Configuring and Managing Multiple Shasta Systems
Best Practices Developed During the Perlmutter Deployment

James Botts - NERSC
Zachary Chrisler - HPE
Aditi Gaur - NERSC
Doug Jacobsen - NERSC
Harold Longley - HPE
Alex Lovell-Troy - HPE
Eric Roman - NERSC
Cory Snavely - NERSC
Chris Samuel - NERSC
Development to Production

- Did you know?
  - Your TDS is the most exciting system on the floor
  - Your production system is a totally boring endpoint
- Recipes static, must test recipe build through operations on TDS
- TDS iteration → regression testing → iterate on TDS → production

Have a working multi-system, many-contributor administration model for XC.

How to do this for Shasta?
With a central goal of continuous operations in mind, it's very important to have sufficient test resources to both test potentially risky procedures as well as provide long-term and short-term development platforms.
Anatomy of a Shasta System

Management Nodes

- master
- worker
- storage

Managed by CSM

Managed Nodes

- Application Nodes
- Login Nodes
- Compute Nodes

Managed by CSM

All nodes in shasta have a job! Nowhere is appropriate for interactive management of the system.
Use Cases for Interactive Systems Management

- **System Configuration**
  - Need a central location to house configuration, secrets, and manipulate the system securely
  - Control administrator access and assure that NERSC security tokens are used
    - NERSC security procedures require administrators log in to a specific admin bastion host

- **Interactive Management**
  - Administrative activities (reboot, debug, etc)
  - Log analysis

- **Store and manage extremely powerful system secrets**
  - Admin ssh user certificate authority
  - Kubeadm generated Kubernetes certificates
  - System root certificates
**NERSC Solution: Management VM**

Highly available VMWare deployment in secured portion of NERSC network

- **OS Image**
- **Persistent Data Storage**
- **MFA Bastion Host**

**NERSC Network**

- **smnet switches**
- **rosetta switches**

**Compute Nodes**

- **ncn-s***
- **ncn-m***
- **ncn-w***

**Application Nodes**

- **CAN/CMN**
- **HSNBorder/CHN**

Dashed lines indicate an interface not used by management VM(s)

- **console server (avocent)**
- **vlan tagged**

**Dashed lines indicate an interface not used by management VM(s)**
Interactive Management

● ssh
  ○ Rewrite NCN/UAN/CN to only accept root login from VM-vault signed ssh certificates and CSM-vault signed ssh certificates
  ○ Only enable root login secured locations

● kubectl
  ○ move /etc/kubernetes/admin.conf to VM and remove from system
  ○ ssh port-forward port 6442 from randomly selected master node to access API gateway

● cray / Istio API Gateway / sat
  ○ Install cray-cli and sat dependencies, (work already done)
  ○ Configure to use api CAN interface, e.g. api.perlmutter.nersc.gov
Configuration Management: git

USING GIT FOR MANAGING CFS CONFIGURATION

- Stores Ansible to apply to nodes at lifecycle events
- All Ansible in git repositories with branches to allow site customization
- Ordered configuration management across multiple repositories
- CFS sessions as part of pre-boot Image Customization as well as post-boot Node Personalization

Each layer is provided by a separate gitea git repository.

Source: HPE CSM v1.4 Overview Presentation
Configuration Management: git

NERSC Gitlab
nersc-cle git repository

- shasta/ansible/cos-config-management
- shasta/ansible/cpe-config-management
- shasta/ansible/sma-config-management
- shasta/ansible/nersc
- shasta/alvarez_vars
- shasta/muller_vars
- shasta/perlmutter_vars

On-system Gitea

- cray/cos-config-management
- cray/cpe-config-management
- cray/sma-config-management
- crayvcs/nersc
- crayvcs/inventory

Appropriate \texttt{<system>\_vars} synced to inventory repo

\texttt{./update\_system\_ex.py} --branch=development
Configuration Management - Secrets

VM

vault

mgrkv/

bmc

ssh/

adminca

hostca

root

syskv/

ssh/

hostkeys

csm

cmm

slurm/

...

shasta cray-vault

vault

secret/

hms-creds/

x1000c0

...

cos

uan

syskv/

ssh/

hostkeys

csm

cmm

slurm/

...

syskv is sync'ed from the VM to cray-vault (using kubectl port forwarding) for use with CFS. cray-vault also gets hashed passwords for deployment on the system.

mgrkv has secrets that are only known on the manager VM, such as plaintext passwords, private keys for the host and admin CAs.

This requires policy changes by patching the Cray "vault/cray-vault" kubernetes object.

We have an automated process for patching this and keeping it up-to-date.

[hash password upon sync]
Configuration Management

- **RPMs**
  - "nersc" product layer RPMs synced to appropriate on-system nexus repositories using /update_system_ex.py (direct integration with nexus API)

- **Helm Charts**
  - NERSC-custom charts are stored in `nersc-cle` git repository
  - Deployed with Cray's `loftsman` from manager VM (leveraging end-user's privileges with kubernetes)

- **Containers**
  - Based on parse of nersc-cle charts, use skopeo to sync containers from external source, registry.nersc.gov, or VM-constructed container to on-system nexus

The container/chart synchronization is a HUGE area for discussion, more depth in paper!
$ cd nersc-cle
$ git checkout development
$ ./update_system_ex.py --branch=development
  # record timestamp as `suffix`
  # syncs secrets
  # writes feature/a ansible directories to dmjtest branches in gitea
  # generates CFS configuration objects, uploads using CFS API
    # compute-development-<suffix>
    # login-development-<suffix>
    # gateway-development-<suffix>
  # generates bos-sessiontemplates-<suffix> with unconfigured images
$ ./shasta/scripts/build_latest_images.sh development
$ cd bos-sessiontemplates-<suffix>
$ ..//shasta/scripts/update_bos_latest_images.sh -i development -d development
  # generates and uploads usable BOS sessiontemplates
  # compute-development, login-development, gateway-development
$ cray bos session create --template-uuid login-development --operation reboot
Conclusions

The manager VM:

- Adds layers on control to add features and enable rich role-based administration capabilities
- Leverages keycloak-enabled role-based workflows by allowing RBAC integration and secrets delegation
- Provides a locale for interactive development and administration
- Enables a straightforward administrative and development workflow that sets relatively simple norms on the system
- Provides a high-fidelity method to develop on a development system and deploy on a production system