will need to have those tags applied by the existing selinux support in Slurm.
  - Podman allows easy configuration disablement while Docker requires command line arguments (at time of writing this slide).

scrun - limitations (23.02)

- No automatic resource selections implemented yet
  - Use of Slurm environment variables allow job property control
  - scrun will currently run the default job with default resources requested
- Container failures may require examining slurmd logs and/or syslogs to determine root cause
  - This appears to be a common issue with container orchestration systems and there are a few potential ways to handle this in future releases

scrun - limitations (23.02)

- Lua must either be compiled with JSON support or the library must be installed.
  - Slurm may need to be compiled after the JSON library is installed in Lua in order to be able to use it.
- scrun will not currently kill or stop the lua script while it is running.
  - If the Lua staging scripts hang, then the job time limit may be triggered and kill the job.
- scrun has the relevant SPANK and clifilter support.
  - These hooks are not a security device and any user may override them same as srun/sbatch/salloc.
  - scrun uses standard Slurm RPCs and user permissions. Any user may modify or ptrace their own processes. Any security must be applied at the controller.

scrun - limitations (23.02)

- One podman/docker instance per user per host
  - scrun does not provide information for jobs other than its own
    - Jobs will be visible via squeue/sacct/slurmrestd
  - docker / podman will be blind to any externally started containers
- MUNGE Authentication
  - scrun currently only works via MUNGE
  - Job submission host must have Slurm installed and be in MUNGE perimeter
- JWT Authentication
  - Not currently implemented
- Container IDs must be unique per user
  - Docker or Podman will hand the container ID to scrun verbatim.
  - scrun will try to search for the container by ID If the local anchor process is dead.

scrun - limitations (23.02)

- All existing limitations for running containers in Slurm still apply:
  - Containers must have a compatible version of Slurm installed to call Slurm commands
  - MUNGE's socket must be mounted in container to use MUNGE based authentication
  - JWT authentication is possible from container but there are no secrets functionality currently available.
    - Slurm does not support step controls/commands via JWT currently.
- User environment must be explicitly set
  - The environment at time of calling docker/podman will not be inherited by the container unless environment variables are supported by Docker/Podman.
  - Any environment variables must be set:
    - This can be done by setting env vars in the config.json in the container image.
    - Docker also supports '--env', '-e', and '--env-file' to set environment variables at runtime.

scrun - limitations (23.02)

- scrun will create a local process that must remain alive for the duration of the Job
  - If the local process is killed, then the job will be killed by Slurm. This is the same requirement as any job run via srun
  - scrun can be started from a batch job to avoid submission host uptime requirements
- Docker current uses an event and poll based system for determining if a container is alive
  - This may result in higher CPU usage on the host running Docker than only running a container directly via srun
  - This is part of the design of Docker and is independent of Slurm.
  - Sites that choose to rootless on login nodes will need to take care to account for the extra CPU usage and memory usage even if the users are not directly interacting with Docker at all times.

scrun - limitations (23.02)

- Builtin Docker/Podman logging is supported by scrun
  - scrun adds support for output of Docker JSON formatted log files
    - Docker JSON formatted log include meta data to determine the source of the log entry
    - Job output to stdout and stderr will annotated (and thus easily queried in Docker)
  - scrun's logs will be tagged as coming from the stdout source
    - Normal slurm commands will continue to print logs to stdout and stderr depending on the severity of the log.

scrun - limitations (23.02)

- scrun requires oci.conf to be fully configured
  - If oci.conf is not configured then scrun will refuse to run and there will be no container support.
- I/O restrictions and other limitations from the submission host will affect staging containers in and out
- Slurm has no control over Docker/Podman
  - Docker and podman will need to configured independently of Slurm
  - Only rootless Docker/Podman is supported
    - rootless docker has varying levels of support with older kernels
    - Sites are recommended to run the latest version of their distro and docker to avoid issues
  - Slurm (scrun) is run as one of the last steps of starting the container in Docker/Podman
- Not all functionality of Docker/Podman is implemented currently

scrun - limitations (23.02)

- Online image repositories exist independently of Slurm and may apply bandwidth or usage restrictions
  - These limitations can falsely imply scrun (and Slurm) being slow
  - Sites are suggested to set up local caching proxies if possible
  - scrun does not cache images
- scrun is not a security solution or antivirus or a new security layer
  - It does not scan or reason about the contents of the container images beyond enforcing basic OCI image formatting
  - It will push the images to the execution hosts where the configured and the OCI runtime in oci.conf will be executed to start the containers
  - Users are responsible to ensure the container images are following site policies and procedures while being free of malicious code

scrun - limitations (23.02)

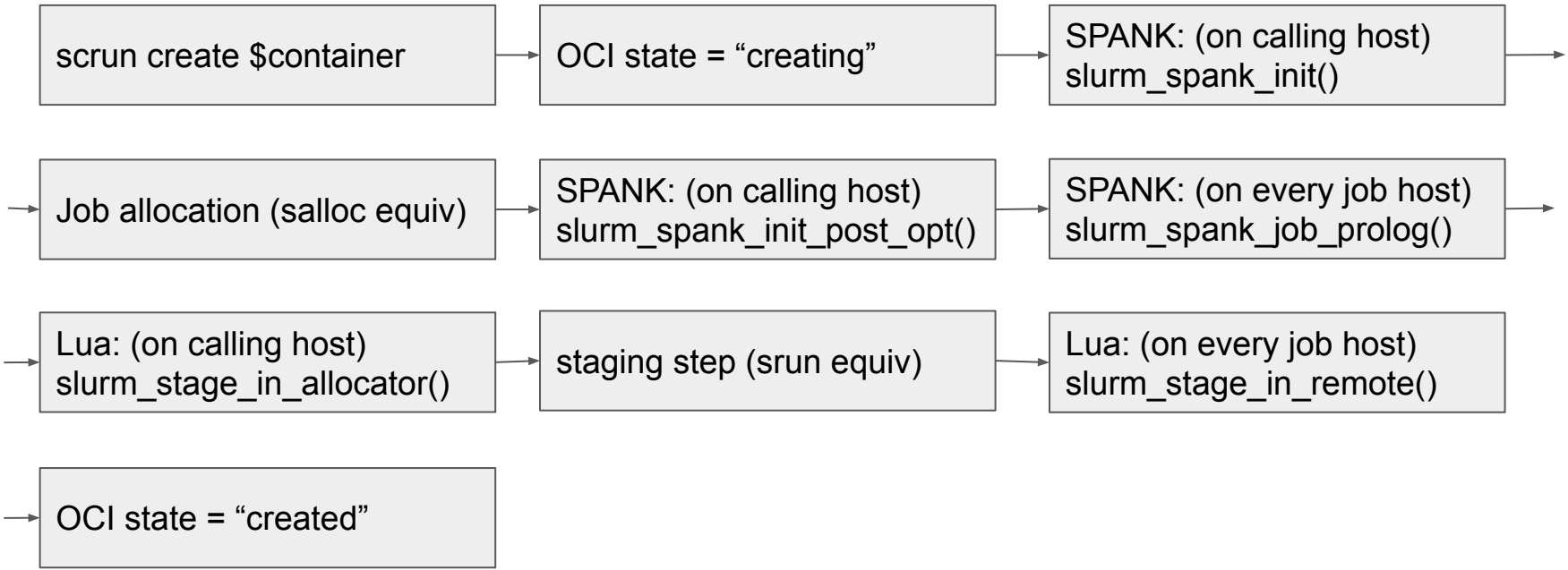
- scrun will only run under the POSIX user/group neither adding or removing abilities/capabilities/permissions from the user and therefore the container processes
- Sites must configure the stage in and stage out Lua scripts to clean up cached images
  - Failure to cleanup the images may result in massive wasteful usage of the filesystems.
- Sites must configure docker/podman to cleanup cached images independently of Slurm
  - Dockers build cache can get very large!
- In order to support `docker build` or `podman build` container staging in and out must be completed to apply the final changes back to the container's private mount space
  - This will only happen on a single compute node so there should not be issues cross-syncing changes from multiple nodes.
  - Lingering container image file trees will allow fewer sync calls but final cleanup must happen outside of scrun

scrun - limitations (23.02)

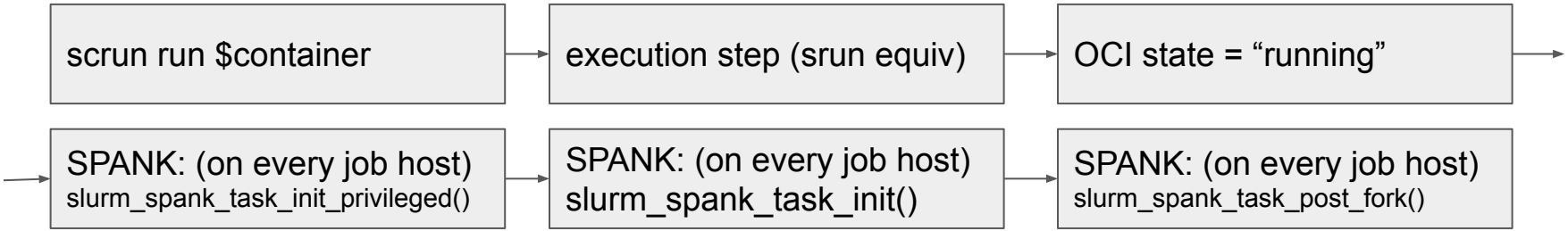
- scrun exec is not yet implemented

Appendix 2: Job lifecycle through scrun

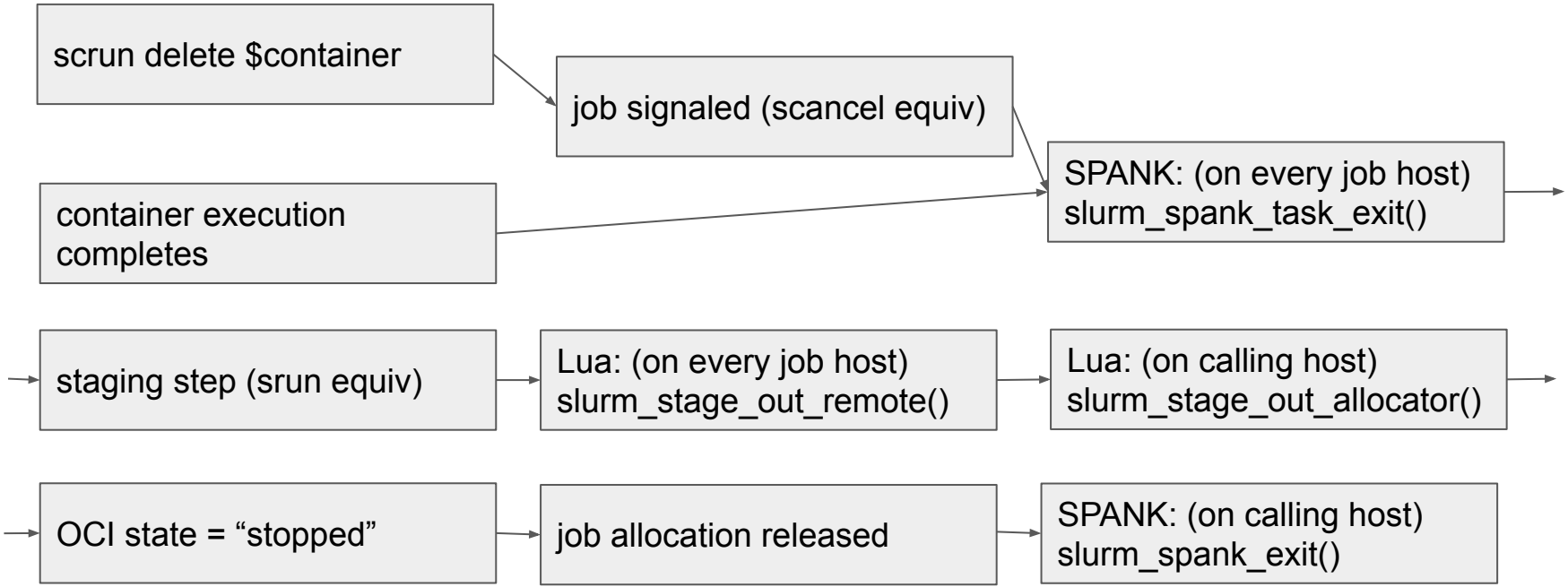
Container creation flow - scrun



Container run flow - scrun



Container cleanup flow - scrun



Appendix: 3 Container Staging for scrun

scrun - container staging

- scrun needs to stage in the image to remote host at startup
- scrun needs to stage out the image from remote host at job end
- Flexibility required as every site has a different shared file system configuration and data ingress and egress rules.
    - scrun avoids making as many assumptions about the request host vs the execution host in Slurm itself as possible.
    - Site admins must configure where and how images are staged.

scrun - container staging via Lua

- scrun's Lua staging plugin allows site to write custom and simple scripts to move the image to and back from the remote storage.
- scrun's staging lua script is located at:
  - /etc/slurm/scrun.lua
- Lua script runs as user avoiding any additional privilege escalation risk
- Lua already has JSON support via libraries
- Sites can write a native Slurm plugin if desired instead of using the Lua plugin.

scrun - Lua container stage in example

Simplified stage in (to common shared filesystem) hook:

```
function slurm_scrun_stage_in(id, bundle, spool_dir, config_file, job_id, user_id,
                              group_id, job_env)
        os.execute(string.format("/usr/bin/env rsync --numeric-ids --delete-after
                              --ignore-errors -a -- %s/ %s/", rootfs, dstfs))

        slurm.set_bundle_path(p)
        slurm.set_root_path(p.."rootfs")

        write_file(jc, json.encode(c))
        return slurm.SUCCESS
end
```

scrun - Lua container stage out example

Simplified stage out hook: (this example only deletes the container)

```
function slurm_scrun_stage_out(id, bundle, orig_bundle, root_path, orig_root_path,
                               spool_dir, config_file, jobid, user_id, group_id)
    os.execute("rm --one-file-system --preserve-root=all -rf "..bundle)
    return slurm.SUCCESS
end
```