Lustre Management with Ansible

Rick Mohr (University of Tennessee)

Nathan Grodowitz (Oak Ridge National Lab)

Christopher Layton (Oak Ridge National Lab)

What is CADES?

- CADES is the Compute and Data Environment for Science
- An effort at Oak Ridge National Lab to centralize data and compute on various platforms
- Currently has three primary center-wide Lustre file systems, one for each protection zone (Open Research, Moderate, and HDSI)
- Zones incorporate VM infrastructure, cluster compute, and specialized HPC resources
- Specialized HPC includes Cray Urika GX, SGI UV300, and Cray XK7 systems
- Compute cluster consists of 425 Cray CS400 nodes spread among the protection zones
- VM Infrastructure consists of a self-service OpenStack environment for research and computational resource

Admin Tasks

Manage the node state

- Config files
- Enable/disable services
- Install/uninstall packages
- Maintain consistency
- Aim for “static state”

Manage the file system

- Many of the same tasks as managing the nodes
- Runtime administration
  - MDT/OST failover
  - Disable OST
- One-shot tasks
  - Format file system

Why Ansible?

- Configuration management for node images
  - In particular, support for chroot environments
- Uses ssh under the hood
  - This what a lot of homegrown Lustre scripts do anyway
- Avoid duplication of information
  - Why put same Lustre info in Ansible and homegrown scripts?
  - Can we make Ansible the definitive source for Luster info?

Ansible for Node Administration

Diskless System Basics

- All Lustre systems in CADES utilize a “diskless” hybrid shared NFSRoot file system
- Allows systems to maintain coherency between all nodes and provide a stateless environment for security and stability
- CADES utilizes GeDI (Generic Diskless Installer) for its hybrid environment across all compute nodes and Lustre nodes
- Hybrid NFSRoot utilizes a read-only NFS mount for the majority of the file system but allows for some writeable areas stored in ramdisks
- Utilize /var, /tmp, and /etc as localized ram disks with /root being a shared writeable NFS mount
- GeDI copies files from NFSRoot on boot to create identical systems, and then runs scripts to customize specific files

Ansible Diskless Management

- GeDI creates a basic root file system image on the NFS server
- Ansible used to fully customize the image to create easily repeatable environments
  - Ansible has provisions for native chroot vs other config management tools (puppet, chef) which require modules
- Some Ansible modules utilize pseudo-tty functionality of SSH and won't be compatible with chroot
  - Ex – service, command

General Design

- Divide roles up by most general to most specific
  - Allows for optimal reuse of code
  - Example
    - Roles for base nfsroot config shared by lustre and compute nodes
    - Additional roles to specialize the nfsroot for lustre or compute node
- Roles used on both normal and nfsroot-based servers should use methods that work in chroot environment
  - Ex - LDAP role copies script to /tmp on node then executes

Playbook Example

- Shortened version of playbook for Lustre installation

```
- hosts: /images/gedi-hydra-lustre-ansible
  remote_user: root
  connection: chroot
  roles:
    - base_auth-ldapx
    - base_setup-hosts_file
    - storage_lustre-hydra_base_config
    - storage_lustre-configure_zfs_repo
    - storage_lustre-dkms_installs
    - storage_lustre-configure_srp
```

Ansible for Lustre Administration

Disclaimer
(a.k.a – I am not an Ansible expert)

- Playbooks, roles, etc. are a work in progress
  - Design decisions presented here may not be final
  - Some choices might be site-specific
  - Missing some edge cases and error checking
- Goal is to generalize this work as much as possible to make it usable by others
- Feedback is certainly welcome

Lustre Administration Tasks

- Manage zpools
  - Create/destroy
  - Import/export
  - Status
- Manage Lustre targets (MGT/MDTs/OSTs)
  - Format targets
  - Start/stop targets
  - Handle target failover
  - Check target status

General Design

- Use roles to group tasks based on functionality
  - zpool stuff in one role, Lustre stuff in another
- Host-specific parameters in /etc/ansible/host_vars/
- Default parameters in /etc/ansible/roles/$ROLE/vars/
- Use tag to select action (and only run when tag is specified)
- Use --limit to restrict actions to specific host(s)
  - Use -e to further restrict targets of actions

Managing zpools (Config)

- /etc/ansible/host_vars/or-oss-d1/zpools.yaml

```
---
zpools:
  - name: oss-d1-ost0
    device: /dev/mapper/test-ddn-d-l00
  - name: oss-d1-ost1
    device: /dev/mapper/test-ddn-d-l01
  - name: oss-d1-ost2
    device: /dev/mapper/test-ddn-d-l02
```

Managing zpools (Role)

- /etc/ansible/roles/zpool/tasks/main.yaml

```
---
- set_fact: abort=true

- name: Create a zpool
  command: /sbin/zpool create {{ item.create_options |
         default(zpool_create_options) }} {{ item.name }} {{ item.device }}
  with_items: "{{zpools}}"
  when:
    - abort is undefined
    - item.name | match(target|default(".*"))
  tags: create_zpool
```

Managing zpools (Playbook)

- Create a very simple playbook for testing

```
- hosts: zfs-nodes
  remote_user: root
  roles:
    - zpool
```

Managing zpools (Examples)

- Create all zpools on all hosts

```
ansible-playbook zpool.yaml --tag create_zpool
```

- Create all zpools on single host

```
ansible-playbook zpool.yaml --tag create_zpool --limit or-oss-d1
```

- Import single zpool on single host

```
ansible-playbook zpool.yaml --tag import_zpool --limit or-oss-d1
-e target=oss-d1-ost1
```

Format Lustre (Config)

- host_vars/$HOST/lustre.yaml

```
---
osts:
  - name: ost0
    index: 0
    zpool: oss-d1-ost0
  - name: ost1
    index: 1
    zpool: oss-d1-ost1
```

- roles/lustre/vars/main.yaml

```
---
lustre:
  fsname: ansible
  backfstype: zfs
  mgsnode: 172.23.85.24@tcp0
```

Format Lustre (Role)

- /etc/ansible/roles/lustre/tasks/main.yaml

```
- name: Format MDTs
  command: /usr/sbin/mkfs.lustre --mdt --backfstype={{ lustre.backfstype }}
           --fsname={{ lustre.fsname }} --index={{ item.index }}
           --mgsnode={{ lustre.mgsnode }} {{ item.zpool }}/{{ item.name }}
  with_items: "{{ mdts }}"
  when:
    - abort is undefined
    - mdts is defined
    - item.name | match(target|default(".*"))
  tags:
    - format_mdts
    - format_lustre
```

Format Lustre (Playbook)

- Test the role using a simple playbook

```
- hosts: lustre-servers
  remote_user: root
  roles:
    - lustre
```

Format Lustre (Example)

- Format entire file system

```
ansible-playbook lustre.yaml --tag format_lustre
```

- Format all MDTs

```
ansible-playbook lustre.yaml --tag format_mdts
```

- Format single OST

```
ansible-playbook lustre.yaml --tag format_osts –e target=ost0
```

- Formate all OSTs on a single host

```
ansible-playbook lustre.yaml --tag format_osts --limit $HOST
```

Start Lustre (Role)

- /etc/ansible/role/lustre/tasks/main.yaml

```
- name: Start OSTs
  command: mount -t lustre {{ item.zpool }}/{{ item.name }}
           /tmp/lustre/{{ item.name }}
  with_items: "{{ osts }}"
  when:
    - abort is undefined
    - osts is defined
    - item.name | match(target|default(".*"))
  tags:
    - start_osts
    - start_lustre
```

Start Lustre (Example)

- Start the entire file system

```
ansible-playbook lustre.yaml --tag start_lustre
```

- Start all OSTs on a host

```
ansible-playbook lustre.yaml --tag start_osts --limit
or-oss-d2
```

Future Work

- Additional functionality and generalization
  - Add support for ldiskfs
  - Support arbitrary layouts for zpools
  - Support failover parameters
- Support multiple Lustre file systems
- Investigate the use of custom facts or modules
- Tie into root image management
  - Use config variables to generate Lustre files

Conclusion

- Ansible is a good configuration management tool for NFSRoot images due to its support for chroot
  - Although some small concessions must be made
- Ansible is also a promising tool for Lustre administration beyond standard configuration management
- Lustre playbooks can help automate many Lustre tasks
  - More work still needs to be done to generalize playbooks
  - Still some questions on how far this work can go

Acknowledgments

This research used resources of the Compute and Data Environment for Science (CADES) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Questions?