Migrating to Slurm from Grid Engine: Politics, Partitions and Problems

Murray Collier

diamond

Background

- Two clusters: Science & Hamilton
- 15,816 cores 404 GPUs (K80, P100, V100, A100
- RHEL 7
- Univa Grid Engine
- Config management via CFEngine
- Use cases
  - Accelerator simulation
  - Initial, “live” processing
  - Post-processing
  - Ad hoc usage

Why change?

- Financial – Grid Engine costly licensing
- Familiarity and community
- API – Kubernetes submission
- Flexibility
- Interoperability
- Cloudbursting submission to STFC Cloud

Hopper

- Hardware verification platform to evaluate workloads on future hardware and software
- 4 Node cluster
  - 1 CPU Node (76 cores, 9GB RAM/core)
  - 3 A100 GPU Nodes (144 cores, 10GB RAM/core)
- First cluster at DLS to use Slurm/RHEL8
- Used prebuilt RedHat provided Slurm RPMs
- Built on a standard DLS RHEL8 PXE/kickstart build (approx. 1600 base packages)
- Was configured in part using CFEngine and by hand
- Various user groups tested workloads on it, this informed the decision to go ahead with the full migration of the Hamilton cluster
- The hardware is due to be integrated into the Wilson cluster imminently

RHEL 8 Upgrade

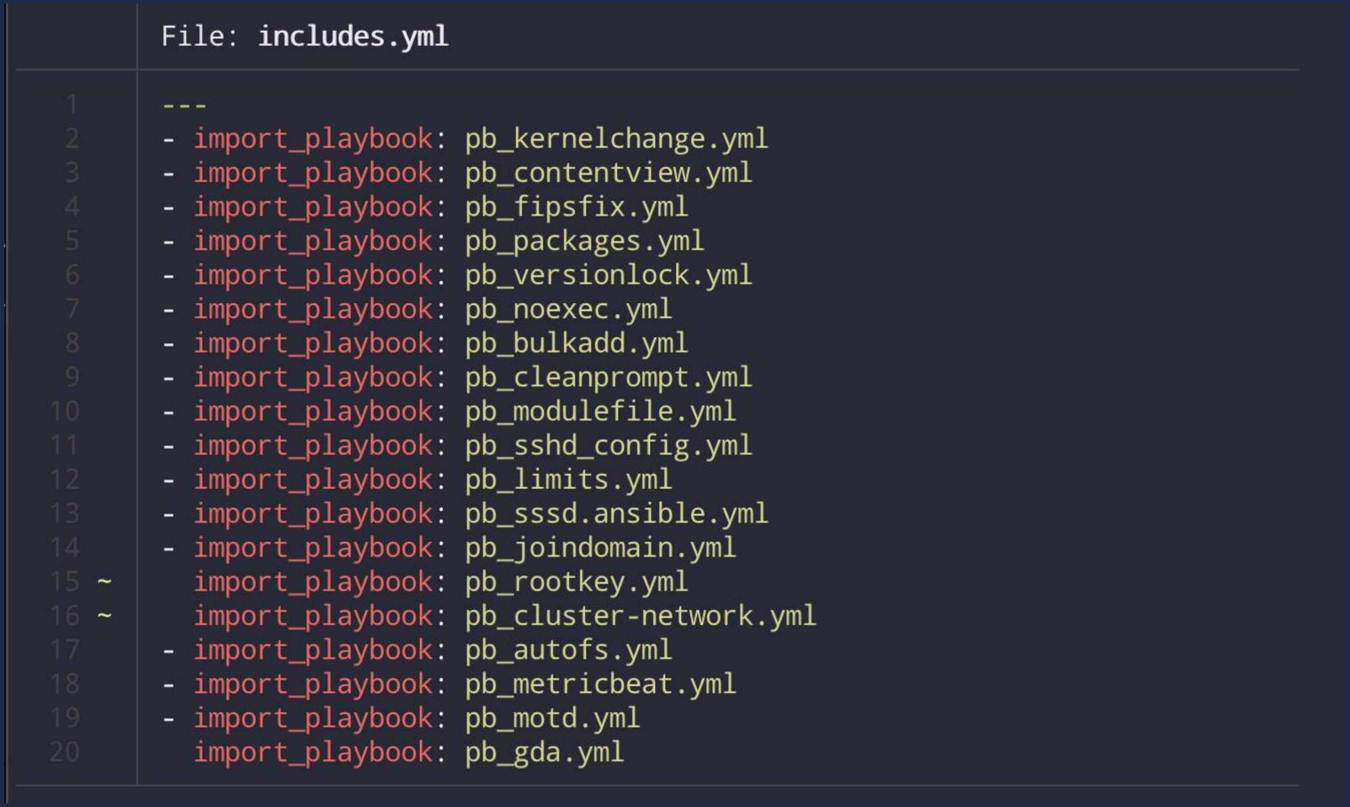
- The rest of DLS was undergoing a RHEL7 -> RHEL8 migration program, it made sense to take the opportunity to upgrade the HPC nodes
- Very minimal quick PXE/kickstart build (~500 packages, minimal requirement to run ansible against).
- Updated firmware needed on Mellanox NICs
- Lots of work done with stakeholders to confirm that their software, scripts and modules were working on RHEL8/Slurm.

Switch to Ansible config management

- Why Ansible?
  - Widely used, excellent community resources and documentation
  - Existing knowledge in the team to tap into
  - Agentless
  - Straightforward to learn
  - FOSS
  - Highly extensible
  - Speed and agility
  - Idempotency (if correctly understood and written)

Switch to Ansible config management

- Playbooks and roles - post build
  - Auth config
  - GPFS
  - Networking
  - NFS mount point
  - Autofs
  - Package Installation
  - Version/kernel locking

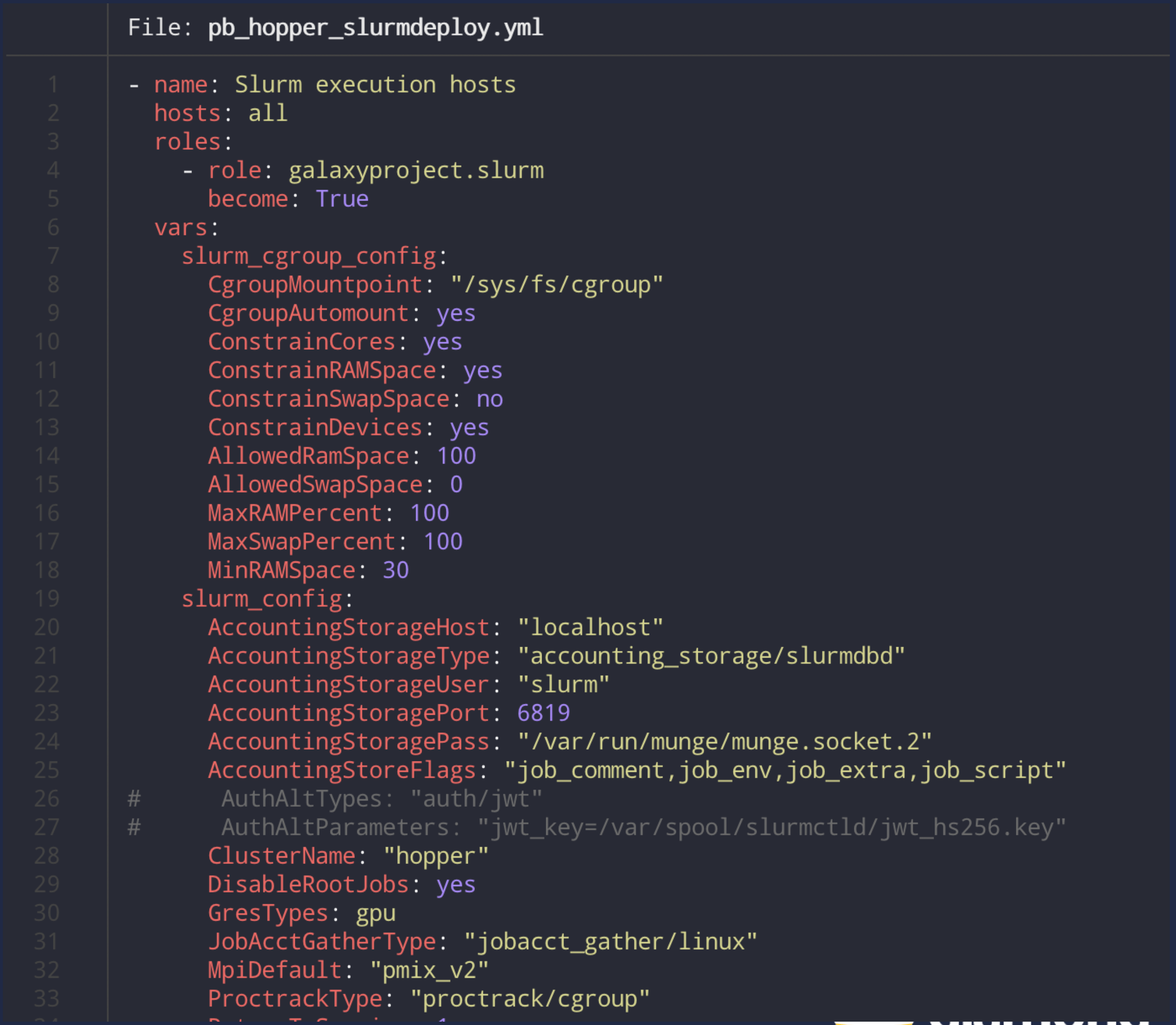


File: includes.yml

```
---
- import_playbook: pb_kernelchange.yml
- import_playbook: pb_contentview.yml
- import_playbook: pb_fipsfix.yml
- import_playbook: pb_packages.yml
- import_playbook: pb_versionlock.yml
- import_playbook: pb_noexec.yml
- import_playbook: pb_bulkadd.yml
- import_playbook: pb_cleanprompt.yml
- import_playbook: pb_modulefile.yml
- import_playbook: pb_sshd_config.yml
- import_playbook: pb_limits.yml
- import_playbook: pb_sssd.ansible.yml
- import_playbook: pb_joindomain.yml
  import_playbook: pb_rootkey.yml
  import_playbook: pb_cluster-network.yml
- import_playbook: pb_autofs.yml
- import_playbook: pb_metricbeat.yml
- import_playbook: pb_motd.yml
  import_playbook: pb_gda.yml
```

Slurm cluster using Ansible

- We're using Ansible now – there must be a way of deploying our Slurm cluster using it?
- Do we write our own or do we use a role from a well-respected project?
- Investigation was done and we decided on a Ansible galaxy role (https://github.com/galaxyproject/ansible-slurm)
- Define config in Ansible variables, define hosts in Ansible inventory, run the playbook. Voila! Working Slurm cluster
- Needed separate Ansible for configuring the DB for slurmdbd
- Overall impressions, very happy with it. Stable, reliable and has caused very few problems.



File: pb_hopper_slurmdeploy.yml

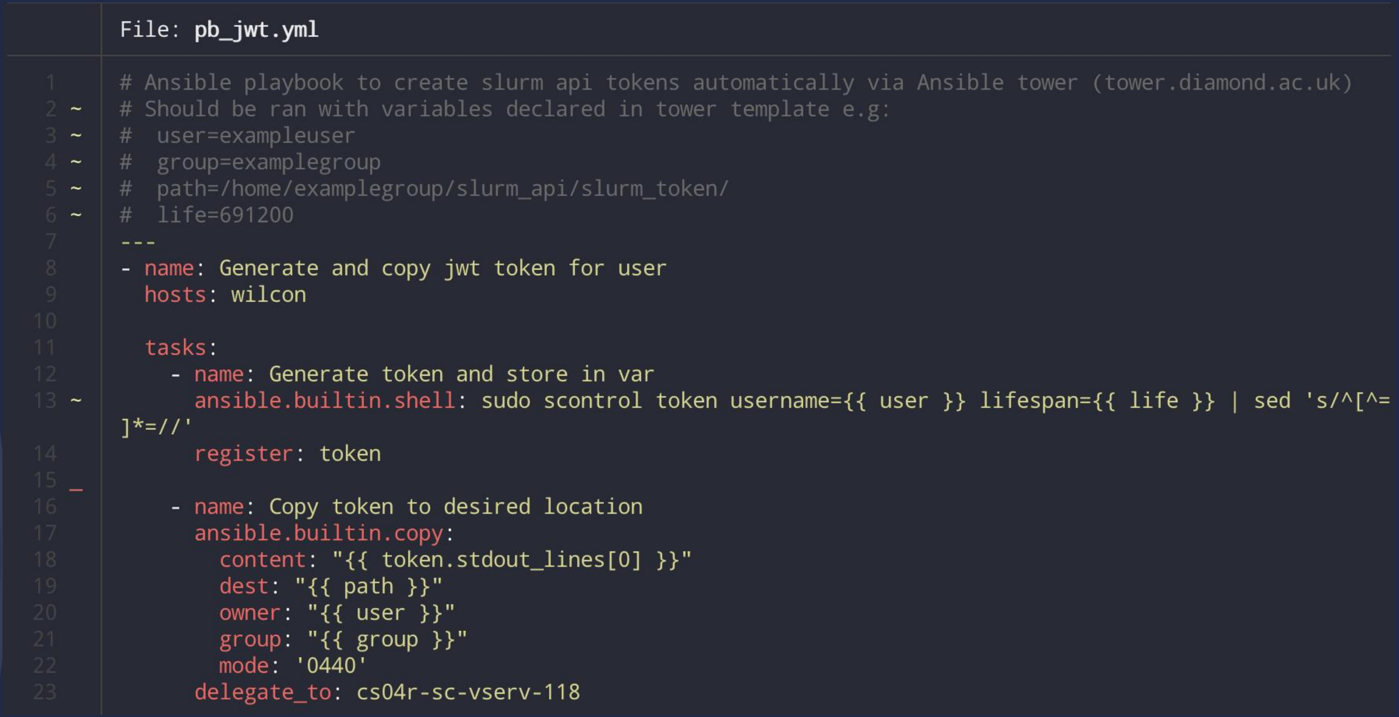
```yaml
- name: Slurm execution hosts
  hosts: all
  roles:
    - role: galaxyproject.slurm
      become: True
  vars:
    slurm_cgroup_config:
      CgroupMountpoint: "/sys/fs/cgroup"
      CgroupAutomount: yes
      ConstrainCores: yes
      ConstrainRAMSpace: yes
      ConstrainSwapSpace: no
      ConstrainDevices: yes
      AllowedRamSpace: 100
      AllowedSwapSpace: 0
      MaxRAMPercent: 100
      MaxSwapPercent: 100
      MinRAMSpace: 30
    slurm_config:
      AccountingStorageHost: "localhost"
      AccountingStorageType: "accounting_storage/slurmdbd"
      AccountingStorageUser: "slurm"
      AccountingStoragePort: 6819
      AccountingStoragePass: "/var/run/munge/munge.socket.2"
      AccountingStoreFlags: "job_comment,job_env,job_extra,job_script"
#      AuthAltTypes: "auth/jwt"
#      AuthAltParameters: "jwt_key=/var/spool/slurmctld/jwt_hs256.key"
      ClusterName: "hopper"
      DisableRootJobs: yes
      GresTypes: gpu
      JobAcctGatherType: "jobacct_gather/linux"
      MpiDefault: "pmix_v2"
      ProctrackType: "proctrack/cgroup"
```

Slurm REST API

- One of the main driving forces for move to Slurm
- Heavily used at Diamond for auto/live processing
- Uses JWT tokens for auth.
  - User generated with scontrol
  - Auto-generated via AWX
- Sits behind NGINX HTTPS proxy
- Seems that development of it moves faster than other parts of Slurm, this has led to several upgrades already

Tokens, Updates and AWX

- Reboot program triggers AWX to run update playbook on cluster nodes (WIP)
- AWX automatically creates and distributes JWT Tokens for several key functional data acquisition accounts



File: pb_jwt.yml

```
# Ansible playbook to create slurm api tokens automatically via Ansible tower (tower.diamond.ac.uk)
# Should be ran with variables declared in tower template e.g:
#  user=exampleuser
#  group=examplegroup
#  path=/home/examplegroup/slurm_api/slurm_token/
#  life=691200
---
- name: Generate and copy jwt token for user
  hosts: wilcon

  tasks:
    - name: Generate token and store in var
      ansible.builtin.shell: sudo scontrol token username={{ user }} lifespan={{ life }} | sed 's/^[^=]*=//'
      register: token

    - name: Copy token to desired location
      ansible.builtin.copy:
        content: "{{ token.stdout_lines[0] }}"
        dest: "{{ path }}"
        owner: "{{ user }}"
        group: "{{ group }}"
        mode: '0440'
      delegate_to: cs04r-sc-vserv-118
```

Stakeholder engagement

- Communicate change with lots of notice
  - Town hall meetings
  - Regular email updates
  - Slack channel
- Collaborate with software teams to make change as transparent as possible to users
- Additional support mechanisms
  - Weekly drop-in sessions
- Adapt plans based on user feedback

Recap

- Hopper Cluster
  - Hardware verification platform to evaluate workloads on future hardware – First cluster at DLS to use Slurm/RHEL8
  -
- Hamilton Cluster -> Wilson migration
  - RHEL 8 Upgrade
    - Rebuilt using pxe and kickstart using a very minimal build
  - Switch to Ansible config management
    - All post build config applied using ansible
  - Switch to Slurm for scheduling using ansible for cluster deployment
    - Using ansible role (https://github.com/galaxyproject/ansible-slurm) Features
    - Applied features to allow users to fine tune their job requirements
  - API via HTTPS/nginx proxy
    - Configured API via HTTPS nginx proxy to facilitate job submission from Kubernetes containers

Lessons

- Stakeholder comms and engagement is crucial
- Enforce limits from the outset
- Build own Slurm packages
- API compatibility between versions
- Adding remote resource is straightforward
- Fan noise is much less with RHEL 8!

Future Plans

- Migrate science cluster to Slurm
- Open OnDemand
- Sort out resource limits
- API upgrade to support
- Ansible best practice
- Still need to separate “live” and post-processing