The Good, Bad and Ugly of systemd

Craig Liddle
Craig.Liddle@suse.com

Alan Epps
Alan.Epps@boeing.com

Effective containment of HPC batch workloads on a shared node

Craig Liddle
Craig.Liddle@suse.com

Alan Epps
Alan.Epps@boeing.com

Agenda

- Welcome & Introductions
- What is systemd
- What is the Good
- What is the Bad & Ugly
- Use Cases
- How to protect the Kernel and core processes

systemd

The new system startup and service manager for Linux, replacing the “old” SysV init

Systemd – The Good

- Offers on-demand starting of daemons
- Uses socket and D-Bus activation for starting services
- Supports snapshotting and restoring of the system state
- Maintains mount and automount points
- Systemd daemons make it is easier to supervise and control processes and parallelized job execution.
- keeps track of processes using Linux cgroups
- systemd offers the systemctl command and cgroups to make your life easier:
- systemctl provides the administrator with more detailed error messages including runtime and start-up errors.
- cgroups, or "control groups", allow for the grouping of processes into a hierarchy for easier management.
- Process attributes such as function and ownership are much easier to ascertain. For example, under Systemd, sub-processes, once spawned, become 'children' and are organized under the appropriate 'parent' group to show inheritance.

Information Technology
& Data Analytics

Effective containment of HPC batch workloads on a shared node

Greg Siekas

Enterprise High Performance Computing Service

Containment is required

Needed the ability to enforce resource limits on HPC batch workloads on a shared node

What is driving this?

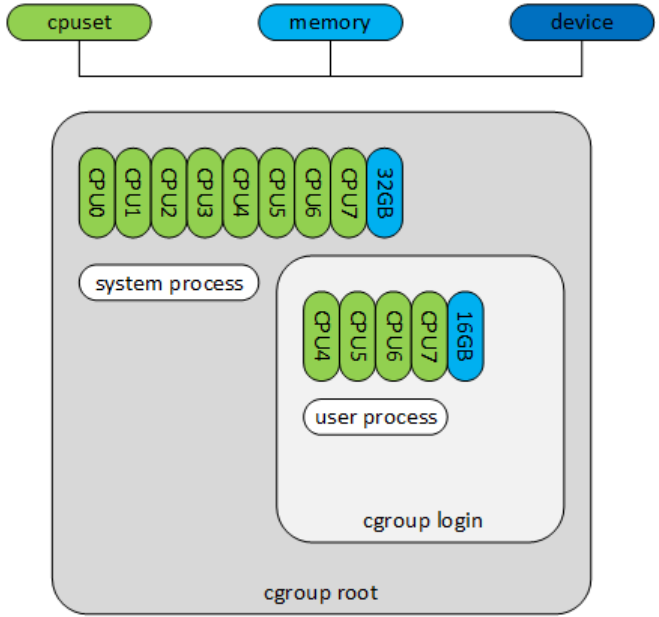
- **Increasing number of cores per node**
- **Do not want to leave idle resources**
- **Cannot afford to allow one job to interrupt another**

We have deployed a multilevel strategy to enforce resource limits under SLES11

SLES11 login node

User containment on login nodes to enforce shared limit on CPU and Memory resources

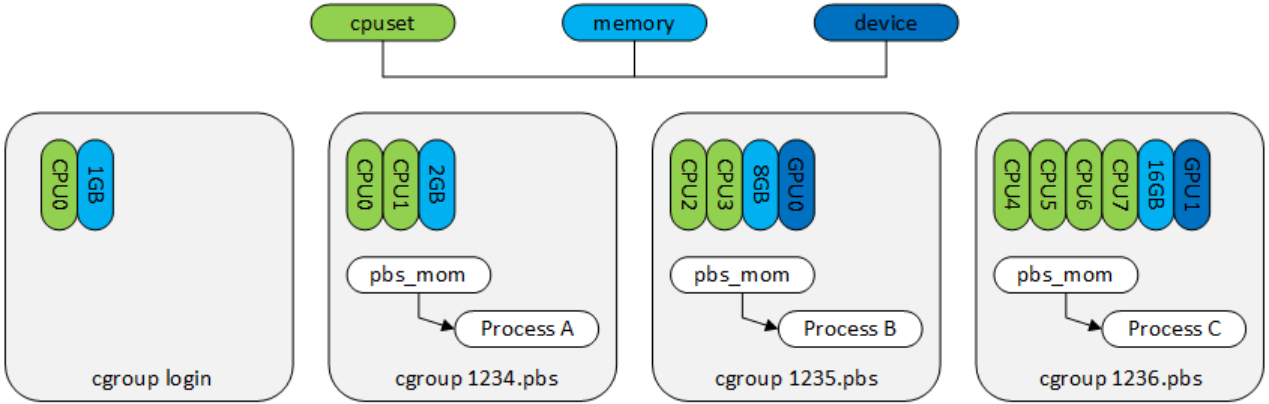
Limit users ability to interrupt login node



SLES11 compute node

Hardened enforcement of CPU, Memory, and GPU resources on batch compute nodes

Effective containment allows for sharing of batch compute resources



SLES11 Containment Strategy

Developers and end users can be sneaky or make mistakes

We cannot just place the batch job script inside the control group and hope for the best

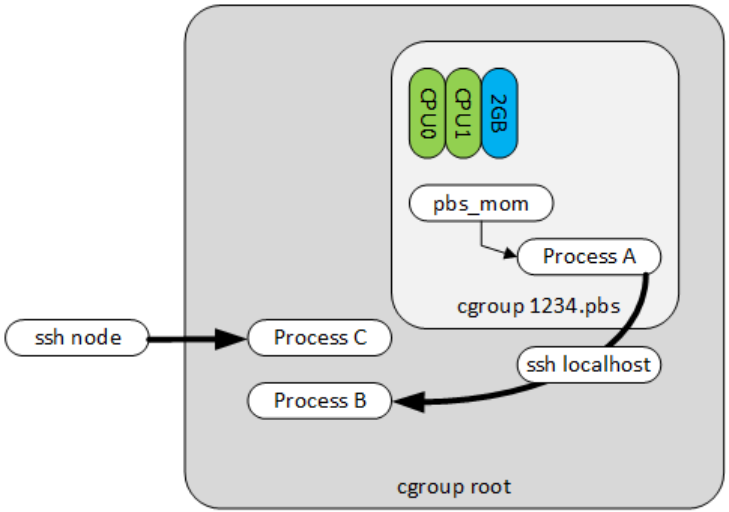
We have deployed a 3 level strategy to enforce resource limits on compute nodes

Level 1 – Simple does not work 100%

PBS prologue and epilogue to create and destroy control group with the requested resources

Works great until…
 # ssh localhost
A simple way to escape from control group!

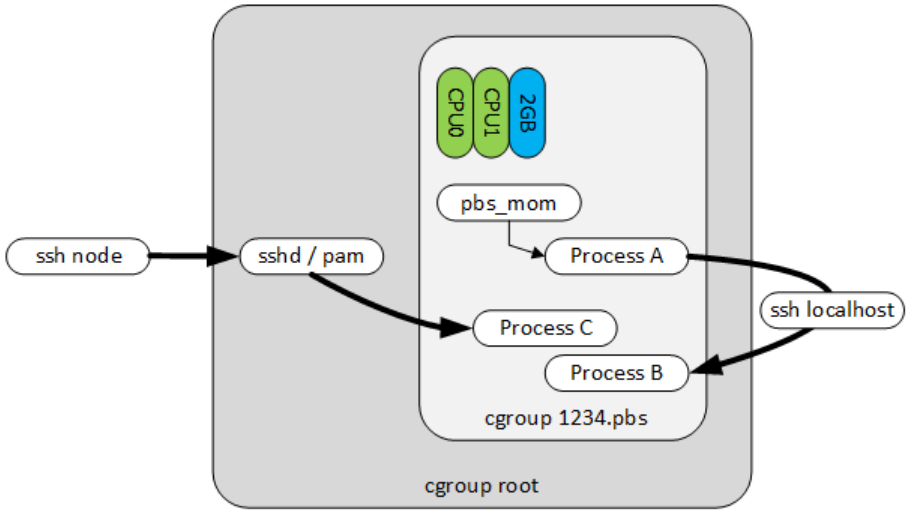
Any SSH access from any remote node does not place the process into the control group



Level 2 – PBS_JOBID and SSH

Modification to SSH configuration to send and accept PBS_JOBID environment variable
Pam module for limiting access to node only if user has a valid PBS job running
Profile.d script to use pbs_attach to make process of child of the pbs_mom

Now SSH access isn't a problem



Level 3 –libcgroup1 and cgrulesengd

Modification to libcgroup1 to use PBS_JOBID in cgrules.conf

user:1234.pbs cpuset,devices,memory 1234.pbs/

PBS prologue and epilogue create and destroy rules in cgrules.conf

Handles situation where pbs_attach does not work

We discovered a memory leak in libcgroup1!
Triggered by updating the cgrules.conf and restarting cgrulesengd multiple times (50k+)
Quickly fixed by SUSE

Modifications to libcgroup1

```
--- libcgroup-0.41.rc1-orig/src/api.c   2017-03-29 12:42:18.001875000 -0700
+++ libcgroup-0.41.rc1/src/api.c        2017-07-19 18:47:06.447814000 -0700
@@ -3905,6 +3905,56 @@
}

/**
+ * Get PBS_JOBID from /proc/<pid>/environ
+ */
+int cg_get_pbs_jobid_from_procfs(pid_t pid, char **procname)
+{
+
+       FILE *f;
+       int ret = ECGFAIL;
+       int c = 0;
+       int len = 0;
+       char path[FILENAME_MAX];
+       char buf[32768];
+
+       sprintf(path, "/proc/%d/environ", pid);
+       f = fopen(path, "re");
+       if (!f)
+              return ECGROUPNOTEXIST;
+
+       while (c != EOF) {
+              c = fgetc(f);
+              if ((c != EOF) && (c != '\0')) {
+                     buf[len] = c;
+                     len++;
+                     /*
+                      * Avoid overflowing due to long environment variable settings
+                      */
+                     if (len >= 32768) {
+                            ret = ECGFAIL;
+                            break;
+                     }
+                     continue;
+              }
+              buf[len] = '\0';
+              if (!strncmp(buf, "PBS_JOBID=",10)) {
+                     *procname = strdup(buf + strlen("PBS_JOBID="));
+                     if (*procname == NULL) {
+                            last_errno = errno;
+                            ret = ECGOTHER;
+                            break;
+                     }
+                     ret = 0;
+                     break;
+              }
+              len = 0;
+       }
+       fclose(f);
+       return ret;
+}
+
+
+/**
 * Get process name from /proc/<pid>/status file.
 * @param pid: The process id
 * @param pname_status : The process name
@@ -4037,6 +4087,7 @@
       int ret;
       char *pname_status;
       char *pname_cmdline;
+       char *pname_env;
       char path[FILENAME_MAX];
       char buf[FILENAME_MAX];

@@ -4045,6 +4096,16 @@
              return ret;

       /*
+        * Get the PBS_JOBID from /proc/<pid>/environ
+        */
+       ret = cg_get_pbs_jobid_from_procfs(pid, &pname_env);
+       if (!ret) {
+              free(pname_status);
+              *procname = pname_env;
+              return 0;
+       }
+
+       /*
        * Get the full patch of process name from /proc/<pid>/exe.
        */
       memset(buf, '\0', sizeof(buf));
```

SLES11 Containment Strategy

All processes are spawned from either PBS or SSH

Uses PBS_JOBID as the key for control group name and rules

We now have a shared multi-node job containment strategy that works!

SLES12 – systemd breaks everything!

With the introduction of SLES12 and systemd we hit a few road blocks…

libcgroup1 is now obsolete and conflicts with systemd!
https://www.suse.com/support/kb/doc/?id=7018741

HELP!

Lots of questions for Craig and SUSE support

Reviewed the SLES12 manuals and information on systemd to find a solution

SLES12 – Simple solution

Simple solution was to modify /etc/systemd/system.conf

JoinControllers=cpu,cpuacct,freezer,cpuset,devices,memory

Collapses the control group directory structure which allows cgrulesengd to work

From the cgrules.conf man page:

First rule which matches the criteria will be executed

Allows our SLES11 functionality to work on SLES12 without modification

How to deploy in your environment

https://doc.opensuse.org/documentation/leap/tuning/html/book.sle.tuning/cha.tuning.cgroups.html

https://www.suse.com/documentation/sled-12/book_sle_admin/data/cha_systemd.html

https://www.suse.com/documentation/sled-12/book_sle_tuning/data/sec_tuning_cgroups_subsys.html

Thank you for joining us today!