Introduction to Configuration Management

for Linux Clusters Institute Introductory Workshop

Mike Renfro, PhD

Information Technology Services, Tennessee Tech University

2025-02-11

Acknowledgments

- Thanks to Alexei Kotelnikov from Rutgers University

Stateful

- OS, settings, some/all applications for compute nodes stored on local disks
- Local files are preserved across reboots
- Primary challenge: **keeping things consistent across nodes**
- Common tools: **Ansible**, **Chef**, **Puppet**, **Salt**
- Model used in LCI workshop labs

Stateless

- OS, applications, settings for compute nodes pulled from central network server
- Root filesystem is a ramdisk, read-only NFS mount, or local disk
- Local files are **not** preserved across reboots
- Primary challenges: network traffic, load on NFS server, managing the contents of the root filesystem contents
- Common tools: **initramfs-tools**, **dracut**, **xCAT**, **Warewulf**
- Model used by OpenHPC project (and others)

Defining configuration management

The management and maintenance of operating systems, applications, and services via automation instead of manual intervention.

More formally:

- **declaring system state in a repeatable and auditable fashion**
- **using tools to impose state and keep systems from deviating**

Adjacent to, but not identical to, "orchestration" (Terraform, Kubernetes, etc.)

Benefits of configuration management

- Centralized, authoritative catalog of system configuration
- Automated enforcement of system state
- Ensured consistency among systems
- Mix and match components
- Collection of system "facts" to inform decision making
- Usually tracked with version control (Git or equivalent)

Cautions about configuration management

- You need some understanding of the more manual processes for systems administration tasks
- Analogous to needing to know how to calculate a formula on paper before writing a program to calculate the same formula
- Automation should be used to make things more consistent, not to just blindly copy/paste

Historical: scripts, make (and scripts)

The DevOps folks didn't invent "infrastructure as code" or "defined state":

- Systems administrators have been writing shell scripts for 50+ years
- http://www.infrastructures.org has references going back to 1988
- Clever use of network booting, make, shell commands, and output files can take you a long way

Modern tools

Most modern tools use:

- a syntax describing desired client system state (usually in a more abstract sense than shell commands)
- facts the client knows about itself (hostname, OS version, "am I a physical or virtual machine?", . . .)
- ordering of steps required to reach the desired system state
- comparison of current system state to desired system state to avoid excess/redundant workj (idempotence)

Sampling of prominent tools

Puppet (2005–) Ruby-based, with agent installed on clients. Communication via HTTPS to central server (port 8140). Ruby-like domain specific language (DSL) for describing state.

Chef Infra (2009–) Ruby-based, with agent installed on clients. Communication via HTTP and HTTPS to central server (ports 443 and 9683). Uses literal Ruby instead of a DSL.

Salt (2011–) Python-based, with agent installed on clients. Communication via AES-encrypted ZeroMQ to central server. Uses YAML for its DSL.

Ansible (2012–) Python-based, no agent installed on clients. Settings pushed from central server or admin workstation over `ssh`. Uses YAML for its DSL.

How Ansible works

- Software installed on any administrative endpoint (`pip install ansible`)
- Uses public-key or host-based `ssh` authentication to managed clients
- Runs several `ssh` connections at once for managing multiple clients
- Elevates privileges with `sudo`
- Builds custom Python scripts to copy to, and execute on, clients
- Lower-level tasks can be composed into playbooks, and playbooks can be composed into roles.
- In HPC, Ansible could be used to configure stateful nodes, or to configure the central server managing stateless nodes.

An Ansible story

- Half-day tutorial at PEARC25 — "OpenHPC: Beyond the Install Guide"
- Similar infrastructure requirements to LCI
- 31 small clusters hosted at NSF's Jetstream2 OpenStack facility
- Created nodes, networks, IPs with Terraform
- Needed to configure each management node (head node) as if we'd just finished installing OpenHPC using instructions from the install guide
- OpenHPC provides an automated installation script with commands generated from a common source with documentation
- I'd already run through the manual steps several times
- I'd already run the provided script several times

Example file structure

For configuring an OpenHPC system management server (head node):

```
% find ansible -type f
ansible/ansible.cfg
ansible/hosts.ini
ansible/0-undocumented-prereqs-unrelated-settings.yaml
ansible/2-install-base-os.yaml
ansible/3-install-openhpc-components.yaml
ansible/a0-installation-template.yaml
ansible/a1-run-recipe.yaml
...
```

ansible.cfg

```
[defaults]
inventory = hosts.ini
remote_user = rocky

[privilege_escalation]
become = true
```

hosts.ini

If an LCI cluster was running OpenHPC:

```
[head]
lci-head-[01:55]-1.ncsa.cloud
```

Tasks for Excerpt of OpenHPC Install Guide 3.2, Section 2

```
# 2-install-base-os.yaml
- name: 2. Install Base Operating System
  hosts: head
  tasks:
    # [sms]# echo ${sms_ip} ${sms_name} >> /etc/hosts
    - name: Add hostname to /etc/hosts
      ansible.builtin.lineinfile:
        path: /etc/hosts
        line: "172.16.0.1 {{ ansible_hostname }}"
    # [sms]# setenforce 0
    - name: Disable SELinux
      ansible.posix.selinux:
        state: disabled
```

Tasks for Excerpt of OpenHPC Install Guide 3.2, Section 3

(Part 1 of 2)

```
# 3-install-openhpc-components.yaml
- name: 3. Install OpenHPC Components
  hosts: head
  tasks:
    - name: Check if crb repository is enabled
      ansible.builtin.shell: |
        dnf repolist crb | grep -q enabled
      register: crb_enabled
      changed_when: false
      ignore_errors: true
```

Tasks for Excerpt of OpenHPC Install Guide 3.2, Section 3

(Part 2 of 2)

```
- name: Ensure the crb repository is enabled
  ansible.builtin.shell: |
    dnf install -y dnf-plugins-core
    dnf config-manager --set-enabled crb
  when: crb_enabled is failed
```

Running a Play

```
% ansible-playbook 2-install-base-os.yaml
```

Conclusions

- Configuration management can provide:
  - better consistency across systems with common attributes
  - fewer manual errors
  - less reinventing wheels
- The specific technology doesn't matter as much as the community support, both locally and globally