Reproducing new and old operational systems on development workstations using containers

Tom Gale

Bureau of Meteorology, Australia

Post-processing

- Statistical correction, probability calibration and blending of data from weather prediction models to produce better forecasts
- Less computation and more I/O than most other earth sciences HPC applications
- No multi compute node processing
- Large numbers of files
  - 5 models * 30 parameters * 240 hourly timesteps * 30 run history = 1 million files
- Our current code is a mix of Python, C and shell script

Previous situation

- Production and staging on a pair of “mid-range” RHEL 5 Linux systems
- Development on a third system with same hardware and OS
- Software dependencies provided using environment modules

- Heavily I/O bound on mechanical disks
- Need for more performance to support higher grid resolution and more advanced post-processing methods
- RHEL 5 end of standard maintenance support period

New data intensive HPC cluster

- Production and staging on a pair of two Cray CS400 cluster systems and two SSD-cached GPFS filesystems with RHEL 7
- Software dependencies provided using environment modules
  - All software versions updated from those on previous mid-range system
- No development equivalent system

How to transition to new system without a matching development environment?

- Use containers to replicate both old and new systems on developer workstations
  - Copy the environment modules as-is into the container image
  - Bind system specific storage paths (eg. /data /scratch) so they are mapped to directories inside the user's home directory on the development system
  - Initially used Docker, moved to Singularity
  - Wrapper shell script for easy module initialisation, environment variables
  - Take care with proprietary software licenses
- Configuration option in Rose/Cylc suite to toggle job submission via PBS scheduler, so can run the suite on both workstation and cluster

Lessons learnt

- Have well tested instructions for development environment setup
- Singularity convenience defaults provide a gentle on-ramp for scientists used to working with modules
- Docker is fiddly, requires much learning/training to progress from follow-the-instructions to good understanding
- Containers work really well for replicating results across a range of systems
  - Use the same container image to run the CI test suite
  - Results should be bit-identical across a variety of underlying OS and hardware

Lessons learnt

- Container usage to replicate a machine is different to “good container practices”
  - Copy everything as-is, rather than trying to minimise container image size
- Maintain a second container image with the same software versions, built in a simple/safe way
- Minimise distribution of container images
  - Container build process should be scripted and run regularly
  - Version control the container build scripts
  - Keep a history of built images, but avoid using them unless needed
  - Rebuilding pulls in any updates to base Linux distro packages eg. security

Results

- Migrated to new data-intensive cluster system without having a development equivalent system available
  - Some suite work needed on staging system, but most software development was carried out on developer workstations
- System operators were worried about not having a development replica system for their familiarisation and testing, but trained them to use our development container setup

- BoM post-processing now running operationally on the data-intensive cluster system with 24x7 support

Even if containers aren’t available on production systems, they’re still great for development work and reproducibility